

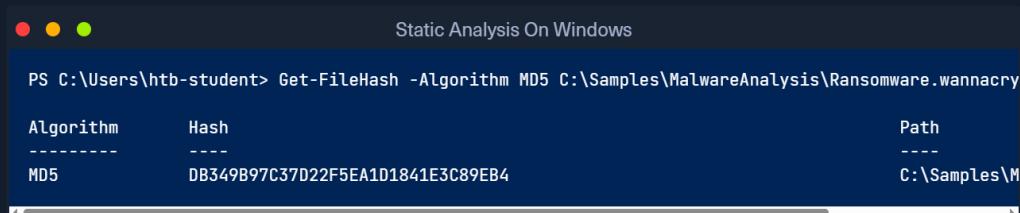


In this stage, our mission is to create a unique identifier for the malware sample. This typically takes the form of a cryptographic hash - MD5, SHA1, or SHA256.

Fingerprinting is employed for numerous purposes, encompassing:

- Identification and tracking of malware samples
- Scanning an entire system for the presence of identical malware
- Confirmation of previous encounters and analyses of the same malware
- Sharing with stakeholders as IoC (Indicators of Compromise) or as part of threat intelligence reports

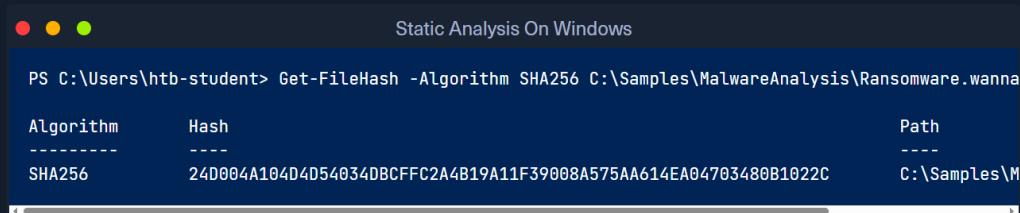
As an illustration, to check the MD5 file hash of the abovementioned malware we can use the `Get-FileHash` PowerShell cmdlet as follows.



```
PS C:\Users\htb-student> Get-FileHash -Algorithm MD5 C:\Samples\MalwareAnalysis\Ransomware.wannacry

Algorithm      Hash                                         Path
-----      ----                                         -----
MD5          DB349B97C37D22F5EA1D1841E3C89EB4
```

To check the SHA256 file hash of the abovementioned malware the command would be the following.



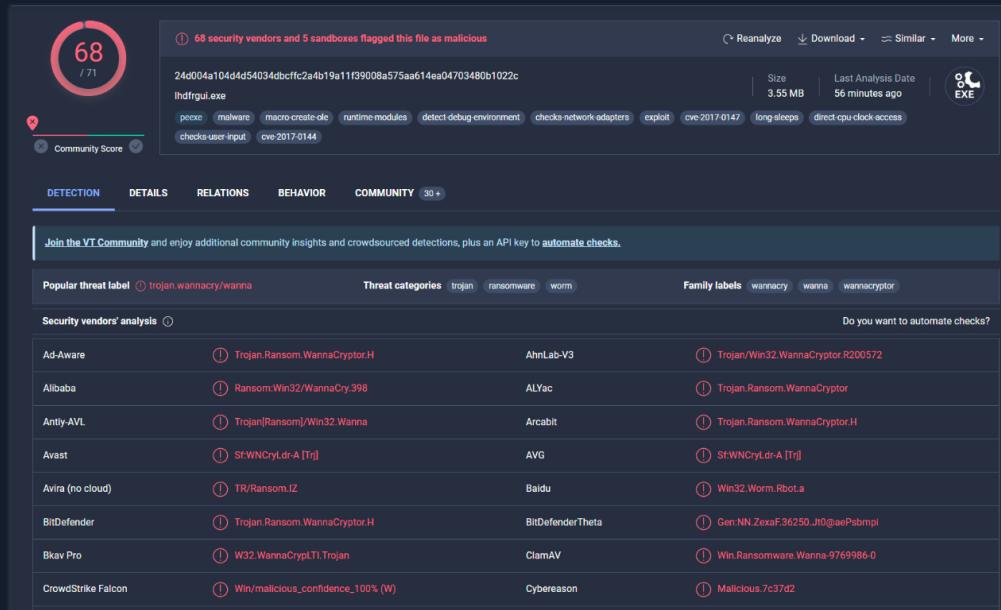
```
PS C:\Users\htb-student> Get-FileHash -Algorithm SHA256 C:\Samples\MalwareAnalysis\Ransomware.wanna

Algorithm      Hash                                         Path
-----      ----                                         -----
SHA256        24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C
```

