Malcode Context of API Abuse

GIAC (GREM) Gold Certification

Author: Ken Dunham, kend@kendunham.org
Advisor: Egan Hadsell

Accepted: April 4, 2011

Abstract

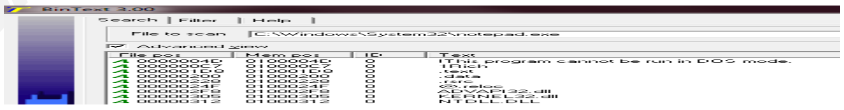
Malcode analysts regularly capture strings (sequence of bytes compiler generated) from memory to identify the language that a code may have been programmed in, downloader URLs, Windows application programming interface (API) calls related to functionality of the code and more. API research can be confusing, especially for a junior analyst who is still learning much about both malicious context and Windows internals. This paper focuses on the challenge understanding API abuse within a *malicious code context* to aid an analyst in research. A methodology is documented in this paper to efficiently and consistently capture ASCII and Unicode strings from malicious code for API string analysis. Recent top codes impacting enterprise and government sectors were used in the sample set analyzed in lab work on over 50 samples. Strings from over 600 bot samples were also analyzed for frequency of API occurrence and those commonly related to abuse. Results of common malcode API abuse research are included in a table at the end of this paper.

1. Introduction

Individuals performing a manual or deep research effort into understanding malicious code need to establish and understand the *malcode context* for success. For example, downloading by a program can be a normal function, such as locating updates for an application. Within the malcode context a downloader event may be related to updating a Trojan or installation of additional malicious payloads on an infected system. As a result, malcode analysts have the unique challenge of identifying how legitimate activities are abused for malicious means.

Establishing malcode context is a process rather than an endpoint. As an analyst works to understand possible maliciousness related to a code of interest static and dynamic analysis provides clues related to functionality. Experienced analysts quickly identify common malcode practices as malicious, such as unauthorized installation of files into the Windows System directory without any EULA, permissions, or notification to the end user.

When working with binaries a common static and dynamic analysis procedure is to capture "strings", sequential characters in ASCII and Unicode found within a compiled binary. Strings quickly reveal to an analyst if a program is a normal portal executable (PE) binary for Windows, such as is revealed via a BinText strings view of notepad.exe:



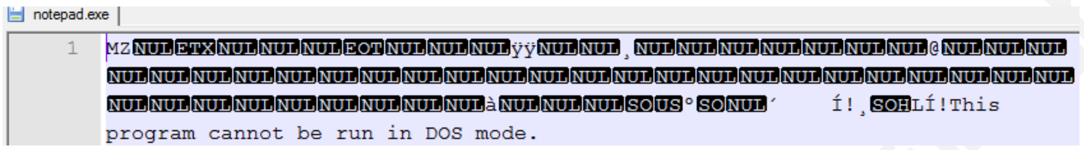
File to scan: C:\Windows\System32\notepad.exe

```
File pos    Mem pos     ID  Text
A 0000004D  0100004D    0   !This program cannot be run in DOS mode.
A 000000C7  010000C7    0   1Rich
A 000001D8  010001D8    0   .text
A 00000200  01000200    0   .data
A 00000228  01000228    0   .rsrc
A 0000024F  0100024F    0   @.reloc
A 000002F8  010002F8    0   ADVAPI32.dll
A 00000305  01000305    0   KERNEL32.dll
A 00000312  01000312    0   NTDLL.DLL
```

Ken Dunham, kend@kendunham.org

Notepad.exe strings reveals it is a Windows program.

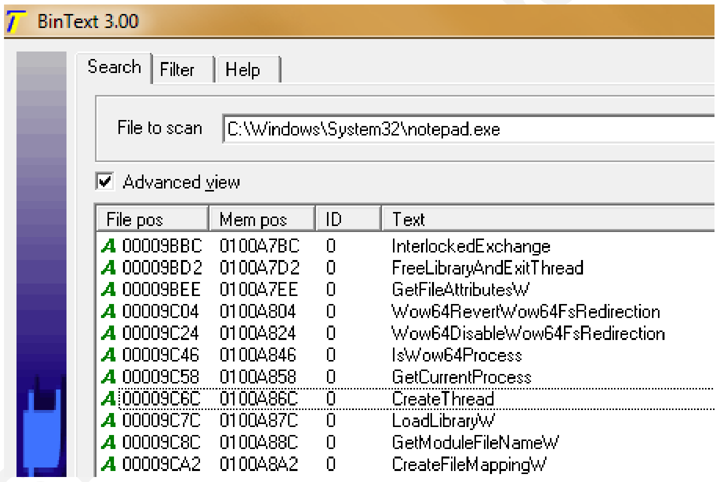
Analysts quickly see the "This program cannot be run in DOS mode." which is common for a PE file with an "MZ" header:



```
notepad.exe
MZNULETXNULNULNULEOTNULNULNULÿÿNULNUL¸NULNULNULNULNULNULNUL@NULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULàNULNULNULSOUS°SONUL´   Í!¸SOHLÍ!This
program cannot be run in DOS mode.
```

Notepad.exe header, viewed in Notepad ++, reveals it is a Windows binary.

A deeper analysis of strings reveals Windows application programming interface (API) strings, such as CreateThread, LoadLibrary, and other functions:



BinText 3.00

Search | Filter | Help

File to scan: C:\Windows\System32\notepad.exe

☑ Advanced view

	File pos	Mem pos	ID	Text
A	00009BBC	0100A7BC	0	InterlockedExchange
A	00009BD2	0100A7D2	0	FreeLibraryAndExitThread
A	00009BEE	0100A7EE	0	GetFileAttributesW
A	00009C04	0100A804	0	Wow64RevertWow64FsRedirection
A	00009C24	0100A824	0	Wow64DisableWow64FsRedirection
A	00009C46	0100A846	0	IsWow64Process
A	00009C58	0100A858	0	GetCurrentProcess
A	00009C6C	0100A86C	0	CreateThread
A	00009C7C	0100A87C	0	LoadLibraryW
A	00009C8C	0100A88C	0	GetModuleFileNameW
A	00009CA2	0100A8A2	0	CreateFileMappingW

API strings found in Notepad.exe.

API strings provide important clues related to possible functionality of code. In the notepad.exe example above, a CreateThread API (Microsoft Corp., 2007) exists within the code is able to create a thread to run in memory. Malcode Analysts review strings to look for APIs that might indicate other possible functions, such as downloading code, APIs that might be abused for concealing a file or process on a system, and more. This is part of how malcode context is created when analyzing malcode.

Experienced analysts are familiar with malcode context, even if it is not documented, and are able to quickly identify possible functionality of code to then evaluate or further research. For example, an analyst may find an API related to downloading and also find a URL within the code (static and/or dynamic analysis). This may then lead to behavioral analysis with a sniffer to see if any actions are taken related to the URL of interest. It may also result in more advanced behavioral analysis and reverse engineering tactics to trigger and analyze what is likely to be the functionality of code based upon various malcode context clues and findings to date.

The focus of this paper is to perform research on how to use freeware tools to efficiently and consistently capture strings of interest to then identify APIs in strings that may help establish malcode context. Several older and more recent codes were used as a way to identify how to best capture specific types of strings, with a specific length, to best perform research against APIs referenced in binaries.

Deep lab research was performed on over 50 different known malicious incidents, with known behavior, from 2010 as the sample set for identifying how APIs may be abused by malcode. Additionally, over 600 known malicious samples were analyzed for strings to identify the top prevalent API references commonly found within codes, thanks to a contribution by Kjell Christian Nilsen. All references to APIs were then researched and vetted out to identify those that are clearly linked to abuse by malicious code.

Results of this output are logged in a reference table as the final component of this research paper. Both inexperienced and experienced analysts may now use this table as a starting point for identifying what an API might be abused for when found in suspected malcode. The reference table of this research may be used in configuration files for programs like APISpy32, YARA signatures (Yara-project, 2011), and more.

2. Understanding Strings

On the surface, understanding strings is a trivial concept, a set of characters in a compiled binary that may be related to comments in code, URLs, API references, and more. To properly analyze and research malicious code based upon strings requires a lab qualified understanding to avoid potential pitfalls in strings analysis.

2.1. ASCII & Unicode Strings

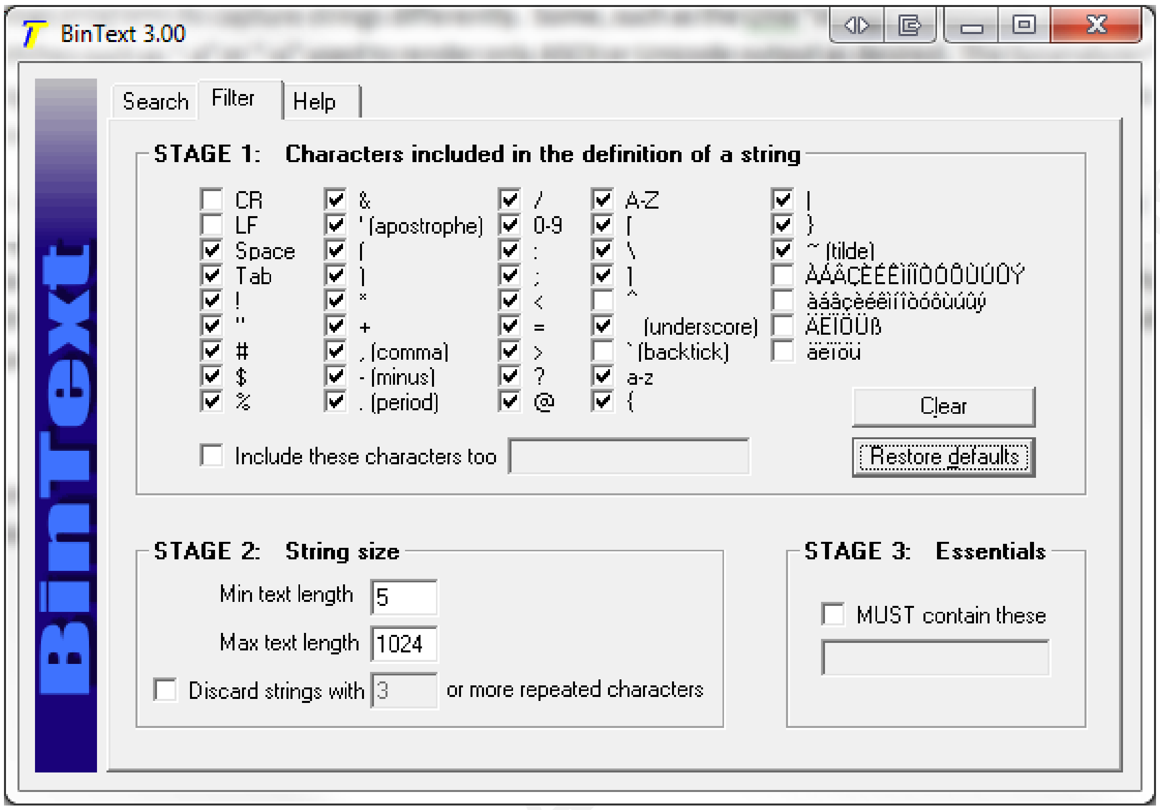
Strings of a compiled binary may contain both **ASCII** (American Standard Code for Information Interchange) and **Unicode** (world language) character sets. Older malicious codes tend to contain more ASCII only string output, where more recent codes have a combination of strings or are heavily Unicode based. Because of a number of caveats related to strings analysis capture of strings for samples of interest, in both ASCII and Unicode, is performed to best collect *all* possible strings of interest related to a hostile binary. For example, using the Unix "`strings`" command, both "-a" or "-u" may be used to capture each character set individually.

When reviewing APIs within a binary those that end with "A" are ASCII while those that end with "W" are Unicode. For example, "SetFileAttributesA" is an ASCII based API reference.

2.2. String Length

The length of strings that one may look for when looking for ASCII and Unicode characters varies, based upon a balance between "junk" or "garbage" output and highly sanitized longer string lengths. For example, if a minimum character length of 10 is selected a string named after an API call like ZwOpenKey would not be in the strings output as it contains just 9 characters. If the minimum is set too low, such as 1 or 2, gibberish is the result with a large number of strings that are meaningless. Low thresholds for strings results in a high noise-to-signal ratio, forcing an analyst to wade through a multitude of meaningless strings.

