

Interested in learning more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Packer Analysis Report-Debugging and unpacking the NsPack 3.4 and 3.7 packer.

I language (in this paper we will use C) so that an analyst can better understand the workings and purpose of the packer....

> **Copyright SANS Institute** Author Retains Full Rights



Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer.

GIAC GREM Gold Certification

Author: Craig S Wright Name, craig.wright@information-defense.com

Advisor: Antonios Atlasis

Accepted: 17 Jun 2010

Abstract

The following report is an analysis of the NsPack 3.4 and 3.7 packer program (by North Star/Liu Xing Ping). Unfortunately, many commercial antivirus vendors have not adequately analyzed the NsPack binary and compression routine. This has led to the unfortunate situation where major anti-malware vendors are misclassifying NsPack (and other PE Packers) as a Trojan This paper provides instructions on how to determine if NsPack was used and on how to unpack NsPack 3.4 and 3.7 using the OllyDbg debugger. The OllyScripts used in this process as well as the custom plug-ins required to automate the process are provided. The custom plug-ins that are required are provided with the source code in the appendixes. This process includes instructions on how to fully restore the import table so the file can be restored to its original state and executed. This also incorporates an analysis of the packer as well as the means to create an unpacker manually and to calculate the OEP.

As NsPack remains one of the most common PE Packers with high rates of reported use and discovery (NsPack is in the top 10 list for PE Packers used on malware samples stored in the AML database) and with the relatively low accuracy rates for detection, it is important that security professionals gain a more comprehensive understanding of this and related packers. For this reason, this paper has been written as a broad analysis of NsPack that will help both the novice and experienced anti-malware professional.

1. Introduction

This document provides instructions on how to unpack NsPack 3.4 and 3.7 using the OllyDbg debugger. The OllyScripts used in this process are included in the appendixes. The custom plug-ins that are used to automate the procedure are provided with the source code. This paper also includes instructions on how to fully restore the import table so the file can be restored to its original state and executed. This is continued further with instructions on how to convert the machine code (assembly language) into a higher level language (in this paper we will use C) so that an analyst can better understand the workings and purpose of the packer.

Unfortunately, many commercial antivirus vendors have not adequately analyzed the NsPack binary and compression routine. This has led to the unfortunate situation where major anti-malware vendors are misclassifying NsPack (and other PE Packers) as Trojans (figure 3.1). In section 6 we will show through both static analysis and dynamic execution that NsPack is not a Trojan but a simple PE compression utility.

NsPack remains one of the most common PE Packers with high rates of reported use and discovery. Oberheide, Bailey, & Jahanian (2009) used the Arbor Network's Arbor Malware Library (AML) to analyze the distribution of PE Packers. The results are displayed in figure 3.2. In these tables we see that NsPack is in the top 10 list for PE Packers used on malware samples stored in the AML database.

While this paper focuses on NsPack, the general principles are designed to enable the reader to learn how to apply the process to other PE Packers. NsPack 3.x is a simple compressor. It does not support Anti-Debug or Anti-Disassembly features. It used configurable section names (defaulting to .nsp). In this document we will walk through both the NsPack 3.4 and 3.7 versions.

Although we will touch on many topics, it is presumed that the reader has a good knowledge of the following:

1. PE file format. Microsoft provides a couple of excellent sources of

knowledge for the budding code analyst¹.

- a. Kath, Randy (1997) "*The Portable Executable File Format from Top to Bottom*" Microsoft Developer Network Technology Group. (Available from: http://www.pelib.com/resources/kath.txt)
- b. Pietrek, Matt (1998) "Windows System Programming Secrets", John Wiley & Sons Inc, USA
- c. Pietrek, Matt (1994) "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format" Microsoft Developer Network Technology Group. (Available from: http://msdn.microsoft.com/en-us/library/ms809762.aspx)
- 2. An acquaintance with SEH (Structured Exception Handling) is required.
- 3. Basic knowledge of the Win32 API (or access to a good guide). In particular, the following APIs are a large part of most packers:
 - a. CreateProcess,
 - b. GetCurrentProcessID,
 - c. GetModuleHandleA,
 - d. GetProcAddress,
 - e. OpenProcess,
 - f. ReadProcessMemory,
 - g. VirtualAlloc,
 - h. VirtualFree, &
 - i. WriteProcessMemory.

The reader should also have a good knowledge of ASM (assembly language). The following sites provide an excellent introduction to this topic:

a. Computer Structures C335 Syllabus (Doyle, 2009),

¹ There are many good online ASM, PE and low level programming tutorials online. One Such example is " Iczelion's tutorial Series" at http://win32assembly.online.fr/tutorials.html

http://homepages.ius.edu/jfdoyle/C335/syllabusc335fall2009.htm

 b. Iczelion's Win32 Assembly Homepage, http://win32assembly.online.fr/tutorials.html

Each of these sites provides a step by step introduction to ASM that is designed to teach the novice assembly coder the fundamentals. Section 9 (Appendix) provides a disassembled code section for NsPack that can be used as an exercise in practicing reversing this packer.

1.1. Tools required:

There are many good debuggers and dissassemblers (including HexRays IDA Pro from the commercial stable). This paper has relied heavily on a use of the following tools:

OllyDebug v1.10,

☑ OllyDump plug-in,

☑ Import ReConstructor 1.6, and

OllyScript Plug-in.

1.2. Why Study PE Packers?

As Guo, Ferrie & Chiueh (2008) note;

"Instead of directly obfuscating malware code, malware authors today heavily rely on packers, which are programs that transform an executable binary into another form so that it is smaller and/or has a different appearance than the original, to evade detection of signature-based anti-virus (AV) scanners. In many cases, malware authors recursively apply different combinations of multiple packers to the same malware to quickly generate a large number of differentlooking binaries for distribution in the wild. The fact that more and more malware binaries are packed seriously degrades the effectiveness of signature-based AV scanners; it also results in an exponential increase in AV signature size, because when an AV vendor cannot effectively unpack a packed threat, it has no choice but to create a separate signature for the threat."

Over 80% of malware is packed (Guo, Ferrie & Chiueh, 2008). The growth of

cybercrime will only lead to more malware and as these products are commercialized, the authors are likely to make more effort (Debrosse, 2009) to create software that is more difficult to detect. By creating packers, the cybercriminal can increase the costs of detecting the software and hence increase their expected returns. For this reason it is important that information security professional understand PE Packers whether they work in the AV industry or for general commercial ventures. In the former instance, an understanding of the packer problem is only likely to become more critical and in the later, an understanding of packers will help the security professional to gain an understanding of the problem in its true extent.

For the majority of security professionals, analyzing malware (and hence packers) becomes most critical when an incident has occurred. Knowing how an attacker has obscured their software can be the key in any successful incident handling exercise involving malware, which is nearly all incidents these days and is only growing worse.

Guo, Ferrie & Chiueh (2008) report that the typical way an AV vendor such as Symantec handles packers involves:

- 1. Recognize the packer,
- 2. Identify the packer,
- 3. Create a recognizer, and
- 4. Create an unpacker.

This paper will incorporate all of these steps for the NsPack packer.

1.3. Paper sections

This paper has been divided into several sections. The first section following the introduction (section 2, What is a packer) details the functions and operation of a PE packer. This section provides the basic functions of a PE packer as well as a synopsis of the PE format. Section 3 (NsPack) provides a synopsis of the NsPack compression function. This section includes an analysis of the execution and operation of the NsPack compression utility and the creation of a packed executable.

Section 4 (Determining the packer), provides a walk-through procedure for

determining the packer used on a PE Executable file with a detailed step-by-step guide to using the tools. The next section (Unpacking in Olly) is a guide to manually unpacking an NsPacked PE executable using the Olly Debugger. This is followed by a manual process that can be used to correct the IAT and make the recovered file executable (for further analysis) in section 6.

Section 7 (NsPack itself) is a detailed analysis of the NsPack compression function and executable. This section goes into the structure and operation of the compression program. In section 8 we draw our conclusions, while, the final section (Appendix) is optional and provides a detailed analysis of the NsPack routine and processes. This section is designed for those wishing to create a functional reversing routine for this packer.

2. What is a Packer?

A 'Packer' is a compression routine that compresses an executable file. These programs originated to minimize disk space and make downloads faster and derive from valid uses such as that of WinZip's compressed executable function. They also obscure the original file and make it more difficult to match the file signature of a compressed file.

Packer programs have been introduced into the world of malicious software so that the authors of the malicious code can extend the expected life of the software. Many valid software authors have used packers to make it more difficult and costly to reverse engineer their software. To this end, packers have become more complex over time and many incorporate complex routines to encrypt the executable that they are protecting.

The packer takes the original program and compresses it. The compressed executable is moved to the data section of the newly created file. As the data is compressed, the PE header and the section header of the original file can no longer be used to run the executable. As such, the packer will add a stub function. This is designed to decrypt and decompress the packed file from the data section of the packed executable into memory where the original file is reconstructed.

Basically, the 'executable' part of the program is a simple routine that is designed to decompress the original file (or at least something that approximates it) into memory and to resume execution at the OEP (Original Entry Point) of the uncompressed program.

Packers generally create a resultant executable that is smaller than the original file. They also change the signature of the file and any hash that can be used to create a simple matching engine (hence making anti-virus software more costly). This comes at a cost. The packing process itself can create a signature that leaves the files being flagged as suspicious (there are valid software products that are packed with unusual packers). There is also a run-time cost as the file needs to be unpacked and/or decrypted before it is run, a process that consumes more cycles than the original executable. This may or may not be an issue to a malicious code author.

Some packers only reconstruct selected calls at a time and are more difficult to dump from memory (an example of such a packer is Themida VM). Many packers also have complex routines to stop memory debugging (or at least make this more difficult and hence costly). NsPack is not one of these and this is outside the scope of this paper.

The weakness of any packer is simple; it needs to be unpacked into memory to be useful. The best that a packer can do is to make the analysis more costly.

2.1. A Quick overview of a PE-COFF Executable File

All 32 and 64 bit executable files in the Microsoft Windows family of operating systems use the Portable Executable (PE) structure. PE is the native file format of all Win32 executable programs. It has a similar specification to the Unix/Linux COFF². It is essential to have an understanding of the PE specification when analysing malware on Windows.

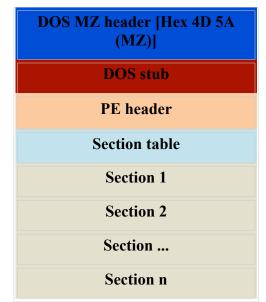


Figure 2.1. The general layout of a PE file.

Portable Executable refers to the universal nature (on the Windows platform) of the file format for executable Windows programs. This is universal as the PE format used by the WIN32 platform is valid on all platforms (Intel, PowerPC etc).

All windows executable files other than VxDs and 16-bit Dlls are created using the PE file format. This definitely includes all malware and packers on Windows.

² Common Object File Format

Figure 2.1 displays the standard PE file format. All PE files begin with a DOS MZ header. When a file is packed, the original executable is compressed and saved as a section in the new (packed) executable file. This is displayed in figure 2.2. Here the compressed data section contains the original executable file.

The new program is really just a decompression routine designed to load the original executable into memory.

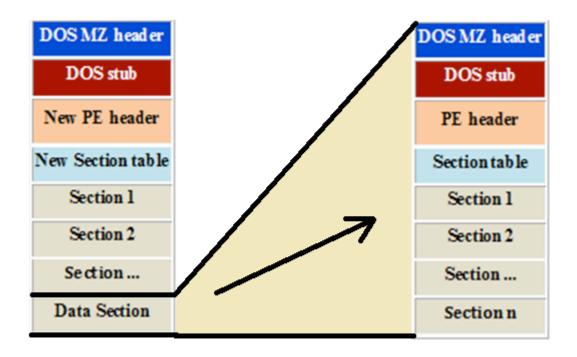


Figure 2.2. The layout of a packed PE file.

When the executable runs, the decompressed version of the original file is loaded into the computer memory.

2.1.1. PE-COFF and the PE Header

The first section of any PE file is the MZ header. This is named after the developer of this format, Mark Zbikowski. The MZ Header starts with the Hex value "4D 5A" and commonly contains a string such as "*This program cannot be run under DOS*" or "*This program must be run under Windows*"³.

The PE Header follows the DOS MZ header. This section contains the data

³ Microsoft has the full PE COFF Schematics available from

http://www.microsoft.com/whdc/system/platform/firmware/PECOFFdwn.mspx

structures for the common execution settings of the file. The PE header is specified in the file at offset 0x3C. This is a 4-byte signature that identifies the file as a PE format image file to the Windows Operating System. This signature is "PE 0^0 " which is "50H 45H 00H 00H" in Hex and represents the letters "P" and "E" followed by two null bytes.

The PE Header includes the following information:

- Machine Machine
- Number Of Sections
- ▼ Time Date Stamp
- 🕱 Pointer To Symbol Table
- Number Of Symbols
- ☑ Size Of Optional Header
- M Characteristics

The PE header is a general term for the PE-related structure also termed the IMAGE_NT_HEADERS. The PE header used by the PE loader which determines the starting offset of the PE header from the DOS MZ header. Windows can actually leave out the DOS stub and start execution at the PE header (this is the true file header). The MZ is used as this allows Windows to determine the type of file more easily.

2.1.2. Section Table

The section table is a reference to the various sections contained in the PE file. The section table has the information displayed in Figure 2.3. This includes the name of each section, the offsets and a set of characteristics describing the file section.

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Charactaristics
.nsp0	0004E000	00001000	0004E000	00001000	F2000060
.nsp1	000131C6	0004F000	000131C6	0004F000	E0000060
.nsp2	0000CEB8	00063000	0000CEB8	00063000	E0000060

Figure 2.3. The sections from an NsPack packed file.

The section table maintains the section permissions for the file. These are used by

Windows when memory pages are allocated. Gustavo Duarte (2009)⁴ has an excellent series of posts on how Windows load s the various page sections into memory for those wanting to learn more about this process.

2.1.3. Sections

Each of the sections that are maintained in the section table contains information related to how the program runs. Each executable section is a collection of data used by the system. Generally, the compiler and linker that are used to turn source code into machine code will group the sections into as few sections as possible. Each of these sections is based on the characteristics of the file. In general, this will be based on the data section permissions (such as a Read, Write and Executable page flag).

In order to preserve memory and make the program run more efficiently, most compilers try to limit the number of sections and a standard PE file may contain the following sections:

- ₩.text
- 🕅 .data
- .rsrc
- I.reloc

Many packers create more segments than are necessary for the program to run as these are more concerned with making the process of reverse engineering the file as complex (and hence economically expensive) as possible. In the example used in this paper, this is not the case. NsPack is a comparatively simple compressor and the packed file will usually be contained in three (3) sections. This is configurable in NsPack and the file may have a number of additional sections so the discovery of more than three sections does not preclude NsPack as the packing engine as we will detail later in this paper.

The grouping of data into sections is based on the common attributes and not on a

⁴ "How The Kernel Manages Your Memory", http://duartes.org/gustavo/blog/post/how-the-kernel-managesyour-memory

logical basis. Each section can contain either "data", "code" or some other logical concept as long as they have identical attributes. A block of data that is read-only should be in a section that is marked as read-only. This can be both data and code, as long as it cannot be changed.

The PE loader (or dynamic linker) begins by mapping the sections into the system memory. The loader then examines the attributes of the sections in the executable. Each memory block within a particular section is then set with the designated attribute. The dynamic loads and links the shared libraries for the executable when executed.

2.1.4. Loading a PE File into Memory

The main stages used to load a PE file into memory (although grossly simplified) are:

- 1. The PE-COFF file is executed (by a user or process). The system starts by examining the DOS MZ header and extracts the offset for the PE header if this exists. On finding the offset, the system jumps to the PE header.
- 2. The PE loader next ensures that the PE header is valid. In the case where the PE header is invalid, the systems will error. Otherwise, the system jumps to the past of the PE header to the start of the section table.
- 3. The PE system next inputs the section table into memory mapping the sections from the table into the systems memory. The attributes of the sections as listed in the section table are mapped in memory.
- 4. Once the PE file has been mapped into the system's memory, the dynamic linker (the PE loader) moves to the logical sections of the PE file. The next jump is to the import table.

The dynamic linker maps each of the sections into memory assigning the specified permissions to each of the sections.

2.1.5. The Import Address Table (IAT)

The dynamic linker moves to the IAT after the PE Header. The system uses the

IAT as a lookup table to find functions that are located in different modules used by the application. The IAT exists as the system does not have the memory location for all of the libraries it uses. Rather, an indirect jump is necessary whenever an API call is completed.

The dynamic linker loads the various modules into memory and connects them together and then writes jump instructions into the IAT slots. The system is then configured such that it is positioned at the memory locations of the consequent library functions. This does have a negative impact on the performance of the system as additional jumps are made outside the calling executable (in place of intra-module calls).

Dynamic libraries (usually in the form of a DLL in Windows) increase the maintainability of the program by removing redundancy. This allows the same code to be reused and updated easily unlike when all executables are separately maintained in statically linked files (static linking builds the code into each executable. This results in larger code as well as a greater requirement to maintain and patch individual programs). OBC (Object Orientated Code) allows the creation and use of common libraries in place of statically linked code. As such, a single DLL can be called from numerous programs. This is extremely beneficial as the user can patch a single file in place of hundreds (or more) statically linked files.

Some examples of calls made by the IAT include those files set from the code as external calls. For instance, a C# program using the following statement could call the "Sleep()",GetDisk(), FreeSpace or " GetCommandLine()" functions⁵:

using System;

For instance, the following call,

PUSH EBP

CALL DWORD PTR [004933FA]

Will return the value stored by the system at location, 004933FA. Looking at this in a hex editor will return a NULL. That is, the Import Address Table will hold the value "00 00 00 00" at address "004933FA". When running (we can see these values in a

⁵ See http://msdn.microsoft.com/en-us/library/ms684847%28VS.85%29.aspx for details on the Kernel32.dll functions

debugger such as Olly as will be detailed later in the paper) a memory location will be returned. For instance if the value "004933FA" points to "AB 0C 59 7C" in memory, the system is calling the GetProcAddress() function⁶.

2.1.6. Relocations

Windows executable files are not based on position-independent code but are compiled to a chosen base address. In the event that a Windows executable cannot be loaded to the chosen address, Windows will rebase the memory location (move to a new base address). This can occur if the chosen address in memory is already used by another program. If this occurs, Windows has to recalculate all absolute addresses. This involves changing the values stored in the PE when the application is loaded into memory and setting new values.

The loader compares the preferred and real load addresses. A delta value representing the difference between the real and preferred start address is calculated. The delta is added by each of the preferred address in the application. The result is the actual memory location that is used by the application when it is executed. The base relocations are then loaded as a list into the system. These are called by the application and loaded into an existing memory location as required.

When loaded into the system memory, the resulting code that is created as a combination of the application and the loaded modules is set as private to the process by the system. When this occurs, the loadable module cannot be shared further.

Microsoft avoids rebasing setting pre-computed and non-overlapping memory addresses to limit the resultant performance hit that this process causes. As most users do not limit their applications to only those from Microsoft (and a number of vendors that follow their set addresses), rebasing will still occur. Malware and other packed software do not conform to the specifications recommended by Microsoft and hence this is one of the many reasons why malware degrades system performance. Rebasing can create extremely efficient code at the expense of additional memory use.

Linux ELF executables are completely position independent. Unlike PE files, ELF

⁶ See http://www.astercity.net/~azakrze3/html/win32_api_functios.html for a list of Windows APIs.

executables use a Global Offset Table. This results in a substitution of execution time in opposition to memory usage. Linux favors tight memory use, Windows PE files the former.

2.2. Further reading and related work

The details of each of these sections are covered fully in the "Microsoft Portable Executable and Common Object File Format Specification"⁷. This document is the ultimate reference guide for all aspects of the PE-COFF format. There are many papers on malware that will provide detail into the various packers if the reader wishes to follow-up this topic in more detail. The techniques presented in this paper can be applied to other packers⁸ equally well.

⁷ This file is provided free by Microsoft and is available online at http://www.microsoft.com/whdc/system/platform/firmware/PECOFFdwn.mspx

⁸ Such as ASPack, FSG, UPX, Themida etc.

3. NsPack

NsPack is a formerly semi-commercial packer. It was written by Liu Xing Ping and distributed by North Star Software in China. Although originally sold under a commercial license, the product was never restricted and was freely distributed through Warez sites and the RE (reverse-engineering) underground.

In section 6 we will analyze the NsPack binary executable itself. From this analysis, we see that NsPack was most likely developed using Microsoft Visual C + + 6.0 and was itself packed using ASProtect, another PE Packer by Alexey Solodovnikov. The likely reason for using a separate packer to pack NsPack itself is in order to make the analysis and reversing of the packing algorithm used more difficult. In section 6, the unpacking process for ASProtect 2.1 will be applied to NsPack 3.7 in order to dump an unpacker version of the original packer.

Unfortunately, many commercial antivirus vendors have not adequately analyzed the NsPack binary and compression routine. This has led to the unfortunate situation where major anti-malware vendors are misclassifying NsPack (and other PE Packers) as Trojans (figure 3.1). In section 6 we will show through both static analysis and dynamic execution that NsPack is not a Trojan but a simple PE compression utility.





Figure 3.1. Classic Misdiagnosis

NsPack remains one of the most common PE Packers with high rates of reported use and discovery. Oberheide, Bailey, & Jahanian (2009) used the Arbor Network's Arbor Malware Library (AML) to analyze the distribution of PE Packers. The results are displayed in figure 3.2. In these tables we see that NsPack is in the top 10 list for PE Packers used on malware samples stored in the AML database.

This is likely the reason for the high rates of misclassification in the industry noted above. There are some valid uses for PE Packers. These include:

Making Reverse Engineering of commercial software more difficult and expensive,

- 1. Hiding internal functions and algorithms from users,
- 2. Penetration tests and the creation of test exploits,
- 3. Minimizing download sizes of files in order to maximize transfer rates.

As such, although packers are commonly associated with malware, the use of PE compressors or Packers cannot be limited to malicious use cases.

SigBuster Identifier	Count
Allaple	22050
UPX	11324
PECompact	5278
FSG	5080
Upack	3639
Themida	1679
NsPack	1645
ASPack	1505
tElock	1332
Nullsoft	1058

PEiD Identifier	Count
UPX	11244
Upack	6079
PECompact	4672
Nullsoft	2295
Themida	1688
FSG	1633
tElock	1398
NsPack	1375
ASpack	1283
WinUpack	1234

Table 1: The top ten packers classes in our AML datasetTable 2: The top ten packers classes in our AML datasetas determined by SigBuster.as determined by PEiD.

Figure 3.2. Packer distributions, Jon Oberheide, Michael Bailey, Farnam Jahanian (2009)

As will be demonstrated later, the Entrypoint of NsPack generally makes use of a JMP instruction followed by a PUSHF and PUSHA command.

NsPack is an executable file compressor for Windows 32 and 64 bit PE based executables. It also has the capability to work on .NET files. In marketing material and in tests (figure 3.3), it is shown that NsPack is capable of compressing the size of a 32-bit or

64-bit Windows executable by up to 60%. It is claimed (NsPack, 2009) that no noticeable performance change will result from this compression. There are better compression programs, but not all of these support 64-bit exe, dll, ocx and scr files. In addition, the far lower deployment rate of NsPack when compared to more common packers (such as UPX) means that less effort has been made to understand and automatically unpack the algorithm used (figure 3.1).

Many anti-virus vendors⁹ simply report the existence of a packer. At the time of writing, Sophos reports NsPacked files as "Mal/Packer" and PcTools as "Packed/NSPack" for example¹⁰. As many files that are packed are not malicious, this leads to a significant increase in the false positive or detection rate and in some industries can pose a significant cost to the software user either through lost productivity or through restricted access to alternative software products.

The greatest challenge posed by NsPack is the ability to recompress an already compressed executable file. NsPack will recompress a PE file that has been compressed using Upx, Aspack, Pecompact, and several other packers. This slows the execution of the packed executable considerably, but make reverse engineering of the program extremely complex. Malware authors use this technique to further obfuscate their payloads. The techniques have not been widely deployed at present due to the inability of many anti-virus vendors to effectively decompress a large number of packers in real time.

⁹ An example of a common malware report for multiple vendors is provided by sources such as Virus total (http://www.virustotal.com/) and Rabid Monkey. A sample analysis is provided at the link below:

¹⁰ Also see http://www.threatexpert.com/threats/packed-nspack.html for further examples.

Windows EXE Packer Test

Last Update : 04/xx/2006

(^^)/ exe packer compression test...

	FFFTP.exe (v1.92a) <u>344.064 bytes</u>	winamp.exe (v5.08e) <u>980,992 bytes</u>	VisualBoyAdvance.exe (Dev v1.7.2) <u>1.773.643 bytes</u>	msnmsgr.exe (v6.2) <u>4.886.528 bytes</u>
FSG (v2.0) by bart	133,909	434,037	(broken)	(broken)
Mew11 SE (v1.2) by Northfox	116,721	391,227	(broken)	(broken)
NsPack (v2.3) by North Star Software (Liu Xing Ping)	108,518 < 2 >	336,792 < 2 >	430,203 < 1 >	(broken)
PECompact (v2.64) by Bitsum Technologies	111,616	339,968	(broken)	2,339,840 < 1 >
Upack / WinUpack (v0.399) by Dwing	105,664 < 1 >	333,444 < 1 >	432,960 < 2 >	(broken)
UPX (v2.0) by Markus & Laszlo	119,296	374,272	503,296	2,628,096 <2>

* All packers are default settings (on Windows XP SP2)

Figure 3.3. Packer Compression tests (Kpan, 2006)¹¹

This has resulted in the false positive issue noted above but also with the lower use of packers due to the increased likelihood of being rejected whether deemed malicious or not. The positive effect of this is that it is less cost effective (in terms of time for a start) for malware authors to pack software and it can be prohibitive to pack malicious code using multiple packers. Conversely, many users have become accustomed to the false positive issue and may run programs that are otherwise likely to be blocked.

In general, a malware analyst will not be interested in NsPack itself, other than removing its compression. NsPack makes the analysis of a packed sample more difficult and time consuming, and hence costly.

3.1. Using NsPack

The executable to be packed is either dragged/dropped onto the main window or the user can select:

¹¹ This table is provided by http://www.geocities.co.jp/Playtown-Yoyo/6130/exepacker.htm

NsPac	k By North Star	
File(<u>F</u>)	Options(<u>O)</u> Help(<u>H</u>)	Ver 3.
🞑 File	🕅 Directory 🔂 Settings 🛛 🕵 About	
	Drop a file into this wi	ndow
	00/	
	0%	
- 1	(C) 🚺 E 140	
Uompre 🖳	ess(<u>C)</u> 🚚 Exit(X)	🛄 Open(<u>O</u>)

File -> Open

Figure 3.4. NsPack in action

Alternatively, the user can select the "Directory" tab and compress all executable files in a directory at one time.

NsPack runs as a standard Windows application (figure 3.4). It is always advisable to run untrusted software onside of a normal production environment. In this case, NsPack was run from a Windows Vista system within a VMware session. Using the program is extremely simple.

This session was configured within a RedHat Linix workstation running SNORT and TCPdump. The reason for this is to capture any traffic to or from the host that could be associated with NsPack. This methodology will allow for the detection of network traffic and allow the determination of a network service if one exists.

As noted in section 3, many anti-virus vendors (see figure 3.1) classify NsPack as a Trojan. If this was the case, the program would either bind to a network port or connect to a remote IP address. Neither occurred when running the program. In fact, running the program for a period of 1 week resulted in no unexpected network traffic.

3.1.1. Options

NsPack is highly configurable (figure 3.5). Relocations, shared sections and section names are all able to be configured.

South Star			
File(E) Options(<u>O</u>) Help(<u>H</u>)	Ver 3.7		
🞑 File 🎁 Directory 🔂 Settings 🕵	About		
Compress resources	✓ Strip relocations		
✓ Create backup copy	✓ Preserve extra data		
✓ Force compression	✓ Save settings on exit		
Handle shared sections	✓ Integrate into context menu		
Auto run after loading	✓ Renew the part of dos stub		
🔽 Use Windows dll loader	Max.compression(win9x:unsupported)		
Reserve the size of file	C compatibility compression		
Section name:			
 User defined(8 chars max):	O Clear all		
O Random numbers(0000000-9999999)	language: English 💌		
💭 Compress(C) 🚽 Exit(🖄)	(D) Open(<u>O</u>)		

Figure 3.5. NsPack Options

NsPack allows the user to change the section headings. In place of the default ".nsp" naming convention, a user defined naming convention or a series of random numbers may be used. Section names can also be cleared using the "Clear all" option.

NsPack has been available in one form or another since 2003 (see figure 3.6) but as the domain is not longer maintained and no new version have appeared, it would appear that the program is no longer being supported. GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer.

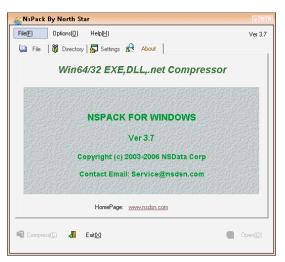


Figure 3.6. About the Packer

Once the file has been loaded, compressing it is simple (figure 3.7). Select:

File -> Compress

× N	lsPack By	North S	tar	_ = ×
File	(E) O(ptions(<u>0</u>)	Help(<u>H</u>)	Ver 3.7
	0pen(<u>0</u>)		🔂 Settings 👷 About	
ŋ	Compress((2)	ISPack 3.7\NsPack 3.7 (Unpacking)\CrackersKit v2.0.exe	
4	Exit(⊠)			
			2047	
			0%	
a	Compress(<u>C</u>)		5.482)	0(0)
<u> </u>	compress[L]	-200	Exit⊠) 🖬	Open(<u>0)</u>

Figure 3.7. Selecting the Packer compress function

The compression routine will run and the results are displayed in NsPack's main window (figure 3.8).

🗸 NsPack By North Star	
File(E) Options(Q) Help(H)	Ver 3.7
🞑 File 🛛 🕅 Directory 🛛 🔂 Settings 🛛 🥵 About	
C:\Data\HCL\NSPack 3.7\NsPack 3.7 (Unpacking)\CrackersKit v2.0.exe	
Verifying PE Format -> (PE32) Number Of PE sections Image Base Address Size Of Image Size Of Headers :000000400(h) Size Of Headers :00001000(h) Section Alignment :00000200(h) Checking/Reconstructing Import Table Checking/Reconstructing resources Checking/Merging Section Compressing Data Soving Image to disk 20667003(B)>143117(B) Compression ratio:[99.3%]	
100%	
10020	
다 Compress(C) 🚚 Exit(조) 🛄 ()pen(<u>0)</u>

Figure 3.8. Running the Packer

At this point, we have a compressed executable.

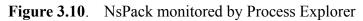
During this process, TCPView (from SysInternals) was run locally (figure 3.9) on the system to monitor for local ports and listening services. Snort will not detect the presence of a listening but not sending service. No ports were opened by NsPack.

a 🔀 😥				
ocess /	Protocol	Local Addre:	25	Remote Address
[System Process]:0 AppleMobileDeviceService.exe:1428 AppleMobileDeviceService.exe:1428 AppleMobileDeviceService.exe:1428 AppleMobileDeviceService.exe:1428 firefox.exe:7576 firefox.exe:7576 firefox.exe:7576 firefox.exe:7576 firefox.exe:7576 ifunesHelper.exe:1672 ifunesHelper.exe:1672 ifunesHelper.exe:1672 ifunesHelper.exe:1672 oUTLOOK.EXE:4176 OUTLOOK.EXE:4176 OUTLOOK.EXE:4176 PGPtray.exe:1424 P		CraigWright-F CraigWright-F	C:27015 C:27015 C:27015 C:27015 C:27015 C:27015 C:52938 C:52939 C:52939 C:52939 C:52939 C:52939 C:49214 C:52901 C:52902 C:52801 C:52801 C:52801 C:52952 C:558058 C:49259 C:33333 C:3333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:33333 C:	proxy01.information-defense.com:3128 localhost:52952 localhost:52952 localhost:52952 localhost:52937 localhost:52937 localhost:52938 localhost:52938 localhost:52938 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:27015 localhost:33333 localhost:52898 localhost:52888 localhost:52888 localhost:52888 localhost:52888 localhost:52888 localhost:52888 localhost:52889 localhost:33333 localhost:33333 localhost:33333 localhost:33333 localhost:33333

Figure 3.9. Monitoring NsPack locally using TCPView

At the same time, NsPack was being monitored by other SysInternal tools. In figure 3.10, Process Explorer is used to see if any other threads are created. Process Monitor was used to log these events. No unusual activity was monitored during these tests.

e <u>O</u> ptions <u>V</u> iew <u>P</u> rocess F	<u>i</u> nd <u>U</u> ser	s <u>H</u> elp				
		LA E	1		💻 🗉 🎛 🔊 📲	S 🕺 👫 🖲
cess	PID	CPU Description	Company Name	Path	Command Line	
🐨 McSvHost.exe	5884	McAfee Service Host	McAfee, Inc.	C:\Program Files\	"C:\Program Files	
McSACore.exe	5948	SiteAdvisor	McAfee, Inc.	C:\Program Files\	"C:\Program Files	
rundll32.exe	4884	Windows host process (Rundl32)	Microsoft Corporation	C:\Windows\Syst	"C:\Windows\syst	
SearchIndexer.exe	5176	Microsoft Windows Search Indexer	Microsoft Corporation	C:\Windows\Syst	C:\Windows\syst	
spoolsv.exe	8032	Spooler SubSystem App	Microsoft Corporation	C:\Windows\Syst	C:\Windows\Syst	
Isass.exe	1036	Local Security Authority Process	Microsoft Corporation	C:\Windows\Syst	C:\Windows\syst	
🗊 lsm.exe	1044	Local Session Manager Service	Microsoft Corporation	C:\Windows\Syst	C:\Windows\syst	
🎒 winlogon.exe	984	Windows Logon Application	Microsoft Corporation	C:\Windows\Syst	winlogon.exe	
🖃 📠 explorer.exe	9772	Windows Explorer	Microsoft Corporation	C:\Windows\expl	explorer.exe	
💭 procexp.exe	7356	4.46 Sysinternals Process Explorer	Sysinternals - www.sysinter	. C:\Program Files\	"C:\Program Files	
🕘 VolPanlu.exe	3724	VolPanlu.exe	Creative Technology Ltd	C:\Program Files\	"C:\Program Files	
🕑 issch.exe	3740	InstallShield Update Service Scheduler	InstallShield Software Corp	. C:\Program Files\	"C:\Program Files	
🗿 VProTray.exe	3768	Tray Application	Symantec Corporation	C:\Program Files\	"C:\Program Files	
🛄 hpwuSchd2.exe	3828	Hewlett-Packard Product Assistant	Hewlett-Packard Co.	C:\Program Files\	"C:\Program Files	
🔂 vmware-tray.exe	3868	VMware Tray Process	VMware, Inc.		"C:\Program Files	
💽 vc9play.exe	3888	0.74 Virtual CD - Player	H+H Software GmbH	C:\Program Files\	"C:\Program Files	
💽 vc9tray.exe	2960	Virtual CD - Quick Start Utility	H+H Software GmbH	C:\Program Files\	"C:\Program Files	
OlpSynch.exe	3900			C:\Program Files\	"C:\Program Files	
🗊 VM ware Tray. exe	2664	VMware Tools tray application	VMware, Inc.		"C:\Program Files	
🗊 VM wareUser. exe	2680	VMware Tools Service	VMware, Inc.	C:\Program Files\	"C:\Program Files	
🛃 jusched.exe	2744	Java(TM) Platform SE binary	Sun Microsystems, Inc.	C:\Program Files\	"C:\Program Files	
<u> j</u> ucheck.exe	3032	Java(TM) Update Checker	Sun Microsystems, Inc.		"C:\Program Files	
🥹 wmdSync.exe	2776	User session Windows Mobile device handler	Microsoft Corporation		"C:\Windows\Wi	
🖰 MacDrive.exe	2768	MacDrive application	Mediafour Corporation	C:\Program Files\	"C:\Program Files	
🛃 iT unesHelper. exe	1672	iT unesHelper	Apple Inc.		. "C:\Program Files	
🙀 sidebar. exe	2436	5.20 Windows Sidebar	Microsoft Corporation		"C:\Program Files	
🕑 AudibleDownloadHelper.exe	2952	Download Manager for Audible content	Audible, Inc.		"C:\Program Files	
🙋 hpqtra08.exe	1660	HP Digital Imaging Monitor	Hewlett-Packard Co.		"C:\Program Files	
PGPtray.exe	1424	PGP Tray	PGP Corporation		"C:\Program Files	
m MOM.exe	3468	Catalyst Control Center: Monitoring program	Advanced Micro Devices I			
ATT CCC.exe	5652	Catalyst Control Centre: Host application	ATI Technologies Inc.		"C:\Program Files	
😔 OUTLOOK.EXE	4176	Microsoft Office Outlook	Microsoft Corporation		"C:\Program Files	
WINWORD.EXE	4632	8.18 Microsoft Office Word	Microsoft Corporation		"C:\Program Files	
KeePass.exe	2876	KeePass	Dominik Reichl		"C:\Program Files	
irefox.exe	7576	4.46 Firefox	Mozilla Corporation		"C:\Program Files	
💶 taskmgr.exe	6540	0.74 Windows Task Manager	Microsoft Corporation		"C:\Windows\syst	
ResHacker.exe	8740	Resource viewer, decompiler & recompiler.			. "C:\Data\HCL\Ot	
grocexp.exe	5120	0.74 Sysinternals Process Explorer	Sysinternals - www.sysinter			
Tcpview.exe	8580	0.74 TCP/UDP endpoint viewer	Sysinternals - www.sysinter			
😽 Nspack 3.7. exe	10120	nSpack Microsoft ??????	North Star	C:\Data\HCL\NS	"C:\Data\HCL\N	



Although, and as noted, several Anti-Virus vendors classify NsPack as a Trojan; we can see that NsPack exhibits no malicious behavior by itself.