## File Hash Lookup

The ensuing step involves checking the file hash produced in the prior step against online malware scanners and sandboxes such as Cuckoo sandbox. For instance, VirusTotal, an online malware scanning engine, which collaborates with various antivirus vendors, allows us to search for the file hash. This step aids us in comparing our results with existing knowledge about the malware sample.

The following image displays the results from [VirusTotal](#) after the SHA256 file hash of the aforementioned malware was submitted.



68 security vendors and 5 sandboxes flagged this file as malicious

lhdfgui.exe

peexe malware macro-create-dll runtime-modules detect-debug-environment checks-network-adapters exploit checks-user-input cve-2017-0147 long-sleeps direct-cpu-clock-access

Community Score

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 30+

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label	Threat categories	Family labels	
trojan.wannacry/wanna	trojan ransomware worm	wannacry wanna wannacryptor	
Security vendors' analysis		Do you want to automate checks?	
Ad-Aware	Trojan.Ransom.WannaCryptor.H	AhnLab-V3	Trojan/Win32.WannaCryptor.R200572
Alibaba	Ransom.Win32/WannaCry.398	ALYac	Trojan.Ransom.WannaCryptor
Anti-AVL	Trojan Ransom Win32.Wanna	Arcabit	Trojan.Ransom.WannaCryptor.H
Avast	St:WNCRYldr-A [Tr]	AVG	St:WNCRYldr-A [Tr]
Avira (no cloud)	TR/Ransom.IZ	Baidu	Win32.Worm.Rbot.a
BitDefender	Trojan.Ransom.WannaCryptor.H	BitDefenderTheta	Gen:NN.Zexaf.36250.Jt0@aePstbimp1
Bkav Pro	W32.WannaCry!T!Trojan	ClamAV	Win.Ransomware Wanna-9769986-0
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	CyberReason	Malicious.7c37d2

Cylance	Unsafe	Cynet	Malicious (score: 100)
Cyren	W32/Trojan.ZTSA-8671	DeepInstinct	MALICIOUS

Even though a file hash like MD5, SHA1, or SHA256 is valuable for identifying identical samples with disparate names, it falls short when identifying similar malware samples. This is primarily because a malware author can alter the file hash value by making minor modifications to the code and recompiling it.

Nonetheless, there exist techniques that can aid in identifying similar samples:

## Import Hashing (IMPHASH)

**IMPHASH**, an abbreviation for "Import Hash", is a cryptographic hash calculated from the import functions of a Windows Portable Executable (PE) file. Its algorithm functions by first converting all imported function names to lowercase. Following this, the DLL names and function names are fused together and arranged in alphabetical order. Finally, an MD5 hash is generated from the resulting string. Therefore, two PE files with identical import functions, in the same sequence, will share an **IMPHASH** value.

We can find the **IMPHASH** in the **Details** tab of the VirusTotal results.

The screenshot shows the VirusTotal analysis interface. At the top left, there's a circular icon with a '68 / 71' rating. Below it is a 'Community Score' bar. The main area has tabs for 'DETECTION' and 'DETAILS', with 'DETAILS' being active. In the 'Basic properties' section, under the 'IMPhash' row, the value '9ecee117164e0b870a53dd187cdd7174' is highlighted with a red rectangle. Other properties listed include MD5, SHA-1, SHA-256, Vhash, Authentihash, and SSDEEP.

Property	Value
MD5	db349b97c37d22f5ea1d1841e3c89eb4
SHA-1	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA-256	24d004a104d4d54034dbcfc2a4b19a11f39008a575aa614ea04703480b1022c
Vhash	036046651d6570b82201cpz31zd025z
Authentihash	1646cad4fe91337460de0d4c2c5451095023e74bdbab331642aaca12647b72f46
IMPhash	9ecee117164e0b870a53dd187cdd7174
Rich PE header	09c088bc95bf88e6f4df4d6ca904611b
hash	
SSDEEP	98304:wDqPoBhz1aRxcSUDk36SAEdhvxa9P593R8yAVp2g3R:wDqPe1Cxcxk3ZAEUadzR8yc4gb

Note that we can also use the `pefile` Python module to compute the **IMPHASH** of a file as follows.