Most programs have a default strings output of 3, 4, or 5. BinText (McAfee, 2011), a popular Windows strings tool with a simple GUI, uses 5 as the default value for strings captures:



STAGE 1: Characters included in the definition of a string

[ ] CR	[✓] &	[✓] /	[✓] A-Z	[✓] |
[ ] LF	[✓] ' (apostrophe)	[✓] 0-9	[✓] [	[✓] }
[✓] Space	[✓] (	[✓] :	[✓] \	[✓] ~ (tilde)
[✓] Tab	[✓])	[✓] ;	[✓]]	[ ] ÀÂÃÇÈÉÊÌÎÏÒÔÕÙÚÛÝ
[✓] !	[✓] *	[✓] <	[ ] ^	[ ] àâãçèéêìîïòôõùúûý
[✓] "	[✓] +	[✓] =	[✓] _ (underscore)	[ ] ÄËÏÖÜß
[✓] #	[✓] , (comma)	[✓] >	[ ] ` (backtick)	[ ] äëïöü
[✓] $	[✓] - (minus)	[✓] ?	[✓] a-z	
[✓] %	[✓] . (period)	[✓] @	[✓] {	

[ ] Include these characters too

STAGE 2: String size

Min text length 5

Max text length 1024

[ ] Discard strings with 3 or more repeated characters

STAGE 3: Essentials

[ ] MUST contain these

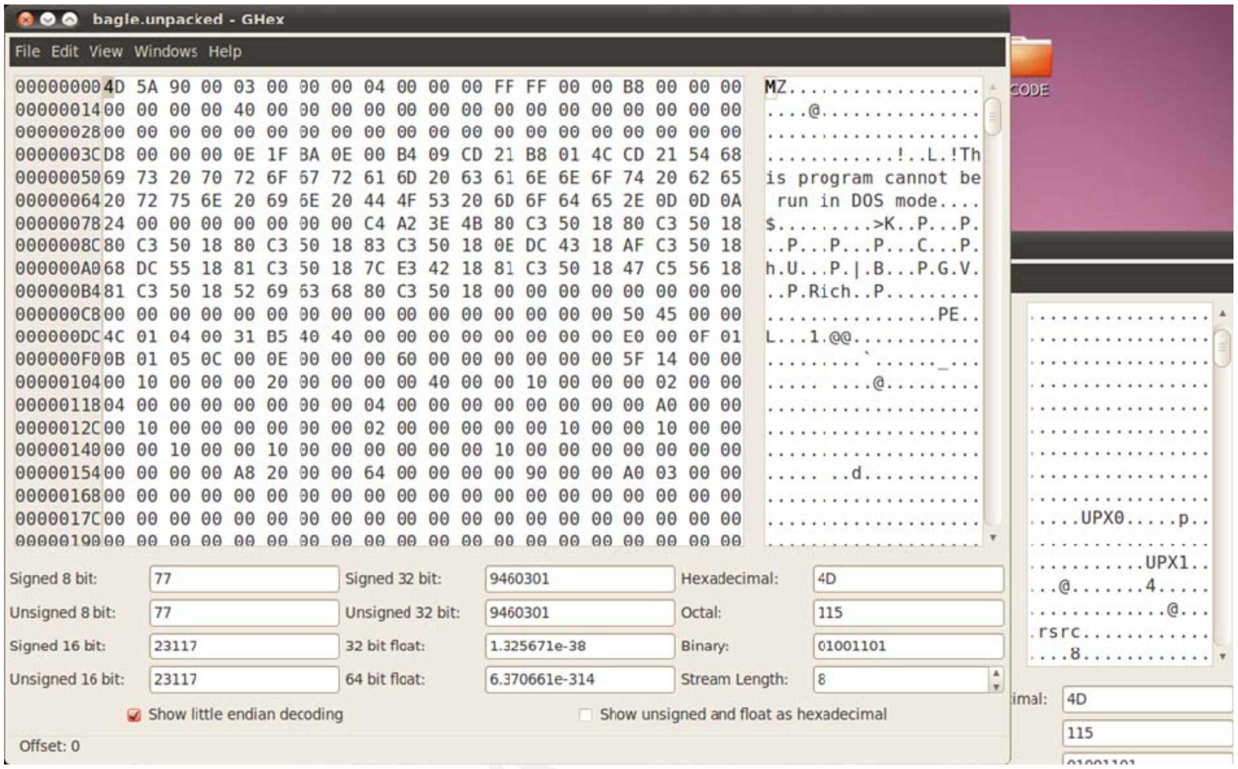
BinText defaults sets the minimum string length to 5 which misses some APIs of interest.

Strings representing API function names are normally four characters or more. Thus, if a tool or method is used to dump strings for a compiled binary that is 5, shorter APIs with a character length of 4, are not included in the list. As of such, the seemingly ideal character length for a comprehensive strings analysis (ASCII and Unicode) is 4.

The Sysanalyzer program was selected as the program for this research project since it exports both ASCII and Unicode strings with the length desired for API analysis (4 characters).

2.3. Unpacked Binaries for Strings Analysis

Strings of most malicious codes cannot be extracted until they are unpacked, as the majority of malcode in the wild in 2011 are packed. The image below shows in a hex editor a variant of the Bagle worm family, with a MZ header in an unpacked sample in the forefront and strings related to the famous UPX packer in the unpacked sample of the worm in the background:



```
000000004D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00  MZ..................
0000001400 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....@...............
0000002800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000003CD8 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ............!..L.!Th
0000005069 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 65  is program cannot be
0000006420 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A   run in DOS mode....
0000007824 00 00 00 00 00 00 00 C4 A2 3E 4B 80 C3 50 18 80 C3 50 18  $.........>K..P...P.
0000008C80 C3 50 18 80 C3 50 18 83 C3 50 18 0E DC 43 18 AF C3 50 18  ..P...P...P...C...P.
000000A068 DC 55 18 81 C3 50 18 7C E3 42 18 81 C3 50 18 47 C5 56 18  h.U...P.|.B...P.G.V.
000000B481 C3 50 18 52 69 63 68 80 C3 50 18 00 00 00 00 00 00 00 00  ..P.Rich..P.........
000000C800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 50 45 00 00  ................PE..
000000DC4C 01 04 00 31 B5 40 40 00 00 00 00 00 00 00 00 E0 00 0F 01  L...1.@@............
000000F00B 01 05 0C 00 0E 00 00 00 60 00 00 00 00 00 00 5F 14 00 00  .........`......_...
0000010400 10 00 00 00 20 00 00 00 00 40 00 00 10 00 00 00 02 00 00  ..... ....@.........
0000011804 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 A0 00 00  ....................
0000012C00 10 00 00 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00  ....................
0000014000 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 00  ....................
0000015400 00 00 00 A8 20 00 00 64 00 00 00 00 90 00 00 A0 03 00 00  ..... ..d...........
0000016800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000017C00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
0000019000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
```

Signed 8 bit:	77	Signed 32 bit:	9460301	Hexadecimal:	4D
Unsigned 8 bit:	77	Unsigned 32 bit:	9460301	Octal:	115
Signed 16 bit:	23117	32 bit float:	1.325671e-38	Binary:	01001101
Unsigned 16 bit:	23117	64 bit float:	6.370661e-314	Stream Length:	8

[✓] Show little endian decoding   [ ] Show unsigned and float as hexadecimal

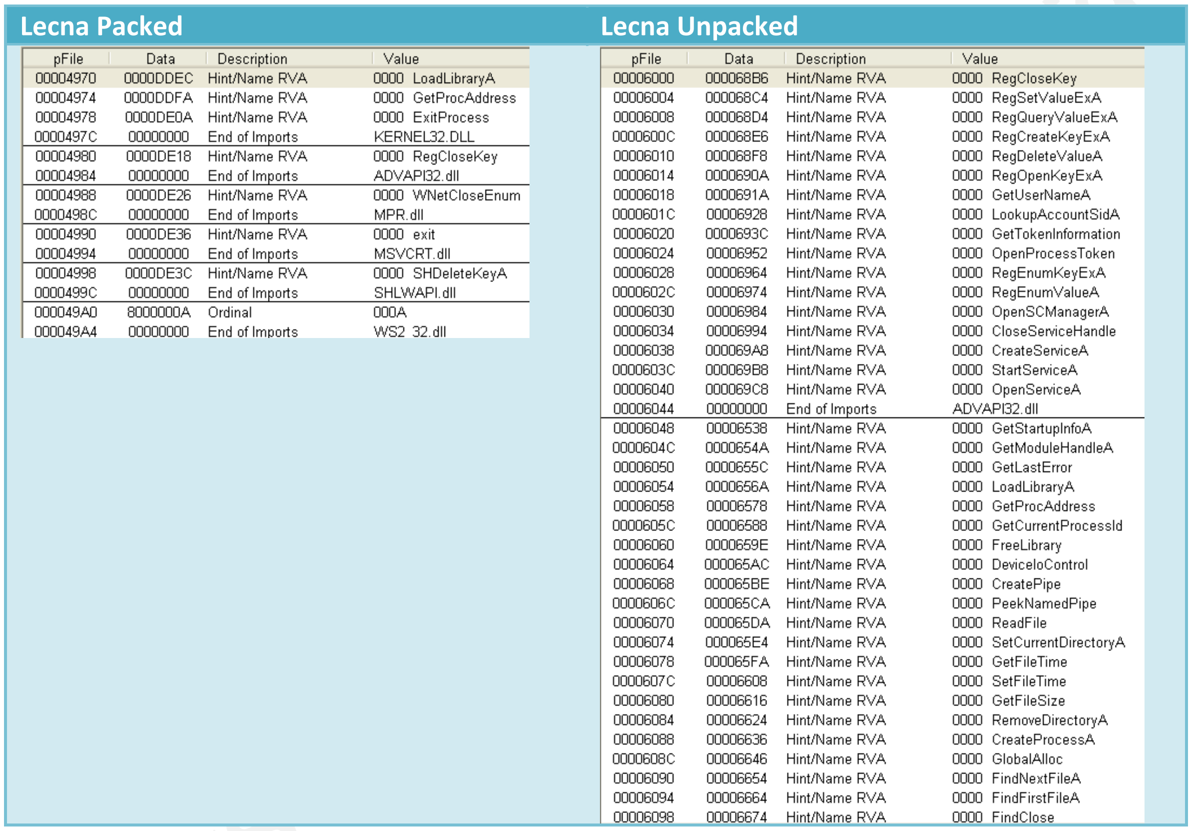
Offset: 0

```
..................
..................
..................
..................
..................
..................
..................
..................
..................
.....UPX0.....p..
..................
............UPX1..
...@........4.....
...............@...
.rsrc.............
....8.............
```

imal: 4D
115

Bagle is packed with UPX, unpacked to reveal more strings of value.

Once a file is unpacked the number of APIs that can be located in strings for the executable is significant and of high value to a malcode researcher. The image below shows PEView (Radburn, 2010) used to view Lecna malcode packed and unpacked, and the associated APIs of interest.