The result of this is that we can be satisfied that NsPack is not a Trojan itself, but simply a packer or compression function. We can also use this to create a number of distinctly different packed executable by manipulating the options associated with NsPack.

4. Determining the packer.

In this section, we will walk through the analysis, disassembly and rebuilding of NsPack, annotating the differences between the 3.4 and 3.7 versions. We will start with the process used to determine what packer (if any) has been used. Although the distinctions are minor, the walkthrough for NsPack 3.4 have been noted in full below where they differ from the processes used with NsPack 3.7.

The first step is to validate that the correct packer has been used on the samples. Different packers require different processes to unpack them. Using the wrong process will at best waste time and could at worst lead to compromising a host.

In the case of NsPack, two (2) tools (PEiD and RDG) are used to ensure that this was correct prior to starting execution. Using the two tools allows us to minimize false positives. PEiD is one of the most accurate packer detectors, but this still means errors occur. By using PEiD and RDG conjointly, the error rate is maintained at an acceptably low level. These tools are available from the following websites:

₩ PEiD (http://www.peid.info/)

RDG Packer Detector v0.6.6 2k8 or later (http://www.rdgsoft.8k.com/)

4.1. PEiD

PEiD is designed to detect the majority of packers, cryptors and compilers used on PE files. It has the ability to detect over 470 distinct packer signatures in PE files. Using PeID is simple. Just run the program (administrator privileges may be required on some systems)

To scan a file, simply:

- 1. Click the ... button on the top right of the main PEiD window to the right of the "file" field.
- 2. Select the file to be analyzed.
- 3. PEiD will automatically scan the loaded file and return the results.

PEiD has a number of options (figure 4.1) that provide the ability to scan multiple files or directories at one time. It is also possible to test the file more completely, but this does increase the time required to report on each file.

PEiD Options	X
 Normal Scan Deep Scan Hardcore Scan 	<u>R</u> egister Shell Extensions <u>Minimize to System Tray</u>
Recurse Subdirectories	Load Plugins (Restart PEiD) Allow Multiple Instances
Use External Signatures	(Restart PEID) jave

Figure 4.1. The PEiD options

The button provides additional information about the file being analyzed (figure 4.2) and provides statistical data as to the entropy of the file (which can also be used in detecting encrypted or compressed segments).

Extra Inform	nation 🛛 🔤
FileName: Detected:	C:\Users\Craig S Wright\AppData\Local\Temp\Rar\$EX00.567\PEiD.e Nothing found *
Scan Mode: Entropy: EP Check:	Normal
Fast Check:	
	<u>O</u> K

Figure 4.2. PEiD options

Loading an NsPack compressed file will quickly return the existence of the packer. In the case the version 3.7 samples, these are readily validated as NsPack 3.7 compressed executables (figure 4.3).

🔛 PEID v0.95					
File: C:\Data\HCL\Tests\Pac	k 3\notepad.exe				
Entrypoint: 000280C9	EP Section: XTg ₁ >				
File Offset: 000032C9	First Bytes: 9C,60,E8,00 >				
Linker Info: 8.0	Subsystem: Win32 GUI >				
nSPack 3.7 -> North Star/Liu Xing Ping <u>Multi Scan Task Viewer Options</u> <u>Exit</u>					
✓ Stay on top	»» ->				

Figure 4.3. PEiD determination of NsPack 3.7

With NsPack version 3.4 compressed executables, the results can be less accurate and a "Hardcore Scan" may be required.

PEiD will also return some basic information concerning the file such as:

☑ Information returned - Entrypoint

- ▼ File Offset
- ☑ Linker Info
- ★ EP section

These are covered in more detail in section 2 and the manual calculation of the File Offset is included in Section 6.3.2.

4.2. RDG Packer Detector

It is important to validate the settings when manually checking samples. Using multiple detection tools in combination significantly increases the probability that we have identified the packer used on the sample correctly.

RDG Pac	ker Detector vO.	5.5 2 ³⁶ - X
C:\Data\HCL\Tes	sts\Pack 3\notepad.exe	Abrir
0	NSPack v3.7	Compilador
Ð	NSPack V3.7	Detectado Posible
Contacto :		Al Frente □ Ø
•	⊙ M-A ○	M-B 123456789M

Figure 4.4. RDG determination of NsPack 3.7

In each case, the samples have been validated as correctly being packed using NsPack version 3.7. This differs slightly to NsPack 3.4, where more errors have been noted in determining the sample packing version.

Many of these tools behave poorly on Windows Vista. The use of either Windows XP or Windows 7 is recommended and even the later can periodically fail.

To use RDG, load the file to be analyzed:

Select Abrir

This will bring up the Explorer 'File' menu and you can then load the program to be tested.

Next, select the **Detectar** button. This will run the analysis and return the result (figure 4.4). This can be clicked to provide more details in some instances (figure 4.4), however, NsPack 3.7 gives little more information. Where multiple packers have been used or the file has been manually altered (e.g. IAT manipulation) this can provide information that may aid in the recreation of the file.



Figure 4.4. RDG has successfully detected NsPack version 3.7.

If the user has mixed the configuration options (figure 3.5) when creating the packed file, NsPack version 3.4 will not always be detected correctly by many tools (figure 4.5).



Figure 4.6. RDG determination of NsPack 3.4

With Version 3.4, there are some discrepancies as to the version (see above). The results do point to NsPack, but the version may not be correctly calculated.

5. Unpacking in Olly

This section details the process to unpack NsPack 3.x compressed executables using Olly (by Oleh Yuschuk). Extreme care and caution should be taken when unpacking possible malware samples in Olly. This should go to the use of isolated hosts (such as VM's or specialized non-networked systems). A debugger runs the executable. Although a good deal of control is maintained over the executable being analyzed in a debugger, it is easy to err and allow the sample to infect the host. VM's are of use here as the snapshot capability allows the reversal of steps that cause problems.

A dissassembler such as IDA (Appendix) does not run the executable. As such, it will not lead to the sample infecting the analysis host. Olly is used for the manual unpacking process. Olly is a powerful Windows debugger and is widely deployed (and being free does not hurt). Olly has a number of enhancements and add-ons that increase its effectiveness.

To load and extract the packed executable images using Olly, we shall start with configuring Olly and the environment it is running within:

- 1. Get the OllyDbg program from <u>http://home.t-online.de/home/Ollydbg/</u>
- Get the OllyDump plug-in from <u>http://www.pediy.com/tools/Debuggers/ollydbg/plugin/OllyDump/OllyDu</u> <u>mp.zip</u>
- Extract the file, ollydump.dll file into OllyDbg's plug-in directory (e.g. C:\Reversing\Olly\Plug-ins).
- 4. Run OllyDbg,
- 5. Click File->open,
- 6. Select the executable to unpack.

GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer.

🌟 OllyDb	g						
File Vie	∾ Debug	Plugins	Options	Window	<u> </u>	elp	
Ready	🖻 📢 :	1 An	alyzeThis	۰.	 	→	LEMTWHC/KBR
С СРИ		2 01	lyDump	×		Dump	debugged process
							EP by Section Hop (Trace into) EP by Section Hop (Trace over)
						Option	5
						About	

Figure 5.1. Check the plug-ins are loaded

It is also good practice to ensure the Plug-in has been correctly loaded. To do this:

- 1. Click File->Plug-ins
- 2. Check that OllyDump has been loaded (figure 5.1).

Olly can be used to analyze malware samples in depth and to investigate the processes started by an unknown application. In this section, we look at analyzing and dumping packed samples. The section may be focused on NsPack, but the same techniques can be used on other types of packers.

5.1. Manual Unpacking

With OllyDebug running and the OllyDump plug-in loaded (figure 5.1), click "**F3**" to load the packed sample. An alert should appear noting that the sample is packed. Click ok (figure 5.2).

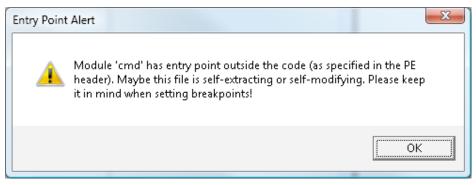


Figure 5.2: Loading a packed sample into Olly

The packed sample will now be loaded into Olly. Note that a warning that the sample is packed is again displayed (figure 5.3). Again, click 'Yes' to continue. As we are

attempting to dump and manually unpack the sample, it is expected that we will receive warnings. Most software is not packed and Olly's ability to analyze the software is limited whilst it is packed.

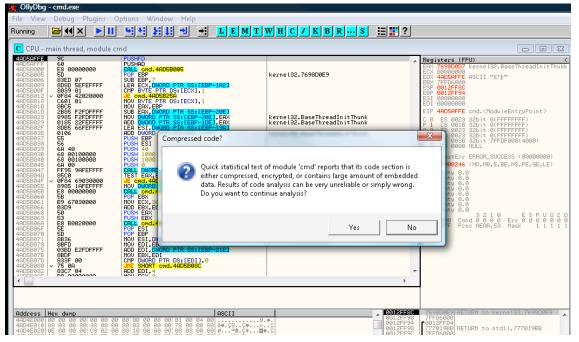


Figure 5.3: Again, Olly lets us know the sample is packed.

To run the program by steps and hence decompress the original, we need to enter "F8" (select the F8 function key) to step through the packed sample (figure 5.4).

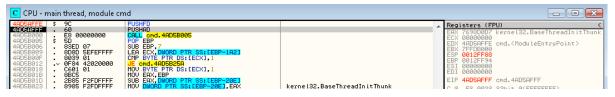


Figure 5.4: Stepping through the program.

At this point you should notice that the Registers have changed (figure 5.5). Olly can display the values help in the systems memory and as the program executes and these update, we can watch the changes that occur.

Registers (FPL)) <	Registers (FPU)
EAX 7698D0D7 k ECX 00000000	ernel32.BaseThreadInitThunk md. <moduleentrypoint></moduleentrypoint>	EAX 7698D0D7 kernel32.BaseThreadInitThur ECX 00000000 EDX 4AD5AFFE cmd. <moduleentrypoint> EBX 7FFD8000 ESP 0012FF88 EBP 0012FF94 ESI 00000000 EDI 00000000</moduleentrypoint>
EIP 4AD5AFFE d	md. <moduleentrypoint></moduleentrypoint>	EIP 4AD5AFFF cmd.4AD5AFFF
P 1 CS 001B 3 A 0 SS 0023 3 Z 1 DS 0023 3 S 0 FS 003B 3 T 0 GS 0000 h D 0 LastErr E	RROR_SUCCESS (00000000)	C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 7FFDF000(4000) T 0 GS 0000 NULL D 0 U LastErr ERROR_SUCCESS (00000000) EFL 00000246 (N0,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0 ST1 empty 0.0 ST2 empty 0.0 ST3 empty 0.0 ST5 empty 0.0 ST5 empty 0.0 ST6 empty 0.0 ST6 empty 0.0 ST7 empty 0.0	NO,NB,E,BE,NS,PE,GE,LE) 3 2 1 0 E S P U O Z D 1 0 0 0 0 Err 0 0 0 0 0 0 5 NEAR,53 Mask 1 1 1 1 1	ST0 empty 0.0 ST1 empty 0.0 ST2 empty 0.0 ST3 empty 0.0 ST4 empty 0.0 ST5 empty 0.0 ST5 empty 0.0 ST6 empty 0.0 ST7 empty 0.0 ST7 empty 0.0

Figure 5.5: The registers will change as the program is stepped through.

From figure 5.5, we can see that the values in the registers have changed with the original load on the left, with the alteration subsequent to hitting F8 on the right. The ESP register is the CPUs 32-bit stack pointer and it stores the *current position* in the stack. When a value is *pushed* to the stack, it is pushed below this address. The ESP register points to the current top of stack.

A **PUSH** subtracts 4 from the SP and copies a 32-bit value onto the top of the stack, **POP** copies a 32-bit value from the top of the stack and adds 4 to SP. SP is the 16-bit stack pointer register. The SP is the low 16 bits of the 32-bit ESP register.

In order to trace the stack, Right Click the ESP register and select "follow in dump" (figure 5.6). Following this value should take us to the OEP.

Olly is a user-mode debugger. User mode debuggers attach to a single process unlike kernel mode debuggers (such as SoftICE or WinDbg) which attach to the entire system and all processes. This limitation is not a problem with respect to unpacking code. With a user mode debugger, it is necessary to know the exact process to be analyzed. In order to unpack a sample this is a prerequisite and hence not a limitation.

Olly displays the registers for the system, and the inability to attach to all running processes is unlikely to affect any unpacking exercise. When analyzing malware, unlike

simply unpacking the sample, the process being analyzed may call other processes. When this occurs, the newly spawned process may not be accessible from within Olly. As noted, this limitation will not impact the unpacking and dumping process as the system will call and unload itself in the same process.

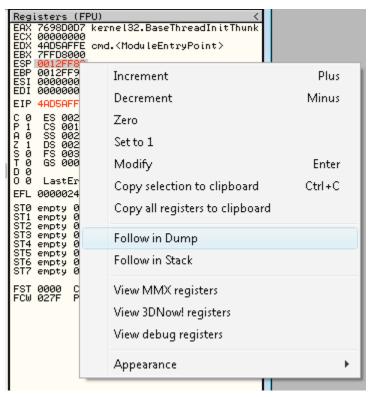


Figure 5.6: Tracing the stack.

The "follow in dump" function provides a dump of executable section that we are going to follow in order to find the "OEP".

This dump is displayed in the figure 5.7. Note the data contained in the Hex dump field displayed in the window at the lower left of the screen.

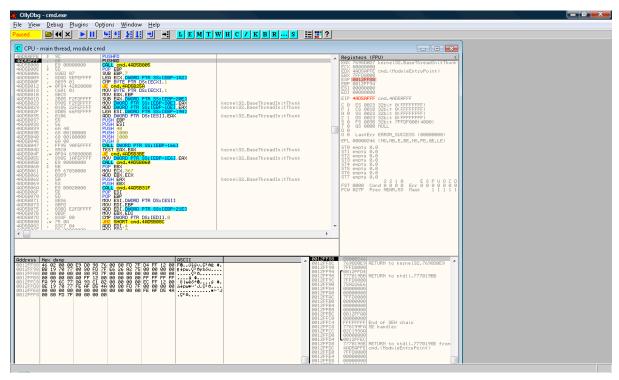


Figure 5.7: The dump of the executable.

The dump below contains the address of the ESP register. You can see that we have highlighted the initial four (4) byte values (as displayed in the figure below with the values highlighted in grey).

Address	Hex du	IMP												ASCII	
0012FF88 0012FF98 0012FF88 0012FF88 0012FF88 0012FF88 0012FF88 0012FF88 0012FF88	00 00 00 00 FA 99 8E 19 00 00	70 7 00 0 60 7 70 0	77 00 00 00 00 A0 77 8A 77 FE 00 00	80 80 93 AF 00	FD 7 FD 7 12 0 C1 0 D5 4 00 0	F E6 F 00 2 00 A 00 0 00	26 00 00 00 80	A2 00 00 00 FD	75 00 00 7F	00 00 FF EC 00	00 00 FF FF 00	00 00 FF 12 00	00 00 FF 00 00	F8. ΰ\$ÿ∪.DzΔἐ ‡. η↓pw.Ç²Δµ&öu .DzΔ .š ‡ÿ ‡. .ölwēō∸8ÿ ‡. ä↓pw≡>'J.DzΔ .DzΔ	

Figure 5.8: The address of the ESP register.

Using these values we want to set a hardware breakpoint. We do this using the following setting:

"Breakpoint -> Hardware, on access -> Dword"

You do this by selecting the highlighted values above and right clicking. This process is displayed in the image on the following page.

Setting a hardware breakpoint allows us to follow the execution of the program to this point and then to stop (or interrupt) the execution of the program.

4AD5801D 4AD58023 4AD58029 4AD58035 4AD58035 4AD58035 4AD58038 4AD58038 4AD58038 4AD58038 4AD58038 4AD58045	. 8		F2	Backı Backı Binar	up	IV DWOF	D PTF	CSS: SS: PTR	SS:[E [EBP-2 [EBP-1 SS:[E SS:[]	0E] E DE] E BP-19	AX AX	ker	rnel32	2.BaseThreadIn 2.BaseThreadIn 2.BaseThreadIn	itTl
4AD5B047 4AD5B04D	- F - 8 - 9	F		Break	point		۱.	N	vlemory	, on a	access		e132	2.BaseThreadIn	itTl
4AD5B04F 4AD5B055 4AD5B05B	.* Ø . 8	F 9		Searc	h for		F	N	vlemory	, on v	write		e132	2.BaseThreadIn	itTl
4AD5B060 4AD5B061 4AD5B066		B 9 3		Go to	1		•	F	Hardwa	re, on	access	×		Byte	
4AD5B068 4AD5B069	: 5	03	<	Hex			•	F	Hardwa	re, on	write	×		Word	P
4AD5B06A 4AD5B06F 4AD5B070	. 5	Ē		Text			•	F	Hardwa	re, on	execution			Dword	
4AD5B071 4AD5B073	. 8	B		Short	:				UDILE						-
4AD5B075 4AD5B07B 4AD5B07D	. 8	3 B 3		Long			F		CEDIJ,		EJ				
4AD5B080 4AD5B082	× 7	5		Float			• ^{mc}	1.4AC)5808C	-					
<				Disas	semble	:									
				Speci	al		۲.								
	Hex du			Арре	arance		۰Ē				SCII				
0012FF98	46 02 BB 19 00 00	70	77	00 80	FD 7F FD 7F	Ê6 26 00 00	A2 7 00 0	5 00	00 00	00 a -	9ů\$ÿ∨.Dz∆ê ↓pw.Dz∆µ&óu .C²∆	• • •			
0012FFB8 0012FFB8 0012FFC8	00 00 FA 99	00 6C	00 77	66 86 A0 FF 8A 93 FE AF	12 00	00 00	- 00 O	0 FF	FF FF	FF	á.‡ jlweō ⊥8 .0	; + .			
0012FFD8 0012FFE8	00 00 FA 99 8E 19 00 00 00 80	70 00	00	FE AF 00 00 00 00	00 00	00 80 00 00	FD 7 00 0	F 00 0 FE	FF 12 00 00 AF D5	00 Ä-	lpw∎≫'J.Dz∆. Dz∆.	". j			
0012FFF8	00 00	PU	1	00 00	00 00					.,					

Figure 5.9: Setting a hardware breakpoint

With our breakpoint, we want to hit the "F9" function key to "run" the executable until it hits the breakpoint that we have set. This takes us to a jump command. This is displayed in the figure below:

4AD58270 E9 22E5FAFF	UMP cmd. 4AD09797	
4RD58275 \$ 88B5 EAFDFFFF 4RD58276 0BF6 0BF6 4RD58270 0BF6 97000000 4RD58287 0BF2 97000000 4RD58288 8895 F2FDFFFF 4RD58288 832E 00 4RD58288 837E 04 4RD58290 837E 04 4RD58294 75 02 4RD58294 75 08 4RD58294 585E 08 4RD58294 52 44 4RD582A1 52 4RD582A5 56 4RD582A6 8DB0 20582A7 37 4D582847 3326 4RD58283	HeLL EST, DWORD PTR-85:[EBP-216] OR ESI, ESI JE cond. 4AD5B31A MOV EDX, DWORD PTR SS:[EBP-20E] ADD ESI, EDX CMP DWORD PTR DS:[ESI],0 JNZ SHORT cmd.4AD5B29E CMP DWORD PTR DS:[ESI+4],0 JNZ SHORT cmd.4AD5B29E CMP DWORD PTR DS:[ESI+4],0 JNZ SHORT cmd.4AD5B29E JMP SHORT cmd.4AD5B29E JMP SHORT cmd.4AD5B29E JMP SHORT cmd.4AD5B318 MOV EBX, DWORD PTR DS:[ESI+8] ADD ESX,EDX PUSH ESX PUSH ESX PUSH ESI LEA EDI, DWORD PTR DS:[ESI+4] ADD ESI,0C PUSH EDI CALL DWORD PTR SS:[EBP-172]	cmd. <moduleentrypoint> cmd.<moduleentrypoint> cmd.<moduleentrypoint></moduleentrypoint></moduleentrypoint></moduleentrypoint>

Figure 5.10: Jumping to the command

We can follow this jump by entering "F7" to "step into" the command. This will

allow us to run a single machine code command and hence to follow where the jump command takes us.

After the jump, you will notice that some of the code looks strange (see the figure below).

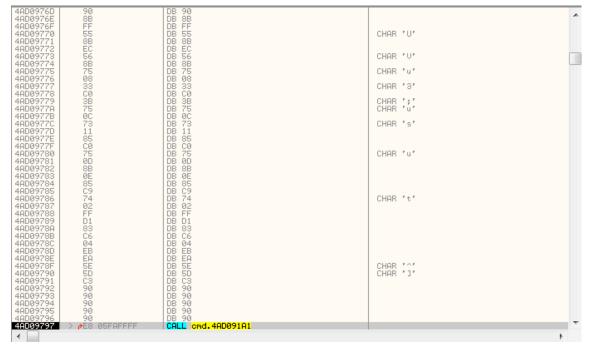


Figure 5.11: Obscured code

Enter "**Ctrl-A**" which will analyze the data and treat is as code. It was not treated as code previously, because before this was all data. When the unpacker executed, the code was written to these memory locations.

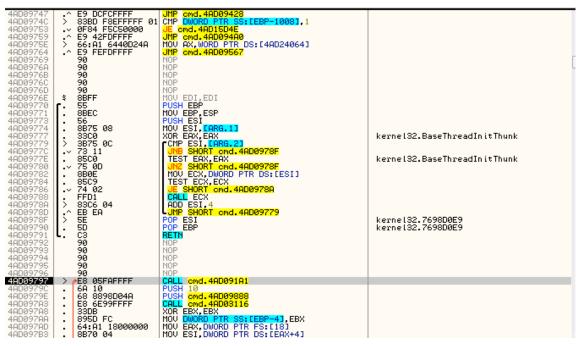


Figure 5.12: The code is now readable

Once this process has completed, you will note that the code is far easier to understand. This is displayed in the previous figure.

Next we want to dump the process.

To do this, we will use the OllyDump plug-in.

To do this, select:

Plug-ins -> OllyDump -> Dump Debugged Process

This process is displayed in the figure below:

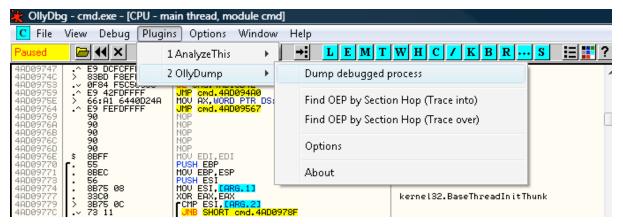


Figure 5.13: Dumping from Olly

Start <u>A</u> c Entry Po	LIEGOE	<u></u>		<u>G</u> et EIP as OB	Dump Cancel
Base of	<u>C</u> ode: 1000	Base of	<u>D</u> ata: 4F000		
🗹 <u>F</u> ix Ra	aw Size & Offset	of Dump Image			
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Charactaristics
.nsp0	0004E000	00001000	0004E000	00001000	F2000060
.nsp1	000131C6	0004F000	000131C6	0004F000	E0000060
.nsp2	0000CEB8	00063000	0000CEB8	00063000	E0000060

When the plug-in is displayed, unselect the "Rebuild Import" option.

Figure 5.14: The section table

When we are working with a highly customized version 3.4 NsPacked executable, (figure 5.15), we see that the sections are not as clearly marked as the NsPack 3.7 packed executable with few obfuscation options (figure 5.14).

Start <u>A</u> d Entry Po		00 <u>S</u> ize	Law and the second	<u>G</u> et EIP as Of	D <u>u</u> mp Ca <u>n</u> cel
Base of	<u>C</u> ode: 1000	Base of	<u>D</u> ata: 14000		
		of Dump Image		D 0% (
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Charactaristics
v[¤•0	00015000	00001000	00015000	00001000	E0000040
v[¤•1	00009000	00016000	00009000	00016000	D0000040
v[¤•2	00001000	0001F000	00001000	0001F000	E0000060
v[¤•3	00007000	00020000	00007000	00020000	E0000060
v[¤•4	000008E8	00027000	000008E8	00027000	E0000060
	ld Import nod <u>1</u> : Search J				

Figure 5.15: Obscured sections

At this point we will not use the Rebuild method from the OllyDump Plug-in.

```
    Bebuild Import
    Method1 : Search JMP[API] | CALL[API] in memory image
    Method2 : Search DLL & API name string in dumped file
```

Figure 5.16: Rebuilding

Select "Dump".

Then select the file to save the dumped executable as.

🔆 Save Dump to File			x
🕖 🖉 🖉 🖉 🖉	ta ▶ HCL ▶ Tests ▶ Pack 2	✓ Search	٩
File <u>n</u> ame:	unpacked_cmd.exe		-
Save as <u>t</u> ype:	Executable file(*.exe)		-
<u> </u>		Save	Cancel

Figure 5.17: Saving the file

By loading this into PEiD we can see that the file is no longer packed:

ſ	🔏 PEiD v0.9	95				x
	File: C:\Da	ata\HCL\Tests\Pa	ck 2\unpacked_cr	md.exe		
	Entrypoint:	00009797	j	EP Section:	.nsp0	>
	File Offset:	00009797	j	First Bytes:	E8,05,FA,FF	>
	Linker Info:	8.0		Subsystem:	Win32 console	>
	Nothing fou	nd *				
	<u>M</u> ulti Scan	<u>T</u> ask Viewe	r <u>O</u> ptions	<u>A</u> bo	ut E <u>x</u> i	t
	Stay on t	top			**	->

Figure 5.18: PEiD show the file is now unpacked

By loading this into RDG Packer Detector we can also see that the file is no

longer packed:



Figure 5.19: The IAT needs to be fixed

However, we have not fixed the IAT and hence the executable will not run as yet.

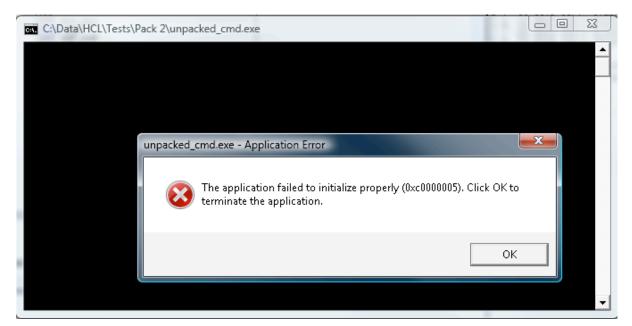


Figure 5.20: Without a fixed IAT, the file will not execute

As such, we need to fix the IAT. To do this at this stage, we will run ImpRec.

First, attach ImpRec to the running process (as displayed in the figure below).

A Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF	
Attach to an Active Process	
c:\data\hcl\tests\pack 2\cmd.exe (00000AE4)	Pick DLL
Imported Functions Found	
	Show Invalid
	Show Suspect
	Auto Trace
	Clear Imports
1	
Log	
Module loaded: c:\windows\system32\kernel32.dll Module loaded: c:\windows\system32\advapi32.dll	
Module loaded: c:\windows\system32\rpcrt4.dll	Clear Log
Module loaded: c:\windows\system32\msvcrt.dll Getting associated modules done.	
Image Base: 4AD 00000 Size:00070000	
IAT Infos needed New Import Infos (IID+ASCII+LOADER)	Options
0EP 0004F20E IAT AutoSearch BVA 00000000 Size 00000000	
	About
RVA 00000000 Size 00001000	Exit
Load Tree Save Tree Get Imports Fix Dump	

Figure 5.21: ImpRec is used to fix the IAT

Notice that the OEP is not correct. Remember, the OEP was supplied using OllyDump (above):

Start <u>A</u> ddress:	4AD 00000	<u>S</u> ize: 700	000	
Entry Point:	4F20E	-> <u>M</u> odify: 979	97	<u>G</u> et EIP as OEP
Base of <u>C</u> ode:	1000	Base of <u>D</u> ata:	4F000	

Figure 5.22: OEP settings

As such, we need to fix up the OEP in ImpRec:

	IAT Infos needed									
0EP 0000	9797 IA1	IAT AutoSearch								
RVA 0000	0000 Size	00001000								
Load Tree	Save Tree	Get Imports								

Figure 5.23: Correcting the OEP

Then select "IAT AutoSearch" to continue.

When ImpREC finds the value, it will display a message, click on "OK":

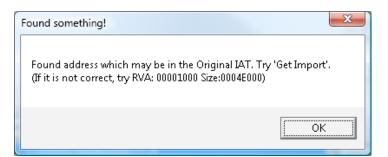


Figure 5.24: Finding the value

Next, get the imports. This is done by clicking "Get Imports" on the lower left of the screen:

Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF	
Attach to an Active Process	
c:\data\hcl\tests\pack 2\cmd.exe (00000AE4)	Pick DLL
Imported Functions Found	
advapi32.dll FThunk:00001000 NbFunc:14 (decimal:20) valid:YES wernel32.dll FThunk:00001054 NbFunc:88 (decimal:136) valid:YES	Show Invalid
Kernelsz, dir Frhunk:00001034 NbFunc:45 (decimal:69) valid:YES T- ntdll.dll FThunk:00001390 NbFunc:A (decimal:10) valid:YES	Show Suspect
	Auto Trace
	Clear Imports
Log	
rva:000011B0 forwarded from mod:ntdll.dll ord:0460 name:RtlSizeHeap rva:000011B4 forwarded from mod:ntdll.dll ord:02AD name:RtlReAllocateHeap Current imports: 4 (decimal:4) valid module(s) (added: +4 (decimal:+4)) EB (decimal:235) imported function(s). (added: +EB (decimal:+235))	Clear Log
Eb (decimal 200) imported function(s), (added, 4Eb (decimal 4200))	
IAT Infos needed New Import Infos (IID+ASCII+LOADER)	Options
OEP 00009797 IAT AutoSearch RVA 00000000 Size 000010084 RVA 00001000 Size 0000038C Image: Add new section Image: Add new section	About
Load Tree Save Tree (Get Imports) Fix Dump	Exit

Figure 5.25: ImpRec to correct the IAT

We can see from the image above that all of the imports have been found successfully. This is demonstrated by the "valid: Yes" flag in the "Imported Functions Found" field. As ImpRec has correctly determined these values, we need to fix the dump. To do this, look at the lower right-hand side of the screen and select "Fix Dump". Ensure that Import ReConstructor is running as the Administrative user on the system or it will not be able to bind to the process.

You will be presented with the location of where you want to save the repaired and unpacked executable.

🍕 Choose your dump file to	o fix			×
💮 🕞 - 🕌 « Data 🕨	HCL 🕨 Tests 🕨 Pack 2	🔻 🍫 Search		Q
🌗 Organize 👻 🏢 View	s 🔻 📑 New Folder			0
Favorite Links	Name	Date modified	Туре	Size
Documents	csv. 2.exe	23/07/2009 12:03	Application	
Recently Changed	ow. B.exe	23/07/2009 12:13	Application	
Desktop	cmd.exe	23/07/2009 9:08 AM	Application	
Recent Places	notepad.exe unpacked_cmd.exe	23/07/2009 9:09 AM 23/07/2009 11:56	Application Application	
Computer	unpacked_cmdexe	23/07/2009 12:37		
Pictures	Up_iat_cmd.exe	23/07/2009 12:02	Application	
Music				
Searches				
Public				
Folders	•			+
File <u>n</u> ame	unpacked_cmdexe	▼ PE	files (*.exe, *.dll)	•
			<u>D</u> pen 🔽 🗌 C	ancel

Figure 5.26: Where to save the corrected file

Enter the name of the dumped executable that you are fixing and select open.

A Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF						
Attach to an Active Process						
c:\data\hcl\tests\pack 2\cmd.exe (00000AE4)	Pick DLL					
Imported Functions Found						
⊕- advapi32.dll FThunk:00001000 NbFunc:14 (decimal:20) valid:YES ⊕- kernel32.dll FThunk:00001054 NbFunc:88 (decimal:136) valid:YES	Show Invalid					
msvort.dll FThunk:00001278 NbFunc:45 (decimal:69) valid:YES ⊡- ntdll.dll FThunk:00001390 NbFunc:A (decimal:10) valid:YES	Show Suspect					
	Auto Trace					
	Clear Imports					
Log						
Fixing a dumped file 4 (decimal: 4) module(s) EB (decimal: 235) imported function(s). **** New section added successfully. RVA:00070000 SIZE:00002000 Image Import Descriptor size: 50; Total length: 1084 C:\Data\HCL\Tests\Pack 2\unpacked_cmdexe saved successfully.	Clear Log					
IAT Infos needed New Import Infos (IID+ASCII+LOADER)	Options					
OEP 00009797 IAT AutoSearch RVA 00000000 Size 000010084 RVA 00001000 Size 000003BC If Add new section If Add new section	About					
Load Tree Save Tree Get Imports	Exit					

Figure 5.27: All fixed

As is displayed above, the log should show that the unpacked executable was saved. In this case (and this is not unusual) the unpacked executable is larger than the original file (before it was initially packed).

os. cmd.exe	23/07/2009 9:08 AM	Application	312 KB
📄 cmd.exe.bak	19/01/2008 5:33 PM	BAK File	312 KB
🥮 notepad.exe	23/07/2009 9:09 AM	Application	148 KB
📄 notepad.exe.bak	19/01/2008 5:33 PM	BAK File	148 KB
📴 unpacked_cmd.exe	23/07/2009 11:56	Application	448 KB
📴 Up_iat_cmd.exe	23/07/2009 12:02	Application	453 KB
unpacked_cmdexe	23/07/2009 12:37	Application	456 KB

Figure 5.28: The directory listing

We see from the figure above, that "cmd.exe", a file that was initially 312Kb in size was packed to just 148Kb, but when it was unpacked, it has grown to 456Kb.

The unpacked file also runs correctly now that the IAT has been repaired (Fig.

5.29).



Figure 5.29: The executable runs now.

6. Fixing the IAT

In this section, a process that is designed to manually correct the IAT is presented.

Figure 6.1: Not all imports are found

We can see from the image above that not all of the imports have been found successfully. This is demonstrated by the "valid: No" flag in the "Imported Functions Found" field. As ImpRec has correctly determined these values, we need to fix the dump. To do this, look at the lower right-hand side of the screen and select "Fix Dump".