Code: `python`

```
import sys
import pefile
import peutils

pe_file = sys.argv[1]
pe = pefile.PE(pe_file)
imphash = pe.get_imphash()

print(imphash)
```

To check the **IMPHASH** of the abovementioned WannaCry malware the command would be the following.

`imphash_calc.py` contains the Python code above.

The screenshot shows a terminal window titled 'Static Analysis On Windows'. The command entered is `C:\Scripts> python imphash_calc.py C:\Samples\MalwareAnalysis\Ransomware.wannacry.exe`. The output is the IMPhash value: `9ecee117164e0b870a53dd187cdd7174`.

## Fuzzy Hashing (SSDEEP)

**Fuzzy Hashing (SSDEEP)**, also referred to as context-triggered piecewise hashing (CTPH), is a hashing technique designed to compute a hash value indicative of content similarity between two files. This technique dissects a file into smaller, fixed-size blocks and calculates a hash for each block. The resulting hash values are then consolidated to generate the final fuzzy hash.

The **SSDEEP** algorithm allocates more weight to longer sequences of common blocks, making it highly effective in identifying files that have undergone minor modifications, or are similar but not identical, such as different variations of a malicious sample.

We can find the **SSDEEP** hash of a malware in the **Details** tab of the VirusTotal results.

We can also use the **ssdeep** tool (available at <C:\Tools\ssdeep-2.14.1>) to calculate the **SSDEEP** hash of a file. To check the **SSDEEP** hash of the abovementioned WannaCry malware the command would be the following.

Static Analysis On Windows

```
C:\Tools\ssdeep-2.14.1> ssdeep.exe C:\Samples\MalwareAnalysis\Ransomware.wannacry.exe  
ssdeep,1.1--blocksize:hash:hash,filename  
98304:wDqPoBhz1aRxcSUDk36SAEdhvxa9P593R8yAvp2g3R:wDqPe1Cxxk3ZAEUadzR8yc4gB,"C:\Samples\MalwareAna
```

① 67 security vendors and 5 sandboxes flagged this file as malicious

24d004a104d4d54034dbcfcc2a4b19a1f39008a575aa614ea04703480b1022c  
lhdfgui.exe

peexe malware macro-create-ole runtime-modules detect-debug-environment checks-network-adapte  
checks-user-input cve-2017-0144

Community Score 71

Generating SSDEEP

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 30 +

```
C:\Tools\ssdeep-2.14.1>ssdeep.exe C:\Samples\MalwareAnalysis\Ransomware.wannacry.exe  
ssdeep,1.1--blocksize:hash:hash,filename  
98304:wDqPoBhz1aRxcSUDk36SAEdhvxa9P593R8yAvp2g3R:wDqPe1Cxxk3ZAEUadzR8yc4gB,"C:\Samples\MalwareAnalysis\Ransomware.wannacry.exe"
```

Basic properties
MDS
SHA-1
SHA-256
Vhash
Authentihash
Imphash
Rich PE header
hash
SSDEEP
TLSH
File type

98304:wDqPoBhz1aRxcSUDk36SAEdhvxa9P593R8yAvp2g3R:wDqPe1Cxxk3ZAEUadzR8yc4gB  
T1B7063A8962DA1BCF0050DB044928557EBFB3C57BBA5A2FCF4045660D43B6F9BC0E61  
Win32 EXE executable windows win32 pe peexe

## Section Hashing (Hashing PE Sections)

**Section hashing**, (hashing PE sections) is a powerful technique that allows analysts to identify sections of a Portable Executable (PE) file that have been modified. This can be particularly useful for identifying minor variations in malware samples, a common tactic employed by attackers to evade detection.

The **Section Hashing** technique works by calculating the cryptographic hash of each of these sections. When comparing two PE files, if the hash of corresponding sections in the two files matches, it suggests that the particular section has not been modified between the two versions of the file.

By applying **section hashing**, security analysts can identify parts of a PE file that have been tampered with or altered. This can help identify similar malware samples, even if they have been slightly modified to evade traditional signature-based detection methods.