Lecna Packed

pFile	Data	Description	Value
00004970	0000DDEC	Hint/Name RVA	0000 LoadLibraryA
00004974	0000DDFA	Hint/Name RVA	0000 GetProcAddress
00004978	0000DE0A	Hint/Name RVA	0000 ExitProcess
0000497C	00000000	End of Imports	KERNEL32.DLL
00004980	0000DE18	Hint/Name RVA	0000 RegCloseKey
00004984	00000000	End of Imports	ADVAPI32.dll
00004988	0000DE26	Hint/Name RVA	0000 WNetCloseEnum
0000498C	00000000	End of Imports	MPR.dll
00004990	0000DE36	Hint/Name RVA	0000 exit
00004994	00000000	End of Imports	MSVCRT.dll
00004998	0000DE3C	Hint/Name RVA	0000 SHDeleteKeyA
0000499C	00000000	End of Imports	SHLWAPI.dll
000049A0	8000000A	Ordinal	000A
000049A4	00000000	End of Imports	WS2_32.dll

Lecna Unpacked

pFile	Data	Description	Value
00006000	000068B6	Hint/Name RVA	0000 RegCloseKey
00006004	000068C4	Hint/Name RVA	0000 RegSetValueExA
00006008	000068D4	Hint/Name RVA	0000 RegQueryValueExA
0000600C	000068E6	Hint/Name RVA	0000 RegCreateKeyExA
00006010	000068F8	Hint/Name RVA	0000 RegDeleteValueA
00006014	0000690A	Hint/Name RVA	0000 RegOpenKeyExA
00006018	0000691A	Hint/Name RVA	0000 GetUserNameA
0000601C	00006928	Hint/Name RVA	0000 LookupAccountSidA
00006020	0000693C	Hint/Name RVA	0000 GetTokenInformation
00006024	00006952	Hint/Name RVA	0000 OpenProcessToken
00006028	00006964	Hint/Name RVA	0000 RegEnumKeyExA
0000602C	00006974	Hint/Name RVA	0000 RegEnumValueA
00006030	00006984	Hint/Name RVA	0000 OpenSCManagerA
00006034	00006994	Hint/Name RVA	0000 CloseServiceHandle
00006038	000069A8	Hint/Name RVA	0000 CreateServiceA
0000603C	000069B8	Hint/Name RVA	0000 StartServiceA
00006040	000069C8	Hint/Name RVA	0000 OpenServiceA
00006044	00000000	End of Imports	ADVAPI32.dll
00006048	00006538	Hint/Name RVA	0000 GetStartupInfoA
0000604C	0000654A	Hint/Name RVA	0000 GetModuleHandleA
00006050	0000655C	Hint/Name RVA	0000 GetLastError
00006054	0000656A	Hint/Name RVA	0000 LoadLibraryA
00006058	00006578	Hint/Name RVA	0000 GetProcAddress
0000605C	00006588	Hint/Name RVA	0000 GetCurrentProcessId
00006060	0000659E	Hint/Name RVA	0000 FreeLibrary
00006064	000065AC	Hint/Name RVA	0000 DeviceIoControl
00006068	000065BE	Hint/Name RVA	0000 CreatePipe
0000606C	000065CA	Hint/Name RVA	0000 PeekNamedPipe
00006070	000065DA	Hint/Name RVA	0000 ReadFile
00006074	000065E4	Hint/Name RVA	0000 SetCurrentDirectoryA
00006078	000065FA	Hint/Name RVA	0000 GetFileTime
0000607C	00006608	Hint/Name RVA	0000 SetFileTime
00006080	00006616	Hint/Name RVA	0000 GetFileSize
00006084	00006624	Hint/Name RVA	0000 RemoveDirectoryA
00006088	00006636	Hint/Name RVA	0000 CreateProcessA
0000608C	00006646	Hint/Name RVA	0000 GlobalAlloc
00006090	00006654	Hint/Name RVA	0000 FindNextFileA
00006094	00006664	Hint/Name RVA	0000 FindFirstFileA
00006098	00006674	Hint/Name RVA	0000 FindClose

PEView reveals a much larger list of APIs available for review in the unpacked version of Lecna.

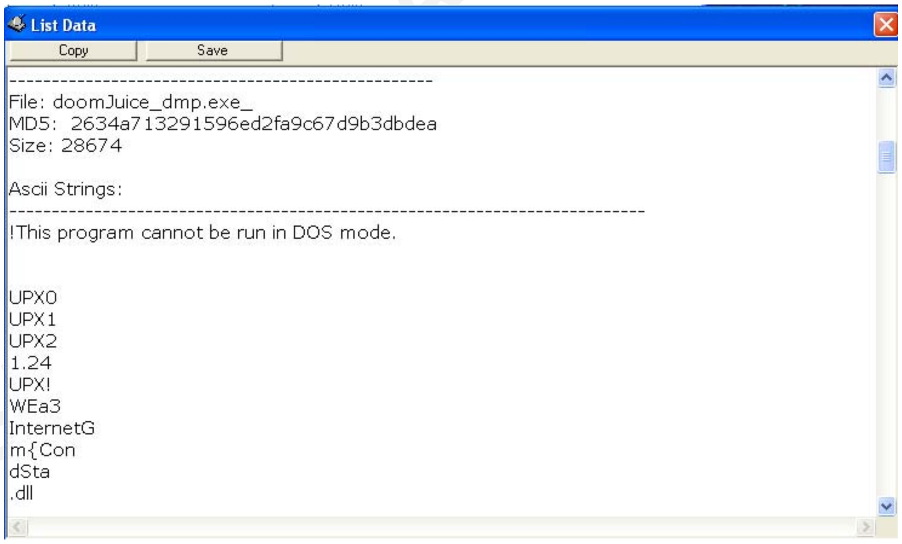
Unpacking files can be a nightmare if done manually. Older methods involved using programs like UPX to then run "upx –d Trojan", where Trojan is the malicious code of interest being unpacked. Such a process can be very time consuming, riddled with many different unpackers and various challenges for each situation.

There exist hundreds of packers, including custom packers. Some of them can be handled using standard unpackers, others using heuristics. Some introduce new techniques to evade all known heuristics. Creating a universal unpacker that can handle all types of packers is infeasible.

Ken Dunham, kend@kendunham.org

To easily capture strings related to a binary of interest that are unpacked performing a strings dump of a process in memory is highly efficient (let the code unpack itself). Tools like Sysanalyzer and Process Explorer (Microsoft Corp., 2010d) make capturing of a specific process fairly simple for an analyst working on a specific code of interest.

Sysanalyzer enables an analyst to run malicious code and then perform a variety of analysis around that malicious code, such as sniffing, monitoring processes, etc. It automatically generates a summary report of changes to a system, including ASCII and Unicode strings that may exist, after about a minute following execution of code. The analyst simply clicks on the Save button to have the report saved. Strings may then be copied from this report into a new text document to then perform strings analysis. The image below displays what an analyst sees when analyzing code with Sysanalyzer:



```
------------------------------------------------
File: doomJuice_dmp.exe_
MD5:  2634a713291596ed2fa9c67d9b3dbdea
Size: 28674

Ascii Strings:
---------------------------------------------------------------------------
!This program cannot be run in DOS mode.


UPX0
UPX1
UPX2
1.24
UPX!
WEa3
InternetG
m{Con
dSta
.dll
```

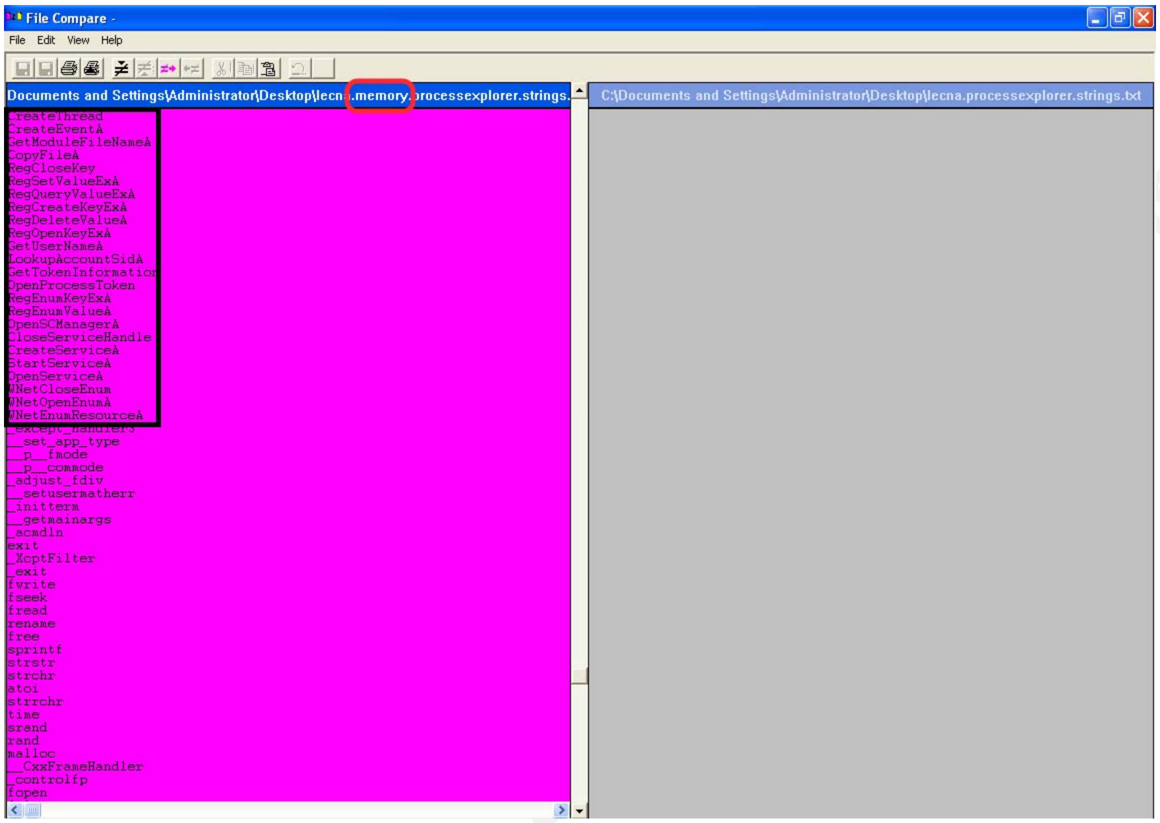
Sysanalyzer generates a report that includes strings.

Strings dumped by Sysanalyzer must be four characters or more in length. Process Explorer is similar but lowers the limit to three characters. As a result, Process Explorer string dumps are longer and contain more data that is not of interest to API research since APIs of interest are four characters and longer.

Process Explorer enables string captures by double-clicking on a process. If desired, an analyst can right-click on a process to temporarily suspend it so that such analysis may be done before a change might take place. Once process properties have been opened (double-clicking) the analyst simply clicks on the Strings tab, selects the Memory radio button, and clicks on save.

When a strings dump is done on an image (packed) and a process (unpacked) using Process Explorer the difference in details is often significant. Using Lecna as an example, 274 strings exist that are three characters or longer in the image while 712 exist in strings captured from the process! Most importantly, APIs and other key data points are visible in the unpacked process that is not seen in the packed image.

Below is an image of a string *comparison* for Lecna for strings of the binary based upon the image (right side) and the unpacked process (left side). A forensic package called Fcompare (Walter Oney Software, 2011) is used to show differences between the two strings dumps, showing a massive difference in purple where such strings *only* exist (in purple) in the unpacked process.



Documents and Settings\Administrator\Desktop\lecn .memory. rocessexplorer.strings.

C:\Documents and Settings\Administrator\Desktop\lecna.processexplorer.strings.txt

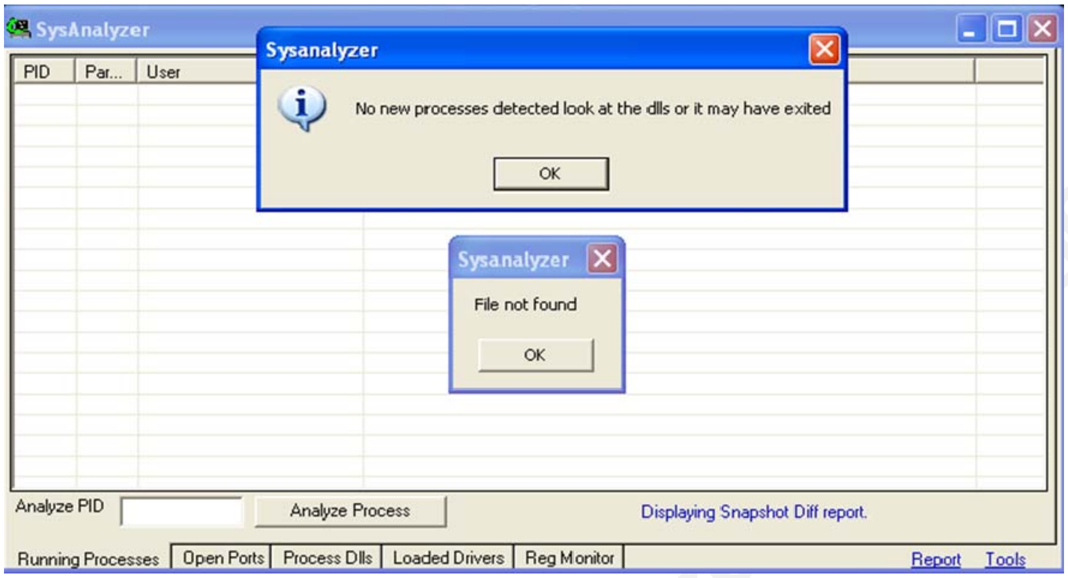
```
CreateThread
CreateEventA
GetModuleFileNameA
CopyFileA
RegCloseKey
RegSetValueExA
RegQueryValueExA
RegCreateKeyExA
RegDeleteValueA
RegOpenKeyExA
GetUserNameA
LookupAccountSidA
GetTokenInformation
OpenProcessToken
RegEnumKeyExA
RegEnumValueA
OpenSCManagerA
CloseServiceHandle
CreateServiceA
StartServiceA
OpenServiceA
WNetCloseEnum
WNetOpenEnumA
WNetEnumResourceA
_except_handler3
__set_app_type
__p__fmode
__p__commode
_adjust_fdiv
__setusermatherr
_initterm
__getmainargs
_acmdln
exit
_XcptFilter
_exit
fwrite
fseek
fread
rename
free
sprintf
strstr
strchr
atoi
strrchr
time
srand
rand
malloc
__CxxFrameHandler
_controlfp
fopen
```

Unpacked strings in memory reveal many APIs of interest not seen in a packed sample.