🔏 Choose your dump file to fix 🛛 💽							
🚱 🗢 💻 Desktop 🔸	🕞 🔍 🗢 📃 Desktop 🕨 🔹 😽 Search						
🖣 Organize 👻 🏢 Views	🔻 📑 New Folder	_	_	0			
Favorite Links	Name	Size	Туре	Date mod 🔶			
 Documents Recently Changed Desktop 	I Craig S. Wright I Public I Computer I Network			E			
Recent PlacesComputer	🔊 Altova DatabaseSpy 🍘 Altova DiffDog	3 KB 3 KB	Shortcut Shortcut	19/07/200 19/07/200			
 Pictures Music 	Altova MapForce Altova SchemaAgent Altova SemanticWorks	3 KB 3 KB 3 KB	Shortcut Shortcut Shortcut	19/07/200 19/07/200 19/07/200			
Public	Altova StyleVision	3 KB 3 KB	Shortcut Shortcut	19/07/200 19/07/200			
Folders 🔨	Altova XMLSov	3 KB	Shortcut	19/07/200 *			
File <u>n</u> ame:	calc2.exe	•	PE files (*.exe, *.dll)	▼			

Figure 6.2: Let's fix the file

Enter the name of the dumped executable that you are fixing and select open.

🔮 Import REConstructor v1.6 FINAL (C) 2001-2003 MackT/uCF					
Attach to an Active Process					
c:\users\craig.wright.info-defense\desktop\calc2.exe (000011E0)	Pick DLL				
Imported Functions Found					
🕒 ? FThunk:0000000 NbFunc:7 (decimal:7) valid:NO 🛛 🔺	Show Invalid				
庄 - ? FThunk:00000020 NbFunc:11 (decimal:17) valid:NO					
庄 - ? FThunk:00000068 NbFunc:9 (decimal:9) valid:NO	Show Suspect				
🗄 - ? FThunk:00000090 NbFunc:8 (decimal:8) valid:NO 😑					
🚊 ? FThunk:000000B4 NbFunc:1 (decimal:1) valid:NO					
庄 - ? FThunk:000000C0 NbFunc:4 (decimal:4) valid:NO	Auto Trace				
庄 - ? FThunk:00000138 NbFunc:6 (decimal:6) valid:NO					
🔄 ? FThunk:0000015C NbFunc:7 (decimal:7) valid:NO					
连 · ? FThunk:00000184 NbFunc:7 (decimal:7) valid:ND 🔍	Clear Imports				
Log					
IAT read successfully.					
Current imports:	Clear Log				
0 (decimal:0) valid module(s)					
51 (decimal:81) imported function(s). (added: +51 (decimal:+81))					
(51 (decimal:81) unresolved pointer(s)) (added: +51 (decimal:+81))					
IAT Infos needed New Import Infos (IID+ASCII+LOADEB)	Options				
0EP 000204E0 IAT AutoSearch RVA 00000000 Size 00000104					
RVA 00000000 Size 00001000	About				
RVA 00000000 Size 000010000 V Add new section	Exit				
Load Tree Save Tree (Get Imports) Fix Dump					

Figure 6.3: Saving the file

As it is displayed above (Fig. 6.3), the log should show that the unpacked executable was saved. In this case (and this is not unusual) the unpacked executable is larger than the original file (before it was initially packed).

Also note, that with many of the NsPack 3.4 options, the IAT does not automatically resolve.

Warning!	×
IAT is still invalid. You have to fix manually all unreso	ved pointers.
	ОК

Figure 6.4: Still no luck

We have to manually fix the IAT when confronted with selected sets of options.

6.1. Automation with OllyScript

To automate this process, we use OllyScript. We begin by loading the sample into OllyDbg (as occurred in the previous section).

Start by opening OllyDbg. Go to:

☑ Plug-ins -> OllyScript -> Run Script -> Load...

This has been displayed in the image below:

bug	Plug	ins Options	Window	/ H	elp			
×		1 AnalyzeThis	+		E M	T	WH	IC/KBR
		2 OllyDump						
		3 OllyScript	•		Run script	•		Load
					Abort Pause Resume			C:\Data\HCL\NSP C:\Data\HCL\nsp: C:\Users\Craiq.Wr
					Step			C:\Data\HCL\NSP
					About		_	C:\Users\Craig.Wr
				_				

Figure 6.5: Using OllyScript

To do this, OllyScript needs to be installed before you open Olly. Again, as in the last section, ensure that you have copied the plug-in to the correct directory.

Load the script first, and then open the sample (as in the first instance).

🔆 OllyDbg - cmd.exe - [CPU - main thread, module cmd]						
C File View Debug Plugin:	s Op <u>t</u> ions <u>W</u> indow <u>H</u> elp					
Paused 🔄 🛃 🗙 🕨 🔢		WHC/KBRS 🗄 📰 ?				
4PD4F20E \$ 9C 4PD4F20F 60 E8 00000000 4PD4F215 \$ 5D 07 4PD4F216 83ED 07 4PD4F215 \$ 4PD4F216 83ED 07 4PD4F216 83ED 07 4PD4F217 8080 6EFEFFFF 4PD4F217 8080 6EFEFFFF 4PD4F2217 8080 07884 42020000 4PD4F228 8EC5 4PD4F228 2B85 02FEFFFF 4PD4F239 8185 32FEFFFF 4PD4F239 8185 32FEFFFF 4PD4F237 8085 76FEFFFF 4PD4F247 9 8185 32FEFFFF 4PD4F245 6106 4PD4F245 6106 90100000 4PD4F245 64 40 4PD4F248 68 90100000 4PD4F250 68 90100000 4PD4F250 68 901000000 4PD4F257 F795 9AFEFFFF 4PD4F251 9F84 69030000 4PD4F255 85C0 9	PUSHED PUSHED PUSHED PUSHED PUSHED SUB EBP.7 LEA ECX, DWORD PTR SS: LEBP-1923 CMP BYTE PTR DS: LECX], 1 UE cmd. 4RD4F464 MOV BYTE PTR DS: LECX], 1 MOV EAX. EBP SUB EAX. DWORD PTR SS: LEBP-1FE].EAX MOV DWORD PTR SS: LEBP-1CE].EAX ADD DWORD PTR SS: LEBP-1CE].EAX ADD DWORD PTR SS: LEBP-16B1 ADD DWORD PTR DS: LESI].EAX PUSH ESI PUSH 1000 PUSH 1000 PUSH 1000 PUSH 1000 PUSH 1000 PUSH 1000 PUSH 1000 PUSH 1000 PUSH 200 PUSH 1000 PUSH 200 PUSH	This is the OEP (original entry point). Kernel32.763FD0E9 kernel32.BaseThreadInitThunk kernel32.BaseThreadInitThunk kernel32.BaseThreadInitThunk kernel32.BaseThreadInitThunk kernel32.BaseThreadInitThunk				
4904E265 8985 29EEEEE	MOU DWORD PTR SS: FERP-1D61.FAX	kernel32.BaseThreadInitThunk				

Figure 6.6: Loaded in Olly and running

Again, we are at the start of our packed executable. This time, instead of manually finding the OEP, we will use the script that we loaded. To do this, go to:

 $\blacksquare Plug-ins -> OllyScript -> Run Script -> C:\Data \...$

🜟 OllyDbg	🔆 OllyDbg - cmd.exe - [CPU - main thread, module cmd]						
C File	C File View Debug Plugins Options Window Help						
Paused	Paused D 🗰 🗰 🗙 1 Analyze This 🕟 📄 👬 📙 E M T W H C 🖊 K B R 💀 S 🗄 📰 ?						
4AD4F20E 4AD4F20F	\$ 9C • 60	2 OllyDump 💦 🕨 🕨		Registers (FPU)			
4AD4F210 4AD4F215 4AD4F216	. E8 000000 \$ 5D . 83ED 07	3 OllyScript 🔹 🕨	Run script 🔹	Load			
4AD4F219 4AD4F21F 4AD4F222	. 8D8D 6EFEFFFF . 8039 01 .∨ 0F84 42020000	LEA ECX, DWORD PTR S CMP BYTE PTR DS: LEO JE cmd. 4AD4F46A	Abort	C:\Data\HCL\NSPack 3.7\Deliverables\OllyScript-NSPack3.7.txt			
4AD4F228 4AD4F228 4AD4F22D	. C601 01 . 8BC5 . 2B85 02FEFFFF	MOV BYTE PTR DS: [E(MOV EAX,EBP SUB EAX,DWORD PTR \$	Pause	C:\Data\HCL\nspack.olly.txt			
4AD4F233 4AD4F239	 8985 02FEFFFF 0185 32FEFFFF 	MOV DWORD PTR SS: LE ADD DWORD PTR SS: LE	Resume	$\label{eq:c:Users} C: \label{eq:c:Users} C$			
4AD4F23F 4AD4F245 4AD4F247	. 8DB5 76FEFFFF . 0106 . 55	LEA ESI, DWORD PTR S ADD DWORD PTR DS: LE PUSH EBP	Step	C:\Data\HCL\NSPack 3.7\NsPack 3.7 (Unpacking)\NsPack 3.7 OEP Finder #2.			
4AD4F248 4AD4F249	. 56 . 6A 40	PUSH ESI PUSH 40	About	C:\Users\Craig.Wright.INFO-DEFENSE\Desktop\nSPack 2.x - 3.x.txt			
4AD4F24B 40D4F250	 68 00100000 68 00100000 	PUSH 1000		0 0 LastErr ERROR_SUCCESS (00			

Figure 6.7: Loading the script in Olly

Select the correct script that is loaded into the OllyScript plug-in. The script used in this instance has been included in the Appendix.

We will first get the statement that this code is packed again. Select "Yes" to continue.

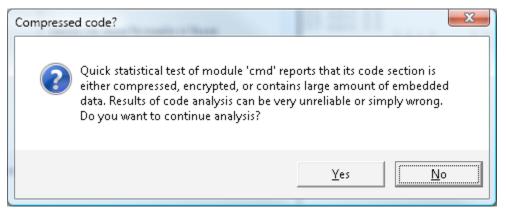
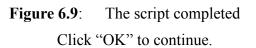


Figure 6.8: The file is packed

The script should complete and return the message displayed in Fig. 6.9.

OllyScript	X
1	Script finished
	ОК



At this point, we should find that we are at the OEP.

4AD09764	.^ E9 FEFDFFFF	JMP cmd.4AD09567	
4AD09769	90	NOP	
4AD0976A	90 90 90	NOP	
4AD0976B	90	NOP	
4AD0976C	90	NOP	
4AD0976D	90	NOP	
4AD0976E	\$ 8BFF	MOV EDI,EDI	
4AD09770	r. 55	PUSH EBP	
4AD09771	. SBEC	MOV EBP,ESP	
4AD09773	. 56	PUSH ESI	
4AD09774	. 8B75 08	MOV ESI, [ARG.1]	
4AD09777	. 3300	XOR EAX, EAX	kernel32.BaseThreadInitThunk
4AD09779	> 3B75 0C	CMP ESI, LARG. 21	ntdll.77AF0FC5
4AD0977C	.~ 73_11	JNB SHORT cmd. 4AD0978F	
4AD0977E	. 8500	TEST EAX,EAX	kernel32.BaseThreadInitThunk
4AD09780	.~ 75_0D	UNZ SHORT cmd.4AD0978F MOV_ECX,DWORD PTR DS:[ESI]	
4AD09782 4AD09784	. 880E . 85C9	MOV_ECX, DWORD PTR DS: [ESI]	
48009784	. 8509	TEST ECX, ECX	
4AD09786	.~ 74 02	JE SHORT cmd. 4AD0978A	
4AD09788	· FFD1	CALL ECX	
4AD0978A	> 8306_04	ADD ESI,4	
4AD0978D	.^ EB EA	UMP SHORT cmd. 4AD09779	1
4AD0978F	> 55	POP ESI	kernel32.763FD0E9
4AD09790	. 5D	POP_EBP	kernel32.763FD0E9
4AD09791	L. C3 90	RETN NOP	
4AD09792 4AD09793	70		
4AD09793	90 90	NOP NOP	
4AD09795	90	NOP	
4AD09796	90	NOP	
4AD09797	> PE8 05FAFFFF	CALL cmd. 4AD091A1	This is the OEP (original entry point). Use this to f
4AD0979C	. 6A 10	PUSH 10	into to the our correginat entry pothers, ose thits to r
4AD0979E	. 68 8898D04A	PUSH cmd. 4AD09888	
4AD097A3	. E8 6E99FFFF	CALL cmd. 4AD03116	
4AD097A8	. 33DB	XOR EBX.EBX	
40000700	· 00000 cc	MOU DUODD DTD CONFERD 41 COV	

Figure 6.10: The OEP

At this point, we will dump and reconstruct the IAT in the same manner as in the

previous section.

6.2. Summary of the process

The summary of the method to uncompress NsPack in OllyDbg involves the following steps:

- 1. At entry point, add a breakpoint in the PUSHA instruction and run the application.
- 2. After it breaks, follow the ESP register value in dump; add a hardware breakpoint with 4 bytes length in the first bytes.
- 3. Run the application again (F9).
- 4. At the next break (BP), the EIP will be at the transfer command.
- 5. Simply single step into it (F8) and the value at EIP will be at the original entry point.

7. Analyzing NsPack itself

Looking at the NsPack executable and using PEiD we see that NsPack is itself packed using ASProtect Version 2.1.x.

🔛 PEiD ∨0.95	- • 💌			
File: C:\Data\HCL\NSPack 3.7\Nspack3.7	.exe			
Entrypoint: 00001000	EP Section:			
File Offset: 00000600	First Bytes: 68,01,D0,5A >			
Linker Info: 6.0	Subsystem: Win32 GUI >			
ASProtect 2.1x SKE -> Alexey Solodovnikov [Overlay] Multi Scan Task Viewer Options About Exit ✓ Stay on top >>> EXIT				

Figure 7.1: NsPack uses ASProtect

It is also possible to quickly get the OEP of NsPack 3.7 using PEiD:

₩ PEiD v0.95		Ì			
File: C:\Data\HCL\NSPack 3.7\Nspack3.7.ex	e				
Entrypoint: 00001000 File Offset: 00000600 Linker Info: 6.0	EP Section: > First Bytes: 68,01,D0,5A > Subsystem: Win32 GUI >				
ASProtect 2.1× SKE -> Alexey Solodovnikov [Multi Scan Task Viewer Options Stay on top	Overlay] About Exit		Plugins	•	Generic OEP Finder
			Normal Scan Deep Scan Hardcore Scan		Krypto ANALyzer PEiD Generic Unpacker
			External Scan		

▼ Plug-ins -> Generic OEP Finder

Figure 7.2: PEiD and the OEP finder

In this case we have the OEP returned at OEP: 004897F7.

GenOEP 💽
Found OEP: 004897F7
ОК

Figure .3: The OEP is found

Next, we start Olly. The following plug-in is essential:

☑ IsDebugPresent API

ASProtect has a debugger detection routine. The plug-in is needed to ensure that the program does not crash prematurely. This file is available from OpenRCE:

Mhttp://www.openrce.org/downloads/details/111/IsDebuggerPresent

To load and enable this plug, go to:

☑ Plug-ins, IsDebugPresent

Select "option"

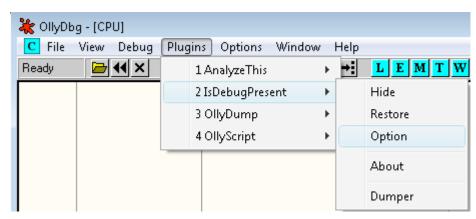


Figure 7.4: The IsDebugPresent flag

The auto-hide function should be set to match the load times of the host running the analysis.

Option	×
AutoHide WARNING this option may cause problems	
Sleep Time 1000 milliseconds	
Save Quit	

Figure 7.5: Setting options

- Next, select whether to automatically hide the debugger not (Autohide). If checked when you load an exe, debugger is hidden and you can choose how long thread will sleep until patch byte API is done (Sleep Time).
- It is also possible to manually hide or restore debugger with menu option.
 Ensure that the exceptions have been disabled (other than Kernel32
 Memory access violations) by entering "Alt-O" in Olly and removing any ticked boxes:

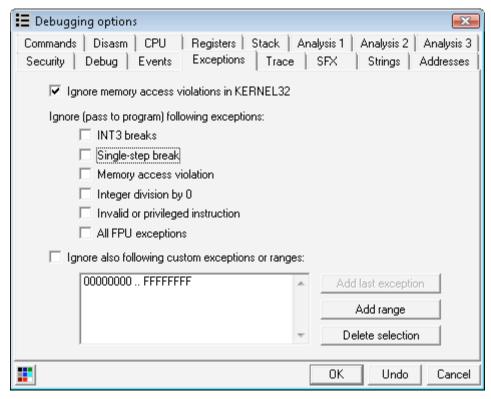


Figure 7.6: Set the options

7.1. Quit and load NsPack 3.7.

Enter "F9" to 'run' the program. This will take us to the first exception:

	004A91D0 D 004A91E0 F 004A91F0 2 004A9200 A)7 07 F 08 28 C9 38 BE	B1 FD F6 87	DC 23 9D B6 DC A5 BC 53	2C F 99 6 9F D 0D E	9 C4 7 D8 D 3C B E4	9D 1B BD 56	16 E6 C3 ØA	87 6C 58 80	38 6A 9C CE	AE 73 65 8A	BC 9E 86 02	6D DA F6 AD	BØ DB C9 3A	<pre>#qA(■+¶Y&. uù±\$\$ I.∰.#,Ø_c8<4m; ■*ØA0gi+µljs×+■ ([F+=∭Afi<+]×£e8+■ ([F+=∭Afi<+]×£e8+= %¥Q4\$\$.ù\$U.Cfi2€i: ±+v=1:% bikt3</pre>
	•														
- î													-		
- 1	Access viola	ation w	hen	writing	to (00	0000	00]•	 use 	۶h	ift+F	-7/F	⁻ 8/F	9 to	pas	s exception to program

Figure 7.7: Exception 1

We will need to count the number of exceptions that are returned. With this information we can restart and step directly to the final exception, BP on the code section (where we should reach the OEP).

Next, dump the program and repair the IAT.

To do this, enter "Shift + F9" for each returned exception. Remember to count the number of exceptions returned.

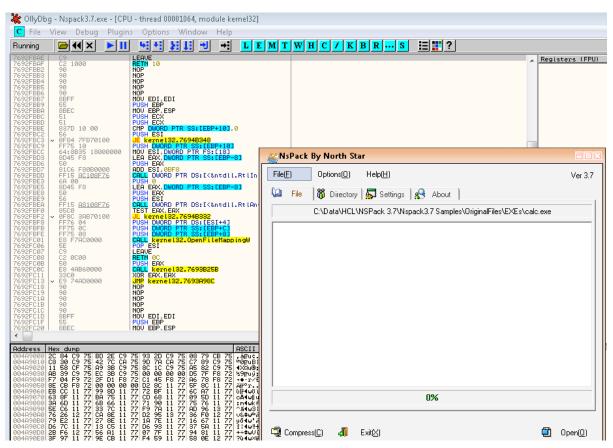


Figure 7.8: Interacting with the program being debugged

Skipping past the exceptions, we can interact with NsPack (Fig. 7.8).

From this we have the message that the executable is packed. This was already known, so simply select "yes" to continue.

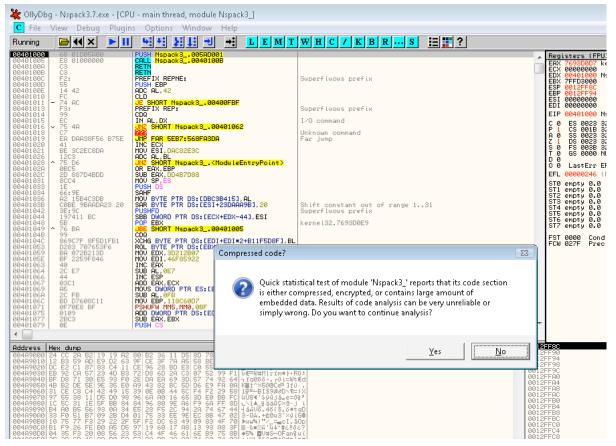


Figure 7.9: Compression noted

Olly has now loaded the module and is awaiting our input. We should go directly to this point if the plug-in (IsDebugPresent) loaded:

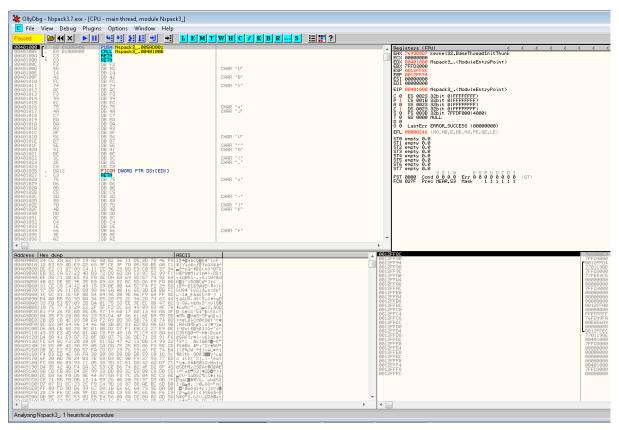


Figure 7.10: Jumping to the exception

The reason for counting the exceptions was to be able to jump directly to final exception.

Select "M" (See below circled in the upper left). This gives us the Memory map (Fig. 7.11).

We now set a breakpoint on the section 'code'. Do this with the mouse (Fig. 7.12), or by entering 'F2'.

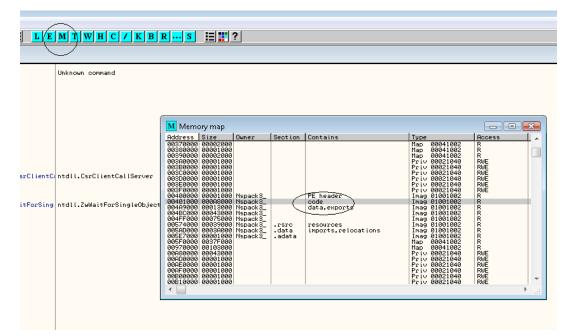


Figure 7.11: A Memory map

M Memor		1	1	1-		×
00380000 0 00300000 0 00300000 0 00320000 0 00350000 0 0040000 0 0040000 0 0040000 0	0002000 0001000 0002000 0001000 0001000 0001000 0001000 0001000 0001000 0001000 Nspack3	-	Contains PE header code	Type Map 00041002 Map 00041002 Priv 00021040 Priv 00021040 Priv 00021040 Priv 00021040 Priv 00021040 Priv 00021040 Priv 00021040 Imag 01001002	Access R R RWE RWE RWE RWE RWE RWE RWE RWE RWE	
0048C000 8 004FF000 8 00574000 8 005AD000 8 005E7000 8 005F0000 8 00970000 8 00970000 8 00A30000 8 00A20000 8 00A20000 8 00A20000 8	00001000 Nspack3 0037F000 00103000 00043000 00001000 00001000 00001000 00001000	.rsrc .data	data,exports resources imports,relocations	Actualize View in Disassem Dump in CPU Dump Search Search next	bler	Enter Ctrl+B Ctrl+L
				Set break-on-acc	ess	F2
				Set memory brea Set memory brea Set access	kpoint on access kpoint on write	Þ
				Copy to clipboar Sort by Appearance	d)

Figure 7.12: Break on access

Using the standard techniques, we can then rebuild the IAT.

7.2. Transfer Command

This is where the program jumps to the real (unpacked) original code entry point (OEP).

00000000 61	POP A	
00000001 9D	POP F	
00000002 E9 ?? ?? ?? ??	JMP	<value></value>

7.3. Entry Point Signature

The entry point signature is the series of unique OP code instructions in a binary that we can use to make a simple detector. It is still necessary to test the hypothesis that the packer detected actually exists (it is not a false positive), but testing a small number of possible files is simpler than testing all files on a system.

00000000 9C 00000001 60 00000002 E8 00000007 5D 00000008 83 0000000B 8D 00000011 80	ED 07 ?? ?? ?? ?? ?? 39 01	PUSH F PUSH A CALL POP SUB LEA CMP IZ	00000003 EBP EBP, 7 ECX, [EBP-value] Byte PTR [ECX], 1 value
00000014 0F	84 ?? ?? ?? ??	JZ	value

9C 60 E8 00 00 00 5D 83 ED 07 8D ?? ?? ?? ?? 80 39 01 0F ?? ?? ?? 00 00

Hence a simple signature could be defined as:

[NsPack 3.7 -> Liu Xing Ping] signature = 9C 60 E8 00 00 00 5D 83 ED 07 8D 85 ?? ?? FF FF ?? 38 01 0F 84 ?? 02 00 00 ?? 00 01 ep_only = true

7.4. Basic Details of NsPack 3.7

In general, NsPacked files report having three sections (.nsp0, .nsp1, and .nsp2). This is user configurable and these can be set to any value. Consequently, the Entry Point Signature (above) is a better means of detecting NsPack than simply using the section headers alone.

7.4.1. PE Structure information

PE Info returns the following information above a generic NsPack compresses

file.

(base data)

entrypointaddress.: 0x7b48e3

(3 sections)

name viradd virsiz rawdsiz ntrpy md5

.nsp0 0x1000 0x3b0000 0x0 0.00 d41d8cd98f00b204e9800998ecf8427e

.nsp1 0x3b1000 0xab000 0xaa6c3 7.99 bc5e2a11a697427c5ec95bb5cabea1dc

.nsp2 0x45c000 0x128b 0x0 0.00 d41d8cd98f00b204e9800998ecf8427e

<u>Note</u>: The section names are variable and can be set to anything by the user.

(1 imports)

> KERNEL32.DLL: LoadLibraryA, GetProcAddress, VirtualProtect, VirtualAlloc, VirtualFree, ExitProcess

As noted, the user can change the section names from the default '.nspX' value.

The first section is unpacked.

7.4.2. Calculating the PE File Execution Start Offset in NsPacked files

In the image below, we see the header information of a typical program that is packed using NsPack. This first example uses the standard options and naming for the section headers.

Member	Offset	Size	Value	Meaning
Machine	000000E4	Word	014C	Intel 386
NumberOfSections	000000E6	Word	0003	
TimeDateStamp	000000E8	Dword	48025287	
PointerToSymbolT	000000EC	Dword	00000000	
NumberOfSymbols	000000F0	Dword	00000000	
SizeOfOptionalHea	000000F4	Word	00E0	
Characteristics	000000F6	Word	010F	Click here

Figure 7.13: The structure of NsPack

This file also has the following Optional Header Section:

Member	Offset	Size	Value	Meaning
Magic	000000F8	Word	010B	PE32
MajorLinkerVersion	000000FA	Byte	07	
MinorLinkerVersion	000000FB	Byte	0A	
SizeOfCode	000000FC	Dword	00000000	
SizeOfInitializedData	00000100	Dword	0000D000	
Size Of Uninitialized Data	00000104	Dword	00030000	
AddressOfEntryPoint	00000108	Dword	000380F9	.nsp0
BaseOfCode	0000010C	Dword	00001000	
BaseOfData	00000110	Dword	00031000	
ImageBase	00000114	Dword	01000000	
SectionAlignment	00000118	Dword	00001000	
FileAlignment	0000011C	Dword	00000200	

Figure 7.14: Optional Headers

With the standard NsPack section naming conventions.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word
.nsp0	00030000	00001000	00000000	00000400	00000000	00000000	0000
.nsp1	0000D000	00031000	0000CFC2	00000400	00000000	00000000	0000
.nsp2	00001538	0003E000	00000000	00000400	00000000	00000000	0000

Figure 7.15: Standard naming of the sections

The address of entry point that is stored in the optional header is a *relative virtual*

address (RVA), where the loader will begin execution. An RVA is simply the offset of an item, relative to where the file is memory-mapped.

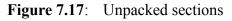
A comparison of the unpacked Notebook.exe and an NsPack version of the same are displayed below loaded into Protection ID to display the section and header values.

2	😻 🔲 📲 🖷 🗇	🌍 🚨 🕀 🛄 🔟					
	FileName C:\Data\HCL\	NSPack 3.	7\Nspack3.7 Sam	ples\OriginalFile:	s\EXEs\notepa	Browse	
	MZ Header	Name	VirtSize	RVA	PhysSize	Offset	Flag
	FileHeader OptionalHeader SectionInfo Import Dir Data Resource Dir	.nsp0 .nsp1 .nsp2	0x00030000 0x0000D000 0x00001538	0x00001000 0x00031000 0x0003E000	0x00000000 0x0000CFC2 0x00000000	0x00000400 0x00000400 0x00000400	0xF0000060 0xE0000060 0xE0000060
		<u>C</u> lear	Process	Mem Used	0 MB 32	2Bit File	
<u>S</u> can	Status Operational				Queue		·

Figure 7.16: The packed sections

And the unpacked version is displayed in Fig. 7.17.

1	😻 🔲 📲 📲 🔲 🖤	🌍 Ω 🕀 🗉 🔟					
	FileName C:\Windows\no	otepad.exe				Browse	
	MZ Header	Name	VirtSize	RVA	PhysSize	Offset	Flag
	FileHeader	(.text	0x00008F40	0x00001000	0x00009000	0x00000400	0x60000020
<u>8</u>	OptionalHeader	.data	0x00002124	0x0000A000	0x00001000	0x00009400	0xC0000040
A 🔜	- SectionInfo ≡	.rsrc	0x00019A18	0x0000D000	0x00019C00	0x0000A400	0x40000040
	🖶 Import Dir	.reloc	0x00000D18	0x00027000	0x00000E00	0x00024000	0x42000040
	- Resource Dir						
😪 🔳	🕀 Fixups Dir						
	 Debug Dir Load Config Dir 						
	Load Config Dir						
🥌 💷							
🎇 🔛	Iree	<u>C</u> lear	Process	Mem Used	0 MB 32	2Bit File	
<u>S</u> can	Status Operational				Queue		



The following are the basic stages used to get to the *file execution start offset*:

1. Determine each section's virtual memory map (that is the virtual start address and end address. The virtual address and virtual size for each section can be found in the section header from the executables PE Header).

- 2. Establish in which section's virtual space the address of entry point is located.
- 3. Validate the offset of that section as per the section header. In the section header, the pointer to raw data field gives us the file-based offset where the section data/bytes begin.
- 4. Calculate the difference between the address of entry point and the virtual address of the section in which the entry point lies. Add this difference to the pointer to raw data, which is the file-based offset of the section, in order to get the file-based execution start offset for the particular file.

Hence, using this data we can calculate the *file execution start offset* for this file:

[(Address of Entry Point) – (Virtual Address)] + (Pointer to Raw Data) = (*file execution start offset*)

The '*Pointer to Raw Data*' value is also called the 'Offset' or 'Raw Address'. Now, by inserting the values from our tables above, we get (these values come from the .nsp0 section header and the main optional headers):

(0x000380F9 - 0x00001000) + 0x00000400 = 0x00038CF9

This calculated value is not necessarily the offset where file execution actually begins with NsPack compressed files.

If we take another example, in this case packed with several NsPack options applied, we get a different type of calculation.

File Optional Header Number of sections: Address of entry point: Image base:		02 00001010 00400000	Section alignment: File alignment:	00001000 00000200
Section Headers Section Virtual	Virtual	Size of	Pointer	Characteristics
name size	Address raw		a raw	data
nsp0 00004000	000010	00000 0000	00B 0000001C	E0000060
nsp1 0000203D	000050	00000 0000	CFD 00000200	E0000060

Hence we have:

(0x00001020 - 0x00001000) + 0x0000001B = 0x0000003B

In Windows, the loader rounds the pointer to raw data to 0x00000000 as it is lower that the *'file alignment value'* (in this example = 0x00000200). As a consequence, the loader assumes that the first section, nsp0, starts at file offset 0 and loads the section accordingly in the memory. So if we round the pointer to raw data, as the loader does, the file execution start offset is calculated as follows:

(0x00001020 - 0x00001000) + 0x00000000 = 0x00000040

The offset 0x00000040 is located within the DOS header of the PE file. Hence this means that it can land within the reserved section of the DOS header (this section is normally filled with zeros). From this location, NsPack inserts a five-byte jump instruction. The reason is that this will transfer control to code further ahead in the program.

<u>Note</u>: It is essential that a check is implemented for occasions where the pointer to raw data is not a multiple of the file alignment. In these instances, this value needs to be rounded to the nearest multiple and the remaining extra bytes should be passed over. For files whose file alignment value is not 0x00000200, the loader rounds it to a multiple of 0x00000200.

As noted, the section header names are variable and as can be seen in the section header table displayed below, these can easily be changed (with a flag in the program) to a different set of values.

Name	Virtual Size	Virtual Address	Ravv Size	Raw Address	Reloc Address	Linenumbers	Relocations N
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word
∧" # 0	0000C000	00001000	00000000	00000200	00000000	00000000	0000
∧"#1	00005000	0000D000	0000404D	00000200	00000000	00000000	0000
^" # 2	0000038C	00012000	00000000	00000200	00000000	00000000	0000

Figure 7.18: Strange section names

Also note that section '.nsp1' (or its equivalent if renamed) can extend beyond the raw file offset of section '.nsp2'

7.5. The decompression algorithm

NsPack uses a single format for compression/decompression. There seems to be little difference between the versions of the program for this function. The initial section of all NsPack 3.7 (and version 3.4) compressed executables is .nsp1 (or the renamed functional equivalent) with first bytes, 9C,60,E8,00. The basic layout the routine is displayed below:

PUSHAD /* PUSHAD saves all the register values onto the stack */ /*de-compression routine here (see appendix)*/ POPAD /*POPAD restores the previously saved data */ /* from the stack to the registers */ JMP OEP /* The Real Original EP */

The Jump to the OEP is made after the de-compression has run and the executable code has been decompressed.

🖽 N 📖	
loc_40D40B: popa popf jmp near ptr <mark>byte_4012A8</mark> ; Jumpt to the Da start endp	ata Se <mark>g</mark> ment – the OEP

Figure 7.19: IDA Pro

The de-compressed code is in effect the original code and it does not have a record that any additional code has been executed prior to arriving at the OEP. The reason for this is that the instructions in the de-compressed code are expecting certain values. These may conflict with any errors that might result from variations in the register values. Consequently, the only instruction that will interact with the values placed on the stack by PUSHAD is the final POPAD instruction.

7.6. Data Structures

Data is stored as a little endian format. This is the least significant bit (LSB) precedes the most significant bit (MSB) in memory. When certain options are used, the COFF line numbers can be removed from the resultant packed executable. COFF symbol

table entries for local symbols can also be removed.

7.7. Functions

The main decompression routine utilizes five (5) functions. These have been disassembled in the Appendices. The entry-point for the decompression function of NsPack begins with a **pushf** and **pusha**. These commands save all the registers (pusha) which are later restored using a popa (Fig. 7.20).

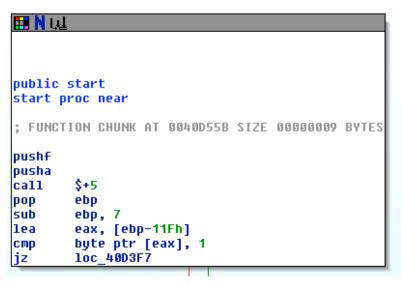
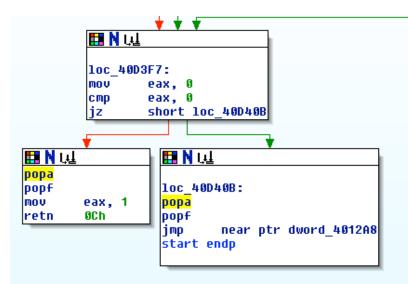


Figure 7.20: IDA Pro showing the ASM function

The popa/popf listed below shows the end of the decompression routine.





Breaking or dumping the decompressed program is possible at the final pop

instructions. The memory image has been decompressed at this stage. The section, .nsp0 (or equivalent name) is a working area used by the program to load the decompressed data. The start of this section later becomes the location of the decompressed function.