Tools such as **pefile** in Python can be used to perform **section hashing**. In Python, for example, you can use the **pefile** module to access and hash the data in individual sections of a PE file as follows.

Code: **python**

```
import sys  
import pefile  
pe_file = sys.argv[1]  
pe = pefile.PE(pe_file)
```

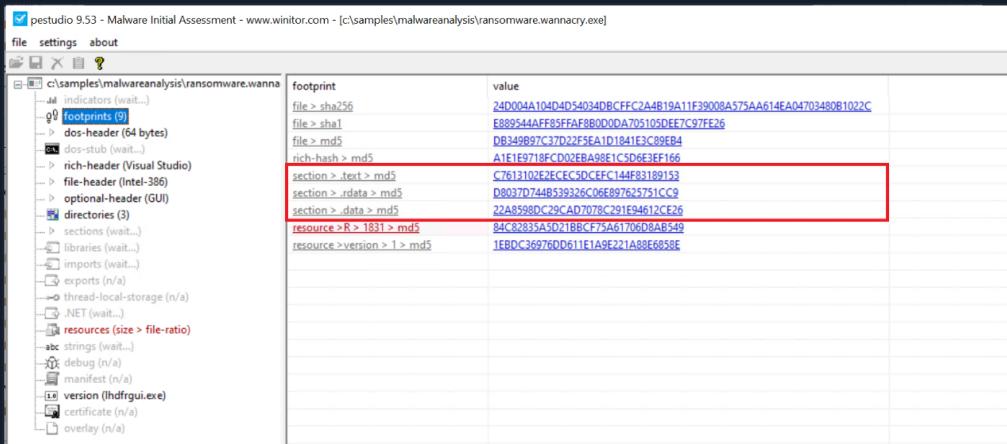
```

for section in pe.sections:
    print (section.Name, "MD5 hash:", section.get_hash_md5())
    print (section.Name, "SHA256 hash:", section.get_hash_sha256())

```

Remember that while **section hashing** is a powerful technique, it is not foolproof. Malware authors might employ tactics like section name obfuscation or dynamically generating section names to try and bypass this kind of analysis.

As an illustration, to check the MD5 file hash of the abovementioned malware we can use **pestudio** (available at [C:\Tools\pestudio\pestudio](#)) as follows.



## String Analysis

In this phase, our objective is to extract strings (ASCII & Unicode) from a binary. Strings can furnish clues and valuable insight into the functionality of the malware. Occasionally, we can unearth unique embedded strings in a malware sample, such as:

- Embedded filenames (e.g., dropped files)
- IP addresses or domain names
- Registry paths or keys
- Windows API functions
- Command-line arguments
- Unique information that might hint at a particular threat actor

The Windows **strings** binary from **Sysinternals** can be deployed to display the strings contained within a malware. For instance, the command below will reveal strings for a ransomware sample named **dharma\_sample.exe** residing in the **C:\Samples\MalwareAnalysis** directory of this section's target.

```

Static Analysis On Windows

C:\Users\htb-student> strings C:\Samples\MalwareAnalysis\dharma_sample.exe

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
gaT
Rich
.text
.rdata
.@data
HQh
9A s
9A$v
---SNIP---
GetProcAddress
LoadLibraryA
WaitForSingleObject
InitializeCriticalSectionAndSpinCount
LeaveCriticalSection
GetLastError
EnterCriticalSection

```

```
ReleaseMutex  
CloseHandle  
KERNEL32.dll  
RSDS%~m  
#ka  
C:\crysis\Release\PDB\payload.pdb  
---SNIP---
```

Occasionally, string analysis can facilitate the linkage of a malware sample to a specific threat group if significant similarities are identified. For [example](#), in the link provided, a string containing a PDB path was used to link the malware sample to the Dharma/Crysis family of ransomware.

## Strings

- 0xc814:\$s1: C:\crysis\Release\PDB\payload.pdb

It should be noted that the **FLOSS** tool is also available for Windows Operating Systems.