2.4. Process Explorer Strings Backup Method

In some cases malicious code can be difficult to work with as it may attempt to immediately terminate or perform other actions hindering strings analysis of a process. In such cases a variety of tactics may be used to capture strings from a process, such as hindering termination of any programs, memory analysis, and more.

One fast and easy method for capturing strings from a process in memory is to *suspend* it using a program like Flypaper (HBGary, 2011) or a right-click option within Process Explorer (or use a debugger, etc). Then use Process Explorer to capture strings from the unpacked process in memory. The downside to using Process Explorer as the strings tool is that it uses a 3 character limit increasing the amount of undesired strings in the sample set when specifically looking for APIs of interest. The image below shows Sysanalyzer being unable to capture strings but Process Explorer, working with a suspended hostile process, used capture strings from memory:

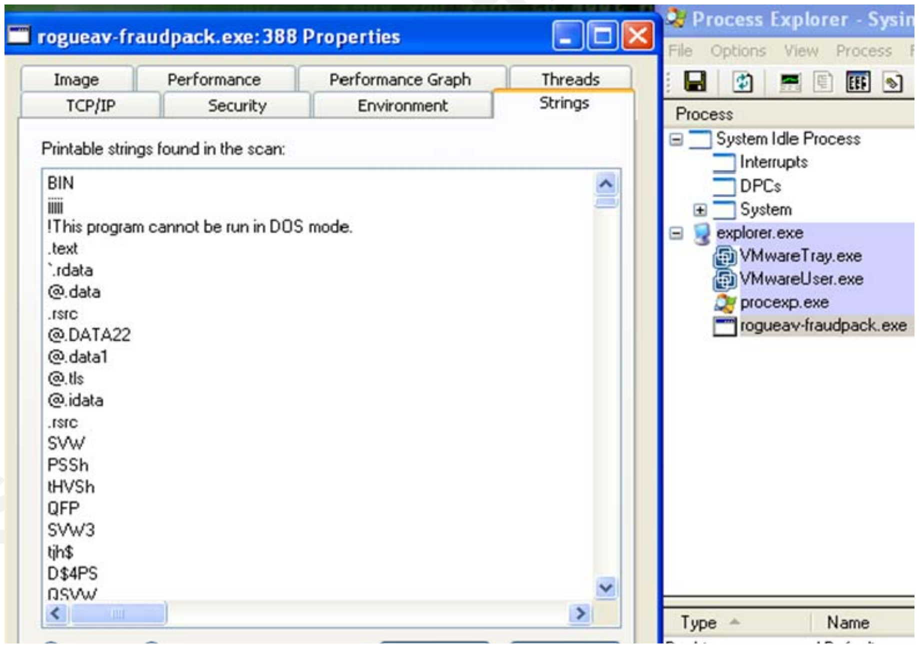


No new processes detected look at the dlls or it may have exited

File not found

Displaying Snapshot Diff report.

Sysanalyzer is unable to analyze a hostile process that may have exited.



Printable strings found in the scan:

```
BIN
iiiii
!This program cannot be run in DOS mode.
.text
`.rdata
@.data
.rsrc
@.DATA22
@.data1
@.tls
@.idata
.rsrc
SVW
PSSh
tHVSh
QFP
SVW3
tjh$
D$4PS
QSVW
```

Process Explorer exporting unpacked strings from a process to a file.

3. Introduction to Application Programming Interface (API)

Windows application programming interface (API) is how Windows uses Dynamic Link Libraries (DLLs) to provide developers with consistent commonly used functions for interacting with the operating system (Microsoft Corp., 2010a).

Microsoft Corp. has created a large number of DLL files that programmers may use as part of their development of software for Microsoft Windows. The DLL files commonly exist in the Windows System directory in Windows 95/98 or the Windows System32 directory in Windows NT/XP/Vista/7.

For example, user32.dll is an API that includes support for the messaging handling, timers, menus, and communications. The "MessageBox" function supports the handling of characters and strings for both 1-byte ANSI and 2-byte Unicode data. When unspecified it defaults to ANSI. A programmer may use this DLL to help display a message containing ANSI or Unicode data. They may also use it for other supported functions, such as a menu. When such DLL executables are run by a user they don't run as one might expect. While an executable with an MZ header they are designed to be run in conjunction with the Windows operating system via API calls from other applications.

Microsoft Corp. has documented APIs in the "Platform Software Development Kit (SDK)" that is shipped with Microsoft Visual C++ and/or Microsoft Developer Network (MSDN) subscriptions. "Platform SDK" can also be downloaded from Microsoft Corp. if a user doesn't have access to an MSDN subscription. This is a good starting point for understanding API calls but users must realize that **not every API is not documented**. As a result a wide variety of Internet posts, snippets in books, and other sources attempt to document such features and implementations. A good example of this is an article online at SecurityXploded (SecurityXploded, 2011).

Third party developers can also create their own API/DLL functions as part of a program. As a result, new DLLs may be created and installed on a system to support third party development. This is also true for malware developers, such as hostile DLL and SYS files commonly associated with Windows rootkits that extend functionality of a system for malicious purposes. Third party applications are not normally considered a library file, even if they have the same format and structure, since they are not in general called by other applications on the same system. In summary this means that Windows maintains control over the official API/DLL files on a system but additional third party add-ons may exist for individual programs, such as alternate browsers, or malware of interest.

APIs functions are not static as a whole, undergoing changes with operating system changes. Over 1,100 new API elements were introduced with Windows 95 (Spinellis, 1997). More recent versions of Windows, such as Windows Vista (Microsoft Corp., 2010c) and Windows 7 (Microsoft Corp., 2010b), have documentation of new API and functions online via Microsoft.com.

There are several Windows DLLs that are commonly used as part of the API support on a computer but are not normally included as something reference by applications (Microsoft Corp., 2009). A brief introduction to those DLLs and their functionality are below as a point of reference to approaching API functionality on a system:

DLL Name	Functionality
Hal.dll Hardware Abstraction Layer	Loaded into the kernel to manage chipset needs. Cannot be called directly by applications so no user mode APIs exist for HAL routines. However, most drivers for hardware are contained in files such as SYS files.
Ntdll.dll Native API	Interface used by user-mode components of Windows related to NT related libraries such as NtDisplayString and use of ntoskrnl.exe (Windows Kernel). Also related to kernel level APIs related to kernel32.dll.

Kernel32.dll	Memory management, input/output operations, process and thread creation, and synchronization efforts. Closely linked to ndtll.dll functions and the main interface to Windows.
Gdi32.dll Graphics Device Interface (GDI)	Drawing functions related to video displays, printers, and font management.
User32.dll	User interface management such as Desktop, windows, and menus. Supports a graphical user interface for Windows. Also related to management of windows, messages, and keyboard/mouse events.
Comctl32.dll	Supports Windows controls including File, Open, Save, Save As, Progress bars, and List views.
Msvcrt.dll Microsoft Visual C++ Run-Time	Library functions to support Visual C++ applications.
Shscrap.dll	Support for shell scrap files as part of the Object Linking and Embedding (OLE) mechanism. For example, how the system supports drag and drop operations.
Ws2_32.dll Winsock	Interface to all functionality related to network and data transmission such as TCP/IP.

3.1. Notable API Groups

APIs are named according to conventions for grouping and/or input requirements. Important groups relevant to malicious code implications are identified below:

Nt	May be kernel mode but not necessarily, related to ntdll.dll; prefix.

Zw	Ensures kernel mode; relevant to kernel level rootkits; prefix.
Ldr	Loader functions for PE file handling and starting of a new process; prefix.
Ex	Windows Executive; second generation API (extension of the original API); appended to API name.
W	Unicode input; appended to API name taking arguments in Unicode format.
A	ASCII input; appended to API name.

Many others exist, such as "Ke" (core kernel routines), "Ks" (kernel streaming), "Ps" (process management), etc.

3.2. Undocumented Windows API Functions

Microsoft Corp. has not publicly documented all functions and features of Windows API. This may be due to a goal to focus programmers on using primary interfaces developed by Microsoft Corp. while providing flexibility about changing components of the operating system. Unfortunately malicious actors are able to identify specific DLLs and API contexts of interest to further investigate and locate undocumented functions. Undocumented APIs, such as SetSfcFileException, have been abused by bad actors to subvert security models of the operating system. Another form of API abuse is to use an alternative API call to accomplish the same task as a more common one, as a way to potentially avoid detection by anti-virus software and/or analysis thereof.

A significant amount of "undocumented" functions related to Windows API have been the topic of many publications to date. One such example exists at ntinternals.net (NTinterlnals.net team, 2008). One example post of an actor initiating research into undocumented features related to concealment of a process is below, revealing at least one approach to researching undocumented APIs of interest (mc_ginley, 2007):

> "Hello everybody. I was playing around with Microsofts Detour 2.1 these days and succesfully "hooked" the well known NtQuerySystemInformation function to hide a process. Funny thing is that this does NOT work if you try tasklist.exe in the windows command prompt.
> So far i found out that tasklist.exe uses (undocumented) functions of winsta.dll to list the running processes of a "WinStation" via Terminal Services. Googleing around i found a topic on this site (http://forum.sysinternals.com/forum_posts.asp?TID=7375&K W=winsta%2Edll) in which someone asked for the calling convention of WinStationKillProcess() which is also part of winsta.dll.
> So, does anyone know how these functions are used or what their parameters are? Unfortunately i'm not really into debugging but i could see (in IDAPro) something like that
> WinStationEnumerateProcesses(x, x)
> WinStationGetAllProcesses(x, x, x, x)
> The "official" documented api to terminal services is Wtsapi32.dll which itself uses winsta.dll functions (i think at least). Any hint would be great. "

Bamital is a sophisticated code that uses an undocumented API as part of an infection routine. Specifically, a data file ("dll") with the MD5 value of 8b0a8d3b0760bd7779b2a8b4fc0682b1, contains encrypted data referencing the undocumented API of interest:

Decoded "dll" file reveals reference to sfc_os.dll which exploits an undocumented API call to inject protected system files.

Ken Dunham, kend@kendunham.org

In order to infect a computer and also remain hidden it attempts to modify a pair of system applications without changing file properties, including size. Explorer.exe (and possibly others) is infected during the routine, including the dllcache copy of the file.

To bypass Windows file protection on Windows XP Bamital uses an undocumented SetSfcFileException API within sfc_os.dll to disable file protection. Once disabled, changes are made to the original file and dllcache backup. Only an MD5 checksum type hash check will reveal to the analyst that the files have changed as file size does not change with this file injection. This "undocumented API" has been reported in a few sources online which may have been leveraged by a malicious actor or researched as a result of an actor identifying and then exploiting such functionality (Shevchenko, 2009).