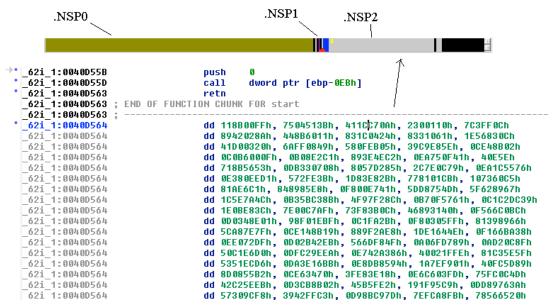


Figure 7.22: The first section

The section, .nsp2 (or equivalent name) holds the compressed (original) executable. This data is called by the decompression routine (.nsp1). This routine takes the data from .nsp2 and copies it (once it is decompressed) into .nsp0.

7.8. Differences in versions

The differences in versions 3.4 and 3.7 of NsPack are small. This can be seen in the table below and in the appendix.

Version 3.7	Version 3.4
.nsp1:4AD5C0B3 mov ecx, edi	_62i_1:0040D251 mov eax, [ebp-163h
.nsp1:4AD5C0B5 mov eax, [ebp-1E6h]	_62i_1:0040D257 add eax, 5AAh
.nsp1:4AD5C0BB add eax, 5AAh	_62i_1:0040D25C call eax
.nsp1:4AD5C0C0 call eax	_62i_1:0040D25E pop ebx
.nsp1:4AD5C0C2 pop ebx	_62i_1:0040D25F pop edx
.nsp1:4AD5C0C3 pop ecx	_62i_1:0040D260 pop ecx
.nsp1:4AD5C0C4 pop edi	_62i_1:0040D261 pop edi

Even when separate options are selected, the decompression routine remains the same (with different section header names and locations).

```
.nsp1:4AD5C13B lea ecx, [ebp-<u>1CAh]</u>____62i_1:0
```

_62i_1:0040D2D8 lea ecx, [ebp-<u>147h]</u>

The differences that result from the options are associated with the location that the data is mapped to in the executable. The structure varies slightly between versions.

7.9. NsPack Dependencies

The following section displays the dependencies used by NsPack. PE Explorer Dependency Scanner is used for this first analysis.

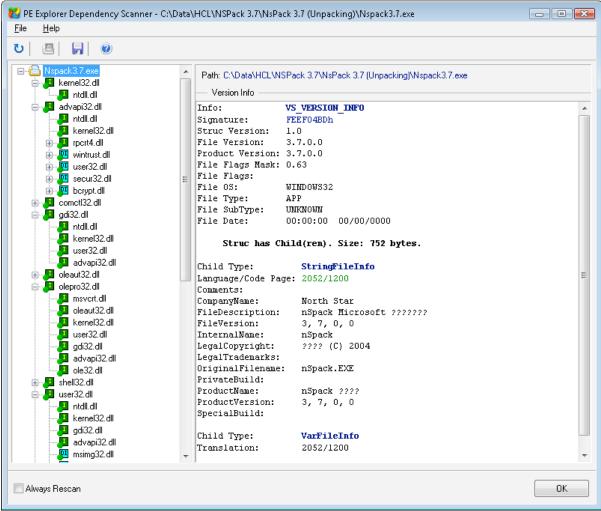


Figure 7.23: The dependencies used by NsPack

The dependencies used by an executable provide us with an insight into the program. These allow us to see the possible system calls that may be made and if the program uses network calls and other such features.

0 🖪 🖥 🖗			
Nspack3.7.exe	Path: C:\Data\HCL\NS	Pack 3.7\NsPack 3.7 (Unpacking)\Nspack3.7.exe	
⊕ <mark>,™</mark> kernel32.dll ⊕ <mark>.™</mark> advapi32.dll			
terreti32.dl	Info:	VS VERSION INFO	
adi32.dll	Signature:	FEEF04BDh	r
⊡ <mark></mark> oleaut32.dll	Struc Version:	1.0	
······································	File Version:	3.7.0.0	
shell32.dll	Product Version:	3.7.0.0	
	File Flags Mask:	0.63	
🗄 🛄 winmm.dll	File Flags:		
🗄 互 winspool.drv	File OS:	WINDOWS32	
🗄 🛄 comdlg32.dll	File Type:	APP	
ie 💶 ole32.dll	File SubType:	UNKNOWN	
🗄 💶 oledlg.dll	File Date:	00:00:00 00/00/0000	
-	Struc has Cl	uld(ren). Size: 752 bytes.	
	Child Type:	StringFileInfo	
	Language/Code Pag	je: 2052/1200	
	Comments:		
	CompanyName:	North Star	
	FileDescription:	nSpack Microsoft ??????	
	FileVersion:	3, 7, 0, 0	
	InternalName:	nSpack ???? (C) 2004	
	LegalCopyright: LegalTrademarks:	???? (C) 2004	
	OriginalFilename:	nSpack.EXE	
	PrivateBuild:	hopdon i braz	
	ProductName:	nSpack ????	
	ProductVersion:	3, 7, 0, 0	
	SpecialBuild:		
	Child Type:	VarFileInfo	
	Translation:	2052/1200	
	J		

Figure 7.24: Detailed info

PE Explorer can be used to display the executables dependencies. Another option

is to use Dependency Walker (http://www.dependencywalker.com/). As the site states:

Dependency Walker is a free utility that scans any 32-bit or 64-bit Windows module (exe, dll, ocx, sys, etc.) and builds a hierarchical tree diagram of all dependent modules. For each module found, it lists all the functions that are exported by that module, and which of those functions are actually being called by other modules. Another view displays the minimum set of required files, along with detailed information about each file including a full path to the file, base address, version numbers, machine type, debug information, and more.

This tool is free and can be used for analyzing the functional dependency tree of a

program.

GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer. 75

File East View Options Profile Window Help	Dependency Walker - [Nspack3.7.exe]	×
NSPACK37.EXE Image: Strength S2DLL Image: S	📑 Elie Edit View Options Profile Window Help	Ξ×
NSPACK37.EXE Image: Strength S2DLL Image: S		
A Module File Time Stamp Link Time Stamp File Size Attr. Link Checksum CPU Subsystem Symbols Preferred Base Actual Bas ∧ 30% FES-FIMS.DLL Error opening file. The system cannot find the file specified (2). IESFIMS.DLL Error opening file. The system cannot find the file specified (2). IESFIMS.DLL 21/11/2009 5:34p 21/11/2009 5:35p 11/06/952 A 0x00.A95309 x86 GUI CV 0x75160000 Unknown AVXPI32.DLL 21/11/2009 5:34p 11/04/2009 5:25p 11/06/2009 5:25p 800,768 A 0x00.09573 0x00.085963 x86 GUI CV 0x77 C80000 Unknown COMCT132.DLL 19/01/2008 6:26p 131/04/2009 5:26p 450,560 A 0x00.07549 0x00.07549 x86 GUI CV 0x77 C80000 Unknown COMDLG32.DLL 11/04/2009 5:28p 11/04/2009 5:26p 450,560 A 0x00.07549 x86 GUI CV 0x77 C80000 Unknown CMDLG32.DLL 11/04/2009 5:28p 11/04/2009 5:22p 450,560 A 0x00.07549 x86 GUI	Image: Strengt 12, DLL Pl Ordinal ^ Hint Function Entry Point Image: Strengt 12, DLL Image: Strengt 12	
IESHIMSDLL Error opening file. The system cannot find the file specified (2). ● INSPACK3.7.EVE Could not find the section that owns the Export Directory. ■ IIFFRAMEDLL 21/11/2009 5:34p 21/11/2009 5:35p 11/06/2009 ▲ DVAPI32.DLL 11/04/2009 5:24p 11/04/2009 5:22p 800,768 A 0x00.049530.9 x86 GUI CV 0x7751600.00 Unknown ■ COMDLG32.DLL 11/04/2009 5:24p 11/04/2009 5:24p 10/06.950 A 0x00.085963 x86 GUI CV 0x77.08.00.00 Unknown ■ COMDLG32.DLL 11/04/2009 5:24p 11/04/2009 5:24p 450,560 A 0x00.07554.9 0x80.07554.9 x86 GUI CV 0x77.08.00.00 Unknown ■ GDI32.DLL 11/04/2009 5:24p 450,560 A 0x00.07554.9 x86 GUI CV 0x77.86.00.00 Unknown • COMDLG32.DLL 11/04/2009 5:24p 450,560 A 0x00.07554.9 x86 GUI CV 0x77.86.00.00 Unknown • GDI32.DLL 11/04/2009 5:24p 220,272.22 200,601.56.C31 0x00.056.C31 0x00.056.C31		Bas A
● INSPACK3.7.EVE Could not find the section that owns the Export Directory. 2 IEFRAME.DLL 21/11/2009 5:34p 21/11/2009 5:35p 11/06/9.952 A 0x0 0.A95309 0x0 0.A95309 x86 GUI CV 0x7 516 0000 Unknown 0 COMCT132.DLL 11/04/2009 5:26p 11/04/2009 5:26p 13/04/2009 5:26p 11/04/2009 5:26p 10/01/05 6:31 0x0 0076 5:49 x86 GUI CV 0x7 7 0.8000.00 Unknown COMDLG32.DLL 11/04/2009 5:26p 130,050 0x0 0076 5:49 x86 GUI CV 0x7 7 0.800.00 Unknown COMDLG32.DLL 11/04/2009 5:28p 11/04/2009 5:22p 450,050 0x0 0076 5:49 x86 GUI CV 0x7 7 0.800.00 Unknown CManigr34 Least one delay-load dependency module was not found.		
	NSPACK3.7.EVE Could not find the section that owns the Export Directory. IFFRAME.DLL 21/11/2009 5:38p 11/069,952 A 0x00 A95309 0x00 A95309 0x86 GUI CV 0x75160000 Unkm ADVAPI32.DLL 11/04/2009 5:28p 11/04/2009 5:22p 800,768 A 0x0002873 0x0006563 x86 GUI CV 0x77800000 Unkm COMCTI32.DLL 19/01/2008 6:26p 531,968 A 0x00076549 0x00076549 x86 GUI CV 0x70800000 Unkm COMDLG32.DLL 11/04/2009 5:28p 11/04/2009 5:22p 200,7564 0x00076549 x86 GUI CV 0x70800000 Unkm GD32.DLL 11/04/2009 5:28p 11/04/2009 5:22p 20,7272 20,400076549 0x00076549 x86 GUI CV 0x77860000 Unkm Console CV 0x77860000 Unkm 0x0076549 x86 GUI CV 0x77860000 Unkm	wn wn wn

Figure 7.25: Using Dependency walker

The decency tree for NsPack is displayed in Fig 7.26. From this we can see no

network modules are loaded.

📫 Dependency Walker - [Nspack3.7.exe]										
■ <u>Eile E</u> dit <u>V</u> iew <u>O</u> ptions <u>P</u> rofile <u>W</u> indow <u>H</u> elp									- & ×	
📄 🚔 📕 🔎 🟠 🗥 🖭 💭 😭 🗒 🍇 🖹 🚿 🖷 🖻	I 🔲 隆									
NSPACK3.7.EXE	PI O	rdinal ^	Hint	Function			Entry Point			
KERNEL32.DLL	C N,	/A	0 (0x0 0 0 0)	GetProcAddress			Not Bound			
ADVAPI32.DLL						Not Bound				
E COMCTL32.DLL	🗖 🖸 🛛 🔊 🗖	(Α	0 (0×0 0 0 0)	LoadLibraryA			Not Bound			
🛨 🔲 GDI32.DLL										
庄 📖 🔲 OLEAUT32.DLL										
🛓 🔲 OLEPRO32.DLL										
💼 🔲 SHELL32.DLL	•				111				- F	
USER32.DLL	E O	rdinal ^	Hint	Function			Entry Point			
🖶 🔲 WINMM.DLL		1 (0x0 0 0 1)		BaseThreadInitThunk			0x0004D0D7		î	
WINSPOOL.DRV		2 (0×0 0 0 2)		InterlockedPushListSLis	t			rlockedPushListSLi	t	
E COMDLG32.DLL		3 (0×0 0 0 3)		AcquireSRWLockExclus				uireSRWLockExclus		
DLE32.DLL		4 (0×0 0 0 4)		AcquireSRWLockShared	ł			uireSRWLockShare	d	
OLEDLG.DLL OLEAUT32.DLL		5 (0x0 0 0 5) 2 (0x0 0 0 2) ActivateActCtx					0×0001C691			
KERNEL32.DLL		l - l - i i i						0x00018086 +		
KERINELSZ.DEL	•			III					•	
^ Module File Time Stamp Link Time Stam	p File S	ize Attr.	Link Checksur	n Real Checksum	CPU	Subsystem	Symbols	Preferred Base	Actual Bas 🗠	
IESHIMS.DLL Error opening file. The system canno										
NSPACK3.7.EXE Could not find the section that owns				1		1	1	1	1	
3 IEFRAME.DLL 21/11/2009 5:34p 21/11/2009 5:3 ADVAPI32.DLL 11/04/2009 5:28p 11/04/2009 5:28p 11/04/2009 5:2		9,952 A 0,768 A	0x00A95309 0x000C9B73	0×00A95309 0×000C9B73	×86 ×86	GUI Console	CV CV	0×75160000 0×77C80000	Unknown Unknown	
ADVAPI32.DLL 11/04/2009 5:28p 11/04/2009 5:2 COMCTL32.DLL 19/01/2008 6:26p 19/01/2008 6:2		1,968 A	0x000C9B73	0x000C9B73	×86	GUI	CV	0x70800000	Unknown	
COMPLG32.DLL 11/04/2009 5:28p 11/04/2009 5:2		0,560 A	0×00076549	0×00076549	×86	GUI	cv	0×6 F C2 0 0 0 0	Unknown	
11/04/2009 5-285 11/04/2009 5-2		7 477 Δ	0-00056031	0-00056031	v86	Console	lov	0v77860000	Unknown T	
									•	
Warning: At least one delay-load dependency module was not fou										
Warning: At least one module has an unresolved import due to a m Warning: At least one module was corrupted or unrecognizable to										
warning: At least one module was corrupted or dimecognizable to	Dependen	cy waiker, bu	it sun appeareu	to be a milliows mouu	ie.					
For Help, press F1										

Figure 7.26: Detailed info in the tree

Another option is to use the tools from SysInternals to display the executable information associated with the NsPack compression program. In Fig 7.27 the detailed information from SysInternal Process Explorer (<u>http://technet.microsoft.com/en-us/sysinternals/default.aspx</u>) is displayed.

💐 Nspac	:k3.7.e	<e:10120< th=""><th>) Propertie</th><th>25</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>x</th></e:10120<>) Propertie	25									x
Image	Perfor	mance	Performa	nce Graph	Threads	TCP/IP	Security	Environment	Job	Strings			_
<u> </u>	solve a	ddresses	:										
Prot	o L	.ocal Ado	tress	Remote A	ddress	State							
L											<u>о</u> к	Cancel	
												Zancer	

Figure 7.27: Details in Process explorer

Selecting the process and then right-clicking to display the options allows the user to choose 'properties'.

Nspac	k3.7.exe:1012() Properties									
Image	Performance	Performance Graph	Threads	TCP/IP	Security	Environment	Job	Strings			
Printable strings found in the scan:											
PMOVMSKB_GdVRdq: required SSE2, useenable-sse option PSUBUSB_VdqWdq: required SSE2, useenable-sse option											
		;; required SSE2, use - q: required SSE2, use									
		required SSE2, usee		•							
		z: required SSE2, use ↔		•							
		lg: required SSE2, use		•							
		required SSE2, usee		•							
		quired SSE2, useen									
		equired SSE2, useer									
PSRA	ND_VdqWdq: re	equired SSE2, useen	able-sse op	tion							
PAVG	iW_VdqWdq: r	equired SSE2, useer	nable-sse op	otion							
		lq: required SSE2, use									
		: required SSE2, use		•							
		required SSE2, usee									
		required SSE2, use		•							
		required SSE2, usee required SSE2, usee		•							
		required SSE2, usee : required SSE2, use		•							
		: required SSE2, use : required SSE2, use									
		equired SSE2, useen									
		quired SSE2, useen/									-
•										•	
								_			_
Image O Memory Eind											
								ſ	<u>0</u> K	⊆and	el :

Figure 7.28: The strings from the executable in memory

This tab will allow us to see what is running on the system. This is less accurate then using a dependency scanner, but does help with a dynamic analysis of the running program.

8. Conclusion

This paper was written to provide a detailed analysis of NsPack (a formerly semicommercial PE packer) written by Liu Xing Ping and distributed from by North Star Software in China. NsPack is an executable file compressor for Windows 32 and 64 bit PE based executables. It also has the capability to work on .NET files. In marketing material and in tests (figure 3.3), it is shown that NsPack is capable of compressing the size of a 32-bit or 64-bit Windows executable by up to 60%. It is claimed (NsPack, 2009) that no noticeable performance change will result from this compression. There are better compression programs, but not all of these support 64-bit exe, dll, ocx and scr files. In addition, the far lower deployment rate of NsPack when compared to more common packers (such as UPX) means that less effort has been made to understand and automatically unpack the algorithm used.

The greatest challenge posed by NsPack is the ability to recompress an already compressed executable file. NsPack will recompress a PE file that has been compressed using Upx, Aspack, Pecompact, and several other packers. This slows the execution of the packed executable considerably, but make reverse engineering of the program extremely complex. Malware authors use this technique to further obfuscate their payloads. The techniques have not been widely deployed at present due to the inability of many anti-virus vendors to effectively decompress a large number of packers in real time. To help combat this, we analyzed the NsPack binary executable in section 6. The Entrypoint of NsPack generally makes use of a JMP instruction followed by a PUSHF and PUSHA command.

The detection and analysis of many common packers remains a mystic art to many people. With more than 80% of malware using some type of packer, this is something that needs to change. As was noted at the start of this paper, the intensification of cybercrime will only end in the development of greater volumes of malware. As these products are commercialized, the authors are likely to escalate their endeavors (Debrosse, 2009) leading to malicious software that is more difficult to detect and stop. Packers, allow the cybercriminal to simply increase the costs of detecting their products. This results in greater expected returns through a more successful campaign. Consequently, it is imperative that information security professional understand PE Packers whether they work in the AV industry or for general commercial ventures. In the former occasion, an understanding of the packer problem is only likely to become more important. In the later, knowledge of packers can only aid the security professional to gain a comprehension of the predicament to its true degree.

For the preponderance of security professionals, an analysis of malware (and hence packers) will be for the most part critical only when an incident has occurred. Knowledge as to the processes that an attacker has used to obscure their software can be the key in any successful incident handling exercise involving malware. As the majority of security incidents are coming to be based on some form of malware, a good understanding of how packers work is becoming more and more crucial. As NsPack remains one of the most common PE Packers with high rates of reported use and discovery (NsPack is in the top 10 list for PE Packers used on malware samples stored in the AML database) and with the relatively low accuracy rates for detection, it is important that security professionals gain a more comprehensive understanding of this and related packers. For this reason, this paper was written as a broad analysis of NsPack that will help both the novice and experienced anti-malware professional.

9. References

Arbor Networks. (2007) Arbor malware library (AML). http://www.arbornetworks.com.

- Chess, D., & White, S. (2000) "*An undetectable computer virus*". In Virus Bulletin Conference, September 2000.
- Cohen. F (1987) "*Computer viruses: Theory and experiments*". In 7th DOD/NBS Computers and Security Conference, volume 6, pages 22--35, September 1987.
- Debrosse, Jeff (2009)"Navigating the New Cybercrime Threatscape" TechNewsWorld (http://www.ecommercetimes.com/story/68067.html), Viewed, 07 Jan 2010.

Doyle, John F. (2009) "Fall 2009 – Computer Structures C335 Syllabus, ASM" http://homepages.ius.edu/jfdoyle/C335/syllabusc335fall2009.htm

Guo, Fanglu, Ferrie, Peter & Chiueh, Tzi-cker (2008) "A Study of the Packer Problem and Its Solutions", Symantec Research Laboratories, Lecture Notes in Computer Science

- Ferrie, Peter. (2006) "Attacks on Virtual Machines". In Proceedings of AVAR Conference.
- Iczelion's Win32 Assembly Homepage, http://win32assembly.online.fr/tutorials.html, Last viewed 20 May 2010.
- Jeong, K., & Lee, H. (2008) "*Code graph for malware detection*". In Information Networking. ICOIN. International Conference on, Jan 2008.
- Kath, Randy (1997) "*The Portable Executable File Format from Top to Bottom*" Microsoft Developer Network Technology Group. (Available from:

http://www.pelib.com/resources/kath.txt)

- Lyda, R., Hamrock, J. (2007) "Using entropy analysis to find encrypted and packed malware", IEEE Security and Privacy (S&P), 5(2), pp. 40-45.
- Masta "masta_'s Tutorial on Win95 ASM Coding Part 0" http://win32assembly.online.fr/w32_00.txt, Last viewed 11 Apr 2010.
- North Star Software. (2008) NsPack. http://www.nsdsn.com/eng/index.htm.
- Oberheide, J., Bailey, M., & Jahanian F. (2009) "PolyPack: An Automated Online Packing Service for Optimal Antivirus Evasion", 3rd USENIX Workshop on Offensive Technologies (WOOT '09)
- PEiD, available at http://www.peid.info/.
- Protection ID the ultimate Protection Scanner, available at

http://pid.gamecopyworld.com/.

- Perdisci, R., Lanzi, A., & Lee, W. (2008) "Classification of Packed Executables for Accurate Computer Virus Detection", Pattern Recognition Letters, pp. 1941-1946, 29(14).
- Pietrek, Matt (1998) "Windows System Programming Secrets", John Wiley & Sons Inc, USA
- Pietrek, Matt (1994) "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format" Microsoft Developer Network Technology Group. (Available from: http://msdn.microsoft.com/en-us/library/ms809762.aspx)
- Shafiq, M. Zubair, Tabish, S. Momina., Farooq, Muddassar (2009) "PE-Probe: Leveraging Packer Detection and Structural Information to Detect Malicious Portable Executables", Next Generation Intelligent Networks Research Center (nexGIN RC) National University of Computer & Emerging Sciences (NUCES-FAST) Islamabad, 44000, Pakistan.

Stewart, Joe. (2006) "OllyBonE v0.1, Break-on-Execute for OllyDbg".Taha, G., (2008) "Counterattacking the packers", McAfee Avert Labs, Aylesbury, UK.

10. Appendix 1 - NsPack Unpacking / De-Compression

The following is the main() unpacking routine from an NsPack 3.7 compressed executable. This is always the 'nsp1' (or the equivalently renamed segment).

.nsp1:4AD5BFFE .nsp1:4AD5BFFE .nsp1:4AD5BFFE .nsp1:4AD5BFFE public start .nspl:4AD5BFFE start proc near .nsp1:4AD5BFFE .nsp1:4AD5BFFE ; FUNCTION CHUNK AT .nsp1:4AD5C3BE SIZE 00000009 BYTES .nsp1:4AD5BFFE .nsp1:4AD5BFFE pushf .nsp1:4AD5BFFF pusha .nsp1:4AD5C000 call \$+5 .nsp1:4AD5C005 pop ebp .nsp1:4AD5C006 sub ebp, 7 lea cmp jz mov .nsp1:4AD5C009 ecx, [ebp-1A2h] .nsp1:4AD5C00F byte ptr [ecx], 1 .nsp1:4AD5C012 loc 4AD5C25A byte ptr [ecx], 1 .nsp1:4AD5C018 .nsp1:4AD5C01B mov eax, ebp .nsp1:4AD5C01D sub eax, [ebp-20Eh] mov .nsp1:4AD5C023 [ebp-20Eh], eax .nsp1:4AD5C029 [ebp-1DEh], eax add .nsp1:4AD5C02F lea add esi, [ebp-19Ah] .nsp1:4AD5C035 add [esi], eax .nsp1:4AD5C037 push ebp .nsp1:4AD5C038 push esi .nsp1:4AD5C039 push 40h .nsp1:4AD5C03B 1000h push .nsp1:4AD5C040 push 1000h .nsp1:4AD5C045 push 0 .nsp1:4AD5C047 call dword ptr [ebp-166h] tes. jz mov call pop .nsp1:4AD5C04D eax, eax .nsp1:4AD5C04F loc_4AD5C3BE .nsp1:4AD5C055 [ebp-1E6h], eax .nsp1:4AD5C05B \$+5 .nsp1:4AD5C060 ebx ecx, 367h .nsp1:4AD5C061 mov .nsp1:4AD5C066 add ebx, ecx .nsp1:4AD5C068 push eax push .nsp1:4AD5C069 ebx .nsp1:4AD5C06A call sub 4AD5C31F .nsp1:4AD5C06F esi pop .nsp1:4AD5C070 ebp pop .nsp1:4AD5C071 mov mov add mov cmp jnz mov esi, [esi] .nsp1:4AD5C073 edi, ebp .nsp1:4AD5C075 edi, [ebp-21Eh] .nsp1:4AD5C07B ebx, edi .nsp1:4AD5C07D dword ptr [edi], 0 .nsp1:4AD5C080 short loc 4AD5C08C add edi, 4 .nsp1:4AD5C082 ecx, 0 .nsp1:4AD5C085 mov jmp short loc 4AD5C0A2 .nsp1:4AD5C08A ·----.nsp1:4AD5C08C ; -----.nsp1:4AD5C08C .nsp1:4AD5C08C loc_4AD5C08C: ; CODE XREF: start+82i ecx, 1 .nsp1:4AD5C08C mov .nsp1:4AD5C091 edi, [ebx] add add .nsp1:4AD5C093 ebx, 4 .nsp1:4AD5C096 ; CODE XREF: start+CFj dword ptr [ebx], 0 short } .nsp1:4AD5C096 loc 4AD5C096: .nsp1:4AD5C096 cmp .nsp1:4AD5C099 short loc_4AD5C0CF jz .nsp1:4AD5C09B add [ebx], edx .nsp1:4AD5C09D mov esi, [ebx] add edi, [ebx+4] .nsp1:4AD5C09F .nsp1:4AD5C0A2 .nsp1:4AD5C0A2 loc_4AD5C0A2: ; CODE XREF: start+8Cj push edi .nsp1:4AD5C0A2

.nsp1:4AD5C0A3 push ecx .nsp1:4AD5C0A4 ebx push push .nsp1:4AD5C0A5 dword ptr [ebp-162h] .nsp1:4AD5C0AB push dword ptr [ebp-166h] .nsp1:4AD5C0B1 edx, esi mov .nsp1:4AD5C0B3 mov ecx, edi .nsp1:4AD5C0B5 mov eax, [ebp-1E6h] .nsp1:4AD5C0BB add eax, 5AAh .nsp1:4AD5C0C0 call eax .nsp1:4AD5C0C2 pop ebx .nsp1:4AD5C0C3 pop ecx .nsp1:4AD5C0C4 pop edi .nsp1:4AD5C0C5 cmp ecx, 0 .nsp1:4AD5C0C8 jz short loc_4AD5C0CF add .nsp1:4AD5C0CA ebx, 8 .nsp1:4AD5C0CD jmp short loc 4AD5C096 .nsp1:4AD5C0CF ; -----.nsp1:4AD5C0CF .nsp1:4AD5C0CF loc_4AD5C0CF: ; CODE XREF: start+9Bj .nsp1:4AD5C0CF ; start+CAj 8000h .nsp1:4AD5C0CF push .nsp1:4AD5C0D4 push 0 .nsp1:4AD5C0D6 push dword ptr [ebp-1E6h] call lea .nsp1:4AD5C0DC dword ptr [ebp-162h] .nsp1:4AD5C0E2 esi, [ebp-1DEh] .nsp1:4AD5C0E8 mov ecx, [esi+8] lea .nsp1:4AD5C0EB edx, [esi+10h] .nsp1:4AD5C0EE mov esi, [esi] .nsp1:4AD5C0F0 mov edi, esi .nsp1:4AD5C0F2 ecx, 0 cmp .nsp1:4AD5C0F5 short loc_4AD5C136 jz .nsp1:4AD5C0F7 .nsp1:4AD5C0F7 loc 4AD5C0F7: ; CODE XREF: start+100j .nsp1:4AD5C0F7 ; start+10Ej .nsp1:4AD5C0F7 mov al, [edi] .nsp1:4AD5C0F9 inc edi .nsp1:4AD5C0FA sub al, OE8h .nsp1:4AD5C0FC .nsp1:4AD5C0FC loc_4AD5C0FC: ; CODE XREF: start+136j .nsp1:4AD5C0FC al, 1 cmp .nsp1:4AD5C0FE ja short loc_4AD5C0F7 .nsp1:4AD5C100 mov eax, [edi] .nsp1:4AD5C102 byte ptr [edx+1], 0 cmp .nsp1:4AD5C106 short loc_4AD5C11C jΖ .nsp1:4AD5C108 bl, [edx] mov .nsp1:4AD5C10A cmp [edi], bl .nsp1:4AD5C10C short loc_4AD5C0F7 jnz .nsp1:4AD5C10E mov bl, [edi+4] .nsp1:4AD5C111 shr ax, 8 .nsp1:4AD5C115 eax, 10h rol .nsp1:4AD5C118 xchq al, ah .nsp1:4AD5C11A short loc_4AD5C126 jmp .nsp1:4AD5C11C ; -----_____ .nsp1:4AD5C11C .nsp1:4AD5C11C loc 4AD5C11C: ; CODE XREF: start+108j bl, [edi+4] .nsp1:4AD5C11C mov xchg .nsp1:4AD5C11F al, ah .nsp1:4AD5C121 rol eax, 10h .nsp1:4AD5C124 al, ah xchg .nsp1:4AD5C126 .nsp1:4AD5C126 loc_4AD5C126: ; CODE XREF: start+11Cj eax, edi .nsp1:4AD5C126 sub .nsp1:4AD5C128 add eax, esi .nsp1:4AD5C12A mov [edi], eax edi, 5 .nsp1:4AD5C12C add .nsp1:4AD5C12F sub bl, OE8h .nsp1:4AD5C132 mov eax, ebx .nsp1:4AD5C134 loc 4AD5C0FC loop .nsp1:4AD5C136 .nsp1:4AD5C136 loc 4AD5C136: ; CODE XREF: start+F7j .nsp1:4AD5C136 call sub 4AD5C275 .nsp1:4AD5C13B ecx, [ebp-1CAh] lea .nsp1:4AD5C141 eax, [ecx+8] mov cmp eax, 0 .nsp1:4AD5C144 loc 4AD5C1CE .nsp1:4AD5C147 jz .nsp1:4AD5C14D mov esi, edx esi, [ecx+10h] .nsp1:4AD5C14F sub short loc_4AD5C1CE [ecx+10h], esi .nsp1:4AD5C152 jz .nsp1:4AD5C154 mov esi, [ebp-19Ah] .nsp1:4AD5C157 lea

.nsp1:4AD5C15D mov esi, [esi] mov esi, [esi] lea ebx, [esi-4] mov eax, [ecx] cmp eax, 1 jz short loc_4AD5C173 mov edi, edx add edi, [ecx+8] mov ecx, [ecx+10h] jmp short loc_4AD5C17B .nsp1:4AD5C15F .nsp1:4AD5C162 .nsp1:4AD5C164 .nsp1:4AD5C167 .nsp1:4AD5C169 .nsp1:4AD5C16B .nsp1:4AD5C16E .nsp1:4AD5C171 .nsp1:4AD5C173 ; ------.nsp1:4AD5C173 .nsp1:4AD5C173 loc 4AD5C173: ; CODE XREF: start+169j mov edi, esi
add edi, [ecx+8]
mov ecx, [ecx+10h] .nsp1:4AD5C173 .nsp1:4AD5C175 .nsp1:4AD5C178 .nsp1:4AD5C17B ; CODE XREF: start+173j .nsp1:4AD5C17B loc_4AD5C17B: .nsp1:4AD5C17B ; start+18Ej .nsp1:4AD5C17B xor eax, eax mov al, [edi] .nsp1:4AD5C17D al, [edi] inc .nsp1:4AD5C17F edi or jz .nsp1:4AD5C180 eax, eax jz short loc_4AD5C1A4 cmp al, 0EFh ja short loc_4AD5C18E .nsp1:4AD5C182 .nsp1:4AD5C184 .nsp1:4AD5C186 .nsp1:4AD5C188 .nsp1:4AD5C188 loc_4AD5C188: ; CODE XREF: start+19Dj .nsp1:4AD5C188 ; start+1A4j .nsp1:4AD5C188 add ebx, eax .nsp1:4AD5C18A add [ebx], ecx short loc_4AD5C17B .nsp1:4AD5C18C jmp _____ .nsp1:4AD5C18E ; -----.nsp1:4AD5C18E .nsp1:4AD5C18E loc_4AD5C18E: ; CODE XREF: start+188j c_4AD5C18E: ; C and al, OFh shl eax, 10h mov ax, [edi] add edi, 2 or eax, eax jnz short loc_4AD5C188 mov eax, [edi] add edi, 4 jmp short loc_4AD5C188 .nsp1:4AD5C18E .nsp1:4AD5C190 .nsp1:4AD5C193 .nsp1:4AD5C196 .nsp1:4AD5C199 .nsp1:4AD5C19B .nsp1:4AD5C19D .nsp1:4AD5C19F .nsp1:4AD5C1A2 .nsp1:4AD5C1A4 ; -----------. .nsp1:4AD5C1A4 .nsp1:4AD5C1A4 loc_4AD5C1A4: ; CODE XREF: start+184j .nsp1:4AD5C1A4 xor ebx, ebx xchg .nsp1:4AD5C1A6 edi, esi .nsp1:4AD5C1A8 mov eax, [esi] .nsp1:4AD5C1AA cmp eax, 0 jz short loc_4AD5C1CE eax, 0 .nsp1:4AD5C1AD .nsp1:4AD5C1AF .nsp1:4AD5C1AF loc_4AD5C1AF: ; CODE XREF: start+1BCj .nsp1:4AD5C1AF lodsd or eax, eax jz short loc_4AD5C1BC add ebx, eax add [edi+ebx], cx jmp short loc_4AD5C1AF .nsp1:4AD5C1B0 .nsp1:4AD5C1B2 .nsp1:4AD5C1B4 .nsp1:4AD5C1B6 .nsp1:4AD5C1BA -----.nsp1:4AD5C1BC ; -----_____ .nsp1:4AD5C1BC _ _ _ ; CODE XREF: start+1B4j xor ebx, ebx shr ecx, 10h .nsp1:4AD5C1BC loc 4AD5C1BC: .nsp1:4AD5C1BC .nsp1:4AD5C1BE .nsp1:4AD5C1C1 ; or eax, eax jz short loc_4AD5C1CE add ebx, eax add for the state .nsp1:4AD5C1C1 loc 4AD5C1C1: ; CODE XREF: start+1CEj lodsd .nsp1:4AD5C1C1 .nsp1:4AD5C1C2 .nsp1:4AD5C1C4 add ebx, eax add [edi+ebx], cx jmp short loc_4AD5C1C1 .nsp1:4AD5C1C6 .nsp1:4AD5C1C8 .nsp1:4AD5C1CC .nsp1:4AD5C1CE ; -----_____ .nsp1:4AD5C1CE .nsp1:4AD5C1CE loc 4AD5C1CE: ; CODE XREF: start+149j ; start+154jnsp1:4AD5C1CE lea esi, [ebp-20Eh] .nsp1:4AD5C1CE mov edx, [esi] lea esi, [ebp-1B2h] mov al, [esi] .nsp1:4AD5C1D4 .nsp1:4AD5C1D6 .nsp1:4AD5C1DC

GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer. 85

.nsp1:4AD5C1DE	cmp	al, 1
.nsp1:4AD5C1E0	jnz	short loc 4AD5C221
-	-	
.nsp1:4AD5C1E2	add	edx, [esi+4]
.nsp1:4AD5C1E5	push	esi
	-	
.nsp1:4AD5C1E6	push	edx
.nsp1:4AD5C1E7	push	esi
.nsp1:4AD5C1E8	push	4
	*	
.nsp1:4AD5C1EA	push	100h
.nsp1:4AD5C1EF	push	edx
.nsp1:4AD5C1F0	call	dword ptr [ebp-16Ah]
-		
.nsp1:4AD5C1F6	pop	edi
.nsp1:4AD5C1F7	pop	esi
.nsp1:4AD5C1F8	cmp	eax, 1
-	-	
.nsp1:4AD5C1FB	jnz	loc_4AD5C3BE
.nsp1:4AD5C201	add	esi, 8
.nsp1:4AD5C204	mov	ecx, 8
-		
.nsp1:4AD5C209	rep mov	sb
.nsp1:4AD5C20B	sub	esi, OCh
.nsp1:4AD5C20E	sub	edi, 8
-		
.nsp1:4AD5C211	push	esi
.nsp1:4AD5C212	push	dword ptr [esi-4]
.nsp1:4AD5C215	push	100h
-	-	
.nsp1:4AD5C21A	push	edi
.nsp1:4AD5C21B	call	dword ptr [ebp-16Ah]
.nsp1:4AD5C221		
.nsp1:4AD5C221 loc_4AD5C221:		; CODE XREF: start+1E2j
.nsp1:4AD5C221	push	ebp
	-	
.nsp1:4AD5C222	pop	ebx
.nsp1:4AD5C223	sub	ebx, 15h
.nsp1:4AD5C229	xor	ecx, ecx
-		
.nsp1:4AD5C22B	mov	cl, [ebx]
.nsp1:4AD5C22D	cmp	cl, 0
.nsp1:4AD5C230	jz	short loc 4AD5C25A
-	-	—
.nsp1:4AD5C232	inc	ebx
.nsp1:4AD5C233	lea	esi, [ebp-20Eh]
.nsp1:4AD5C239	mov	edx, [esi]
.nsp1:4AD5C23B		
.nsp1:4AD5C23B nsp1:4AD5C23B loc 4AD5C23B		: CODE XREF: start+25Ai
.nsp1:4AD5C23B loc_4AD5C23B:		; CODE XREF: start+25Aj
	push	; CODE XREF: start+25Aj esi
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B	-	-
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C	push	esi ecx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23D	push push	esi ecx ebx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C	push	esi ecx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E	push push push	esi ecx ebx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F	push push push push	esi ecx ebx edx esi
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C23F	push push push push push	esi ecx ebx edx esi dword ptr [ebx]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F	push push push push	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242	push push push push push push	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245	push push push push push push mov	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248	push push push push push mov add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245	push push push push push push mov	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248	push push push push push mov add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C24A .nsp1:4AD5C24B	push push push push push mov add push call	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C24A .nsp1:4AD5C24B .nsp1:4AD5C24B .nsp1:4AD5C24B	push push push push mov add push call pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23D .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C24A .nsp1:4AD5C24B	push push push push push mov add push call	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah]
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C24B .nsp1:4AD5C251 .nsp1:4AD5C251	push push push push push mov add push call pop pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C252 .nsp1:4AD5C253	push push push push push mov add push call pop pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254	push push push push push mov add push call pop pop pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C252 .nsp1:4AD5C253	push push push push push mov add push call pop pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254	push push push push push mov add push call pop pop pop pop add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi esi ebx, 0Ch
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C255 .nsp1:4AD5C258	push push push push push mov add push call pop pop pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C243 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C258	push push push push push mov add push call pop pop pop pop add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C255 .nsp1:4AD5C258	push push push push push mov add push call pop pop pop pop add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi esi ebx, 0Ch
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C255 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop pop add	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C24E .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A loc_4AD5C25A: .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop add loop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C255 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop pop add loop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C24E .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A loc_4AD5C25A: .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop add loop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C255 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop pop add loop	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C243 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C252 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop add loop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C252 .nsp1:4AD5C253 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop add loop mov cmp jz popa	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C243 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C252 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A	push push push push push mov add push call pop pop pop add loop	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C25A	push push push push push mov add push call pop pop add loop mov cmp jz popa pop	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C265 .nsp1:4AD5C264	push push push push push mov add push call pop pop pop pop add loop mov cmp jz popf mov	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C24S .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C252 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C264 .nsp1:4AD5C266 .nsp1:4AD5C266	push push push push push posh call pop pop add loop jz popa popf mov retn	esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C265 .nsp1:4AD5C264	push push push push push posh call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C24C .nsp1:4AD5C242 .nsp1:4AD5C243 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C252 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C26A .nsp1:4AD5C264 .nsp1:4AD5C266 .nsp1:4AD5C26B .nsp1:4AD5C26B	push push push push push posh call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 0Ch</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C264 .nsp1:4AD5C265 .nsp1:4AD5C266 .nsp1:4AD5C268 .nsp1:4AD5C268 .nsp1:4AD5C268 .nsp1:4AD5C268 .nsp1:4AD5C268 .nsp1:4AD5C268	push push push push push posh call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 OCh</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C256 .nsp1:4AD5C266	push push push push push posh call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 0Ch</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23B .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C254 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C256 .nsp1:4AD5C266	push push push push push mov add push call pop pop add loop mov cmp jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 OCh</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C26A .nsp1:4AD5C264 .nsp1:4AD5C266	push push push push push mov add push call pop pop pop add loop jz popa popf mov cmp jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 OCh</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C24S .nsp1:4AD5C24C .nsp1:4AD5C24S .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C252 .nsp1:4AD5C252 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C258 .nsp1:4AD5C258 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C262 .nsp1:4AD5C262 .nsp1:4AD5C266 .nsp1:4AD5C26E .nsp1:4AD5C26E loc_4AD5C26E: .nsp1:4AD5C26E	push push push push push mov add push call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 short loc_4AD5C26E eax, 1 OCh ; CODE XREF: start+264j</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C245 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C26A .nsp1:4AD5C264 .nsp1:4AD5C266	push push push push push mov add push call pop pop pop add loop jz popa popf mov cmp jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, OCh loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 eax, 0 short loc_4AD5C26E eax, 1 OCh</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C262 .nsp1:4AD5C262 .nsp1:4AD5C26E	push push push push push mov add push call pop pop add loop mov cmp jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 short loc_4AD5C26E eax, 1 OCh ; CODE XREF: start+264j</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C255 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C262 .nsp1:4AD5C262 .nsp1:4AD5C26E ; .nsp1:4AD5C26E loc_4AD5C26E: .nsp1:4AD5C26F .nsp1:4AD5C26F .nsp1:4AD5C270 .nsp1:4AD5C270 start	push push push push push mov add push call pop pop add loop jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 short loc_4AD5C26E eax, 1 OCh ; CODE XREF: start+264j</pre>
.nsp1:4AD5C23B loc_4AD5C23B: .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23C .nsp1:4AD5C23E .nsp1:4AD5C23F .nsp1:4AD5C240 .nsp1:4AD5C242 .nsp1:4AD5C242 .nsp1:4AD5C248 .nsp1:4AD5C248 .nsp1:4AD5C251 .nsp1:4AD5C251 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C253 .nsp1:4AD5C254 .nsp1:4AD5C254 .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C25A .nsp1:4AD5C262 .nsp1:4AD5C262 .nsp1:4AD5C26E	push push push push push mov add push call pop pop add loop mov cmp jz popa popf mov retn	<pre>esi ecx ebx edx esi dword ptr [ebx] dword ptr [ebx+4] eax, [ebx+8] eax, edx eax dword ptr [ebp-16Ah] edx ebx ecx esi ebx, 0Ch loc_4AD5C23B ; CODE XREF: start+14j ; start+232j eax, 0 short loc_4AD5C26E eax, 1 OCh ; CODE XREF: start+264j</pre>

10.1. Reversing Assembly to produce C/C++ Code

In this section we cover the reversing process used to take assembled code and create a C/C++ representation of the code. This is used for automated unpackers that can be added into other programs as a routine. There are a number of problems that are associated with reverse engineering code. Some of these are noted below. Assembly code loses many of the richness of source code when it is compiled. It is generally expected that comments are lost and would not be recoverable in executable code, however, classes, macros, templates and include files are also lost during compilation. This does not mean that we cannot recover many useful aspects of the code. Executable code (especially when disassembled) retains:

- Dynamic Links
- Functional calls
- ☑ Local Variables (although the richness of naming will be lost)
- Parameters
- Switch statements

Variables do not hold data; they are a pointer to the location where the data has been stored by the system.

10.1.1. Execution Control

The Instruction Pointer (IP) always points to the next instruction when executing. This is the point where the next or subsequent *fetch* is to occur. Altering the IP allows different sections of the algorithm to be executed in place of the following sequential instruction.

The Intel processor supports the following three execution control methods:

- Sequential
- **W** Unconditional branching
- Conditional branching.

Sequential branching is the normal execution process. As the name suggests, with sequential branching, one instruction follows after another in order. This is a standard *fetch/execute* cycle with the IP¹² incrementing in numerical order.

An Unconditional branch involves the non-sequential execution or redirection of instructions. An unconditional branch jumps to another address and *unconditionally* executes the instructions at that point.

A Conditional branch is also executed non-sequentially. The difference to an unconditional branch is that the branch to the instruction to be executed occurs *conditionally*. This means that where a logical condition returns a true result, the instruction at the specified address is executed. In the event that a false result is returned, the instruction is executed sequentially.

10.1.2. Decompiling with HexRays

HexRays has a semi-automated decompiler. We will use this to examine the code and as an aid to reversing the assembled code into C++.

```
62i 1:0040D412 ; ======== S U B R O U T I N E
 _____
 _62i_1:0040D412
  62i 1:0040D412
; CODE XREF: start:loc 40D2D3p
                                           or esi, esi
jz loc_40D4B7
mov edx, [ebp-18Bh]
add esi, edx
_62i_1:0040D420

      _62i_1:0040D420

      _62i_1:0040D428
      ; COL

      _62i_1:0040D428
      cmp
      dword ptr [esi], 0

      _62i_1:0040D42B
      jnz
      short loc_40D43B

      _62i_1:0040D42D
      cmp
      dword ptr [esi+4], 0

      _62i_1:0040D431
      jnz
      short loc_40D43B

      _62i_1:0040D433
      cmp
      dword ptr [esi+8], 0

      _62i_1:0040D437
      jnz
      short loc_40D43B

      _62i_1:0040D439
      jmp
      short loc_40D43B

  62i 1:0040D426
                                                                                   ; CODE XREF: sub 40D412+61j
                 _____
_62i_1:0040D43B
_62i_1:0040D43B loc 40D43B:
                                                                                  ; CODE XREF: sub_40D412+19j
                                                                                  ; sub_40D412+1Fj ...
  62i 1:0040D43B
 62i 1:0040D43B
                                            mov ebx, [esi+8]
 _62i_1:0040D43E
                                                         ebx, edx
                                              add
_62i_1:0040D440
_62i_1:0040D441
                                             push
                                                         ebx
                                               push
                                                            edx
  62i 1:0040D442
                                               push
                                                            esi
                                              lea
 62i 1:0040D443
                                                           edi, [ebp-9Fh]
__62i_1:0040D449
__62i_1:0040D44C
                                                         edi, [esi+4]
esi, OCh
                                               add
                                               add
```

¹² Instruction pointer

_62i_1:0040D44F push edi call 62i 1:0040D450 dword ptr [ebp-0FFh] all aword ptr [ebp-0] pop edi pop edx pop ebx cmp eax, 0 jz short loc_40D4B7 mov [ebp-183h], eax add edi, [esi] add esi, 4 _62i_1:0040D456 _62i_1:0040D457 _62i_1:0040D458 _62i_1:0040D459 62i 1:0040D45C _62i_1:0040D45E _62i_1:0040D464 62i_1:0040D466 _62i_1:0040D469 ; CODE XREF: sub 40D412+A1j _62i_1:0040D475 ; _____ ______ _62i_1:0040D475 _62i_1:0040D475 loc_40D475: _62i_1:0040D475 mov eax, edi _add edi, ecx _edx ; CODE XREF: sub_40D412+5Ej add edi, ecx push edx push ebx push eax cmp byte ptr [eax], OFFh jnz short loc_40D489 inc eax mov eax, [eax] and eax, 7FFFFFFh _62i_1:0040D479 _62i_1:0040D47A 62i_1:0040D47B _62i_1:0040D47C _62i_1:0040D47F __62i_1:0040D481 __62i_1:0040D482

 -62:
 -62:
 1:0040D482

 -62:
 1:0040D489
 anu

 -62:
 1:0040D489
 ic

 -62:
 1:0040D489
 mov
 cl, [edi]

 -62:
 1:0040D489
 mov
 byte ptr [edi], 0

 -62:
 1:0040D48B
 mov
 byte ptr [edi], 0

 -62:
 1:0040D48E
 push
 eax

 -62:
 1:0040D48F
 push
 dword ptr [ebp-183h]

 -62:
 1:0040D490
 push
 dword ptr [ebp-0FBh]

 -62:
 1:0040D49C
 pop
 ecx

 -62:
 1:0040D49E
 pop
 edx

 -62:
 1:0040D49E
 pop
 edx

 -62:
 1:0040D49E
 pop
 edx

 -62:
 1:0040D4A3
 jz
 short loc_40D4B7

 -62:
 1:0040D4A3
 jz
 short loc_40D4B7

 -62:
 1:0040D4A3
 jz
 short loc_40D4B7

 -62:
 1:0040D4A5
 mov
 [edi], cl

 -62:
 1:0040D4A5
 mov
 [edi], cl

 -62:
 1:0040D4A5
 mov
 [esi-4]

 ; CODE XREF: sub 40D412+6Dj _____ _62i_1:0040D4B5 _621_1:0040D4B5 loc_40D4B5: ; CODE XREF: sub 40D412+27j 62i_1:0040D4B5 clc _____62i_1:0040D4B6 retn 62i 1:0040D4B7 ; _____ _62i_1:0040D4B7 _____i:0040D4B7 jmp loc_40D55B __62i_1:0040D4B7 sub_40D412 endp __62i_1:0040D4B7 __62i_1:0040D4B7 __62i_1:0040D4BC ; CODE XREF: sub 40D412+8j ; sub 40D412+4Aj ...

10.1.3. Decompiled code

The results of using HexRays to decompile the assembled code (10.1.2) are displayed below. As we stated in 10.1.1, a good deal of information is lost.

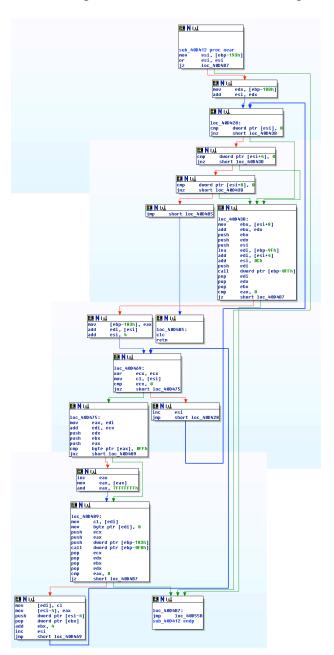
```
void __usercall sub_40D412(int a1<ebp>)
{
  int v1; // esi@1
 int v2; // edx@2
 int v3; // esi@2
int v4; // eax@7
 int v5; // ebx@7
 int v6; // esi@7
 int v7; // STOC 407
 int v8; // edi@8
 int v9; // esi@8
 int v10; // ecx@9
 int v11; // eax011
 int v12; // ST14 4011
 int v13; // eax013
 char v14; // ST08 1013
 int v15; // ST14 407
 int v16; // ST10 407
 int v17; // edi07
 char v18; // cl013
  v1 = * (DWORD *) (a1 - 403);
  if ( v1 )
   v2 = * (DWORD *) (a1 - 395);
   v3 = v2^{+} v1;
   while ( *( DWORD *)v3 || *( DWORD *)(v3 + 4) || *( DWORD *)(v3 + 8) )
   {
          v15 = v2 + *(DWORD *)(v3 + 8);
          v16 = v2;
          v7 = v3;
          v17 = *( DWORD *)(v3 + 4) + a1 - 159;
          v6 = v3 + 12;
                         _stdcall **)(int))(a1 - 255))(v17);
          v4 = (*(int (
          v2 = v16;
          v5 = v15;
          if ( !v4 )
           break;
          * ( DWORD *) (a1 - 387) = v4;
          v8 = *(_DWORD *)v6 + v7;
          v9 = v6 + 4;
          while (1)
          {
            v10 = *( BYTE *)v9;
            if ( !v10 )
                  break;
            v11 = v8;
            v8 += v10;
            v12 = v2;
            if ( *(\_BYTE \ *)v11 == -1 )
                  v11 = * ( DWORD *) (v11 + 1) & 0x7FFFFFF;
            v18 = *( BYTE *)v8;
            *( BYTE *)v8 = 0;
            v1\overline{4} = v18;
            v13 = (*(int ( stdcall **)( DWORD, int))(a1 - 251))(*( DWORD *)(a1 -
387), v11);
            v^2 = v^{12};
            if ( !v13 )
                  return;
            *( BYTE *)v8 = v14;
            *( DWORD *)(v9 - 4) = v13;
            *(_DWORD *)v5 = *(_DWORD *)(v9 - 4);
```

```
v5 += 4;
++v9;
}
v3 = v9 + 1;
}
```

To workout what we have lost and to recreate this information, we will use the graphing functions of IDA Pro. As is displayed in 10.1.4, a flow graph provides more information as to the calls and jumps used by a routine.

10.1.4. Function 1 – The first function

The first function is used by the decompression routine. It conducts a series of comparisons against the various registers (using compare functions) and processing the values using the stack where values in the registers are set to be updated.



1.1.1. Function Pseudo Code

The following is a low level representation of the MASM function which will be converted to a High level pseudo code and C++.

1.1.2. ASM

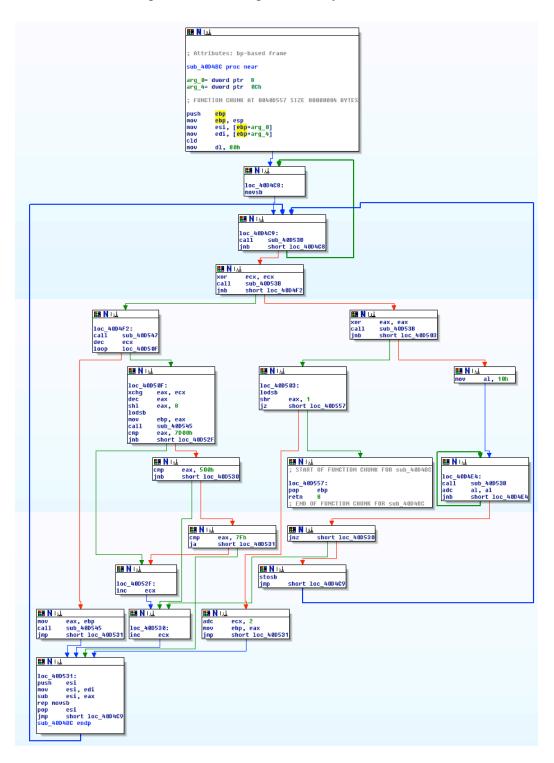
_62i_1:0040D412 ; ========	=== S U	BROUTINE
62i 1:0040D412		=
62i 1:0040D412		
62i 1:0040D412 sub 40D412	proc n	ear ; CODE XREF: start:loc 40D2D3p
62i 1:0040D412	mov	esi, [ebp-193h]
	or	esi, esi
	jz	loc 40D4B7
	mov	edx, [ebp-18Bh]
_62i_1:0040D426	add	esi, edx
_62i_1:0040D428		
_62i_1:0040D428 loc_40D428:		; CODE XREF: sub_40D412+61j
_62i_1:0040D428	cmp	dword ptr [esi], 0
_62i_1:0040D42B	jnz	short loc_40D43B
_62i_1:0040D42D	cmp	dword ptr [esi+4], 0
_62i_1:0040D431	jnz	short loc_40D43B
_62i_1:0040D433 62i 1:0040D437	cmp	dword ptr [esi+8], 0
62i 1:0040D439	jnz jmp	short loc_40D43B short loc 40D4B5
62i 1:0040D43B ;	Jmp	51101 0 100_100405
,		
62i 1:0040D43B		
_62i_1:0040D43B loc_40D43B:		; CODE XREF: sub_40D412+19j
_62i_1:0040D43B		; sub_40D412+1Fj
_62i_1:0040D43B	mov	ebx, [esi+8]
_62i_1:0040D43E	add	ebx, edx
_62i_1:0040D440	push	ebx
_62i_1:0040D441	push	edx
_62i_1:0040D442	push	esi adi (abu OFb)
_62i_1:0040D443 62i 1:0040D449	lea add	edi, [ebp-9Fh] edi, [esi+4]
621 1:0040D44C	add	esi, OCh
62i 1:0040D44F	push	edi
62i 1:0040D450	call	dword ptr [ebp-0FFh]
62i 1:0040D456	pop	edi
62i 1:0040D457	pop	edx
	pop	ebx
	cmp	eax, O
	jz	short loc 40D4B7
_62i_1:0040D45E	mov	[ebp-183h], eax
_62i_1:0040D464	add	edi, [esi]
_62i_1:0040D466	add	esi, 4
_62i_1:0040D469		
_62i_1:0040D469 loc_40D469:		; CODE XREF: sub_40D412+A1j
_62i_1:0040D469	xor	ecx, ecx
_62i_1:0040D46B	mov	cl, [esi]
_62i_1:0040D46D 62i 1:0040D470	cmp	ecx, 0 short loc 40D475
621_1:0040D470	jnz inc	short loc_40D475 esi
62i 1:0040D473	jmp	short loc 40D428
62i 1:0040D475 ;	Jmp	
_62i_1:0040D475		
_62i_1:0040D475 loc_40D475:		; CODE XREF: sub_40D412+5Ej
_62i_1:0040D475	mov	eax, edi
_62i_1:0040D477	add	edi, ecx
_62i_1:0040D479 62i 1:0040D47A	push push	edx ebx
621_1:0040D47B	push	eax
621 1:0040D47C	cmp	byte ptr [eax], OFFh
	- 1-	<u> </u>

GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer. 93

_62i_1:0040D47F _62i_1:0040D481 _62i_1:0040D482 _62i_1:0040D484 _62i_1:0040D489	jnz inc mov and	short loc_40D489 eax eax, [eax] eax, 7FFFFFFFh
 	mov push push call pop pop pop jz mov mov push pop add inc	; CODE XREF: sub_40D412+6Dj cl, [edi] byte ptr [edi], 0 ecx eax dword ptr [ebp-183h] dword ptr [ebp-0FBh] ecx edx edx edx eax, 0 short loc_40D4B7 [edi], cl [esi-4], eax dword ptr [esi-4] dword ptr [ebx] ebx, 4 esi short loc_40D469
_62i_1:0040D4B5 _62i_1:0040D4B5 loc_40D4B5: _62i_1:0040D4B5 _62i_1:0040D4B5 _62i_1:0040D4B6 _62i_1:0040D4B7 ;	clc retn	; CODE XREF: sub_40D412+27j
_62i_1:0040D4B7 _62i_1:0040D4B7 loc_40D4B7: _62i_1:0040D4B7 _62i_1:0040D4B7 _62i_1:0040D4B7 _62i_1:0040D4B7 sub_40D412 _62i_1:0040D4B7 _62i_1:0040D4BC	jmp endp	; CODE XREF: sub_40D412+8j ; sub_40D412+4Aj loc_40D55B

1.2. Function 2 – Memory Functions

The function processes and copies memory from across sections.



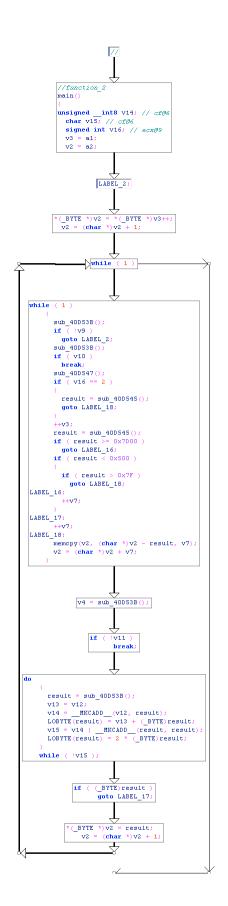
1.2.1. Function Pseudo Code

```
//function 2
unsigned int __cdecl sub_40D4BC(int a1, void *a2)
{
  void *v2; // edi@1
  int v3; // esi@1
 unsigned int v4; // eax@5
  int v5; // ecx@5
  unsigned int result; // eax06
  unsigned int v7; // ecx@6 \,
 unsigned __int8 v8; // cf@11
char v9; // cf@3
  char v10; // cf@4
  char v11; // cf@5
  char v12; // cf@6
  char v13; // tt@6
  unsigned __int8 v14; // cf@6
  char v15; // cf@6
 signed int v16; // ecx09
 v3 = a1;
  v2 = a2;
LABEL 2:
  *( BYTE *)v2 = *( BYTE *)v3++;
  v2 = (char *)v2 + 1;
  while (1)
  {
    while (1)
    {
      sub_40D53B();
      if ( !v9 )
       goto LABEL 2;
      sub 40D53B();
      if ( v10 )
        break;
      sub 40D547();
      if ( v16 == 2 )
      {
        result = sub 40D545();
        goto LABEL 18;
       }
      ++v3;
      result = sub 40D545();
      if ( result \geq 0 \times 7 D 0 0 )
        goto LABEL 16;
      if ( result < 0x500 )
       {
        if ( result > 0x7F )
           goto LABEL 18;
LABEL 16:
        ++v7;
       }
LABEL_17:
      ++v7;
LABEL_18:
     memcpy(v2, (char *)v2 - result, v7);
      v^2 = (char *)v^2 + v^7;
    }
    v4 = sub 40D53B();
    if ( !v11 )
      break;
    do
    {
      result = sub 40D53B();
      v13 = v12;
      v14 = __MKCADD__ (v12, result);
LOBYTE(result) = v13 + (_BYTE)result;
v15 = v14 | __MKCADD__ (result, result);
```

```
LOBYTE(result) = 2 * (_BYTE)result;
}
while ( !v15 );
if ( (_BYTE)result )
   goto LABEL_17;
*(_BYTE *)v2 = result;
v2 = (char *)v2 + 1;
}
LOBYTE(v4) = *(_BYTE *)v3++;
v8 = _MKCSHR__(v4, 1);
result = v4 >> 1;
if ( result )
{
   v7 = v8 + v5 + 2;
   goto LABEL_18;
}
return result;
```

}

GIAC GREM Gold: Packer Analysis Report – Debugging and unpacking the NsPack 3.4 and 3.7 packer. 97



1.2.2. ASM

62i 1:0040D4BC ; ======= S U B R O U T I N E - -_62i_1:0040D4BC _62i_1:0040D4BC ; Attributes: bp-based frame 62i 1:0040D4BC ______ 62i_1:0040D4BC sub_40D4BC proc near ________. ; CODE XREF: start+6Bp _62i_1:0040D4BC _021_1:0040D4BC _62i_1:0040D4BC arg_0 = dword ptr 8 _62i_1:0040D4BC arg_4 = dword ptr 0Ch 62i_1:0040D4BC _62i_1:0040D4BC _62i_1:0040D4BC ; FUNCTION CHUNK AT _62i_1:0040D557 SIZE 00000004 BYTES 62i 1:0040D4BC push ebp mov ebp, esp mov esi, [ebp+arg_0] mov edi, [ebp+arg_4] cld mov dl, 80h _62i_1:0040D4BC _62i_1:0040D4BD _62i_1:0040D4BF _62i_1:0040D4C2 _62i_1:0040D4C5 _62i_1:0040D4C6 _62i_1:0040D4C8 _62i_1:0040D4C8 loc_40D4C8: _62i_1:0040D4C8 mov sb ; CODE XREF: sub 40D4BC+12j _62i_1:0040D4C8 _62i_1:0040D4C9 _62i_1:0040D4C9 loc_40D4C9: ; CODE XREF: sub_40D4BC+34j c_40D4C9: call sub_40D53B jnb short loc_40D4C8 xor ecx, ecx ; Reset ECX = 0 call sub_40D53B jnb short loc_40D4F2 xor eax, eax ; Reset EAX = 0 call sub_40D53B jnb short loc_40D4F2 xor eax, aax ; Reset EAX = 0 call sub_40D53B jnb short loc_40D503 mov al, 10h _62i_1:0040D4C9 _62i_1:0040D4C9 62i_1:0040D4CE _62i_1:0040D4D0 _62i_1:0040D4D2 _62i_1:0040D4D7 _62i_1:0040D4D9 _62i_1:0040D4DB _62i_1:0040D4E0 62i 1:0040D4E2 62i 1:0040D4E4

 62i 1:0040D4E4
 ; CODE XREF: sub_40D4BC+2Fj

 62i 1:0040D4E4
 call sub_40D53B

 62i 1:0040D4E9
 adc al, al

 62i 1:0040D4EB
 jnb short loc_40D4E4

 62i 1:0040D4EB
 jnz short loc_40D4E4

 62i 1:0040D4EF
 stosb

 62i 1:0040D4EF
 stosb

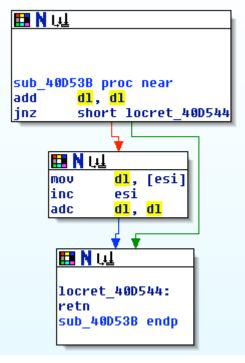
 62i 1:0040D4F0
 jmp short loc_40D4C9

 62i 1:0040D4F2 ; -----_62i_1:0040D4F2 _62i_1:0040D4F2 loc_40D4F2: ; _62i_1:0040D4F2 call sub_40D547 _62i_1:0040D4F7 dec ecx _62i_1:0040D4F8 loop loc_40D50F _62i_1:0040D4FA mov eax, ebp _62i_1:0040D4FC call sub_40D545 _62i_1:0040D501 jmp short loc_40D531 _62i_1:0040D4F2 ; CODE XREF: sub_40D4BC+1Bj 62i 1:0040D503 ; . ; CODE XREF: sub 40D4BC+24j _____ _62i_1:0040D50F ; CODE XREF: sub 40D4BC+3Cj _62i_1:0040D510

_62i_1:0040D514 _62i_1:0040D515 _62i_1:0040D517 _62i_1:0040D51C _62i_1:0040D521 _62i_1:0040D523 _62i_1:0040D528 _62i_1:0040D52A _62i_1:0040D52D _62i_1:0040D52F	cmp jnb cmp jnb cmp	ebp, eax sub_40D545 eax, 7D00h short loc_40D52F eax, 500h short loc_40D530 eax, 7Fh short loc_40D531	
_62i_1:0040D52F loc_40D52F: _62i_1:0040D52F _62i 1:0040D530	inc	; ecx	CODE XREF: sub_40D4BC+65j
_62i_1:0040D530 loc_40D530: _62i_1:0040D530			CODE XREF: sub_40D4BC+31j sub_40D4BC+6Cj
_62i_1:0040D530 62i 1:0040D531	inc	ecx	
_62i_1:0040D531 loc_40D531: 62i_1:0040D531			CODE XREF: sub_40D4BC+45j sub 40D4BC+51j
_62i_1:0040D538 _62i_1:0040D539	sub rep mov pop	esi, edi esi, eax sb	

10.2. Function 3 – Math

This function is called from within Function_5 (below) as well as from within Function_2 (above). ESI is the 32 bit Data Pointer for source of string operations.



ADC is a large number addition function – used here to double the DL register. The DL general purpose register is the 8 bit I/O Pointer value associated with the 32 bit EDX register.

1.2.3. Function Pseudo Code

```
DL = DL + DL
                               (Double DL or DL = 2x DL)
(Not Equal to 0, EFlag, ZF = 0)
                               (Double DL
      (DL <> 0)
ΙF
        (DL <> U)
[ESI] = [ESI] + DL
                               ([ESI] is the memory location pointed to by the value held
                                              In the ESI register)
               ESI = ESI + 1
                                      (or, DL = DL + DL)
               DL = 2x DL
       Return
 _int64 __usercall sub_40D53B<edx:eax>(char a1<dl>, int a2<esi>)
  unsigned __int8 v2; // cf@1
   int64 result; // qax@1
  char v4; // zf@1
        _MKCADD__(a1, a1);
 v2 =
 v_4^2 = \overline{2} * a1 = 0;
  BYTE4(result) = 2 * a1;
  if ( v4 )
   BYTE4(result) = 2 * (v2 + *(BYTE *)a2);
 return result;
}
```

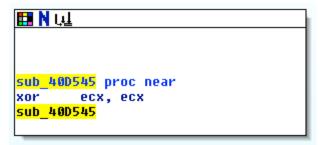
1.2.4. ASM

_62i_1:0040D53B ; ============== S U B R O U T I N E					
62i_1:0040D53B 62i_1:0040D53B		-			
_62i_1:0040D53B sub_40D53B sub 40D4BC:loc 40D4C9p	proc ne	ar	; CODE XREF:		
_62i_1:0040D53B			; sub_40D4BC+16p		
62i 1:0040D53B	add	dl, dl			
62i 1:0040D53D	jnz	short locret 401	0544		
62i 1:0040D53F	mov	dl, [esi]			
		;add to the ; the conte	e memory pointed to by ESI ents of dl		
62i 1:0040D541	inc	esi			
62i 1:0040D542	adc	dl, dl			
62i 1:0040D544					
62i 1:0040D544 locret 40D544:			; CODE XREF: sub 40D53B+2j		
62i 1:0040D544	retn				
_62i_1:0040D544 sub_40D53B _62i_1:0040D544	endp				

10.3. Function 4 – Clear ECX

The function below clears the ECX register. This sets the ECX register = 0x000.

ECX is a 32 bit register that is used as a counter for string and loop functions.



10.3.1. Function Pseudo Code

XOR ECX, ECX (Clear the ECX register)

10.3.2. ASM

_62i_1:0040D545 ; ======= S U B R O U T I N E _62i_1:0040D545 _62i_1:0040D545 _62i_1:0040D545 sub_40D545 proc near ; CODE XREF: sub_40D4BC+40p _62i_1:0040D545 xor ecx, ecx _62i_1:0040D545 sub_40D545 endp ; _62i_1:0040D545 sub_40D545 endp ;

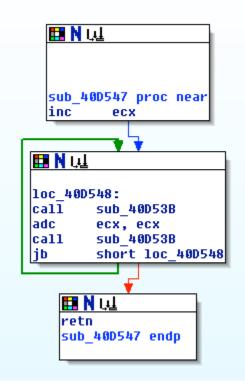
Craig S Wright

10.4. Function 5 – Add ECX register

The function below starts by incrementing ECX ()- this is adding 1 to ECX then looping while calling Function_3 and doubling the ECX register.

ECX is a 32 bit register that is used as a counter for string and loop functions.

ADC is a large number addition function – used here to double the register.