The command below will reveal strings for a malware sample named **shell.exe** residing in the **C:\Samples\MalwareAnalysis** directory of this section's target.

```
Static Analysis On Windows  
  
C:\Samples\MalwareAnalysis> floss shell.exe  
INFO: floss: extracting static strings...  
finding decoding function features: 100%|██████████| 85/85 [00:00  
INFO: floss.stackstrings: extracting stackstrings from 56 functions  
INFO: floss.results: AQAPRQVH1  
INFO: floss.results: JJM1  
INFO: floss.results: RAQH  
INFO: floss.results: AXAX^YZAXAYAZH  
INFO: floss.results: XAYZH  
INFO: floss.results: ws232  
extracting stackstrings: 100%|██████████|  
INFO: floss.tightstrings: extracting tightstrings from 4 functions...  
extracting tightstrings from function 0x402a90: 100%|██████████|  
INFO: floss.string_decoder: decoding strings  
emulating function 0x402a90 (call 1/1): 100%|██████████|  
INFO: floss: finished execution after 25.20 seconds
```

FLARE FLOSS RESULTS (version v2.3.0-0-g037fc4b)

file path	shell.exe
extracted strings	
static strings	254
stack strings	6
tight strings	0
decoded strings	0

### FLOSS STATIC STRINGS

FLOSS STATIC STRINGS: ASCII (254)
-----------------------------------

!This program cannot be run in DOS mode.  
.text  
P` .data  
.rdata  
`@.pdata  
QQ.xdata  
QQ.bss  
.idata  
.CRT

```
HcP<H
D$ H
AUATUWVSH
D$ L
---SNIP---
C:\Windows\System32\notepad.exe
Message
Connection sent to C2
[-] Error code is : %lu
AQAPRQVH1
JJM1
RAOH
AXAX^YZAXAYAZH
XAYZH
ws2_32
PPM1
APAPH
WWMM1
VPAPAPAPI
Windows-Update/7.6.7600.256 %s
1Lbcfr7sAHTD9CgdQo3HTMTkV8LK4ZnX71
open
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
WindowsUpdater
---SNIP---
TEMP
svchost.exe
%v%v
http://ms-windows-update.com/svchost.exe
45.33.32.156
Sandbox detected
iuderfsodp9ifjaposdfjhgosurijfaewrwegwea.com
SOFTWARE\VMware, Inc.\VMware Tools
InstallPath
C:\Program Files\VMware\VMware Tools\
Failed to open the registry key.
Unknown error
Argument domain error (DOMAIN)
Overflow range error (OVERFLOW)
Partial loss of significance (PLOSS)
Total loss of significance (TLOSS)
The result is too small to be represented (UNDERFLOW)
Argument singularity (SIGN)
_matherr(): %s in %S(%g, %g) (retval=%g)
MinGW-w64 runtime failure:
Address %p has no image-section
    VirtualQuery failed for %d bytes at address %p
    VirtualProtect failed with code 0x%x
    Unknown pseudo relocation protocol version %d.
    Unknown pseudo relocation bit size %d.
 pdata
RegCloseKey
RegOpenKeyExA
RegQueryValueExA
RegSetValueExA
CloseHandle
CreateFileA
CreateProcessA
CreateRemoteThread
DeleteCriticalSection
EnterCriticalSection
GetComputerNameA
GetCurrentProcess
GetCurrentProcessId
GetCurrentThreadId
GetLastError
GetStartupInfoA
GetSystemTimeAsFileTime
GetTickCount
InitializeCriticalSection
LeaveCriticalSection
OpenProcess
QueryPerformanceCounter
RtlAddFunctionTable
RtlCaptureContext
RtlLookupFunctionEntry
RtlVirtualUnwind
SetUnhandledExceptionFilter
Sleep
TerminateProcess
TlsGetValue
UnhandledExceptionFilter
VirtualAllocEx
VirtualProtect
```

```
VirtualQuery
WriteFile
WriteProcessMemory
__C_specific_handler
__getmainargs
__initenv
__iob_func
__lconv_init
__set_app_type
__setusermatherr
_acmdln
_amsg_exit
_cexit
_fmode
_initterm
_onexit
_vsnprintf
abort
calloc
exit
fprintf
free
fwrite
getenv
malloc
memcpy
printf
puts
signal
sprintf
strcmp
strlen
strncmp
vfprintf
ShellExecuteA
MessageBoxA
InternetCloseHandle
InternetOpenA
InternetOpenUrlA
InternetReadFile
WSACleanup
WSAStartup
closesocket
connect
freeaddrinfo
getaddrinfo
htonl
inet_addr
socket
ADVAPI32.dll
KERNEL32.dll
msvcrt.dll
SHELL32.dll
USER32.dll
WININET.dll
WS2_32.dll
```

```
+-----+
| FLOSS STATIC STRINGS: UTF-16LE (0) |
+-----+
```