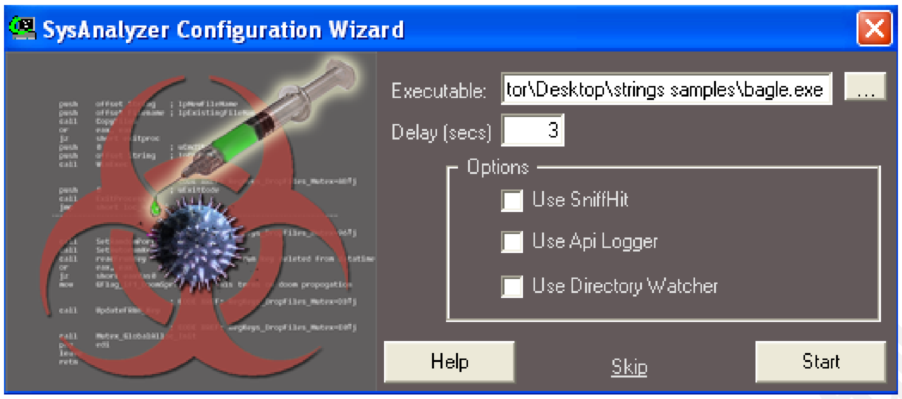
Strings in Bamital are encrypted, and several files are used as part of the infection routine. This hinders the common methodology used in this research from being used on codes like Bamital. Instead, debugging and reverse engineering is required to properly understand Bamital and the various API references made during infection.

4. Methodology of API Abuse Research

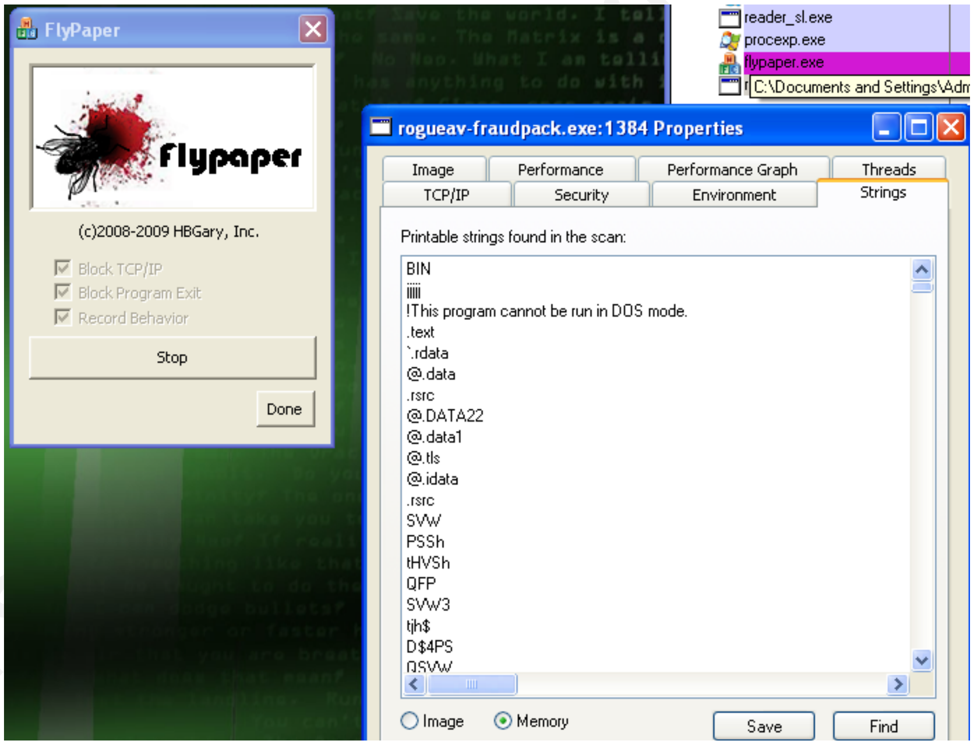
Dozens of lab qualified malicious codes obtained from incident response and malicious code research with iSIGHT Partners was used in conjunction with deep reports on each sample used within this study. This enables the researcher to quickly perform API research on codes with known behaviors that are repeatable in a lab environment, helpful in understanding and analyzing abuse of APIs associated with each malicious code. In short, the methodology is as follows:

1. Use Flypaper and Sysanalyzer to run code in a VMware lab environment.

2. Use Sysanalyzer to export strings of the hostile process. This, by default, exports both ASCII and Unicode strings with a minimum length of 4.


SysAnalyzer Configuration Wizard
Executable: tor\Desktop\strings samples\bagle.exe
Delay (secs) 3
Options
Use SniffHit
Use Api Logger
Use Directory Watcher
Help
Skip
Start

3. If Sysanalyzer is unable to locate a new hostile process use Process Explorer for a strings capture of the process suspended in memory using Flypaper at runtime.



Printable strings found in the scan:

```
BIN
iiiii
!This program cannot be run in DOS mode.
.text
`.rdata
@.data
.rsrc
@.DATA22
@.data1
@.tls
@.idata
.rsrc
SVW
PSSh
tHVSh
QFP
SVW3
tjh$
D$4PS
QSVW
```

4. Export strings from the malicious process, using Process Explorer.

Ken Dunham, kend@kendunham.org

5. Perform an API strings analysis upon the strings captured to leverage in this research of what the code is known to do behaviorally.

The identified method for this research works in both virtual environments, like that of VMware, and native environments. The tools selected for this research are all in the freeware domain enabling security experts in training to quickly use all such tools to replicate and leverage this research when analyzing code in the lab.

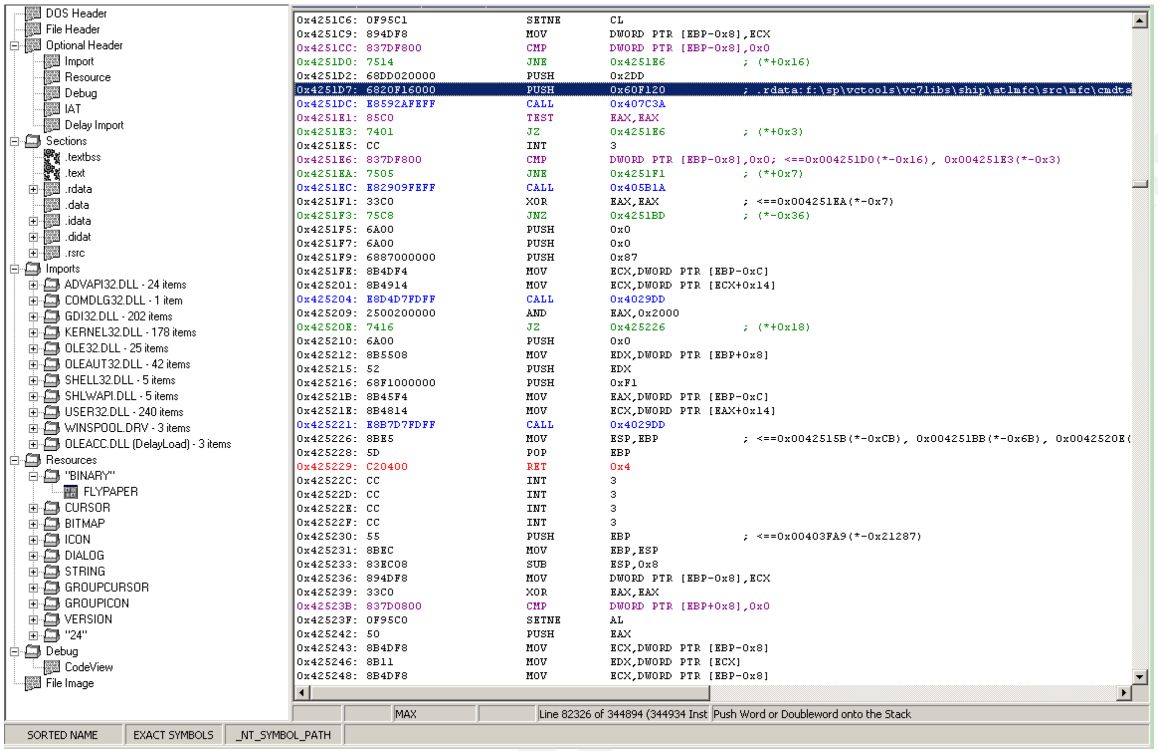
4.1. Other Methods Considered but Rejected

Other methods also exist for capturing strings of hostile processes that may terminate rapidly, such as placing a virtual machine into hibernation mode and the analyzing the VMEM file created on the host computer or performing a dump of all memory to a file (e.g. windd32). These methods unfortunately involve analysis of *all* artifacts in memory which requires much time and effort than is necessary for the method adopted for this research.

Windows Memory Forensic Toolkit (WMFT) also has the ability to dump physical memory, but it also fails to work well due to a race condition with codes that quickly exit (the hostile code may simply exit too quickly). User Mode Process dumps require an installation and are not the type of output desired for rapid strings analysis. Pmdump also has the wrong type of output for analysis focused upon in this research. Windows Vista and Windows 7 Task Manager (TM) also enables a memory dump for a process, but this is again, a race condition with malcode that quickly exits or injects making such dumps unreliable using TM.

4.1. Strings Analysis for APIs of Interest

This is where the real work begins for the malcode analyst. Strings contain a large number of data points that may allude to various functions or APIs commonly abused by malcode. Strings typically reveal the structure and flow of a binary, much like that seen in PEBrowse Professional Interactive (Osterlund, 2011):



```
DOS Header
File Header
Optional Header
    Import
    Resource
    Debug
    IAT
    Delay Import
Sections
    .textbss
    .text
    .rdata
    .data
    .idata
    .didat
    .rsrc
Imports
    ADVAPI32.DLL - 24 items
    COMDLG32.DLL - 1 item
    GDI32.DLL - 202 items
    KERNEL32.DLL - 178 items
    OLE32.DLL - 25 items
    OLEAUT32.DLL - 42 items
    SHELL32.DLL - 5 items
    SHLWAPI.DLL - 5 items
    USER32.DLL - 240 items
    WINSPOOL.DRV - 3 items
    OLEACC.DLL (DelayLoad) - 3 items
Resources
    "BINARY"
        FLYPAPER
    CURSOR
    BITMAP
    ICON
    DIALOG
    STRING
    GROUPCURSOR
    GROUPICON
    VERSION
    "24"
Debug
    CodeView
File Image
```

```
0x4251C6: 0F95C1          SETNE    CL
0x4251C9: 894DF8          MOV      DWORD PTR [EBP-0x8],ECX
0x4251CC: 837DF800        CMP      DWORD PTR [EBP-0x8],0x0
0x4251D0: 7514            JNE      0x4251E6           ; (*+0x16)
0x4251D2: 68DD020000      PUSH     0x2DD
0x4251D7: 6820F16000      PUSH     0x60F120           ; .rdata:f:\sp\vctools\vc7libs\ship\atlmfc\src\mfc\cmdta
0x4251DC: E8592AFEFF      CALL     0x407C3A
0x4251E1: 85C0            TEST     EAX,EAX
0x4251E3: 7401            JZ       0x4251E6           ; (*+0x3)
0x4251E5: CC              INT      3
0x4251E6: 837DF800        CMP      DWORD PTR [EBP-0x8],0x0; <==0x004251D0(*-0x16), 0x004251E3(*-0x3)
0x4251EA: 7505            JNE      0x4251F1           ; (*+0x7)
0x4251EC: E82909FEFF      CALL     0x405B1A
0x4251F1: 33C0            XOR      EAX,EAX            ; <==0x004251EA(*-0x7)
0x4251F3: 75C8            JNZ      0x4251BD           ; (*-0x36)
0x4251F5: 6A00            PUSH     0x0
0x4251F7: 6A00            PUSH     0x0
0x4251F9: 6887000000      PUSH     0x87
0x4251FE: 8B4DF4          MOV      ECX,DWORD PTR [EBP-0xC]
0x425201: 8B4914          MOV      ECX,DWORD PTR [ECX+0x14]
0x425204: E8D4D7FDFF      CALL     0x4029DD
0x425209: 2500200000      AND      EAX,0x2000
0x42520E: 7416            JZ       0x425226           ; (*+0x18)
0x425210: 6A00            PUSH     0x0
0x425212: 8B5508          MOV      EDX,DWORD PTR [EBP+0x8]
0x425215: 52              PUSH     EDX
0x425216: 68F1000000      PUSH     0xF1
0x42521B: 8B45F4          MOV      EAX,DWORD PTR [EBP-0xC]
0x42521E: 8B4814          MOV      ECX,DWORD PTR [EAX+0x14]
0x425221: E8B7D7FDFF      CALL     0x4029DD
0x425226: 8BE5            MOV      ESP,EBP            ; <==0x0042515B(*-0xCB), 0x004251BB(*-0x6B), 0x0042520E(
0x425228: 5D              POP      EBP
0x425229: C20400          RET      0x4
0x42522C: CC              INT      3
0x42522D: CC              INT      3
0x42522E: CC              INT      3
0x42522F: CC              INT      3
0x425230: 55              PUSH     EBP                ; <==0x00403FA9(*-0x21287)
0x425231: 8BEC            MOV      EBP,ESP
0x425233: 83EC08          SUB      ESP,0x8
0x425236: 894DF8          MOV      DWORD PTR [EBP-0x8],ECX
0x425239: 33C0            XOR      EAX,EAX
0x42523B: 837D0800        CMP      DWORD PTR [EBP+0x8],0x0
0x42523F: 0F95C0          SETNE    AL
0x425242: 50              PUSH     EAX
0x425243: 8B4DF8          MOV      ECX,DWORD PTR [EBP-0x8]
0x425246: 8B11            MOV      EDX,DWORD PTR [ECX]
0x425248: 8B4DF8          MOV      ECX,DWORD PTR [EBP-0x8]
```