10.4.1. Function Pseudo Code

```
ECX = ECX + 1
      Do While (ECX < Function_3(Returned))
                                                     (Carry Flag; EFLAGS,
CF = 1)
             Run Function_3
ECX = ECX x2
                                          (ADC is a doubling function for large
numbers)
             Run Function 3
      Return
int __cdecl sub_40D547()
{
 int result; // eax@1
char v1; // cf@1
 do
  {
   sub 40D53B();
   result = sub_40D53B();
  }
```

while (v1);
return result;
}

1.2.5. ASM

10.5. Comparison of Main() Functions between NsPack 3.4& 3.7

Version 3.7

Version 3.4

.nsp1:4AD5BFFE	_62i_1:0040D19B
.nsp1:4AD5BFFE ; ====== S U B R	62i_1:0040D19B ; ===== S U B R
O U T I N E ===	O U T I N E ===
.nsp1:4AD5BFFE	_62i_1:0040D19B
.nsp1:4AD5BFFE	_62i_1:0040D19B
.nsp1:4AD5BFFE public start	_62i_1:0040D19B public start
.nsp1:4AD5BFFE start proc near	_62i_1:0040D19B start proc near
.nsp1:4AD5BFFE	_62i_1:0040D19B
.nsp1:4AD5BFFE ; FUNCTION CHUNK AT .nsp1:4AD5C3BE SIZE 00000009 BYTES	_62i_1:0040D19B ; FUNCTION CHUNK AT 62i 1:0040D55B SIZE 00000009 BYTES
1	
.nsp1:4AD5BFFE	_62i_1:0040D19B
.nsp1:4AD5BFFE pushf	_62i_1:0040D19B
.nsp1:4AD5BFFF pusha	_62i_1:0040D19C
.nsp1:4AD5C000 call \$+5	_62i_1:0040D19D call \$+5
.nsp1:4AD5C005 pop ebp	_62i_1:0040D1A2 pop ebp
.nsp1:4AD5C006 sub ebp, 7	_62i_1:0040D1A3 sub ebp, 7
.nsp1:4AD5C009 lea ecx, [ebp-1A2h]	_62i_1:0040D1A6 lea eax, [ebp-11Fh]
.nsp1:4AD5C00F cmp byte ptr [ecx], 1	_62i_1:0040D1AC cmp byte ptr [eax], 1
.nsp1:4AD5C012 jz loc_4AD5C25A	_62i_1:0040D1AF jz loc_40D3F7
.nsp1:4AD5C018 mov byte ptr [ecx], 1	_62i_1:0040D1B5 mov byte ptr [eax], 1
.nsp1:4AD5C01B mov eax, ebp	_62i_1:0040D1B8 mov edx, ebp
.nsp1:4AD5C01D sub eax, [ebp-20Eh]	_62i_1:0040D1BA sub edx, [ebp-18Bh]
.nsp1:4AD5C023 mov [ebp-20Eh], eax	_62i_1:0040D1C0 mov [ebp-18Bh], edx
.nsp1:4AD5C029 add [ebp-1DEh], eax	_62i_1:0040D1C6 add [ebp-15Bh], edx
.nsp1:4AD5C02F lea esi, [ebp-19Ah]	_62i_1:0040D1CC lea esi, [ebp-117h]
.nsp1:4AD5C035 add [esi], eax	_62i_1:0040D1D2 add [esi], edx

.nsp1:4AD5C037 push ebp	
.nsp1:4AD5C038 push esi	_62i_1:0040D1D4
.nsp1:4AD5C039 push 40h	_62i_1:0040D1D5
.nsp1:4AD5C03B push 1000h	_62i_1:0040D1D7
.nsp1:4AD5C040 push 1000h	_62i_1:0040D1DC
.nsp1:4AD5C045 push 0	_62i_1:0040D1E1
.nsp1:4AD5C047 call dword ptr [ebp-166h]	_62i_1:0040D1E3 call dword ptr [ebp-0F3h]
.nsp1:4AD5C04D test eax, eax	_62i_1:0040D1E9 test eax, eax
.nsp1:4AD5C04F jz loc_4AD5C3BE	_62i_1:0040D1EB jz loc_40D55B
.nsp1:4AD5C055 mov [ebp-1E6h], eax	_62i_1:0040D1F1 mov [ebp-163h], eax
.nsp1:4AD5C05B call \$+5	_62i_1:0040D1F7 call \$+5
.nsp1:4AD5C060 pop ebx	_62i_1:0040D1FC pop ebx
.nsp1:4AD5C061 mov ecx, 367h	_62i_1:0040D1FD mov ecx, 368h
.nsp1:4AD5C066 add ebx, ecx	_62i_1:0040D202 add ebx, ecx
.nsp1:4AD5C068 push eax	_62i_1:0040D204
.nsp1:4AD5C069 push ebx	_62i_1:0040D205
.nsp1:4AD5C06A call sub_4AD5C31F	_62i_1:0040D206 call sub_40D4BC
.nsp1:4AD5C06F pop esi	
.nsp1:4AD5C070 pop ebp	_62i_1:0040D20B
.nsp1:4AD5C071 mov esi, [esi]	_62i_1:0040D20C mov esi, [esi]
.nsp1:4AD5C073 mov edi, ebp	_62i_1:0040D20E mov edi, ebp
.nsp1:4AD5C075 add edi, [ebp-21Eh]	_62i_1:0040D210 add edi, [ebp-19Bh]
.nsp1:4AD5C07B mov ebx, edi	_62i_1:0040D216 mov ebx, edi
.nsp1:4AD5C07D cmp dword ptr [edi], 0	_62i_1:0040D218 cmp dword ptr [edi], 0
.nsp1:4AD5C080 jnz short loc_4AD5C08C	_62i_1:0040D21B jnz short loc_40D227
.nsp1:4AD5C082 add edi, 4	_62i_1:0040D21D add edi, 4
.nsp1:4AD5C085 mov ecx, 0	_62i_1:0040D220 mov ecx, 0
.nsp1:4AD5C08A jmp short loc_4AD5C0A2	_62i_1:0040D225 jmp short loc_40D23D
.nsp1:4AD5C08C ;	_62i_1:0040D227 ;
.nsp1:4AD5C08C	_62i_1:0040D227

.nsp1:4AD5C08C loc_4AD5C08C: ; CODE XREF: start+82j	_62i_1:0040D227 loc_40D227: ; CODE XREF: start+80j
.nsp1:4AD5C08C mov ecx, 1	_62i_1:0040D227 mov ecx, 1
.nsp1:4AD5C091 add edi, [ebx]	_62i_1:0040D22C add edi, [ebx]
.nsp1:4AD5C093 add ebx, 4	_62i_1:0040D22E add ebx, 4
.nsp1:4AD5C096	_62i_1:0040D231
.nsp1:4AD5C096 loc_4AD5C096: ; CODE XREF: start+CFj	_62i_1:0040D231 loc_40D231: ; CODE XREF: start+CFj
.nsp1:4AD5C096 cmp dword ptr [ebx], 0	_62i_1:0040D231 cmp dword ptr [ebx], 0
.nsp1:4AD5C099 jz short loc_4AD5C0CF	_62i_1:0040D234 jz short loc_40D26C
.nsp1:4AD5C09B add [ebx], edx	_62i_1:0040D236 add [ebx], edx
.nsp1:4AD5C09D mov esi, [ebx]	_62i_1:0040D238 mov esi, [ebx]
.nsp1:4AD5C09F add edi, [ebx+4]	_62i_1:0040D23A add edi, [ebx+4]
.nsp1:4AD5C0A2	_62i_1:0040D23D
.nsp1:4AD5C0A2 loc_4AD5C0A2: ; CODE XREF: start+8Cj	_62i_1:0040D23D loc_40D23D: ; CODE XREF: start+8Aj
.nsp1:4AD5C0A2 push edi	_62i_1:0040D23D
.nsp1:4AD5C0A3 push ecx	_62i_1:0040D23E
	_62i_1:0040D23F push edx
.nsp1:4AD5C0A4 push ebx	_62i_1:0040D240
.nsp1:4AD5C0A5 push dword ptr [ebp-162h]	_62i_1:0040D241 push dword ptr [ebp-0EFh]
.nsp1:4AD5C0AB push dword ptr [ebp-166h]	_62i_1:0040D247 push dword ptr [ebp-0F3h]
.nsp1:4AD5C0B1 mov edx, esi	_62i_1:0040D24D mov edx, esi
.nsp1:4AD5C0B3 mov ecx, edi	_62i_1:0040D24F mov ecx, edi
.nsp1:4AD5C0B5 mov eax, [ebp-1E6h]	_62i_1:0040D251 mov eax, [ebp-163h]
.nsp1:4AD5C0BB add eax, 5AAh	_62i_1:0040D257 add eax, 5AAh
.nsp1:4AD5C0C0 call eax	_62i_1:0040D25C call eax
.nsp1:4AD5C0C2 pop ebx	_62i_1:0040D25E
	_62i_1:0040D25F
.nsp1:4AD5C0C3 pop ecx	_62i_1:0040D260
.nsp1:4AD5C0C4 pop edi	_62i_1:0040D261
.nsp1:4AD5C0C5 cmp ecx, 0	_62i_1:0040D262

.nsp1:4AD5C0C8 jz short loc_4AD5C0CF	_62i_1:0040D265 jz short loc_40D26C
.nsp1:4AD5C0CA add ebx, 8	_62i_1:0040D267 add ebx, 8
.nsp1:4AD5C0CD jmp short loc_4AD5C096	_62i_1:0040D26A jmp short loc_40D231
.nsp1:4AD5C0CF ;	_62i_1:0040D26C ;
.nsp1:4AD5C0CF	_62i_1:0040D26C
.nsp1:4AD5C0CF loc_4AD5C0CF: ; CODE XREF: start+9Bj	_62i_1:0040D26C loc_40D26C: ; CODE XREF: start+99j
.nsp1:4AD5C0CF ; start+CAj	_62i_1:0040D26C ; start+CAj
.nsp1:4AD5C0CF push 8000h	_62i_1:0040D26C
.nsp1:4AD5C0D4 push 0	_62i_1:0040D271
.nsp1:4AD5C0D6 push dword ptr [ebp-1E6h]	_62i_1:0040D273 push dword ptr [ebp- 163h]
.nsp1:4AD5C0DC call dword ptr [ebp-162h]	_62i_1:0040D279 call dword ptr [ebp- 0EFh]
.nsp1:4AD5C0E2 lea esi, [ebp-1DEh]	_62i_1:0040D27F lea esi, [ebp- 15Bh]
.nsp1:4AD5C0E8 mov ecx, [esi+8]	_62i_1:0040D285 mov ecx, [esi+8]
.nsp1:4AD5C0EB lea edx, [esi+10h]	_62i_1:0040D288 lea edx, [esi+10h]
.nsp1:4AD5C0EE mov esi, [esi]	_62i_1:0040D28B mov esi, [esi]
.nsp1:4AD5C0F0 mov edi, esi	_62i_1:0040D28D mov edi, esi
.nsp1:4AD5C0F2 cmp ecx, 0	_62i_1:0040D28F cmp ecx, 0
.nsp1:4AD5C0F5 jz short loc_4AD5C136	_62i_1:0040D292 jz short loc_40D2D3
.nsp1:4AD5C0F7	_62i_1:0040D294
	_62i_1:0040D294 loc_40D294: ; CODE XREF: start+100j
.nsp1:4AD5C0F7 ; start+10Ej	_62i_1:0040D294 ; start+10Ej
.nsp1:4AD5C0F7 mov al, [edi]	_62i_1:0040D294 mov al, [edi]
.nsp1:4AD5C0F9 inc edi	_62i_1:0040D296 inc edi
.nsp1:4AD5C0FA sub al, 0E8h	_62i_1:0040D297 sub al, 0E8h
.nsp1:4AD5C0FC	_62i_1:0040D299
.nsp1:4AD5C0FC loc_4AD5C0FC: ; CODE XREF: start+136j	_62i_1:0040D299 loc_40D299: ; CODE XREF: start+136j
.nsp1:4AD5C0FC cmp al, 1	_62i_1:0040D299 cmp al, 1

.nsp1:4AD5C0FE ja short loc_4AD5C0F7	_62i_1:0040D29B ja short loc_40D294			
.nsp1:4AD5C100 mov eax, [edi] 62i 1:0040D29D mov eax, [edi]				
.nsp1:4AD5C102 cmp byte ptr [edx+1], 0	_62i_1:0040D29F cmp byte ptr [edx+1], 0			
.nsp1:4AD5C106 jz short loc_4AD5C11C [62i_1:0040D2A3 jz short loc_40D2B9				
$[nsp1:4AD5C108 mov bl, [edx] $ $[62i_1:0040D2A5 mov bl, [edx]$				
.nsp1:4AD5C10A cmp [edi], bl	_62i_1:0040D2A7 cmp [edi], bl			
.nsp1:4AD5C10C jnz short loc_4AD5C0F7	_62i_1:0040D2A9 jnz short loc_40D294			
.nsp1:4AD5C10E mov bl, [edi+4]	_62i_1:0040D2AB mov bl, [edi+4]			
.nsp1:4AD5C111 shr ax, 8	_62i_1:0040D2AE shr ax, 8			
.nsp1:4AD5C115 rol eax, 10h	_62i_1:0040D2B2 rol eax, 10h			
.nsp1:4AD5C118 xchg al, ah	_62i_1:0040D2B5 xchg al, ah			
.nsp1:4AD5C11A jmp short loc_4AD5C126	_62i_1:0040D2B7 jmp short loc_40D2C3			
.nsp1:4AD5C11C ;	_62i_1:0040D2B9 ;			
.nsp1:4AD5C11C	_62i_1:0040D2B9			
.nsp1:4AD5C11C loc_4AD5C11C: ; CODE XREF: start+108j	_62i_1:0040D2B9 loc_40D2B9: ; CODE XREF: start+108j			
.nsp1:4AD5C11C mov bl, [edi+4]	_62i_1:0040D2B9 mov bl, [edi+4]			
.nsp1:4AD5C11F xchg al, ah	_62i_1:0040D2BC xchg al, ah			
.nsp1:4AD5C121 rol eax, 10h	_62i_1:0040D2BE rol eax, 10h			
.nsp1:4AD5C124 xchg al, ah	_62i_1:0040D2C1 xchg al, ah			
.nsp1:4AD5C126	_62i_1:0040D2C3			
.nsp1:4AD5C126 loc_4AD5C126: ; CODE XREF: start+11Cj	_62i_1:0040D2C3 loc_40D2C3: ; CODE XREF: start+11Cj			
.nsp1:4AD5C126 sub eax, edi	_62i_1:0040D2C3 sub eax, edi			
.nsp1:4AD5C128 add eax, esi	_62i_1:0040D2C5 add eax, esi			
.nsp1:4AD5C12A mov [edi], eax	_62i_1:0040D2C7 mov [edi], eax			
.nsp1:4AD5C12C add edi, 5	_62i_1:0040D2C9 add edi, 5			
.nsp1:4AD5C12F sub bl, 0E8h	_62i_1:0040D2CC sub bl, 0E8h			
.nsp1:4AD5C132 mov eax, ebx	_62i_1:0040D2CF mov eax, ebx			
.nsp1:4AD5C134 loop loc_4AD5C0FC	_62i_1:0040D2D1 loop loc_40D299			
.nsp1:4AD5C136	_62i_1:0040D2D3			

.nsp1:4AD5C136 loc_4AD5C136: ; CODE XREF: start+F7j	_62i_1:0040D2D3 loc_40D2D3: ; CODE XREF: start+F7j			
.nsp1:4AD5C136 call sub_4AD5C275	_62i_1:0040D2D3 call sub_40D412			
.nsp1:4AD5C13B lea ecx, [ebp-1CAh]	_62i_1:0040D2D8 lea ecx, [ebp- 147h]			
.nsp1:4AD5C141 mov eax, [ecx+8]	_62i_1:0040D2DE mov eax, [ecx+8]			
.nsp1:4AD5C144 cmp eax, 0	_62i_1:0040D2E1 cmp eax, 0			
.nsp1:4AD5C147 jz loc_4AD5C1CE _62i_1:0040D2E4 jz loc_40D36B				
.nsp1:4AD5C14D mov esi, edx62i_1:0040D2EA mov esi, edx				
.nsp1:4AD5C14F sub esi, [ecx+10h]	_62i_1:0040D2EC sub esi, [ecx+10h]			
.nsp1:4AD5C152 jz short loc_4AD5C1CE	_62i_1:0040D2EF jz short loc_40D36B			
.nsp1:4AD5C154 mov [ecx+10h], esi	_62i_1:0040D2F1 mov [ecx+10h], esi			
.nsp1:4AD5C157 lea esi, [ebp-19Ah]	_62i_1:0040D2F4 lea esi, [ebp-117h]			
.nsp1:4AD5C15D mov esi, [esi]	_62i_1:0040D2FA mov esi, [esi]			
.nsp1:4AD5C15F lea ebx, [esi-4]	_62i_1:0040D2FC lea ebx, [esi-4]			
.nsp1:4AD5C162 mov eax, [ecx]	_62i_1:0040D2FF mov eax, [ecx]			
.nsp1:4AD5C164 cmp eax, 1	_62i_1:0040D301 cmp eax, 1			
.nsp1:4AD5C167 jz short loc_4AD5C173	_62i_1:0040D304 jz short loc_40D310			
.nsp1:4AD5C169 mov edi, edx	_62i_1:0040D306 mov edi, edx			
.nsp1:4AD5C16B add edi, [ecx+8]	_62i_1:0040D308 add edi, [ecx+8]			
.nsp1:4AD5C16E mov ecx, [ecx+10h]	_62i_1:0040D30B mov ecx, [ecx+10h]			
.nsp1:4AD5C171 jmp short loc_4AD5C17B	_62i_1:0040D30E jmp short loc_40D318			
.nsp1:4AD5C173 ;	_62i_1:0040D310 ;			
.nsp1:4AD5C173	_62i_1:0040D310			
.nsp1:4AD5C173 loc_4AD5C173: ; CODE XREF: start+169j	_62i_1:0040D310 loc_40D310: ; CODE XREF: start+169j			
.nsp1:4AD5C173 mov edi, esi	_62i_1:0040D310 mov edi, esi			
.nsp1:4AD5C175 add edi, [ecx+8]	_62i_1:0040D312 add edi, [ecx+8]			
.nsp1:4AD5C178 mov ecx, [ecx+10h]	_62i_1:0040D315 mov ecx, [ecx+10h]			
.nsp1:4AD5C17B	_62i_1:0040D318			
.nsp1:4AD5C17B loc_4AD5C17B: ; CODE XREF: start+173j	_62i_1:0040D318 loc_40D318: ; CODE XREF: start+173j			
.nsp1:4AD5C17B ; start+18Ej	_62i_1:0040D318 ; start+18Ej			

.nsp1:4AD5C17B xor eax, eax	_62i_1:0040D318 xor eax, eax
.nsp1:4AD5C17D mov al, [edi]	_62i_1:0040D31A mov al, [edi]
.nsp1:4AD5C17F inc edi	_62i_1:0040D31C inc edi
.nsp1:4AD5C180 or eax, eax	_62i_1:0040D31D or eax, eax
.nsp1:4AD5C182 jz short loc_4AD5C1A4	_62i_1:0040D31F jz short loc_40D341
.nsp1:4AD5C184 cmp al, 0EFh	_62i_1:0040D321 cmp al, 0EFh
.nsp1:4AD5C186 ja short loc_4AD5C18E	_62i_1:0040D323 ja short loc_40D32B
.nsp1:4AD5C188	_62i_1:0040D325
.nsp1:4AD5C188 loc_4AD5C188: ; CODE XREF: start+19Dj	_62i_1:0040D325 loc_40D325: ; CODE XREF: start+19Dj
.nsp1:4AD5C188 ; start+1A4j	_62i_1:0040D325 ; start+1A4j
.nsp1:4AD5C188 add ebx, eax	_62i_1:0040D325 add ebx, eax
.nsp1:4AD5C18A add [ebx], ecx	_62i_1:0040D327 add [ebx], ecx
.nsp1:4AD5C18C jmp short loc_4AD5C17B	_62i_1:0040D329 jmp short loc_40D318
.nsp1:4AD5C18E ;	_62i_1:0040D32B ;
.nsp1:4AD5C18E	_62i_1:0040D32B
.nsp1:4AD5C18E loc_4AD5C18E: ; CODE XREF: start+188j	_62i_1:0040D32B loc_40D32B: ; CODE XREF: start+188j
.nsp1:4AD5C18E and al, 0Fh	_62i_1:0040D32B and al, 0Fh
.nsp1:4AD5C190 shl eax, 10h	_62i_1:0040D32D shl eax, 10h
.nsp1:4AD5C193 mov ax, [edi]	_62i_1:0040D330 mov ax, [edi]
.nsp1:4AD5C196 add edi, 2	_62i_1:0040D333 add edi, 2
.nsp1:4AD5C199 or eax, eax	_62i_1:0040D336 or eax, eax
.nsp1:4AD5C19B jnz short loc_4AD5C188	_62i_1:0040D338 jnz short loc_40D325
.nsp1:4AD5C19D mov eax, [edi]	_62i_1:0040D33A mov eax, [edi]
.nsp1:4AD5C19F add edi, 4	_62i_1:0040D33C add edi, 4
.nsp1:4AD5C1A2 jmp short loc_4AD5C188	_62i_1:0040D33F jmp short loc_40D325
.nsp1:4AD5C1A4 ;	_62i_1:0040D341 ;
.nsp1:4AD5C1A4	_62i_1:0040D341

CODE XREF: start+184j	XREF: start+184j		
.nsp1:4AD5C1A4 xor ebx, ebx	_62i_1:0040D341 xor ebx, ebx		
.nsp1:4AD5C1A6 xchg edi, esi	_62i_1:0040D343 xchg edi, esi		
.nsp1:4AD5C1A8 mov eax, [esi]	_62i_1:0040D345 mov eax, [esi]		
.nsp1:4AD5C1AA cmp eax, 0	_62i_1:0040D347 cmp eax, 0		
.nsp1:4AD5C1AD jz short loc_4AD5C1CE	_62i_1:0040D34A jz short loc_40D36B		
.nsp1:4AD5C1AF	_62i_1:0040D34C		
.nsp1:4AD5C1AF loc_4AD5C1AF: ; CODE XREF: start+1BCj	_62i_1:0040D34C loc_40D34C: ; CODE XREF: start+1BCj		
.nsp1:4AD5C1AF lodsd	_62i_1:0040D34C lodsd		
.nsp1:4AD5C1B0 or eax, eax	_62i_1:0040D34D or eax, eax		
.nsp1:4AD5C1B2 jz short loc_4AD5C1BC	_62i_1:0040D34F jz short loc_40D359		
.nsp1:4AD5C1B4 add ebx, eax	_62i_1:0040D351 add ebx, eax		
.nsp1:4AD5C1B6 add [edi+ebx], cx	_62i_1:0040D353 add [edi+ebx], cx		
.nsp1:4AD5C1BA jmp short loc_4AD5C1AF	_62i_1:0040D357 jmp short loc_40D34C		
.nsp1:4AD5C1BC ;	_62i_1:0040D359 ;		
.nsp1:4AD5C1BC	_62i_1:0040D359		
.nsp1:4AD5C1BC loc_4AD5C1BC: ; CODE XREF: start+1B4j	_62i_1:0040D359 loc_40D359: ; CODE XREF: start+1B4j		
.nsp1:4AD5C1BC xor ebx, ebx	_62i_1:0040D359 xor ebx, ebx		
.nsp1:4AD5C1BE shr ecx, 10h	_62i_1:0040D35B shr ecx, 10h		
.nsp1:4AD5C1C1	_62i_1:0040D35E		
.nsp1:4AD5C1C1 loc_4AD5C1C1: ; CODE XREF: start+1CEj	_62i_1:0040D35E loc_40D35E: ; CODE XREF: start+1CEj		
.nsp1:4AD5C1C1 lodsd	_62i_1:0040D35E lodsd		
.nsp1:4AD5C1C2 or eax, eax	_62i_1:0040D35F or eax, eax		
.nsp1:4AD5C1C4 jz short loc_4AD5C1CE	_62i_1:0040D361 jz short loc_40D36B		
.nsp1:4AD5C1C6 add ebx, eax	_62i_1:0040D363 add ebx, eax		
.nsp1:4AD5C1C8 add [edi+ebx], cx	_62i_1:0040D365 add [edi+ebx], cx		
.nsp1:4AD5C1CC jmp short loc_4AD5C1C1	_62i_1:0040D369 jmp short loc_40D35E		
.nsp1:4AD5C1CE ;	_62i_1:0040D36B ;		
I	I		