---

#### FLOSS STACK STRINGS

---

```
AQAPRQVH1
JJM1
RAQH
AXAX^YZAXAYAZH
XAYZH
ws232
```

---

#### FLOSS TIGHT STRINGS

---

## Unpacking UPX-packed Malware

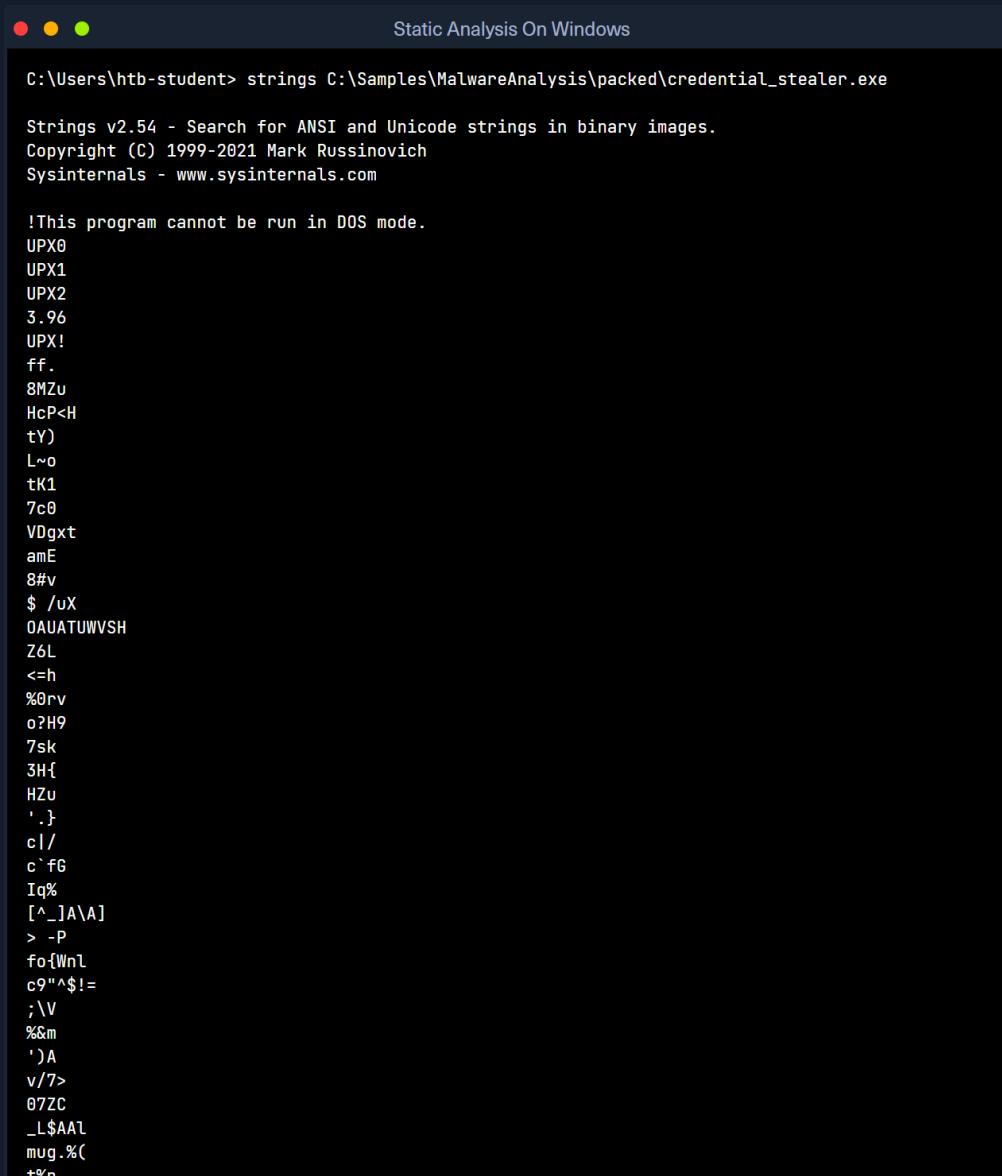
In our static analysis, we might stumble upon a malware sample that's been compressed or obfuscated using a technique referred to as packing. Packing serves several purposes:

- It obfuscates the code, making it more challenging to discern its structure or functionality.
- It reduces the size of the executable, making it quicker to transfer or less conspicuous.
- It confounds security researchers by hindering traditional reverse engineering attempts.

This can impair string analysis because the references to strings are typically obscured or eliminated. It also replaces or camouflages conventional PE sections with a compact loader stub, which retrieves the original code from a compressed data section. As a result, the malware file becomes both smaller and more difficult to analyze, as the original code isn't directly observable.

A popular packer used in many malware variants is the [Ultimate Packer for Executables \(UPX\)](#).

Let's first see what happens when we run the `strings` command on a UPX-packed malware sample named `credential_stealer.exe` residing in the `C:\Samples\MalwareAnalysis\packed` directory of this section's target.



```
● ● ● Static Analysis On Windows
C:\Users\htb-student> strings C:\Samples\MalwareAnalysis\packed\credential_stealer.exe

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.
UPX0
UPX1
UPX2
3.96
UPX!
ff.
8MZu
HcP<H
tY)
L~o
tK1
7c0
VDgxt
amE
8#v
$ /uX
0AUATUWVSH
Z6L
<=h
%0rv
o?H9
7sk
3H{
HZu
'.}
c|/
c fG
Iq%
[^_]A\A]
> -P
fo{Wnl
c9"^\$!=
;\V
%Sm
')A
v/7>
07zC
_L$AA1
mug.%(
t%
```

```
#8%,X  
e]`^  
(hk  
Dks  
zC:  
Vj<  
w~5  
m<6  
|$PD  
c(t  
\3_  
---SNIP---
```

Observe the strings that include UPX, and take note that the remainder of the output doesn't yield any valuable information regarding the functionality of the malware.

```
C:\Tools>strings credential_stealer.exe
```

Strings v2.54 - Search for ANSI and Unicode strings in binary images.  
Copyright (C) 1999-2021 Mark Russinovich  
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.

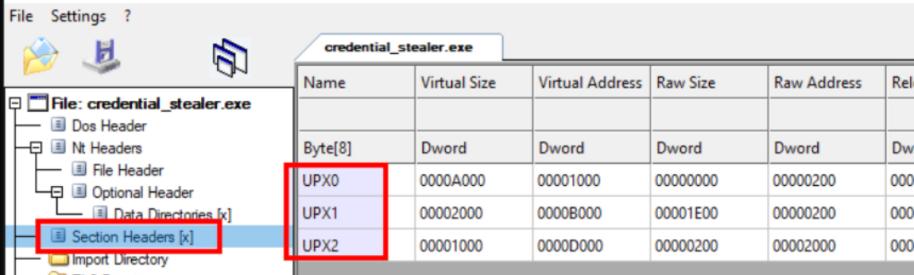
JPX0  
JPX1  
JPX2

4.00  
UPX!  
ff.  
8Mzu  
HcP<H  
tY)  
L~o  
tK1  
7c0  
VDgxt  
amE  
8#v  
\$ /uX

File Settings ?

credential\_stealer.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc
Byte[8]	Dword	Dword	Dword	Dword	Dword
UPX0	0000A000	00001000	00000000	00000200	00000
UPX1	00002000	0000B000	00001E00	00000200	00000
UPX2	00001000	0000D000	00000200	00002000	00000



We can unpack the malware using the UPX tool (available at <C:\Tools\upx\upx-4.0.2-win64>) with the following command.

```
Static Analysis On Windows
```

```
C:\Tools\upx\upx-4.0.2-win64> upx -d -o unpacked_credential_stealer.exe C:\Samples\MalwareAnalysis\  
Ultimate Packer for eXecutables  
Copyright (C) 1996 - 2023  
UPX 4.0.2 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 30th 2023  
  
File size Ratio Format Name  
-----  
16896 <- 8704 51.52% win64/pe unpacked_credential_stealer.exe  
  
Unpacked 1 file.
```

Let's now run the strings command on the unpacked sample.

```
Static Analysis On Windows
```

```
C:\Tools\upx\upx-4.0.2-win64> strings unpacked_credential_stealer.exe
```

Strings v2.54 - Search for ANSI and Unicode strings in binary images.  
Copyright (C) 1999-2021 Mark Russinovich  
Sysinternals - www.sysinternals.com

!This program cannot be run in DOS mode.