MAX
Line 82326 of 344894 (344934 Inst
Push Word or Doubleword onto the Stack
SORTED NAME
EXACT SYMBOLS
_NT_SYMBOL_PATH

Flypaper viewed with PEBrowse Professional Interactive.

Notice in the image above DOS Header, File Header, sections, imports and other components common to a Windows executable. When reviewing strings this is often seen in the initial strings, in sequence, such as the following:

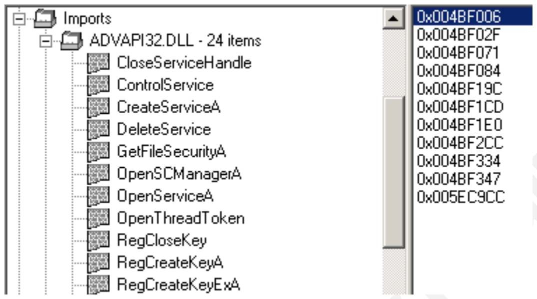
This program must be run under Win32
.text
`.itext
`.data
.bss
.idata
.tls
.rdata

When researching the above sections a few like ".itext" may lead to the discovery of documentation about Borland Delphi containing such sections[1], "PE files produced by Delphi may contain these sections which must be located: section CODE, .itext, DATA, BSS, .tls, .rdata, and .idata." Additional strings may then exist just after the sections of the PE file further suggesting it is a program compiled in Borland Delphi, "FastMM

[1] http://www.on-time.com/rtos-32-docs/rttarget-32/programming-manual/compiling/borland-delphi.htm

Borland", "Edition 2004, 2005 Pierre le Riche / Professional Software Development", and "SOFTWARE\Borland\Delphi\RTL".

PEBrowse Professional Interactive can also help to reveal the flow of imports and their relationships to various DLLs, as shown below:

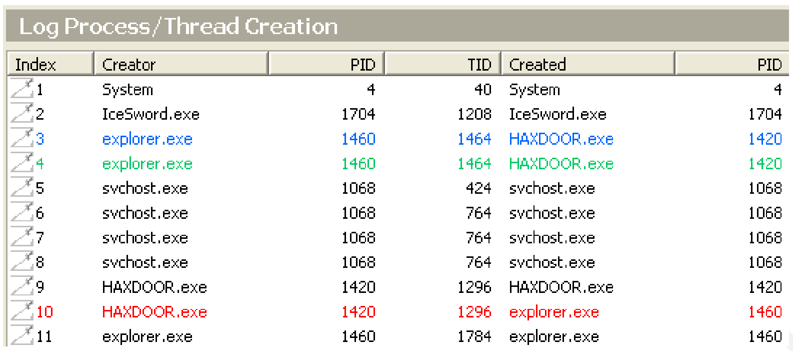


```
Imports
  ADVAPI32.DLL - 24 items
    CloseServiceHandle     0x004BF006
    ControlService         0x004BF02F
    CreateServiceA         0x004BF071
    DeleteService          0x004BF084
    GetFileSecurityA       0x004BF19C
    OpenSCManagerA         0x004BF1CD
    OpenServiceA           0x004BF1E0
    OpenThreadToken        0x004BF2CC
    RegCloseKey            0x004BF334
    RegCreateKeyA          0x004BF347
    RegCreateKeyExA        0x005EC9CC
```

DLLs and their imports are revealed in PEBrowse Professional.

APIs that accept ASCII input end in "A" and those that accept Unicode end in "W". Windows Executive APIs, using the extension "Ex", are second generation APIs. Then there are APIs that don't have any such naming appension as they only accept numeric or binary data, such as CryptHasData. As a final note for this research, as API groups is an exhaustive topic, there are also undocumented APIs making such research even more challenging. Analysts must look for TitleCase strings of interest to then research via sites like http://msdn.microsoft.com/en-us/library/ and general Internet queries to identify possible functionality or context.

For example, an analyst may perform a query at msdn.microsoft.com for WriteProcessMemory to arrive at http://msdn.microsoft.com/en-us/library/ms681674%28VS.85%29.aspx which then identifies this as a documented API responsible for writing data to an area of memory in a specified process. A Google query for "WriteProcessMemory Worm" (or Trojan, etc) then reveals multiple results discussing code injection, providing additional interpretive context. This can further be qualified by looking at behaviors of the code, such as seeing injection in IceSword or using similar tools and tactics in analysis:



Log Process/Thread Creation

Index	Creator	PID	TID	Created	PID
1	System	4	40	System	4
2	IceSword.exe	1704	1208	IceSword.exe	1704
3	explorer.exe	1460	1464	HAXDOOR.exe	1420
4	explorer.exe	1460	1464	HAXDOOR.exe	1420
5	svchost.exe	1068	424	svchost.exe	1068
6	svchost.exe	1068	764	svchost.exe	1068
7	svchost.exe	1068	764	svchost.exe	1068
8	svchost.exe	1068	764	svchost.exe	1068
9	HAXDOOR.exe	1420	1296	HAXDOOR.exe	1420
10	HAXDOOR.exe	1420	1296	explorer.exe	1460
11	explorer.exe	1460	1784	explorer.exe	1460

IceSword reveals injection of Haxdoor rootkit into explorer.exe (Windows).

Interpretation of strings is also logical and sequential. For example, strings commonly begin with data related to executable statements and structure (sections of a binary, etc). Additionally, nearby strings provide context as to what is possibly taking place in a binary or a series of operations that are related. For example, one hostile binary may contain strings suggesting command and control HTTP communications:

```
HttpSendRequestExA
HttpQueryInfoA
HttpSendRequestExW
InternetQueryDataAvailable
InternetReadFileExA
HttpSendRequestW
GetUrlCacheEntryInfoW
```

An analyst may review each of the above strings for possible functionality and how it may be related to possible malicious behavior reported or seen in behavioral tests. After strings are compared to known functionality of code, or further investigated with reverse-engineering, a lab qualified context for abuse of APIs is then known.

4.2. APIs Commonly Found in Malcode

Strings from over 600 malicious samples were collected by Nilsen in support of this research project. A simple count was then performed to identify those strings most common to the codes analyzed. The more common an API string reference is the most likely it is to be abused by malicious code (higher reliability).

Count	API
571	LoadLibraryA

133	GetUserNameA
119	GetComputerNameA
116	GetVersionExA
104	GetModuleFileNameA
101	GetStartupInfoA
96	IsCharAlphaA
92	IsBadStringPtrA
84	IsCharUpperA
78	GetWindowTextA
68	IsCharAlphaNumericA
67	IsCharLowerA
67	GetWindowTextLengthA
38	GetModuleHandleA
37	MessageBoxA
36	GetCommandLineA
19	LCMapStringA
19	GetStringTypeA
19	FreeEnvironmentStringsA
19	ChooseFontA

LoadLibraryA is the most common API referenced within the considered bots.

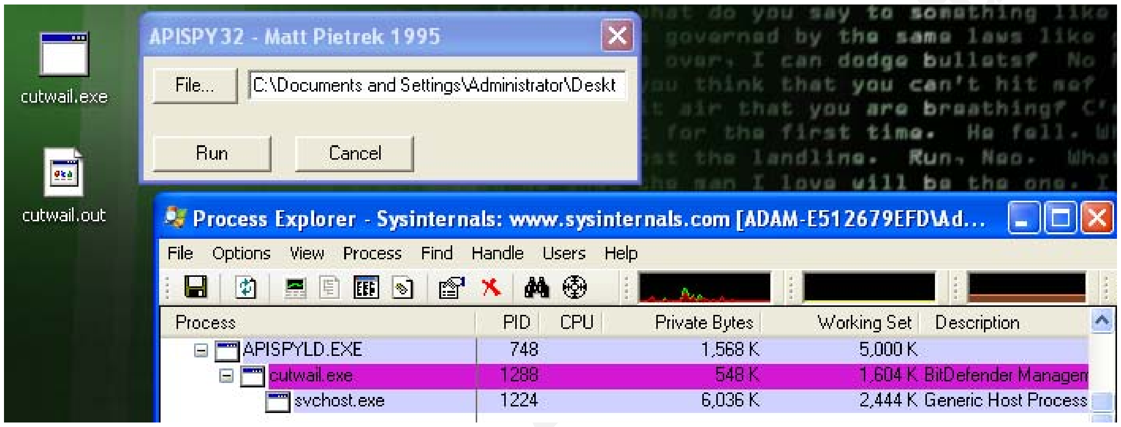
Reliability is not within the scope of this paper but is addressed in part. Reliability is useful for a contextual understanding of how to work through the art of understanding API abuse. For example, "ChooseFont" is a very common API that provides little malcode context and is not included in the final list of commonly abused APIs in this research. Other APIs, such as LoadLibrary, is commonly used by malicious code to execute malicious code or perform DLL injection, highly relevant and common for a malicious context.

The large group statistics (prevalence) for APIs found in bots are also biased towards the codes used within the group studied. Malcode has changed significantly over the past two years and varies greatly based upon what is being evaluated. The data also largely represents just a few families of bots and is not that diverse (Conficker, Palevo, and Rbot are primary). Analysts are able to accommodate such considerations in their research to create custom contexts and reliability based upon the family of code being analyzed. For example, analysts may create API lists related to families or types of codes commonly analyzed, such as the family of Zeus or bots types.

Ken Dunham, kend@kendunham.org

4.3. APISpy32

One popular tool for analyzing API calls is APISpy32 (Pietrek, 2011). This tool requires that a user create a configuration file to identify which APIs to monitor, which requires some knowledge on behalf of the analyst. A screenshot of the tool loading a code and creating a ".out" log file is below:


cutwail.exe
cutwail.out
APISPY32 - Matt Pietrek 1995
File...
Run
Cancel
Process Explorer - Sysinternals: www.sysinternals.com [ADAM-E512679EFD\Ad...
File Options View Process Find Handle Users Help

C:\Documents and Settings\Administrator\Deskt

Process	PID	CPU	Private Bytes	Working Set	Description
APISPYLD.EXE	748		1,568 K	5,000 K	
cutwail.exe	1288		548 K	1,604 K	BitDefender Managem
svchost.exe	1224		6,036 K	2,444 K	Generic Host Process

APISpy32 loads malcode cutwail.exe and logs hooked APIs to cutwail.out.