Insp1:4AD5C1CE loc_4AD5C1CE:62i_1:0040D36B loc_40D36B:; CODECODE XREF: start+149istart+154j62i_1:0040D36B; start+154jInsp1:4AD5C1CEicstart+154j62i_1:0040D36Bica esi, [cbp-18Bh]Insp1:4AD5C1D4movedx, [esi]62i_1:0040D371movedx, [esi]Insp1:4AD5C1D6leaesi, [ebp-1B2h]62i_1:0040D373leaesi, [ebp-12Fh]Insp1:4AD5C1D6moval, [esi]62i_1:0040D379moval, [esi]Insp1:4AD5C1D6moval, [esi]62i_1:0040D379moval, [esi]Insp1:4AD5C1D6moval, [esi]62i_1:0040D378cmpal, 1Insp1:4AD5C1D6moval, [esi]62i_1:0040D379moval, [esi]Insp1:4AD5C1E6gizshort loc_4AD5C22162i_1:0040D377moval, [esi]Insp1:4AD5C1E6gizshort loc_4AD5C22162i_1:0040D382pushedxInsp1:4AD5C1E6gizdidcdx, [esi+4]62i_1:0040D382pushedxInsp1:4AD5C1E6gizdidcdz_1:0040D384pushedxInsp1:4AD5C1E7gizhoth62i_1:0040D385push4Insp1:4AD5C1E7gizlooh62i_1:0040D385push4Insp1:4AD5C1E7gizlooh62i_1:0040D385push4Insp1:4AD5C1E7gizloo62i_1:0040D385push4Insp1:4AD5C1E7gizloo62i_1:0040D385push4Insp1:4AD5C1E7giz<	.nsp1:4AD5C1CE	_62i_1:0040D36B
nsp1:4AD5C1CEleaesi, [ebp-20Eh] $[62i_1:0040D36B$ leaesi, [ebp-18Bh]nsp1:4AD5C1D4movedx, [esi] $[62i_1:0040D371$ movedx, [esi]nsp1:4AD5C1D6leaesi, [ebp-182h] $[62i_1:0040D373$ leaesi, [ebp-12Fh]nsp1:4AD5C1DCmoval, [csi] $[62i_1:0040D373$ moval, [csi]nsp1:4AD5C1D2impal, 1 $[62i_1:0040D378$ mmpal, 1nsp1:4AD5C1E0jnzshort loc_4AD5C221 $[62i_1:0040D376]$ moval, [csi]nsp1:4AD5C1E5pushedx, [esi+4] $[62i_1:0040D382]$ pushedxnsp1:4AD5C1E6pushedx $[62i_1:0040D383]$ pushedxnsp1:4AD5C1E7pushesi $[62i_1:0040D384]$ pushesinsp1:4AD5C1E8pushedx $[62i_1:0040D385]$ push4nsp1:4AD5C1E7pushesi $[62i_1:0040D386]$ push4nsp1:4AD5C1E8push4 $[62i_1:0040D386]$ push4nsp1:4AD5C1E7pushedx $[62i_1:0040D386]$ push4nsp1:4AD5C1E7pushedx $[62i_1:0040D386]$ push4nsp1:4AD5C1E6pushedx $[62i_1:0040D386]$ push4nsp1:4AD5C1E7pushedx $[62i_1:0040D386]$ push4nsp1:4AD5C166popedi $[62i_1:0040D396]$ pushedxnsp1:4AD5C1F6popedi $[62i_1:0040D396]$ popesinsp1:4AD5C1F8	/	
nsp1:4AD5C1D4movedx, [esi] $[62]$ 1:0040D371movedx, [esi]nsp1:4AD5C1D6leaesi, [ebp-1B2h] $[62i]$ 1:0040D373leaesi, [ebp-12Fh]nsp1:4AD5C1DCmoval, [esi] $[62i]$ 1:0040D379moval, [esi]nsp1:4AD5C1DEmpal, 1 $[62i]$ 1:0040D37Bcmpal, 1nsp1:4AD5C1E0jnzshort loc_4AD5C221 $[62i]$ 1:0040D37Djnzshort loc_40D3BEnsp1:4AD5C1E5pushesi $[62i]$ 1:0040D37Faddedx, [esi+4]nsp1:4AD5C1E6pushedx $[62i]$ 1:0040D382pushesinsp1:4AD5C1E7pushesi $[62i]$ 1:0040D383pushedxnsp1:4AD5C1E8pushedx $[62i]$ 1:0040D385push4nsp1:4AD5C1E7pushesi $[62i]$ 1:0040D387pushedxnsp1:4AD5C1E8push4 $[62i]$ 1:0040D385push4nsp1:4AD5C1F6pushedx $[62i]$ 1:0040D387push100hnsp1:4AD5C1F6popedi $[62i]$ 1:0040D380culdword ptr [cbp-0F7h]nsp1:4AD5C1F6popedi $[62i]$ 1:0040D393popedinsp1:4AD5C1F8empeax, 1 $[62i]$ 1:0040D395cmpeax, 1nsp1:4AD5C1F8inzeax, 8 $[62i]$ 1:0040D394popesinsp1:4AD5C1F8inzeax, 8 $[62i]$ 1:0040D395cmpeax, 8nsp1:4AD5C201addesi, 8 $[62i]$ 1:0040D394popesi </td <td>.nsp1:4AD5C1CE ;start+154j</td> <td>_62i_1:0040D36B ; start+154j</td>	.nsp1:4AD5C1CE ;start+154j	_62i_1:0040D36B ; start+154j
nsp1:4AD5C1D6leaesi, [ebp-1B2h] $62i_1:0040D373$ leaesi, [ebp-12Fh]nsp1:4AD5C1DCmoval, [esi] $62i_1:0040D379$ moval, [esi]nsp1:4AD5C1DEinzshort loc_4AD5C221 $62i_1:0040D37B$ cmpal, 1nsp1:4AD5C1E2addedx, [esi+4] $62i_1:0040D37D$ jnzshort loc_40D3BEnsp1:4AD5C1E5pushesi $62i_1:0040D37F$ addedx, [esi+4]nsp1:4AD5C1E6pushesi $62i_1:0040D382$ pushesinsp1:4AD5C1E6pushedx $62i_1:0040D383$ pushedxnsp1:4AD5C1E7pushesi $62i_1:0040D385$ push4nsp1:4AD5C1E8push4 $62i_1:0040D387$ push4nsp1:4AD5C1E7pushedx $62i_1:0040D387$ push4nsp1:4AD5C1E8push4 $62i_1:0040D387$ push4nsp1:4AD5C1F6popedi $62i_1:0040D387$ push100hnsp1:4AD5C1F6popedi $62i_1:0040D387$ pushedxnsp1:4AD5C1F6popedi $62i_1:0040D387$ pushedxnsp1:4AD5C1F6popedi $62i_1:0040D387$ pushedxnsp1:4AD5C1F6popedi $62i_1:0040D387$ pushedinsp1:4AD5C1F7popesi $62i_1:0040D389$ popesinsp1:4AD5C1F8inzeax, 1 $62i_1:0040D398$ inzloc_40D55Bnsp1:4AD5C207addesi, 8 $62i_1:$.nsp1:4AD5C1CE lea esi, [ebp-20Eh]	_62i_1:0040D36B lea esi, [ebp- 18Bh]
$nsp1:AD5C1DC$ mov $al, [esi]$ $(ci_1:0040D379$ mov $al, [esi]$ $nsp1:AD5C1DE$ cmp $al, 1$ $(ci_1:0040D37B$ cmp $al, 1$ $nsp1:AD5C1E0$ jnz $short loc_4AD5C221$ $(ci_1:0040D37D$ jnz $short loc_40D3BE$ $nsp1:AD5C1E2$ add $cdx, [esi+4]$ $(ci_1:0040D37F)$ add $cdx, [esi+4]$ $nsp1:AD5C1E5$ $push$ esi $(ci_1:0040D382)$ $push$ esi $nsp1:AD5C1E6$ $push$ edx $(ci_1:0040D383)$ $push$ edx $nsp1:AD5C1E7$ $push$ esi $(ci_1:0040D384)$ $push$ esi $nsp1:AAD5C1E7$ $push$ esi $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1E8$ $push$ 4 $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1E6$ $push$ edx $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1E7$ $push$ edx $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1F6$ $push$ edx $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1F6$ $push$ edx $(ci_1:0040D387)$ $push$ edx $nsp1:AAD5C1F6$ pop esi $(ci_1:0040D393)$ pop esi $nsp1:AAD5C1F8$ mp $eax, 1$ $(ci_1:0040D394)$ pop esi $nsp1:AAD5C201$ add $esi, 8$ $(ci_1:0040D394)$ pop esi $nsp1:AAD5C201$ add $esi, 8$ $(ci_1:0040D394)$ mov $ecx, 8$ <td< td=""><td>.nsp1:4AD5C1D4 mov edx, [esi]</td><td>_62i_1:0040D371 mov edx, [esi]</td></td<>	.nsp1:4AD5C1D4 mov edx, [esi]	_62i_1:0040D371 mov edx, [esi]
Insp1:4AD5C1DEcmpal, 1 $nsp1:4AD5C1DE$ jnzshort loc_4AD5C221 $62i_1:0040D37B$ cmpal, 1 $nsp1:4AD5C1E2$ addedx, [esi+4] $62i_1:0040D37F$ addedx, [esi+4] $nsp1:4AD5C1E5$ pushesi $62i_1:0040D37F$ addedx, [esi+4] $nsp1:4AD5C1E6$ pushesi $62i_1:0040D382$ pushesi $nsp1:4AD5C1E6$ pushedx $62i_1:0040D383$ pushedx $nsp1:4AD5C1E7$ pushesi $62i_1:0040D385$ pushedx $nsp1:4AD5C1E8$ push4 $62i_1:0040D385$ push4 $nsp1:4AD5C1E7$ pushedx $62i_1:0040D387$ push100h $nsp1:4AD5C1E7$ pushedx $62i_1:0040D387$ pushedx $nsp1:4AD5C1E7$ pushedx $62i_1:0040D387$ pushedx $nsp1:4AD5C1E7$ pushedx $62i_1:0040D380$ pushedx $nsp1:4AD5C1F6$ calldword ptr [ebp-16Ah] $62i_1:0040D393$ popedi $nsp1:4AD5C1F8$ rmpeax, 1 $62i_1:0040D394$ popesi $nsp1:4AD5C1F8$ rmpeax, 1 $62i_1:0040D395$ rmpeax, 1 $nsp1:4AD5C201$ addesi, 8 $62i_1:0040D394$ popesi $nsp1:4AD5C204$ rovecx, 8 $62i_1:0040D394$ popesi $nsp1:4AD5C204$ rovecx, 8 $62i_1:0040D394$ potesi $nsp1:4AD5C205$ subesi, 8 $62i_1:0040D3A4$ <td< td=""><td>.nsp1:4AD5C1D6 lea esi, [ebp-1B2h]</td><td>_62i_1:0040D373 lea esi, [ebp-12Fh]</td></td<>	.nsp1:4AD5C1D6 lea esi, [ebp-1B2h]	_62i_1:0040D373 lea esi, [ebp- 12Fh]
nsp1:4AD5C1E0jnzshort loc_4AD5C221 $[62i_1:0040D37D$ jnzshort loc_40D3BEnsp1:4AD5C1E2addedx, [esi+4] $[62i_1:0040D37F$ addedx, [esi+4]nsp1:4AD5C1E5pushesi $[62i_1:0040D382$ pushesinsp1:4AD5C1E6pushedx $[62i_1:0040D382$ pushesinsp1:4AD5C1E6pushedx $[62i_1:0040D383$ pushedxnsp1:4AD5C1E7pushesi $[62i_1:0040D385$ pushedxnsp1:4AD5C1E8push4 $[62i_1:0040D387$ push100hnsp1:4AD5C1E7pushedx $[62i_1:0040D387$ push100hnsp1:4AD5C1E7pushedx $[62i_1:0040D380$ pushedxnsp1:4AD5C1F6pushedx $[62i_1:0040D380$ pushedxnsp1:4AD5C1F6popedi $[62i_1:0040D380$ pushedinsp1:4AD5C1F8cmpeax, 1 $[62i_1:0040D394$ popesinsp1:4AD5C1F8impeax, 1 $[62i_1:0040D398$ jnzloc_40D55Bnsp1:4AD5C1F8impeax, 8 $[62i_1:0040D396$ impexx, 8nsp1:4AD5C201addesi, 8 $[62i_1:0040D3A1$ movecx, 8nsp1:4AD5C205subesi, 0Ch $[62i_1:0040D3A6$ rep movsbnsp1:4AD5C206subesi, 8 $[62i_1:0040D3AE$ pushesinsp1:4AD5C211pushesi $[62i_1:0040D3AE$ pushesinsp1:4AD5C212pushdword ptr [.nsp1:4AD5C1DC mov al, [esi]	_62i_1:0040D379 mov al, [esi]
$nsp1:4AD5C1E2$ addedx, [esi+4] $edi_1:0040D37F$ addedx, [esi+4] $nsp1:4AD5C1E5$ pushesi $edi_1:0040D382$ pushesi $nsp1:4AD5C1E6$ pushedx $edi_1:0040D383$ pushesi $nsp1:4AD5C1E7$ pushesi $edi_1:0040D384$ pushesi $nsp1:4AD5C1E8$ push4 $edi_1:0040D385$ push4 $nsp1:4AD5C1E8$ push4 $edi_1:0040D385$ push4 $nsp1:4AD5C1EA$ push100h $edi_1:0040D387$ push100h $nsp1:4AD5C1EF$ pushedx $edi_1:0040D387$ push100h $nsp1:4AD5C1F6$ calldword ptr [ebp-16Ah] $edi_1:0040D380$ pushedx $nsp1:4AD5C1F6$ popedi $edi_1:0040D393$ popedi $nsp1:4AD5C1F7$ popesi $edi_1:0040D393$ popedi $nsp1:4AD5C1F8$ cmpeax, 1 $edi_1:0040D395$ cmpeax, 1 $nsp1:4AD5C1F8$ popesi $edi_1:0040D395$ cmpeax, 1 $nsp1:4AD5C201$ addesi, 8 $edi_1:0040D398$ jnzloc_40D55B $nsp1:4AD5C209$ rep movsb $edi_1:0040D3A6$ rep movsbesi, 8 $nsp1:4AD5C208$ subesi, 0Ch $edi_1:0040D3A8$ subesi, 0Ch $nsp1:4AD5C208$ subesi, 0Ch $edi_1:0040D3A8$ subesi, 8 $nsp1:4AD5C211$ pushesi $edi_1:0040D3A8$ subesi $nsp1:4AD5C212$ push<	.nsp1:4AD5C1DE cmp al, 1	_62i_1:0040D37B cmp al, 1
nsp1:4AD5C1E5pushesi $62i_1:0040D382$ pushesinsp1:4AD5C1E6pushedx $62i_1:0040D383$ pushedxnsp1:4AD5C1E7pushesi $62i_1:0040D383$ pushedxnsp1:4AD5C1E8push4 $62i_1:0040D385$ push4nsp1:4AD5C1E8push4 $62i_1:0040D387$ push100hnsp1:4AD5C1E7pushedx $62i_1:0040D387$ push100hnsp1:4AD5C1E7pushedx $62i_1:0040D387$ pushedxnsp1:4AD5C1F0calldword ptr [ebp-16Ah] $62i_1:0040D380$ calldword ptr [ebp-0F7h]nsp1:4AD5C1F7popedi $62i_1:0040D393$ popedinsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D393$ popedinsp1:4AD5C1F8rmpeax, 1 $62i_1:0040D395$ cmpeax, 1nsp1:4AD5C1F8inceax, 8 $62i_1:0040D398$ jnzloc_40D55Bnsp1:4AD5C1F8incexx, 8 $62i_1:0040D346$ rep movsbnsp1:4AD5C201addesi, 8 $62i_1:0040D3A6$ rep movsbnsp1:4AD5C208subesi, 0Ch $62i_1:0040D3A8$ subesi, 8nsp1:4AD5C208subesi, 0Ch $62i_1:0040D3A8$ subedi, 8nsp1:4AD5C211pushesi $62i_1:0040D3A5$ pushesinsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3A5$ pushesinsp1:4AD5C215pushdword ptr [esi-4] 62	.nsp1:4AD5C1E0 jnz short loc_4AD5C221	_62i_1:0040D37D jnz short loc_40D3BE
$nsp1:4AD5C1E6$ pushedx $62i_1:0040D383$ pushedx $nsp1:4AD5C1E7$ pushesi $62i_1:0040D384$ pushesi $nsp1:4AD5C1E8$ push4 $62i_1:0040D385$ push4 $nsp1:4AD5C1EA$ push100h $62i_1:0040D387$ push100h $nsp1:4AD5C1EF$ pushedx $62i_1:0040D387$ pushedx $nsp1:4AD5C1E6$ pushedx $62i_1:0040D380$ pushedx $nsp1:4AD5C1F0$ calldword ptr [ebp-16Ah] $62i_1:0040D393$ popedi $nsp1:4AD5C1F6$ popedi $62i_1:0040D394$ popesi $nsp1:4AD5C1F7$ popesi $62i_1:0040D395$ cmpeax, 1 $nsp1:4AD5C1F8$ cmpeax, 1 $62i_1:0040D395$ cmpeax, 1 $nsp1:4AD5C1F8$ inpeax, 1 $62i_1:0040D395$ cmpeax, 1 $nsp1:4AD5C201$ addesi, 8 $62i_1:0040D3A1$ movecx, 8 $nsp1:4AD5C209$ rep movsb $62i_1:0040D3A6$ rep movsb $nsp1:4AD5C208$ subesi, 0Ch $62i_1:0040D3A8$ subesi, 0Ch $nsp1:4AD5C211$ pushesi $62i_1:0040D3A5$ pushesi $nsp1:4AD5C211$ pushesi $62i_1:0040D3A5$ pushesi $nsp1:4AD5C211$ pushesi $62i_1:0040D3A5$ pushesi $nsp1:4AD5C211$ pushesi $62i_1:0040D3A5$ pushesi $nsp1:4AD5C211$ pushesi $62i_1:0040D3A5$ <	.nsp1:4AD5C1E2 add edx, [esi+4]	_62i_1:0040D37F add edx, [esi+4]
$nsp1:4AD5C1E7$ $push$ esi $62i_1:0040D384$ $push$ esi $nsp1:4AD5C1E8$ $push$ 4 $62i_1:0040D385$ $push$ 4 $nsp1:4AD5C1EA$ $push$ $100h$ $62i_1:0040D387$ $push$ $100h$ $nsp1:4AD5C1EF$ $push$ edx $62i_1:0040D38C$ $push$ edx $nsp1:4AD5C1F0$ $call$ $dword ptr [ebp-16Ah]$ $62i_1:0040D38D$ $call$ $dword ptr [ebp-0F7h]$ $nsp1:4AD5C1F6$ pop edi $62i_1:0040D393$ pop edi $nsp1:4AD5C1F7$ pop esi $62i_1:0040D394$ pop esi $nsp1:4AD5C1F8$ cmp $eax, 1$ $62i_1:0040D395$ cmp $eax, 1$ $nsp1:4AD5C1F8$ pop $esi, 8$ $62i_1:0040D398$ pop esi $nsp1:4AD5C1F8$ pop $eax, 1$ $62i_1:0040D398$ pop $eax, 1$ $nsp1:4AD5C201$ add $esi, 8$ $62i_1:0040D3A1$ mov $ecx, 8$ $nsp1:4AD5C208$ sub $esi, 0Ch$ $62i_1:0040D3A8$ sub $esi, 0Ch$ $nsp1:4AD5C208$ sub $esi, 0Ch$ $62i_1:0040D3AE$ $push$ esi $nsp1:4AD5C211$ $push$ esi $62i_1:0040D3AF$ $push$ esi $nsp1:4AD5C212$ $push$ $dword ptr [esi-4]$ $62i_1:0040D3AF$ $push$ $dword ptr [esi-4]$ $nsp1:4AD5C215$ $push$ $dword ptr [esi-4]$ $62i_1:0040D3AF$ $push$ $dword ptr [esi-4]$ $nsp1:4AD5C215$ $push$ $dword ptr [esi-4]$ <	.nsp1:4AD5C1E5 push esi	_62i_1:0040D382
nsp1:4AD5C1E8push4 $62i_1:0040D385$ push4nsp1:4AD5C1EApush100h $62i_1:0040D387$ push100hnsp1:4AD5C1EFpushedx $62i_1:0040D38C$ pushedxnsp1:4AD5C1F0calldword ptr [ebp-16Ah] $62i_1:0040D38D$ calldword ptr [ebp-0F7h]nsp1:4AD5C1F6popedi $62i_1:0040D393$ popedinsp1:4AD5C1F7popesi $62i_1:0040D393$ popesinsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D395$ cmpeax, 1nsp1:4AD5C1F8inzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55Bnsp1:4AD5C201addesi, 8 $62i_1:0040D3A1$ movecx, 8nsp1:4AD5C209rep movsb $62i_1:0040D3A5$ subesi, 0Chnsp1:4AD5C20Esubesi, 8 $62i_1:0040D3A8$ subesi, 0Chnsp1:4AD5C20Esubedi, 8 $62i_1:0040D3A5$ pushedi, 8nsp1:4AD5C211pushesi $62i_1:0040D3A5$ pushedi, 8nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3A5$ pushedi, 8nsp1:4AD5C215pushdword ptr [esi-4] $62i_1:0040D3A5$ pushedi, 8nsp1:4AD5C215pushdword ptr [esi-4] $62i_1:0040D3A5$ pushedi, 8nsp1:4AD5C215pushdword ptr [esi-4] $62i_1:0040D3A5$ pushedinsp1:4AD5C215pushdword ptr [esi-4] $62i_1:0040D3A5$ pushed	.nsp1:4AD5C1E6 push edx	_62i_1:0040D383
1 -2 1 1 $100h$ $62i_1:0040D387$ $push$ $100h$ $100h$ $100h$ $62i_1:0040D387$ $push$ $100h$ $100h$ $100h$ $62i_1:0040D38C$ $push$ $100h$.nsp1:4AD5C1E7 push esi	_62i_1:0040D384
$nsp1:4AD5C1EF$ pushedx $62i_1:0040D38C$ pushedx $nsp1:4AD5C1F0$ calldword ptr [ebp-16Ah] $62i_1:0040D38D$ calldword ptr [ebp-0F7h] $nsp1:4AD5C1F6$ popedi $62i_1:0040D393$ popedi $nsp1:4AD5C1F7$ popesi $62i_1:0040D394$ popesi $nsp1:4AD5C1F8$ cmpeax, 1 $62i_1:0040D395$ cmpeax, 1 $nsp1:4AD5C1F8$ jnzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55B $nsp1:4AD5C201$ addesi, 8 $62i_1:0040D3A1$ movecx, 8 $nsp1:4AD5C209$ rep movsb $62i_1:0040D3A6$ rep movsb $nsp1:4AD5C20E$ subesi, 0Ch $62i_1:0040D3AB$ subesi, 0Ch $nsp1:4AD5C20E$ subedi, 8 $62i_1:0040D3AE$ pushesi $nsp1:4AD5C211$ pushesi $62i_1:0040D3AE$ pushesi $nsp1:4AD5C212$ pushdword ptr [esi-4] $62i_1:0040D3AF$ pushesi $nsp1:4AD5C215$ push100h $62i_1:0040D3AE$ pushesi	.nsp1:4AD5C1E8 push 4	_62i_1:0040D385 push 4
.nsp1:4AD5C1F0calldword ptr [ebp-16Ah] $62i_1:0040D38D$ calldword ptr [ebp-0F7h].nsp1:4AD5C1F6popedi $62i_1:0040D393$ popedi.nsp1:4AD5C1F7popesi $62i_1:0040D394$ popesi.nsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D395$ cmpeax, 1.nsp1:4AD5C1F8jnzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55B.nsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8.nsp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A8$ subesi, 0Ch.nsp1:4AD5C20Esubesi, 0Ch $62i_1:0040D3AE$ pushesi.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AE$ pushesi.nsp1:4AD5C215push100h $62i_1:0040D3AE$ pushesi	.nsp1:4AD5C1EA push 100h	_62i_1:0040D387
.nsp1:4AD5C1F6popedi $62i_1:0040D393$ popedi.nsp1:4AD5C1F7popesi $62i_1:0040D394$ popesi.nsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D395$ cmpeax, 1.nsp1:4AD5C1FBjnzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55B.nsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8.nsp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Esubesi, 0Ch $62i_1:0040D3AE$ subesi, 0Ch.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushesi.nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1EF push edx	_62i_1:0040D38C
Inspl:4AD5C1F7popesi $62i_1:0040D394$ popesi.nsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D395$ cmpeax, 1.nsp1:4AD5C1FBjnzloc_4AD5C3BE $62i_1:0040D395$ cmpeax, 1.nsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8.nsp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3AB$ subesi, 0Ch.nsp1:4AD5C20Esubedi, 8 $62i_1:0040D3AE$ pushesi.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushesi.nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1F0 call dword ptr [ebp-16Ah]	_62i_1:0040D38D call dword ptr [ebp- 0F7h]
.nsp1:4AD5C1F8cmpeax, 1 $62i_1:0040D395$ cmpeax, 1.nsp1:4AD5C1FBjnzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55B.nsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8.nsp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3A8$ subesi, 0Ch.nsp1:4AD5C20Esubedi, 8 $62i_1:0040D3AB$ subedi, 8.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushesi.nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1F6 pop edi	_62i_1:0040D393 pop edi
Insp1:4AD5C1FBjnzloc_4AD5C3BE $62i_1:0040D398$ jnzloc_40D55BInsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8Insp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8Insp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsbInsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3A8$ subesi, 0ChInsp1:4AD5C20Esubedi, 8 $62i_1:0040D3AB$ subedi, 8Insp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesiInsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushdword ptr [esi-4]Insp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1F7 pop esi	_62i_1:0040D394 pop esi
IJ $ -$.nsp1:4AD5C201addesi, 8 $62i_1:0040D39E$ addesi, 8.nsp1:4AD5C204movecx, 8 $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3A8$ subesi, 0Ch.nsp1:4AD5C20Esubedi, 8 $62i_1:0040D3AB$ subedi, 8.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushdword ptr [esi-4].nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1F8 cmp eax, 1	_62i_1:0040D395 cmp eax, 1
.nsp1:4AD5C204movecx, 8.nsp1:4AD5C209rep movsb $62i_1:0040D3A1$ movecx, 8.nsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Esubedi, 8 $62i_1:0040D3A8$ subesi, 0Ch.nsp1:4AD5C211pushesi $62i_1:0040D3AB$ subedi, 8.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AE$ pushesi.nsp1:4AD5C215pushdword ptr [esi-4] $62i_1:0040D3AE$ pushdword ptr [esi-4].nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C1FB jnz loc_4AD5C3BE	_62i_1:0040D398 jnz loc_40D55B
.nsp1:4AD5C209rep movsb $62i_1:0040D3A6$ rep movsb.nsp1:4AD5C20Bsubesi, 0Ch $62i_1:0040D3A8$ subesi, 0Ch.nsp1:4AD5C20Esubedi, 8 $62i_1:0040D3AB$ subedi, 8.nsp1:4AD5C211pushesi $62i_1:0040D3AE$ pushesi.nsp1:4AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushdword ptr [esi-4].nsp1:4AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C201 add esi, 8	_62i_1:0040D39E add esi, 8
$nsp1:4AD5C20B$ subesi, 0Ch $62i_1:0040D3A8$ subesi, 0Ch $nsp1:4AD5C20E$ subedi, 8 $62i_1:0040D3AB$ subedi, 8 $nsp1:4AD5C211$ pushesi $62i_1:0040D3AE$ pushesi $nsp1:4AD5C212$ pushdword ptr [esi-4] $62i_1:0040D3AF$ pushdword ptr [esi-4] $nsp1:4AD5C215$ push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C204 mov ecx, 8	_62i_1:0040D3A1 mov ecx, 8
.nsp1:4AD5C20Esubedi, 8.nsp1:4AD5C211pushesi.nsp1:4AD5C212pushdword ptr [esi-4].nsp1:4AD5C215push100h.nsp1:4AD5C215push100h	.nsp1:4AD5C209 rep movsb	_62i_1:0040D3A6 rep movsb
.nsp1:4AD5C211pushesi.nsp1:4AD5C212pushdword ptr [esi-4].nsp1:4AD5C215push100h.nsp1:4AD5C215push100h	.nsp1:4AD5C20B sub esi, 0Ch	_62i_1:0040D3A8 sub esi, 0Ch
Image: Insplit AD5C212pushdword ptr [esi-4] $62i_1:0040D3AF$ pushdword ptr [esi-4]Insplit AD5C215push100h $62i_1:0040D3B2$ push100h	.nsp1:4AD5C20E sub edi, 8	_62i_1:0040D3AB sub edi, 8
.nsp1:4AD5C215 push 100h [62i_1:0040D3B2 push 100h	.nsp1:4AD5C211 push esi	_62i_1:0040D3AE push esi
	.nsp1:4AD5C212 push dword ptr [esi-4]	_62i_1:0040D3AF push dword ptr [esi-4]
.nsp1:4AD5C21A push edi _62i_1:0040D3B7 push edi	.nsp1:4AD5C215 push 100h	_62i_1:0040D3B2
	.nsp1:4AD5C21A push edi	_62i_1:0040D3B7

.nsp1:4AD5C21B call dword ptr [ebp-16Ah]	_62i_1:0040D3B8 call dword ptr [ebp- 0F7h]
.nsp1:4AD5C221	_62i_1:0040D3BE
.nsp1:4AD5C221 loc_4AD5C221: ; CODE XREF: start+1E2j	_62i_1:0040D3BE loc_40D3BE: ; CODE XREF: start+1E2j
.nsp1:4AD5C221 push ebp	_62i_1:0040D3BE push ebp
.nsp1:4AD5C222 pop ebx	_62i_1:0040D3BF pop ebx
.nsp1:4AD5C223 sub ebx, 15h	_62i_1:0040D3C0 sub ebx, 21h
.nsp1:4AD5C229 xor ecx, ecx	_62i_1:0040D3C6 xor ecx, ecx
.nsp1:4AD5C22B mov cl, [ebx]	_62i_1:0040D3C8 mov cl, [ebx]
.nsp1:4AD5C22D cmp cl, 0	_62i_1:0040D3CA cmp cl, 0
.nsp1:4AD5C230 jz short loc_4AD5C25A	_62i_1:0040D3CD jz short loc_40D3F7
.nsp1:4AD5C232 inc ebx	_62i_1:0040D3CF inc ebx
.nsp1:4AD5C233 lea esi, [ebp-20Eh]	_62i_1:0040D3D0 lea esi, [ebp-18Bh]
.nsp1:4AD5C239 mov edx, [esi]	_62i_1:0040D3D6 mov edx, [esi]
.nsp1:4AD5C23B	_62i_1:0040D3D8
.nsp1:4AD5C23B loc_4AD5C23B: ; CODE XREF: start+25Aj	_62i_1:0040D3D8 loc_40D3D8: ; CODE XREF: start+25Aj
.nsp1:4AD5C23B push esi	_62i_1:0040D3D8
.nsp1:4AD5C23C push ecx	_62i_1:0040D3D9
.nsp1:4AD5C23D push ebx	_62i_1:0040D3DA push ebx
.nsp1:4AD5C23E push edx	_62i_1:0040D3DB push edx
.nsp1:4AD5C23F push esi	_62i_1:0040D3DC push esi
.nsp1:4AD5C240 push dword ptr [ebx]	_62i_1:0040D3DD push dword ptr [ebx]
.nsp1:4AD5C242 push dword ptr [ebx+4]	_62i_1:0040D3DF push dword ptr [ebx+4]
.nsp1:4AD5C245 mov eax, [ebx+8]	_62i_1:0040D3E2 mov eax, [ebx+8]
.nsp1:4AD5C248 add eax, edx	_62i_1:0040D3E5 add eax, edx
.nsp1:4AD5C24A push eax	_62i_1:0040D3E7
.nsp1:4AD5C24B call dword ptr [ebp-16Ah]	_62i_1:0040D3E8 call dword ptr [ebp- 0F7h]
.nsp1:4AD5C251 pop edx	_62i_1:0040D3EE pop edx
.nsp1:4AD5C252 pop ebx	_62i_1:0040D3EF pop ebx
.nsp1:4AD5C253 pop ecx	_62i_1:0040D3F0 pop ecx

.nsp1:4AD5C254 pop esi	_62i_1:0040D3F1 pop esi		
.nsp1:4AD5C255 add ebx, 0Ch	_62i_1:0040D3F2 add ebx, 0Ch		
.nsp1:4AD5C258 loop loc_4AD5C23B	_62i_1:0040D3F5 loop loc_40D3D8		
.nsp1:4AD5C25A	_62i_1:0040D3F7		
.nsp1:4AD5C25A loc_4AD5C25A: ; CODE XREF: start+14j	_62i_1:0040D3F7 loc_40D3F7: ; CODE XREF: start+14j		
.nsp1:4AD5C25A ; start+232j	_62i_1:0040D3F7 ; start+232j		
.nsp1:4AD5C25A mov eax, 0	_62i_1:0040D3F7 mov eax, 0		
.nsp1:4AD5C25F cmp eax, 0	_62i_1:0040D3FC cmp eax, 0		
.nsp1:4AD5C262 jz short loc_4AD5C26E	_62i_1:0040D3FF jz short loc_40D40B		
.nsp1:4AD5C264 popa	_62i_1:0040D401 popa		
.nsp1:4AD5C265 popf	_62i_1:0040D402 popf		
.nsp1:4AD5C266 mov eax, 1	_62i_1:0040D403 mov eax, 1		
.nsp1:4AD5C26B retn 0Ch	_62i_1:0040D408 retn 0Ch		
.nsp1:4AD5C26E ;	_62i_1:0040D40B ;		
.nsp1:4AD5C26E	_62i_1:0040D40B		
.nsp1:4AD5C26E loc_4AD5C26E: ; CODE XREF: start+264j	_62i_1:0040D40B loc_40D40B: ; CODE XREF: start+264j		
.nsp1:4AD5C26E popa	_62i_1:0040D40B popa		
.nsp1:4AD5C26F popf	_62i_1:0040D40C popf		
.nsp1:4AD5C270 jmp near ptr 4AD09797h	_62i_1:0040D40D jmp near ptr dword_4012A8		
.nsp1:4AD5C270 start endp	_62i_1:0040D40D start endp		
.nsp1:4AD5C270	_62i_1:0040D40D		

11. Appendix – Olly Scipt

The following samples are Olly-script that can be used to unpack the NsPack version 3.7 packer.

```
/*
; -----
;
; NsPack 3.7 unpacking script for Olly
; Date : 22/07/2009
;
; ------
;
*/
var cpa
var errorcnt
 tryAgain:
find eip, #619DE9??????#
cmp $RESULT,0
je _tryNSPack
mov cpa,$RESULT
add cpa,2
bp cpa
run
bc cpa
sto
cmt eip, "This is the OEP (original entry point). Use this to fix the IAT"
an eip
ret
_tryNSPack:
cmp errorcnt,1
je notNSPack
mov errorcnt,1
sto
JMP _tryAgain
notNSPack:
msg "*** This executable does not look to be packed using NsPack***"
ret
```

11.1. Olly Scipt OEP locator

The following samples are olly-script that can be used to unpack the NsPack version 3.4 packer.

/*

```
; -----
;
; NsPack 3.4 OEP finder script for Olly
;
; Date : 10/08/2009
;
; ------
;
*/
var t
sti
sti
mov t,esp
bphws t,"r"
run
bphwc t
sti
sti
cmt eip, "This is the OEP (original entry point). "
msg "Dump & use this to fix the IAT!"
ret
```

12. More Appendixes...

The following provide unpacking code samples for use in analyzing NsPack.

12.1. The Unpack Code

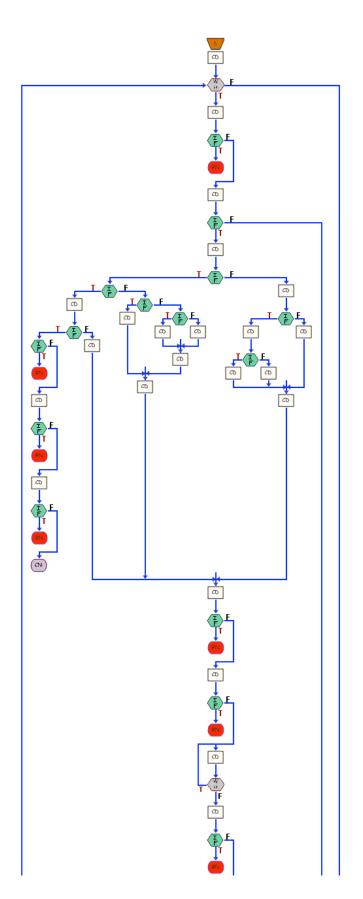
The following is the code used to unpack the embedded executable file:

```
/\star loop to unpack the code from the compressed data \!\!\!\!\!/
/*
read struct(self, struct)
read struct(struct)
                              --> Read structure from file into memory.
                                     This loads the file into memory from
                                      disk without executing it.
'*in table' is a pointer referencing the position in the data
ssize is the 32 bit value at the point of the data we have read into the function.
* /
 while (true) {
   uint32 t former size = initial byte & amount unpacked to date;
    uint32_t table_position;
   uint32 t temp value = point in table1;
   if (read struct.error) return 1;
       /* check once per mainloop - if there is an error, end */
    if (!load single bit from table(&table[(point in table1<<4) + former size],
&read struct) ) {
     uint32 t shft = 8 - (table rem&Oxff);
      shft &= 0xff;
     table position = (point_in_table2>>shft) +
((put&amount_unpacked_to_date) << (table_rem&0xff));
     table position \overline{*}=3;
      table position<<=8;</pre>
      point in table2 = last bit = 1;
      if (load_single_bit_from_table(&table[point_in_table1+0xc0], &read_struct)) {
       if (!load_single_bit_from_table(&table[point_in_table1+0xcc], &read_struct)) {
         table position = point in table1+0xf;
         table_position <<=4;</pre>
         table position += former size;
         if (!load single bit from table(&table[table position], &read struct)) {
           if (!amount_unpacked_to_date) return point_in_table2;
           point in table1 = 2*((int32 t)point in table1>=7)+9;
           if (!buffer_bounded(destination_point, dsize,
&destination_point[amount_unpacked_to_date - last_bytes[0]], 1)) return 1;
           point in table2 = (uint8 t)destination point[amount unpacked to date -
last_bytes[0]];
            destination point[amount unpacked to date] = point in table2;
           amount unpacked to date++;
           if (amount unpacked to date>=dsize) return 0;
           continue;
         } else {
           former size = load a variable number of bits from table(&table[0x534],
```

```
&read struct, former size);
           point in table1 = ((int32_t)point_in_table1>=7);
           point_in_table1 = ((point_in_table1-1) & 0xffffffd)+0xb;
         }
       } else {
         if (!load single bit from table(&table[point in table1+0xd8], &read struct)) {
           table position = last bytes[1];
         } else {
           if (!load single bit from table(&table[point in table1+0xe4], &read struct)) {
             table position = last bytes[2];
           } else {
             table position = last bytes[3];
             last bytes[3] = last bytes[2];
           }
           last_bytes[2] = last_bytes[1];
         }
         last bytes[1] = last bytes[0];
         last_bytes[0] = table position;
         former size = load a variable number of bits from table(&table[0x534],
&read struct, former size);
        point in table1 = ((int32 t)point in table1>=7);
         point_in_table1 = ((point_in_table1-1) & 0xffffffd)+0xb;
      } else {
       last bytes[3] = last bytes[2];
       last bytes[2] = last bytes[1];
       last bytes[1] = last bytes[0];
       point_in_table1 = ((int32_t)point_in_table1>=7);
       point in table1 = ((point in table1-1) & 0xffffffd)+0xa;
       former size = load a variable number of bits from table(&table[0x332],
&read struct, former size);
       table_position = ((int32_t)former_size>=4)?3:former_size;
       table position<<=6;</pre>
       table position = load n bits from table(&table[0x1b0+table position], 6,
&read_struct);
       if (table_position>=4) {
         uint32_t s = table_position;
         s>>=1;
         s--:
         temp value = (table position & point in table2) | 2;
         temp value<<=(s&0xff);</pre>
         if ((int32 t)table position<0xe) {
           temp value += load bitmap(&table[(temp value-table position)+0x2af], s,
&read struct);
         } else {
           s += 0xffffffc;
           table position = get bitmap(&read struct, s);
           table position <<=4;</pre>
           temp value += table position;
           temp value += load bitmap(&table[0x322], 4, &read struct);
         }
       } else {
         last bytes[0] = temp value = table position;
       }
       last bytes[0] = temp value+1;
```

```
}
/* nspack unpacking function end */
       if (last_bytes[0] > amount_unpacked_to_date) return point_in_table2;
       former_size +=2;
       if (!buffer_bounded(destination_point, dsize,
&destination_point[amount_unpacked_to_date], former_size) ||
        !buffer_bounded(destination_point, dsize,
        &destination_point[amount_unpacked_to_date - last_bytes[0]], former_size)
         ) {
        return 1;
       }
       do {
        destination point[amount unpacked to date] =
destination_point[amount_unpacked_to_date - last_bytes[0]];
        amount_unpacked_to_date++;
       } while (--former size && amount unpacked to date<dsize);
      point_in_table2 = (uint8_t) destination_point[amount_unpacked_to_date - 1];
       if (amount unpacked to date>=dsize) return 0;
    }
  }
}
```

```
This function is described in detail below.
```

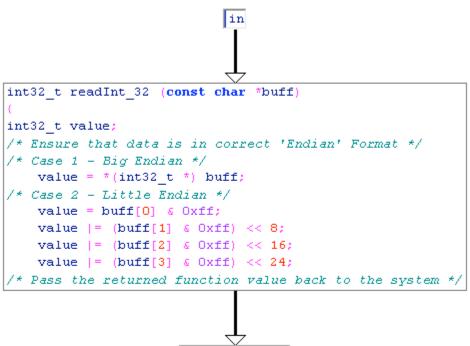


12.1.1. Functions used in packing routines

The following are functions/routines used in the following sections:

readInt_32

```
int32 t readInt 32 (const char *buff)
int32 t value;
/* Ensure that data is in correct 'Endian' Format */
/* Case 1 - Big Endian */
/* If the data is Big Endian - set the value this way */
CASE 1 (BE)
  value = *(int32 t *) buff;
/* case 1 is not likely to apply in NsPack, \  \  \star/
/* but we should check to be sure */
/* Case 2 - Little Endian */
/* If the data is Big Endian - set the value this way */
CASE 2 - LE (expected)
   value = buff[0] & 0xff;
    value |= (buff[1] & 0xff) << 8;</pre>
    value |= (buff[2] & 0xff) << 16;</pre>
    value |= (buff[3] & 0xff) << 24;</pre>
/* We are processing the data as 'Big Endian' */
/\,\star\, So we want to reverse the format that we expect \,\star/\,
/* As the buffer is read into the system */
/* Pass the returned function value back to the system */
   return value;
}
/* This function reads the data - a 32bit word and returns ^{\star/}
/* it to the system in Big Endian format*/
/* for the standard data stream in an Intel Little Endian */ /* system, we will reverse
the order of the data for ^{\star/}
/* processing */
```



return value;

cli_context

Based on the libGDS library

See: <u>http://libgds.info.ucl.ac.be/</u> http://libgds.info.ucl.ac.be/doc/html/cli ctx 8c-source.html

Structure – DeNSP

The following structure used in this document relates to the results of the 'read_struct' function.

struct DeNSP read struct;

buffer_bounded

This function ensures that the second buffer is contained within the first (i.e. buffer2 is contained inside buffer1).

buffer_bounded(buffer1, buffer1_size, buffer2, buffer2_size);

```
/*
This function is a check routine to ensure that a buffer does not cause an overrun.
If data is returned that exceeds the initial buffer being processed, the function will
exit instead of writing past the end of the first buffer.
* /
    ( buffer1_size > 0
    buffer2_size > 0
                                                           & &
                                                                                     \

    buffer2_size > 0

    buffer2_size <= buffer1_size</td>

    &&

    &

                                                                                     1
                                                           & &
                                                                             \backslash
                                                                             buffer2 + buffer2_size <= buffer1 + buffer1_size && \
buffer2 + buffer2_size > buffer1 );
/*The function checks the 2 buffers, buffer1 and buffer2 with respective size (length):
                buffer1 buffer1 size
                buffer2 buffer2 size
        This is done to ensure that buffer (when an additional amount of data is added) is
larger or equal in length to the buffer it is being compared to (or that one buffer does
not exceed the other buffer that is being written over).
The function returns 'TRUE' logically if all of the following conditions are met:
                 buffer1 has a size of > 0
                buffer2 has a size of > 0
All values of the function have to be logically 'TRUE' for the function to return a value
of 'TRUE'.
```

Basically a bounds check to stop security and other errors. $\star/$

load_bitmap

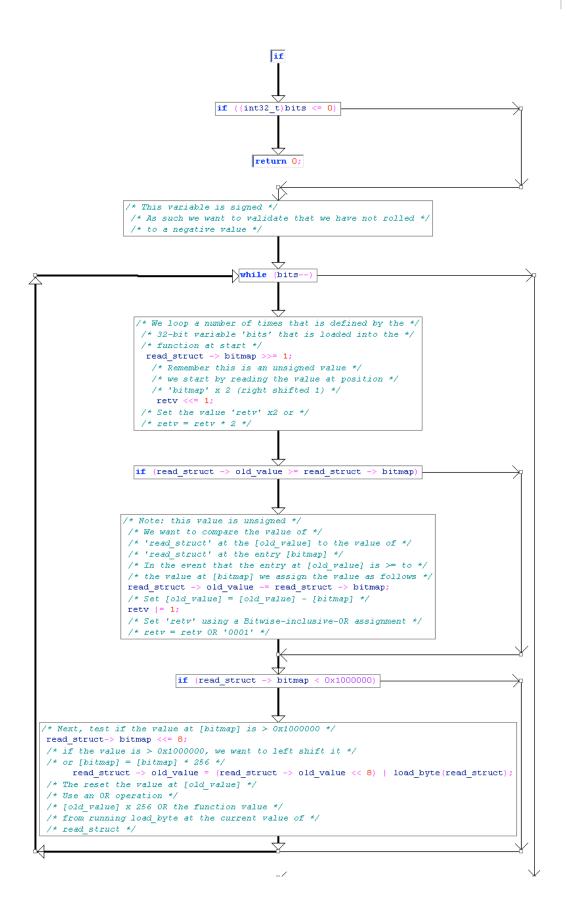
Load the data as a bitmapped variable.

```
uint32_t load_bitmap(struct DeNSP *read_struct, uint32_t bits) {
```

```
uint32_t retv = 0;
```

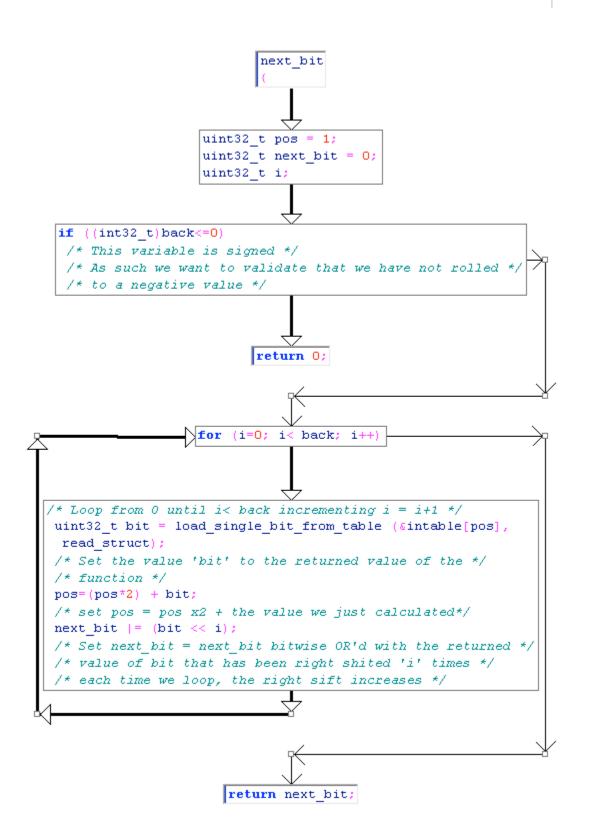
```
/*
read struct(self, struct)
read struct(struct)
                               --> Read structure from file into memory.
                                      This loads the file into memory from
                                       disk without executing it.
retv is a 32bit variable used as a marker and is initially set = 0.
bits is the 32 bit value at the point of the data we have read into the function.
* /
if ((int32 t)bits <= 0)
       return 0;
       /* This variable is signed */
        /\,{}^{\star} As such we want to validate that we have not rolled {}^{\star}/
        /* to a negative value */
       while (bits--) {
        /* We loop a number of times that is defined by the */
       /* 32-bit variable 'bits' that is loaded into the */
        /* function at start */
               read_struct -> bitmap >>= 1;
                       /* Remember this is an unsigned value */
                       /\,\star we start by reading the value at position \,\star/\,
                       /* 'bitmap' x 2 (right shifted 1) */
    retv <<= 1;
       /* Set the value 'retv' x2 or */
       /* retv = retv * 2 */
    if (read struct -> old value >= read struct -> bitmap) {
        /* Note: this value is unsigned */
        /* We want to compare the value of */
       /* 'read_struct' at the [old_value] to the value of */
       /* 'read struct' at the entry [bitmap] */
        /* In the event that the entry at [old_value] is >= to */
       /* the value at [bitmap] we assign the value as follows */
       read_struct -> old_value -= read_struct -> bitmap;
       /* Set [old_value] = [old_value] - [bitmap] */
       retv |= 1;
        /* Set 'retv' using a Bitwise-inclusive-OR assignment */
        /* retv = retv OR '0001' */
        1
    if (read struct -> bitmap < 0x100000) {
        /* Next, test if the value at [bitmap] is > 0x1000000 */
       read struct-> bitmap <<= 8;</pre>
        /* if the value is > 0x1000000, we want to left shift it */
        /* or [bitmap] = [bitmap] * 256 */
      read struct -> old value = (read struct -> old value << 8) |
```

```
load_byte(read_struct);
    /* The reset the value at [old_value] */
    /* Use an OR operation */
    /* [old_value] x 256 OR the function value */
    /* from running load_byte at the current value of */
    /* read_struct */
    }
}
return retv;
    /* Output the new value of retv */
}
return 0;
```