.text  
P` .data  
.rdata  
`@.pdata  
QQ.xdata  
QQ.bss  
.idata  
.CRT  
.tls  
ff.  
8Mzu  
HcP<H

```
--SNIP--  
D$(  
D$  
D$0  
D$(  
D$  
t'H  
%5T  
@A\A]A`  
SeDebugPrivilege  
SE Debug Privilege is adjusted  
lsass.exe  
Searching lsass PID  
Lsass PID is: %lu  
Error is - %lu  
lsassmem.dmp  
LSASS Memory is dumped successfully  
Err 2: %lu  
@u@  
`p@  
Unknown error  
Argument domain error (DOMAIN)  
Overflow range error (OVERFLOW)  
Partial loss of significance (PLOSS)  
Total loss of significance (TLOSS)  
The result is too small to be represented (UNDERFLOW)  
Argument singularity (SIGN)  
_matherr(): %s in %s(%g, %g) (retval=%g)  
Mingw-w64 runtime failure:  
Address %p has no image-section  
    VirtualQuery failed for %d bytes at address %p  
    VirtualProtect failed with code 0x%  
    Unknown pseudo relocation protocol version %d.  
    Unknown pseudo relocation bit size %d.  
.pdata  
@@  
@@@  
`E@  
`E@  
@v@  
hy@  
`y@  
@p@  
@v@  
Pp@  
AdjustTokenPrivileges  
LookupPrivilegeValueA  
OpenProcessToken  
MiniDumpWriteDump  
CloseHandle  
CreateFileA  
CreateToolhelp32Snapshot  
DeleteCriticalSection  
EnterCriticalSection  
GetCurrentProcess  
GetCurrentProcessId  
GetCurrentThreadId  
GetLastError  
GetStartupInfoA  
GetSystemTimeAsFileTime  
GetTickCount  
InitializeCriticalSection  
LeaveCriticalSection  
OpenProcess  
Process32First  
Process32Next  
QueryPerformanceCounter  
RtlAddFunctionTable  
RtlCaptureContext  
RtlLookupFunctionEntry  
RtlVirtualUnwind  
SetUnhandledExceptionFilter  
Sleep  
TerminateProcess  
TlsGetValue  
UnhandledExceptionFilter  
VirtualProtect  
VirtualQuery  
__C_specific_handler  
__getmainargs  
__initenv  
__job_func  
__lconv_init  
__set_app_type
```

```
__setusermatherr
_acmdln
_amsg_exit
_cexit
_fmode
_initterm
_onexit
abort
calloc
exit
fprintf
free
fwrite
malloc
memcpy
printf
puts
signal
strcmp
strlen
strncmp
vfprintf
ADVAPI32.dll
dbghelp.dll
KERNEL32.DLL
msvcrt.dll
```

Now, we observe a more comprehensible output that includes the actual strings present in the sample.

#### VPN Servers

**⚠ Warning:** Each time you "Switch", your connection keys are regenerated and you must re-download your VPN connection file.

All VM instances associated with the old VPN Server will be terminated when switching to a new VPN server.

Existing PwnBox instances will automatically switch to the new VPN server.

US Academy 3

Medium Load

#### PROTOCOL

UDP 1337    TCP 443

[DOWNLOAD VPN CONNECTION FILE](#)



#### Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

#### Pwnbox Location

UK

163ms

**!** Terminate Pwnbox to switch location

[Start Instance](#)

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions  

## Questions

Answer the question(s) below to complete this Section and earn cubes!



Download VPN  
Connection File

Target(s): Click here to spawn the target system!

 RDP to with user "htb-student" and password "HTB\_academy\_stdnt!"

+ 1  In the C:\Samples\MalwareAnalysis directory of this section's target, there is a file called dharma\_sample.exe. Calculate the file's SHA256 hash and enter it as your answer.

BFF6A1000A86F8EDF3673D576786EC75B80BED0C458A8CA0BD52D12B74099071

 Submit

 Previous

Next 

 Mark Complete & Next

Powered by 