The updated version reference in this report comes with a good start for configuration compared to older versions of the tool. For example, the tool includes the following LoadLibrary hooks in the configuration file by default:

```
API:KERNEL32.dll:LoadLibraryA
   LPSTR

API:KERNEL32.dll:LoadLibraryExA
   LPSTR
   HANDLE
   DWORD

API:KERNEL32.dll:LoadLibraryExW
   LPWSTR
   HANDLE
   DWORD
```

NOTE: APISpy32 includes common Windows Data Types as part of its configuration (LPSTR, HANDLE, DWORD). More information can be found on MSDN (Microsoft Corp., 2008) related to these data types.

If an analyst extracts APIs of interest from a strings sample and populates the APISpy32 "APISPY32.API" configuration file accordingly the tool becomes extremely valuable, revealing handles called by malcode loaded and monitored by the tool. For example, populating an APISpy32 configuration file with APIs commonly abused by malcode, the primary output of this research, is a fantastic application for default malcode monitoring.

The table below reveals some of the APISpy32 output using default configuration values, with strings from the hostile binary in bold to show those monitored. An analyst can simply add additional items to the configuration of the tool before using it to run the malcode to then capture details aiding in API analysis:

Select Strings in Malcode	APISpy32 Log Output (using default configuration)
…	…
GetCurrentProcess	**GetProcAddress**(HANDLE:7C800000,LPSTR:00419
GetCurrentThread	0AF:"CompareStr")
GetProcAddress	GetProcAddress returns: 7C80D293
LoadLibraryExW	GetProcAddress(HANDLE:7C800000,LPSTR:004190
LeaveCriticalSection	BF:"ExitProces")
HeapAlloc	GetProcAddress returns: 7C81CAA2
EnterCriticalSection	GetProcAddress(HANDLE:7C800000,LPSTR:004190
LCMapStringW	CC:"GetWindows")
FreeLibrary	GetProcAddress returns: 7C82293B
lstrcpyW	**LoadLibrary**A(LPSTR:0041B0A5:"ADVAPI32.d")
ExitProcess	LoadLibraryA returns: 77DD0000
LoadLibraryA	GetProcAddress(HANDLE:77DD0000,LPSTR:004190
QueryPerformanceCounter	EB:"LsaClose")
…	…

If the analyst takes other strings of interest, such as GetCurrentProcess, APISpy32 will attempt to hook and log all such handles and events accordingly.

Ken Dunham, kend@kendunham.org

5. Abuse of APIs within a Malcode Context

After performing research on countless samples and correlating to malicious behavior, the following pointers have been aggregated for how *Windows* APIs may be abused by malicious code within specific contexts. This is not conclusive as a malicious code context must be qualified before these pointers are of any value, followed with additional lab qualified research accordingly. Additionally, more general APIs abused are not included in the list compiled in this research, such as InternetOpen which is a common call to initiate WinINet functions. Such APIs are useful for creating context for nearby APIs in strings but are not included in the malcode context chart for this paper.

Other APIs that are relevant, discovered in analysis of samples, are not included such as those related to Mozilla Firefox APIs and strings like "Autorun.inf" which may be related to a removable drive infection routine but are not a Windows API. For example, FireFox related APIs hooked by Zeus, PR_OpenTCPSocket and PR_Read, or SpyEye hooked PR_Poll and PR_SetError, are not included in the table output for this research. These APIs are authored and documented by Mozilla.

Appended characters such as "A", "W", "ExW", and "ExA" are removed to globalize the API references, such as having only DNSQuery in the table instead of two entries for DNSQueryA and DNSQueryW. This makes the list much more manageable while still serving core string identification purposes. For example, looking for the string value of "HttpSendRequest" in a Zeus sample successfully locates all four variants that exist within the sample: HttpSendRequestW, HttpSendRequestA, HttpSendRequestExW, and HttpSendRequestExA.

Analysts who use this table with tools like APISpy32 or custom scripts may find benefit in assigning *reliability*. Some API references are almost always malicious, such as URLDownloadToFile, while others have a high false positive rate, such as Process32First. A practical example in this study is "BeginPaint", hooked by Zeus to disable local screen changes when a VCN session is activated. While this is rarely used by malcode families for malicious means it is used by Zeus, one of the most common families of malcode in the wild in 2011. While ratings of "reliability" are beyond the scope of this research the author has attempted to not include APIs that are very common and not specific to malicious context.

For a table reference of APIs identified within the malicious context as part of this research please see Appendix A of this report.

6. References

Special thanks to **David Zimmer** and **Robert Wang**, iSIGHT Partners, for their assistance in research and peer review of this research report. Their support and expertise is invaluable and greatly appreciated.

Additionally, I want to thank the countless professionals that I work with in various public and private groups globally. The following individuals specifically helped to identify malicious APIs of interest, further supplementing findings in this report: Felix Leder, Kjell Christian Nilsen, and Kayne Naughton.

HBGary. (2011). Flypaper. Retrieved from https://www.hbgary.com/popular/flypaper/

mc_ginley. (2007). undocumented api: winsta.dll. Retrieved from http://forum.sysinternals.com/undocumented-api-winstadll_topic9968_post43557.html

McAfee (2011). Bintext 3.03. Retrieved from http://www.mcafee.com/us/downloads/free-tools/bintext.aspx

Microsoft Corp. (2007). Createthread function. Retrieved from http://msdn.microsoft.com/en-us/library/ms682453%28VS.85%29.aspx

Ken Dunham, kend@kendunham.org

Microsoft Corp. (2008). Windows data types. Retrieved from http://msdn.microsoft.com/en-us/library/aa383751%28v=vs.85%29.aspx

Microsoft Corp. (2009). Introducing libraries. Retrieved from http://msdn.microsoft.com/en-us/magazine/dd861346.aspx

Microsoft Corp. (2010a). Dynamic-link libraries. Retrieved from http://msdn.microsoft.com/en-us/library/ms682589%28v=vs.85%29.aspx

Microsoft Corp. (2010b). Windows 7 api list. Retrieved from http://msdn.microsoft.com/en-us/library/ee461765%28v=vs.85%29.aspx

Microsoft Corp. (2010c). Windows Vista api list. Retrieved from http://msdn.microsoft.com/en-us/library/ee461768%28v=VS.85%29.aspx

Microsoft Corp. (2010d). Process explorer v14.01. Retrieved from http://technet.microsoft.com/en-us/sysinternals/bb896653

NTinternals.net team. (2008). The undocumented functions. Retrieved from http://undocumented.ntinternals.net/

Osterlund, R. (2011). Pebrowse professional interactive windows debugger. Retrieved from http://www.smidgeonsoft.prohosting.com/pebrowse-pro-interactive-debugger.html

Pietrek, M. (2011). Apispy32 updated. Retrieved from http://www.wheaty.net/APISPY32.zip

Radburn, W. (2010). Peview version 0.9.8. Retrieved from http://www.magma.ca/~wjr/

SecurityXploded. (2011). Hidden rootkit process detection . Retrieved from http://securityxploded.com/hidden-process-detection.php

Shevchenko, A. (2009). Advancing malware techniques 2008. Retrieved from http://www.virusbtn.com/virusbulletin/archive/2009/01/vb200901-advancing-malware-techniques

Spinellis, D. (1997). A critique of the windows application programming interface. Retrieved from http://dmst.aueb.gr/dds/pubs/jrnl/1997-CSI-WinApi/html/win.html

Walter Oney Software. (2011). File compare utility for windows (fcompare) . Retrieved from http://www.oneysoft.com/fcompare.htm

Yara-project (2011). Yara-Project. Retrieved from http://code.google.com/p/yara-project/

Appendix A – Table of APIs in a Malicious Context

Items in red are for undocumented APIs.

API	Malicious Context
accept	Accepts an incoming connection attempt on a socket; possible backdoor.
AddCredentials	Adds credentials to a logon session.
bind	May indicate backdoor Trojan.
CertDeleteCertificateFromStore	Deletes the specified certificate context from the certificate store.
CheckRemoteDebuggerPresent	Checks for a debugger.
closesocket	Closes an existing socket. Zeus hooks this API to free resources allocated in other hooked APIs.
connect	Establishes a connection to a specified socket, potentially related to downloader, notification, reverse shell, or Internet connectivity events.
ConnectNamedPipe	May indicate a reverse shell backdoor. Look for a hidden cmd process.
ControlService	Sends a control code to a service. Conficker uses this to control a hostile service.
ConsentPromptBehaviorAdmin	Defines settings that enable the administrator to configure the behavior of the User Account Control (UAC).
CopyFile	Malcode copies an existing file to a new file during installation.
CreateDirectory	Creates a directory.
CreateFile	Create or opens a file, likely an installer. Sinowal uses this to modify the MBR by opening Device\Harddisk0\DR0; SpyEye.
CreateMutex	Use a mutex with a new process to run only one instance of code in memory.
CreateNamedPipe	May indicate a reverse shell backdoor. Look for a hidden cmd process.
CreateProcess	Used to run a process.
CreateProcessAsUser	Used to run a process.
CreateRemoteThread	Creates a thread that runs in the virtual address space of another process. A common way of DLL injection is create remote thread with start address to LoadLibrary and parameter point to the DLL to be injected; used by Conficker and Zeus.

Ken Dunham, kend@kendunham.org

CreateService	Conficker uses this to create a service.
CreateThread	Creates a thread to execute within the virual address space of the calling processing; Bamital.
CreateToolhelp32Snapshot	Possibly capturing a snapshot of current processes, as well as heaps, modules, and threads, for reconnaissance.
CryptCreateHash	Intiates hashing of a stream of data, possibly used to encrypt or obfuscate stolen data, netflow, or configuration information.
CryptEncrypt	Encrypts data; SpyEye.
CryptGetHashParam	Possible encryption or obfuscation of data stolen and/or configuration data used by the malcode.
DebugActiveProcess	Used by an advanced multi-process packer.
DeleteFile	Deletes a file.
DeleteService	Deletes a service. May be used to disable Windows Security Center services to lower security and avoid detection.
DeviceIoControl	Sends a control code to a driver.
DisconnectNamedPipe	May indicate a reverse shell backdoor. Look for a hidden cmd process.
DNSQuery	Conficker hooks this API to block access to a list of security-related domains.
EnableExecuteProtectionSupport	Modifications to Windows DEP security controls. Undocumented API.
EnumProcesses	Lists running processes.
EnumWindows	Enumerate Windows open on a system.
ExitProcess	Exits a process.
ExitThread	Monitoring of threads for malicious means.
FindFirstChangeNotification	Monitoring of changes to a specified directory. May be used by malcode to protect itself from deletion or modifications.
FindFirstFile	Searches a directory for files and subdirectories. May be hooked to hide files/directories protected by user-level rootkit.
FindNextFile	A function hook to hide a user-level rootkits system files.
FindWindow	Searches for a top-level windows with specified class name and/or window name.
FinNextChangeNotification	Malcode may monitor a thread or process and/or recreate a file if deleted or modified.
FltRegisterFilter	Keyloggers hook wh_keyboard and WH CALLWINDPROC.
FlushInstructionCache	Flushes cache for specified process; SpyEye.
FreeEnvironmentStrings	Frees up environmental variables related to a process managed by malcode.
FtpGetFile	Downloads a file from an FTP server.

FtpOpenFile	Initiates access to a remote file on an FTP server for reading or writing.
GetClipboardData	Zeus hooks this API to steal data from the clipboard.
GetCommandLine	May support command line parameters.
GetComputerName	Obtains the NetBIOS name of the computer, possibly as a form of reconnaissance for infection tracking.
GetDriveType	Determines wheter a disk drive is a removable, fixed, CD-ROM, RAM Disk, or network drive. Potentially related to a removal drive (USB) or network spreader.
GetDiskFreeSpace	Identifies free space on a drive.
GetEnvironmentVariable	Obtains variable from environmental block of calling process. SpyEye uses this function to transfer a password between two modules.
GetFileAttributes	Zeus hooks this API.
GetHostByAddr	Host information related to an IP address.
GetHostByName	Host information by name.
GetHostName	Retrieve a host name for for the local computer.
GetMenu	Retrieves a handle to the menu assigned to specified window.
GetMessagePos	Retrieves cursor position for last message retrieved by the GetMessage funciton.
GetModuleFileName	Retrieves file path for a module in memory.
GetModuleHandle	Retrieves a module handle.
GetNativeSystemInfo	Obtains information about current system to an application running under WOW64, possibly for reconnaissance.
GetNetworkParams	Retrieves network parameters for the local computer.
GetProcAddress	Retrieves the address of an exported function from a DLL.
GetStartupInfo	Identifies the Windows station, desktop, standard handle, and appearance of the main windows for a process.
GetStringType	Character conversions and manipulation as part of possible data conversion or de-obfuscation.
GetSystemDirectory	Retreives the patch of the system directory as part of a possible configuration or installation of code.
GetTempFileName	Creates a filename for a temporary file.
GetTempPath	Retrieves temporary files path.
GetTickCount	A function used to retrieve the number of ms since the system was started up, possibly related to a sleep function; Bamital.
GetTimeZoneInformation	Retrieves current time zone settings.
GetUpdateRect	Zeus hooks this API.
GetUpdateRgn	Zeus hooks this API.
GetUrlCacheEntryInfo	Gathers cache entry data related to URLs; Koobface.

GetUserName	Obtains the username associated with the current thread, used as possible reconnaisance or installation of code.
GetVersion	Malcode may only work on specified operating system(s).
GetWindowText	Copies the text of a specified windows title bar, possibly used for monitoring triggers for malcode.
GetWindowThreadProcessId	Retrieves the identifier of the thread that created the specified window and, optionally, the identifier of the process that created the window.
htons	Conversion of host TCP/IP data to nework byte order.
HttpAddRequestHeaders	Modification of HTTP headers, possibly abused for C&C operations; SpyEye.
HttpOpenRequest	Creates an HTTP request handle.
HttpQueryInfo	Zeus hooks this API to steal HTTP authentication data of interest. Conficker uses it to obtain date and time from remote websites.
HttpSendRequest	Zeus hooks this API to steal HTTP authentication data of interest; SpyEye.
IcmpSendEcho	Sends out a PING.
InternetCloseHandle	Zeus hooks this API to free resources allocated by other hooked APIs; SpyEye.
InternetConnect	Opens an FTP, Gopher, or HTTP session for a given site. May be hooked by malcode for exfiltrating data.
InternetCrackUrl	Cracks a URL into component parts.
InternetGetConnectedState	Retrieves the connected state of a local system. Conficker hooks this API.
InternetOpen	Initializes an application use of WinINet functions. May be downloading a hostile file.
InternetOpenURL	Open a FTP or HTTP link.
InternetQueryDataAvailable	Zeus hooks this API to steal HTTP data of interest; SpyEye.
InternetQueryOption	Hooked by malcode for stealing online data; SpyEye.
InternetReadFile	Zeus hooks this API to steal HTTP data of interest; SpyEye.
InternetSetOption	Hooked by malcode for stealing online data; Zeus.
InternetSetStatusCallback	Zeus uses this API for HTTP authentication theft.
InternetWriteFile	Writes data to an open Internet file; Spyeye.
IsCharAlpha	A check to see if a character is alphabetical as part of possible data conversion or de-obfuscation.
IsDebuggerPresent	Checks for a debugger.
LdrLoadDll	Low level API to load a library. Hooked by Zeus to install nspr4.dll hook; SpyEye. Undocumented API.
listen	May indicate backdoor Trojan.
LoadLibrary	Used to execute malicious code or load other DLL modules and/or injection.

Ken Dunham, kend@kendunham.org

lstrcat	A function that should not be used since it can be abused for injecting executable code into a process; Bamital.
ModifyExecuteProtectionSupport	Modifications to Windows DEP security controls. Undocumented API.
Module32First	Possible module enumeration and manipulation.
Module32Last	Possible module enumeration and manipulation.
MoveFile	Moves a file, possible during installation of malcode.
NdisRegisterProtocol	Registers an NDIS driver protocol. A method to bypass firewalls used by Rustock and others.
NdrClientCall2	Transmits data to a remote server.
NeEnumerateKey	Acquires information about a certain Windows registry key.
NetpwPathCanonicalize	Conficker hooks this API to avoid reinfections from other machines.
NoExecuteAddFileOptOutList	Adds executable file into DEP exclusion list. Undocumented API.
NtCreateFile	Zeus hooks this API for Murofet file infection.
NtCreateThread	Zeus hooks this API for thread injection.
NtCreateUserProcess	Zeus hooks this API for thread injection.
NtDeviceIoControlFile	This function is called by netstat and all other ws32 applications, including winsock creation and opening of a socket. MigBot reroutes control flow from this kernel function.
NtDuplicateObject	Used to duplicate a handle in memory.
NtEnumerateValueKey	Obtains information about the value of an open Windows registry key; SpyEye.
NtLoadDriver	Loads a driver.
NtOpenProcess	Opens an existing process. May be hooked by rootkits to manage processes opened or terminated.
NtQueryDirectoryFile	Retrieves contents of a directory; Spyeye.
NtQueryInformationProcess	Checks for a debugger.
NtQuerySystemInformation	Obtain information the system.
NtQueueApcThread	Conficker enumerates all threats running inside a targeted process, adding to the queue an Asynchronous Procedure Call (APC). Leads to LoadLibraryExA to run the code. Bamital uses it for thread injection and downloading. Undocumented API.
NtResumeThread	Low-level implementation of ResumeThread related to DLL injection; SpyEye; Undocumented API.
NtVdmControl	NT Virtual DOS Machine API for working with DOS emulated program support; hooked by SpyEye. Undocumented API.
NtWriteVirtualMemory	Possible injection of a malicious component into a process.

Ken Dunham, kend@kendunham.org

ObtainUserAgentString	Malcode capturing user-agent strings from netflow data.
OpenInputDesktop	Zeus hooks this API to to use a different desktop for a VNC module.
OpenProcess	Opens a process.
OpenProcessToken	Opens an access token associated with a process.
OpenScManager	Establishes a connection to athe service control manager.
OpenThread	Opens a thread within a process.
OutputDebugString	Sends a string to a debugger; sometimes left in code by developers.
PeekMessage	Zeus hooks this API.
PeekNamedPipe	May indicate a reverse shell backdoor. Look for a hidden cmd process.
PFXImportCertStore	Rohimafo, SpyEye, Sinowal and many others hook this API to steal Internet Explorer passwords and/or certificates.
PrintWindow	Copies a visual window into the specified device context, typically a printer.
Process32First	Lists running processes.
Process32Next	Lists running processes.
PsSetCreateProcessNotifyRoutine	Manages drive-supplied callback routines related to process creation and deletion.
ReadDirectoryChanges	Malcode may monitor a thread or process and/or recreate a file if deleted or modified.
ReadProcessMemory	Stealing information from targeted process.
recv	Receives data from a socket. May indicate a backdoor Trojan or downloading of a file.
RegCloseKey	Closes a handle to a specified Windows registry key.
RegCreateKey	Creates a specified registry key.
RegDeleteKey	Deletes specified registry key.
RegDeleteValue	Deletes a named value from specified registry key.
RegEnumKey	Enumerates subkeys of a specified open registry key.
RegOpenKey	Opens a specified Windows registry key.
RegQueryValue	Retrieves data associated with a Windows registry key.
RegSetValue	Sets data and type of a Windows registry key.
RpcMgmtIsServerListening	Identify if a remote server is listening for remote procedure calls.
SeAccessCheck	Determines whether the requested access rights can be granted. MigBot reroutes control flow from this kernel function.
send	Sends data related to a socket. Zeus uses this API to exfiltrate data to a remote server; SpyEye.
sendto	Sends data to a specific destination. Conficker hooks this API as part of blocking access to security-related domains.

SetCapture	Set the mouse capture to a specified window.
SetEnvironmentalVariable	Sets variable from environmental block of calling process. SpyEye uses this function to transfer a password between two modules.
SetFileAttributes	Sets file attributes of a file, such as making it hidden to conceal it from a user.
SetKeyboardState	Keylogging.
SetPrivilege	Change privileges.
SetSfcFileException	Bamital uses this undocumented API to disable Windows File Protection of system files to inject into explorer.exe on Windows XP. Undocumented API.
SetSystemTime	Sets the current system time and date.
SetThreadContext	Possible thread injection.
SetTimer	Possible logic bomb or timed event.
SetWindowsHook	Installs and application-defined hook procedure into a hook chain. May be used by keyloggers; Bayrob.
SetWindowsText	Zeus hooks this API to steal logon data when the user is forced to enter new credentials for Full Tilt Poker.
SetWinEventHook	Sets an event hook function for a range of events. Possible keylogging or DLL injection, able to install and uninstall a Windows hook.
ShellExecute	Possibly used to run a 32-bit PE file without knowing the name of the extension, such as running an executable with an extension such as .tmp or some other proprietary assignment.
Sleep	Suspends execution of a thread for the interval (time) specified.
socket	Create a socket for netflow operations.
StartService	Starts a service.
Thread32First	Possible enumeration and manipulate of threads.
Thread32List	Possible enumeration and manipulate of threads.
Toolhelp32ReadProcessMemory	Possible theft of targeted process.
TranslateMessage	Zeus hooks this API; SpyEye.
UnhookWindowsHook	Possible keylogging or DLL injection, able to install and uninstall a Windows hook.
UnhookWinEvent	Possible keylogging or DLL injection, able to install and uninstall a Windows hook.
URLDownloadToCacheFile	Downloads data to the Internet cache, such as related to a drive-by exploitation and installation event.
URLDownloadToFile	Downloads from the Internet and saves to a file.
UrlUnescape	Unescape function to convert a URL into ordinary characters. Other strings suggest manipulation or construction of a URL from encoded data.

WinExec	Related to possible opening of a new process.
WNetUseConnection	Connection to a network resource; Bamital.
WriteFile	Creating a file, likely an installer.
WriteProcessMemory	Writes data to an area of memory in a specified process. Conficker uses this to perform thread injection.
WSASend	Sends data on a connected socket. Zeus hooks this API to steal authenticated data.
WSASocket	Creates a socket for netflow operations.
WSAStartup	Initiates use of the Winsock DLL by a process.
ZwConnectPort	TDL3 hooks this kernel level API in its own memory space and attempts to connect to a named port to establish a Local Procedure Call connection.
ZwCreateFile	Kernel level file creation possibly related to a rootkit.
ZwCreateKey	Kernel level routine to open an existing registry key.
ZwCreateLinkObject	Kernel level method to create a new link object.
ZwDeviceIoControlFile	Kernel mode control of a specific device driver.
ZwLoadDriver	Loads a device or file system driver into the currently running system.
ZwOpenFile	Opens an existing directory, device, or volume at kernel level, possibly related to a rootkit.
ZwOpenKey	Kernel level routine to open an existing registry key.
ZwOpenProcess	Kernel level routine to open a handle to a process object and set accrss rights.
ZwOpenSection	Kernel level routine to open a handle, possibly related to a rootkit.
ZwProtectVirtualMemory	Possible poisoning (API-splicing) of browser process, website redirection, and remote C&C communications. TDL3; Undocumented API.
ZwQueryDirectoryFile	Carberp uses this kernel mode control to conceal itself on a file system.
ZwQueryInformationFile	Possible rootkit retrieving kernel level information.
ZwQueryInformationPort	Possible rootkit retrieving kernel level information.
ZwQueryInformationProcess	Possible rootkit retrieving kernel level information.
ZwQuerySystemInformation	Hooked by a rootkit to manipulate a system.
ZwQuerySystemThread	Hooked by a rootkit to manipulate a system.
ZwResumeThread	Management of threads at the kernel level.
ZwSetInformationFile	Possible rootkit retrieving kernel level information.
ZwSetInformationInformation	Possible rootkit retrieving kernel level information.

ZwSetInformationPort	Possible rootkit retrieving kernel level information.
ZwSetInformationProcess	Possible rootkit retrieving kernel level information.
ZwSetInformationThread	Possible rootkit retrieving kernel level information.
ZwSetSystemInformation	Subversion of ServerLock to load a rootkit DLL. May also interact with Windows Service Control Manager (SCM) without using common method. Undocumented API.
ZwSetValueKey	Creates or replaces a registry key value.
ZwSystemDebugControl	Execute code into kernal mode; Undocumented API; Bredolab.
ZwWriteVirtualMemory	Possible poisoning (API-splicing) of browser process, website redirection, and remote C&C communications. TDL3; Undocumented API.