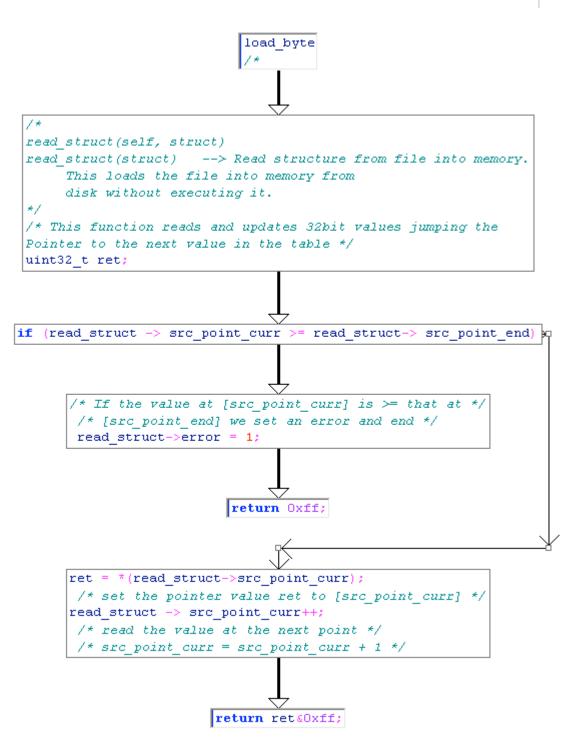
next_bit

```
uint32 t next bit (uint16 t *intable, uint32 t back, struct DeNSP *read struct)
{
/*
read struct(self, struct)
read_struct(struct)
                               --> Read structure from file into memory.
                                    This loads the file into memory from
                                      disk without executing it.
'*in table' is a pointer referencing the position in the data
ssize is the 32 bit value at the point of the data we have read into the function.
*/
/* start counting from 1 - we want to read in 0x100 bits */
/* FF or 256 decimal*/
uint32 t pos = 1;
uint32_t next_bit = 0;
uint32 t i;
if ((int32 t)back<=0)
       /* This variable is signed */
       /\,{}^{\star} As such we want to validate that we have not rolled {}^{\star}/
       /* to a negative value */
       return 0;
for (i=0; i< back; i++) {
       /* Loop from 0 until i< back incrementing i = i+1 */
       uint32 t bit = load single bit from table (&intable[pos],
               read struct);
       /* Set the value 'bit' to the returned value of the */
       /* function */
       pos=(pos*2) + bit;
       /* set pos = pos x2 + the value we just calculated*/
       next bit |= (bit << i);</pre>
       /* Set next_bit = next_bit bitwise OR'd with the returned */
       /* value of bit that has been right shited 'i' times */
       /\,\star\, each time we loop, the right sift increases \,\star/\,
       }
return next bit;
       /* Output the new value 'next_bit' */
}
```



load_byte

```
uint32 t load byte(struct DeNSP *read struct) {
/*
read struct(self, struct)
read struct(struct)
                               --> Read structure from file into memory.
                                     This loads the file into memory from
                                      disk without executing it.
*/
/\star This function reads and updates 32bit values jumping the
Pointer to the next value in the table */
uint32 t ret;
if (read_struct -> src_point_curr >= read_struct-> src_point_end) {
       /* If the value at [src point curr] is >= that at */
       /* [src point end] we set an error and end */
       read struct->error = 1;
       return 0xff;
}
ret = *(read struct->src point curr);
       /* set the pointer value ret to [src_point_curr] */
read_struct -> src_point_curr++;
       /\,\star\, read the value at the next point \,\star/\,
       /* src_point_curr = src_point_curr + 1 */
return ret&0xff;
       /\,\star\, return ret after we have \,\star/\,
       /* cleared the values of ret other than the last 256 bits */
}
```

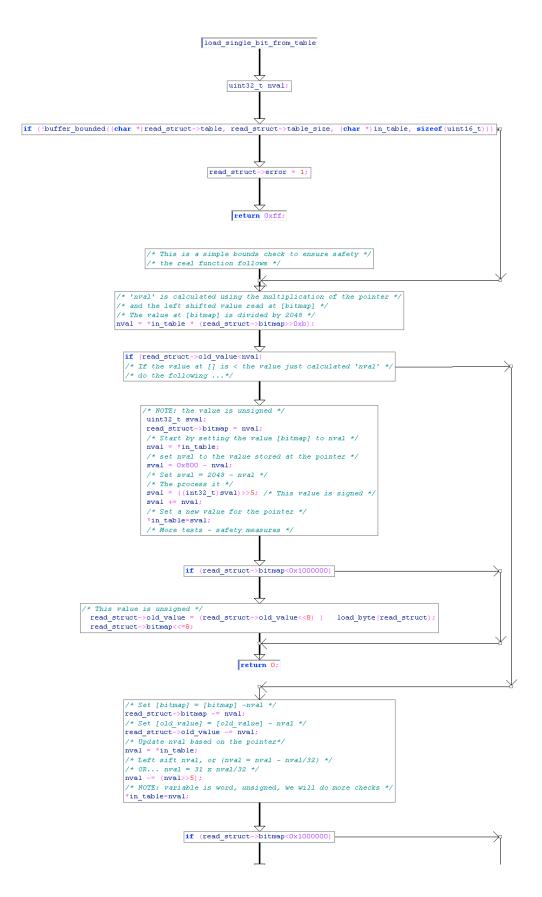


load_single_bit_from_table

int load_single_bit_from_table(uint16_t *in_table, struct DeNSP *read_struct) {

```
/*
read struct(self, struct)
read_struct(struct)
                              --> Read structure from file into memory.
                                     This loads the file into memory from
                                      disk without executing it.
'*in table' is a pointer referencing the position in the data
ssize is the 32 bit value at the point of the data we have read into the function.
* /
/*
There are 2 real parts to this function. Basically, the function reads, swaps and
processes values using marker variables and pointers. As the values are unsigned, the
function also has a check routine (security, function etc).
*/
uint32 t nval;
if (!buffer bounded((char *)read struct->table, read struct->table size, (char
*)in table, sizeof(uint16 t)))
{
       read struct->error = 1;
       return 0xff;
/* This is a simple bounds check to ensure safety ^{\star/}
/* the real function follows */
}
/* 'nval' is calculated using the multiplication of the pointer */
/* and the left shifted value read at [bitmap] */
/* The value at [bitmap] is divided by 2048 */
nval = *in table * (read struct->bitmap>>0xb);
if (read struct->old_value<nval)
/* If the value at [] is < the value just calculated 'nval' */
/* do the following \ldots*/
/* NOTE: the value is unsigned */
       uint32 t sval;
       read struct->bitmap = nval;
       /* Start by setting the value [bitmap] to nval */
       nval = *in_table;
       /* set nval to the value stored at the pointer */
       sval = 0x800 - nval;
       /* Set sval = 2048 - nval */
       /* The process it */
       sval = ((int32_t)sval)>>5; /* This value is signed */
       sval += nval;
       /* Set a new value for the pointer */
       *in table=sval;
       /* More tests - safety measures */
       if (read struct->bitmap<0x1000000) {
       /* This value is unsigned */
               read struct->old value = (read struct->old value<<8) |</pre>
       load_byte(read struct);
              read struct->bitmap<<=8;</pre>
        }
```

```
return 0;
}
/* Set [bitmap] = [bitmap] -nval */
read_struct->bitmap -= nval;
/* Set [old_value] = [old_value] - nval */
read struct->old value -= nval;
/* Update nval based on the pointer*/
nval = *in_table;
/* Left sift nval, or (nval = nval - nval/32) */
/* OR... nval = 31 x nval/32 */
nval -= (nval >>5);
/* NOTE: variable is word, unsigned, we will do more checks ^{\star/}
*in table=nval;
if (read struct->bitmap<0x100000)
/* More security checks as the value is unsigned */
       read struct->old_value = (read_struct->old_value<<8) |</pre>
       load byte(read struct);
       read struct->bitmap<<=8;</pre>
}
return 1;
}
```



load_100_bits_from_table

This function loads 0x100 (or 256) bits from the table.

uint32_t load_100_bits_from_table(uint16_t *in_table, struct DeNSP *read_struct, uint32_t ssize) {

```
/*
read struct(self, struct)
                             --> Read structure from file into memory.
read struct(struct)
                                    This loads the file into memory from
                                     disk without executing it.
'*in table' is a pointer referencing the position in the data
ssize is the 32 bit value at the point of the data we have read into the function.
*/
/* start counting from 1 - we want to read in 0x100 bits */
/* FF or 256 decimal*/
uint32 t count = 1;
/* Run once before looping*/
       uint32 t left position, table position;
/* define marker variables (left position and table position) */
/* These are used to read information while shifting information */
       /* Clear the last 256 bit value of ssize */
       left position = ssize&0xff;
       /* Clear the values of ssize other than the last 256 bits */
       /* variable that has been AND'd with 0xff in order to */
       /* clear the right most 256 bits. */
       /* this is: */
       /* Binary mask the right most 256 bits (leave left most ok) */
       /* Right shift 1 - that is double left position */
       /* Bitwise 'OR' the values*/
       ssize=(ssize & 0xffffff00) | ((left_position<<1)&0xff);</pre>
       /* Right shift the value 'left position' seven times*/
       /* or left_position = left_position / 128*/
       left position>>=7;
       /* Set the variable 'table position' as equalling the value */
       /* 'left position' plus 1 */
       table position = left position+1;
       /* Left shift the variable eight times */
       /* table position = Table position x 256 */
       table position<<=8;</pre>
       /* Set 'table position += count' */
       /* Add Table position and the value at count*/
       /* Set the new value of table position to this value */
       table_position = table_position + count;
       /* Load a new value from the file as it was read into memory */
       /* and stored in an array */
table position = load single bit from table (&in table[table position], read struct);
       /* Set the new value of count as equal to double the previous */
```

```
/* value 'OR'd with the new value just loaded */
        count=(count*2)|table position;
        /* Check if the variable 'left position' is not equal to */
        /* 'table_position'. If true, loop, otherwise end. */
/* The loop runs until these 2 values are equal */
        if (left position!=table position) {
        /* The next loop occurs in a different way to the first ^{\star/}
        /* This loop has to run until the value 'count' is less */
        /* than 256 or 0x100*/
                while (count<0x100)
        /* Process the new value of count and load the values from */
        /\star the array that is associated with the value of the file \star/
        /* as read into memory and stored in the array */
                 count = (count*2)|load_single_bit_from_table(&in_table[count],
read struct);
    }
  }
        /* Reset the variable count */
        /* Adding 255 to the value of count effectively sets */
        /* the returned value back to 0 */
       return count&0xff;
}
```

```
START

CODE

I F

FALSE

CODE

WHILE

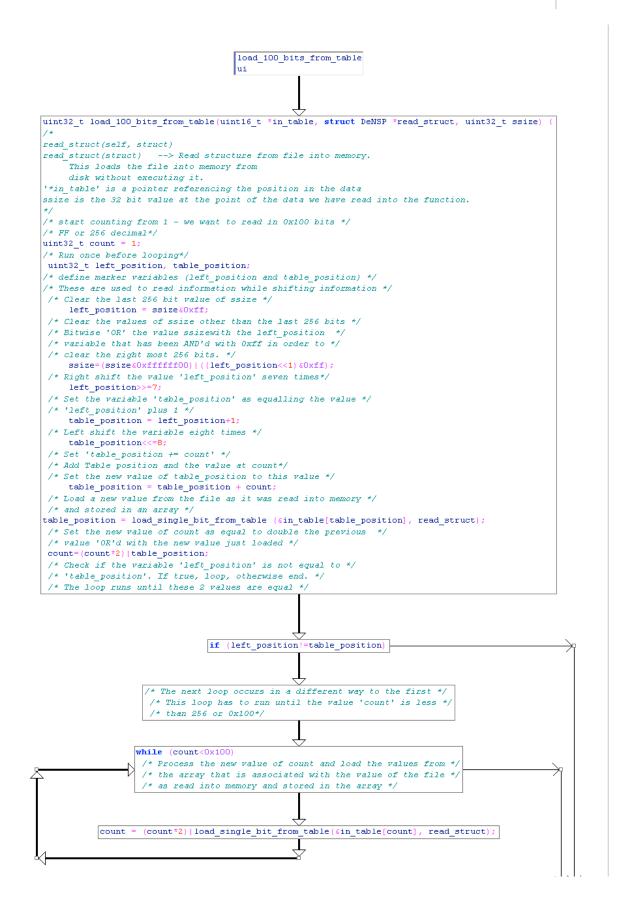
FALSE

TRUE

CODE

TRUE

CODE
```



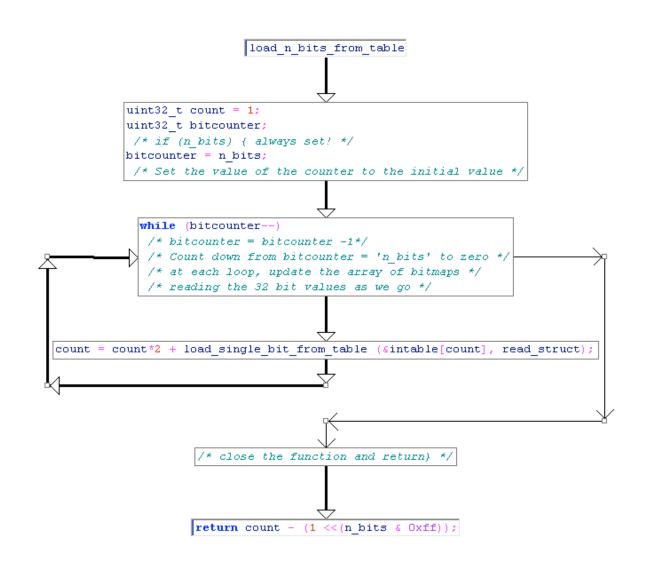
load_n_bits_number_of_bits_from_table

This function allows a variable number of bits (n) to be loaded from the table.

uint32_t load_n_bits_from_table(uint16_t *intable, uint32_t n_bits, struct UNSP
*read_struct) {

```
/*
read struct(self, struct)
read_struct(struct) --> Read structure from file into memory.
                                     This loads the file into memory from
                                     disk without executing it.
'*in table' is a pointer referencing the position in the data
'n bits' is the 32 bit value that determines how many loops this function makes.
*/
/* start counting from 1 - we want to read in 'n bits' \# of bits */
uint32_t count = 1;
uint32 t bitcounter;
       /* if (n bits) { always set! */
bitcounter = n bits;
       /* Set the value of the counter to the initial value */
while (bitcounter--)
       /* bitcounter = bitcounter -1*/
       /* Count down from bitcounter = 'n bits' to zero */
       /* at each loop, update the array of bitmaps */
       /\,\star\, reading the 32 bit values as we go \,\star/\,
count = count*2 + load single bit from table (&intable[count], read struct);
       /* close the function and return} */
return count - (1 <<(n bits & 0xff));
       /* The value returned is count - a right sifted value */
       /* The right sift takes the value 1 and right sifts it */
       /\ast based on the initial value 'n bits' selected with \ast/
       /* an 'AND' operation to clear the values of 'n_bits' other */
       /* then the last 256 bits */
```

}

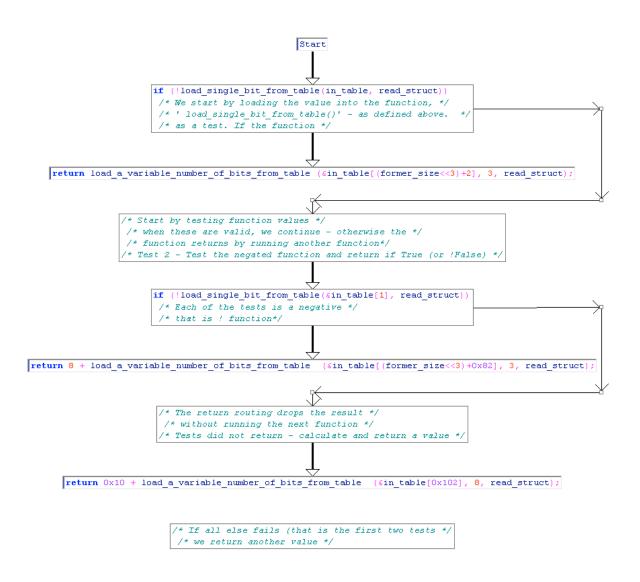


load_a_variable_number_of_bits_from_table

This function allows a variable number of bits (n) to be loaded from the table.

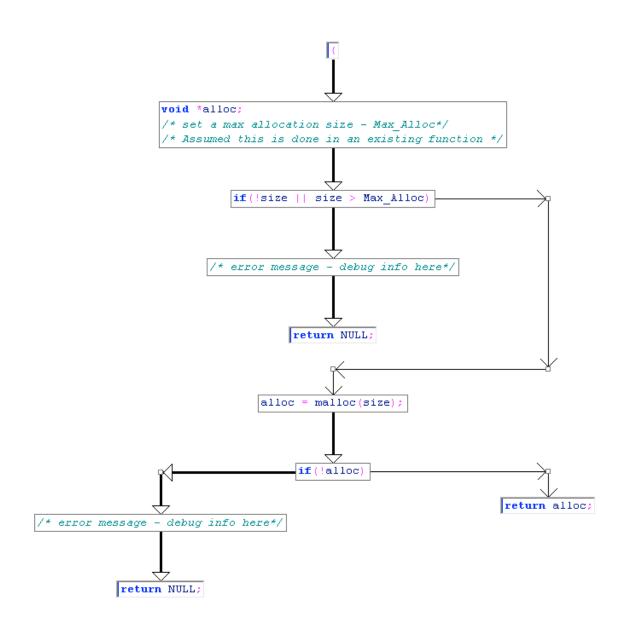
uint32_t load_a_variable_number_of_bits_from_table(uint16_t *in_table, struct DeNSP
*read_struct, uint32_t former_size) {

```
/*
read struct(self, struct)
read struct(struct)
                              --> Read structure from file into memory.
                                    This loads the file into memory from
                                     disk without executing it.
'*in table' is a pointer referencing the position in the data
former size is the 32 bit value at the point of the data we have read into the function.
*/
/* Test 1 - Test the negated function and return if True (or !False) */
 if (!load single bit from table(in table, read struct))
       /* We start by loading the value into the function, */
       /* ' load_single_bit_from_table()' - as defined above. */
       /* as a test. If the function */
   return load a variable number of bits from table (&in table[(former size<<3)+2], 3,
read struct);
       /* Start by testing function values */
       /* when these are valid, we continue - otherwise the */
       /* function returns by running another function*/
/* Test 2 - Test the negated function and return if True (or !False) */
 if (!load_single_bit_from_table(&in_table[1], read_struct))
       /* Each of the tests is a negative */
       /* that is ! function*/
    return 8 + load a variable number of bits from table
       (&in_table[(former_size<<3)+0x82], 3, read_struct);
       /* The return routing drops the result */
       /* without running the next function */
/* Tests did not return - calculate and return a value */
 return 0x10 + load a variable number of bits from table
                                                            (&in table[0x102], 8,
read struct);
       /* If all else fails (that is the first two tests */
       /\,\star we return another value \,\star/\,
}
```



check_malloc

```
void *check malloc(size t size)
{
      void *alloc;
/* set a max allocation size - Max_Alloc */
/* Assumed this is done in an existing function */
/* This is a catch to ensure that memory is not exhausted */
   if(!size || size > Max Alloc) {
size_tma
/* error message - debug info here*/
      return NULL;
   }
   alloc = malloc(size);
   if(!alloc) {
/* error message - debug info here*/
/* If the function cannot allocate sufficient memory or if the buffer */
/* will cause an overflow, return an error and exit gracefully */
      return NULL;
   } else return alloc;
```

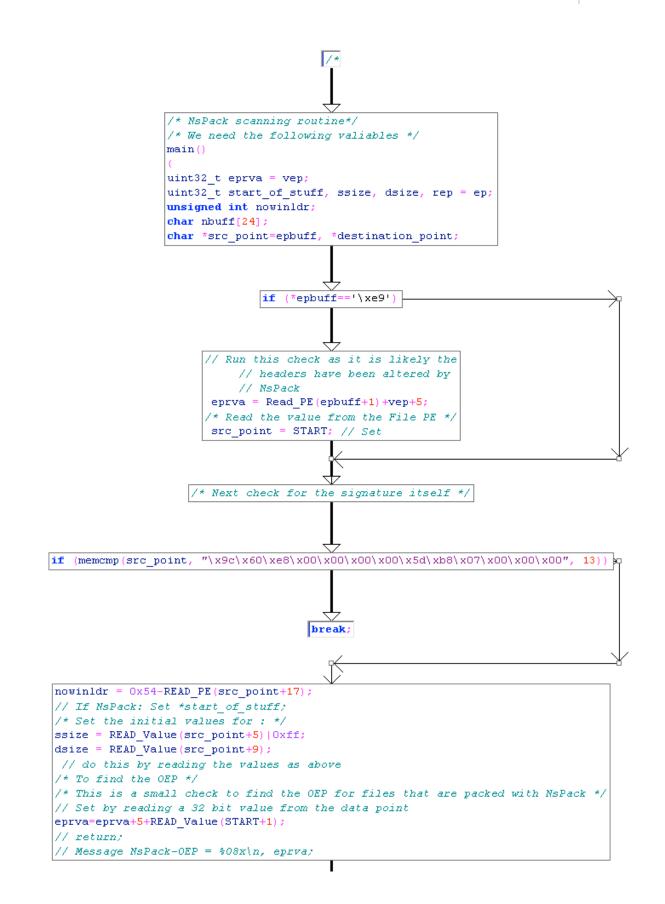


The Determination and call to the Unpacking Function

We define some of the values used in unpacking in the scanning of NsPack in the first instance.

Here we find the values in the file that we use in the unpacking sections (below).

```
/* NsPack scanning routine*/
/* We need the following valiables */
uint32_t eprva = vep;
uint32 t start of stuff, ssize, dsize, rep = ep;
unsigned int nowinldr;
char nbuff[24];
char *src point=epbuff, *destination point;
if (*epbuff=='\xe9') { // Run this check as it is likely the
                                 // headers have been altered by
                                     // NsPack
       eprva = Read PE(epbuff+1)+vep+5;
/* Read the value from the File PE */
       src point = START; // Set
}
/* Next check for the signature itself */
if (memcmp(src_point, "\x9c\x60\xe8\x00\x00\x00\x00\x5d\xb8\x07\x00\x00\x00\x00", 13)) break;
nowinldr = 0x54-READ PE(src point+17);
// If NsPack: Set *start of stuff;
/* Set the initial values for : */
ssize = READ Value(src point+5)|0xff;
dsize = READ_Value(src_point+9);
       // do this by reading the values as above
/* To find the OEP */
/* This is a small check to find the OEP for files that are packed with NsPack */
// Set by reading a 32 bit value from the data point
eprva=eprva+5+READ Value(START+1);
// return;
// Message NsPack-OEP = %08x\n, eprva;
```



The Unpacking Function itself (nspack_unpacking_function)

uint32_t nspack_unpacking_function(uint16_t *table, uint32_t table_size, uint32_t table_rem, uint32_t allocsize, uint32_t initial_byte, char *src_point, uint32_t ssize, char *destination_point, uint32_t dsize) {

/* Read in the data */

```
/*
read struct(self, struct)
                              --> Read structure from file into memory.
read_struct(struct)
                                  This loads the file into memory from
                                     disk without executing it.
'*in table' is a pointer referencing the position in the data
ssize is the 32 bit value at the point of the data we have read into the function.
*/
struct DeNSP read struct;
/* Start by setting 'i'. This is a */
/* Clear the values of (allocsize+table rem)other than the last 256 bits */
/* We multiply 768 by the last values (\overline{0}\text{-}255\text{)} calculated and add 1846*/
uint32 t i = (0x300<<((allocsize+table rem)&0xff)) + 0x736;
/* Initialise the variables - we start with nothing unpacked. \star/
/* We are doing a shift operation */
/* These are used to hold the prior values of the data as we conduct ^{\star/}
/* swaps */
uint32 t last bit = 0;
uint32_t amount_unpacked_to_date = 0;
// These values have been changes from that which is included in a comment
// below. The array is used for bitshift operations and other bitwise
// calculations.
                             // We use a 4 element array of 32 bit values
uint32_t last_bytes[4];
                      \ensuremath{{//}} to manipulate the data section that we
                      // read from the uncompressed file.
/* The array is initialised with an initial value of 0x0001 */
for (i=0; i<4; ++i) last bytes[i] = 1;</pre>
/*
                                                           // last bytes[0];
uint32_t former_bytes_value = 1;
* /
/* Initialise the variables - and the point is at the start of the data. */
uint32 t point in table1 = 0;
uint32 t point in table2 = 0;
/* */
uint32_t put = (1<<(allocsize&0xff))-1;</pre>
/* Set the initial start value as 0x01 left shifted */
/* This value is calculated outside of this function and is given as ^{\prime}
/* input */
initial byte = (1<<(initial byte&0xff))-1;</pre>
```

```
/* We need to check that we do not exceed the bounds */
if (table size < i*sizeof(uint16 t)) return 2;
/* initialise the table and prep it */
/* this is the array of memory to process the decompression */
while (i) table [--i] = 0x400;
       read_struct.error = 0;
       read struct.old value = 0;
       read struct.src point curr = src point;
       read struct.bitmap = 0xfffffff;
       read_struct.src_point_end = src_point + ssize;
       read struct.table = (char *)table;
       read_struct.table_size = table_size;
/* */
for ( i = 0; i<5 ; i++) read_struct.old_value = (read_struct.old_value<<8) |</pre>
get byte(&read struct);
if (read struct.error) return 1;
/* if (!dsize) return 0; - check to ensure valid*/
/* Check for exceptions etc. */
/* loop to unpack the code from the compressed data*/
while (1) {
   uint32_t former_size = initial_byte & amount_unpacked_to_date;
   uint32 t table position;
   uint32_t temp_value = point_in_table1;
    if (read struct.error) return 1;
/* We need to check once per mainloop for errors and exceptions */
/* Not a part of the decompression itself, but still needed */
    if (!load single bit from table(&table[(point in table1<<4) + former size],
&read struct)) {
    /* Check that jumps to one function if true or processes differently */
    /* if not found
                    */
      // We start with setting a shift variable used in the process
      uint32_t shft = 8 - (table_rem&0xff);
      shft &= 0xff;
                                     // We only want the last bits
      /* These values are used to Right Shift values */
      /* these operations change the 32 bit value 'table position' that
      /* is used to store values in the data
      table position = (point in table2>>shft) +
((put&amount unpacked to date) << (table rem&0xff));
      table position *=3;
      table position<<=8;</pre>
      /* Next, remember that these values (below) are signed */
      if ((int32_t)point_in_table1>=4)
                                                     // signed
            if ((int32 t)point in table1>=0xa)
                                             // signed
       {
         point in table1 -= 6;
       } else {
         point in table1 -= 3;
       }
      //Here is the alternate run if the first test value if found
      else
      {
              point in table1=0;
      }
      if (last bit) {
       if (!buffer bounded(destination point, dsize,
&destination point[amount unpacked to date - last bytes[0]], 1)) return 1;
       ssize = (ssize&0xffffff00) | (uint8_t)destination_point[amount_unpacked_to_date -
```

```
last bytes[0]];
       point in table2 = load 100 bits from tablesize(&table[table position+0x73],
&read struct, ssize);
       last bit=0;
      } else {
       point in table2 = load 100 bits from tablesize(&table[table position+0x736],
&read struct);
      }
/* At this point we unpack a single byte of data */
/* this is repeated many times */
/* We start by doing some bounds checks */
      if (!buffer bounded(destination point, dsize, &
destination_point[amount_unpacked_to_date], 1))
       return 1;
      destination point[amount unpacked to date] = point in table2;
      amount_unpacked_to_date++;
/* Check bounds */
      if (amount unpacked to date>=dsize) return 0;
      continue;
      } else {
     point in table2 = last bit = 1;
      if (load_single_bit_from_table(&table[point_in_table1+0xc0], &read_struct)) {
       if (!load single bit from table(&table[point in table1+0xcc], &read struct)) {
         table_position = point_in_table1+0xf;
         table_position <<=4;</pre>
         table position += former size;
         if (!load_single_bit_from_table(&table[table_position], &read_struct)) {
           if (!amount unpacked to date) return point in table2;
           point in table1 = 2*((int32 t)point in table1>=7)+9; /* Note: we are using a
signed value */
           if (!buffer bounded(destination point, dsize,
&destination point[amount unpacked to date - last bytes[0]], 1)) return 1;
           point in table2 = (uint8_t) destination_point[amount_unpacked_to_date -
last bytes[0]];
           /* unpack one byte - real */
           destination point[amount unpacked to date] = point in table2;
           amount unpacked to date++;
           if (amount unpacked to date>=dsize) return 0;
           continue;
         } else {
           former size = load a variable number of bits from table(\&table[0x534],
&read struct, former size);
           point_in_table1 = ((int32 t)point in table1>=7); /* signed */
           point in table1 = ((point in table1-1) & 0xffffffd)+0xb;
           /* jmp checkloop and backcopy (uses edx) */
         } /* gotbit_uno ends */
       } else { /* gotbit_due */
         if (!load_single_bit_from_table(&table[point_in_table1+0xd8], &read_struct)) {
           table position = last bytes[1];
         } else {
           if (!load single bit from table(&table[point in table1+0xe4], &read struct)) {
             table position = last bytes[2];
           } else {
             table position = last bytes[3];
             last bytes[3] = last bytes[2];
           last_bytes[2] = last_bytes[1];
         }
         last bytes[1] = last bytes[0];
         last bytes[0] = table position;
```

```
former size = load a variable number of bits from table(&table[0x534],
&read struct, former size);
         point_in_table1 = ((int32_t)point_in_table1>=7);
/* Value used is signed */
        point_in_table1 = ((point in table1-1) & 0xffffffd)+0xb;
         /* jmp checkloop_and_backcopy (uses edx) */
       }
      } else {
/\,{}^{\star} Here we swap the stored values repeatedly {}^{\star}/
/* The values in the table are cycled as we add new ones to process */
       last_bytes[3] = last_bytes[2];
       last bytes[2] = last bytes[1];
       last bytes[1] = last bytes[0];
       point in table1 = ((int32_t)point_in_table1>=7);
       point_in_table1 = ((point_in_table1-1) & 0xffffffd)+0xa;
       former_size = load_a_variable_number_of_bits_from_table(&table[0x332],
&read struct, former size);
       table_position = ((int32_t)former_size>=4)?3:former_size;
       table_position<<=6;</pre>
       table position = load n bits from table(&table[0x1b0+table position], 6,
&read struct);
       if (table position>=4) {
         uint32 t s = table position;
         s>>=1;
         s--;
         temp value = (table position & point in table2) | 2;
         temp value<<=(s&0xff);</pre>
         if ((int32 t)table position<0xe) {
           temp_value += load_bitmap(&table[(temp_value-table_position)+0x2af], s,
&read struct);
         } else {
           s += 0xffffffc;
           table position = get bitmap(&read struct, s);
           table_position <<=4;</pre>
           temp value += table position;
           temp_value += load_bitmap(&table[0x322], 4, &read_struct);
         }
       } else {
          /* gotbit out1 */
         last_bytes[0] = temp value = table position;
       }
       /* gotbit out2 */
       last bytes[0] = temp value+1;
       /* jmp checkloop_and_backcopy (makes use of EDX) */
      }
      /* checkloop and backcopy */
      if (!last bytes[0]) return 0;
/* nspack unpacking function end */
      if (last bytes[0] > amount unpacked to date) return point in table2;
      former size +=2;
      if (!buffer bounded(destination point, dsize,
&destination_point[amount_unpacked_to_date], former_size) ||
         !buffer bounded(destination point, dsize,
&destination_point[amount_unpacked_to_date - last_bytes[0]], former_size)
         ) {
       return 1;
```

```
}
do {
    destination_point[amount_unpacked_to_date] =
destination_point[amount_unpacked_to_date - last_bytes[0]];
    amount_unpacked_to_date++;
    } while (--former_size && amount_unpacked_to_date<dsize);
    point_in_table2 = (uint8_t) destination_point [amount_unpacked_to_date - 1];
    if (amount_unpacked_to_date>=dsize) return 0;
    }
/* while true ends */
}
```

Basically, the function does a series of reads and shift operations based on the previously listed and detailed functions.

The diagram below is complex, but does demonstrate this flow.

*/

Upcoming SANS Training Click Here for a full list of all Upcoming SANS Events by Location \mathbf{n}

SANS Secure Singapore 2013	Singapore, SG	Feb 25, 2013 - Mar 02, 2013	Live Event
SANS 2013	Orlando, FLUS	Mar 08, 2013 - Mar 15, 2013	Live Event
SEC573 Python for Pen Testers	Washington, DCUS	Mar 18, 2013 - Mar 22, 2013	Live Event
SANS Secure Canberra 2013	Canberra, AU	Mar 18, 2013 - Mar 23, 2013	Live Event
What Works in Cyber Threat Intelligence Summit	Washington, DCUS	Mar 22, 2013 - Mar 22, 2013	Live Event
SANS Monterey 2013	Monterey, CAUS	Mar 22, 2013 - Mar 27, 2013	Live Event
SANS Abu Dhabi 2013	Abu Dhabi, AE	Mar 23, 2013 - Mar 28, 2013	Live Event
SANS Delhi 2013	New Delhi, IN	Apr 01, 2013 - Apr 12, 2013	Live Event
SANS Northern Virginia 2013	Reston, VAUS	Apr 08, 2013 - Apr 13, 2013	Live Event
SANS Secure Europe 2013	Amsterdam, NL	Apr 15, 2013 - Apr 27, 2013	Live Event
SANS Cyber Guardian 2013	Baltimore, MDUS	Apr 15, 2013 - Apr 20, 2013	Live Event
Management 442- BETA	Washington, DCUS	Apr 19, 2013 - Apr 20, 2013	Live Event
AppSec 2013	Austin, TXUS	Apr 22, 2013 - Apr 27, 2013	Live Event
SANS CyberCon 2013	Online, VAUS	Apr 22, 2013 - Apr 27, 2013	Live Event
SANS CDK Seoul 2013	Seoul, KR	Apr 22, 2013 - Apr 27, 2013	Live Event
Critical Security Controls International Summit	London, GB	Apr 26, 2013 - May 02, 2013	Live Event
SANS Secure India @Bangalore 2013	Bangalore, IN	Apr 29, 2013 - May 04, 2013	Live Event
SANS Security West 2013	San Diego, CAUS	May 07, 2013 - May 16, 2013	Live Event
SANS at IT Web Security Summit 2013	Johannesburg, ZA	May 09, 2013 - May 10, 2013	Live Event
SANS South Africa May 2013	Johannesburg, ZA	May 13, 2013 - May 25, 2013	Live Event
SANS Brisbane 2013	Brisbane, AU	May 13, 2013 - May 18, 2013	Live Event
RSA Conference 2013	OnlineCAUS	Feb 24, 2013 - Feb 25, 2013	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced