THE UNIVERSITY of NEW ORLEANS

CSCI 4621/5621 Intro to CyberSecurity

04: MALWARE (MALICIOUS SOFTWARE)

Vassil Roussev

vassil@cs.uno.edu

READING: Oorschot [ch7]

Definition

- Software **intentionally** designed or deployed to have effects contrary to the best interests of one or more users (or system owners or administrators), including potential damage related to resources, devices, or other systems.
  - » *Damage might involve data, software, hardware, or compromise of privacy.*
  - » *Given full knowledge, most users would (if given a choice) **not** allow it to run.*
- *Harmful software:*
  - » *a more general term, which also includes software that may cause harm without explicit intent.*
  - » ***not** the primary focus in this course*

Malware concerns

- Attack vectors
  - *How does the malware get placed onto a computer system?*
  - *worms, viruses, drive-by-downloads, etc.*
- Detection problems
  - *What makes malware hard to detect?*
  - *to some degree, it is an issue of malware definition*
    - e.g., is adware malware?
- Installation
  - *How does malware get installed?*
  - *users are often the weak link, and often have the final authorization decision*
  - *ease of update has a downside when it comes to malware installs*

VIRUSES & WORMS

- "A program that can infect other programs or files by modifying them to include a possibly evolved copy of itself." --F. Cohen
  - » *a virus replicates by spreading to other programs*
  - » *user actions can help spread it to other systems*
    - e.g., USB drives, network shares, etc.

Computer virus	Computer worm

loop
  remain_dormant_until_host_runs();
  propagate_with_user_help();
  if trigger_condition_true() then
  run_payload();
endloop;
	loop
	  propagate_over_network();
	  if trigger_condition_true() then
	  run_payload();
	endloop

[CREDIT: Oorschot]

Overall structure

- Dormancy
  - » *a virus is typically dormant until the host program runs.*
- Propagation
  - » *when/how the malware spreads*
- Trigger condition
  - » *controls when the payload is executed*
- Payload
  - » *functionality delivered by the malware (other than propagation)*
  - » *actions range from relatively benign (scare/amuse the user) to severe destruction, even bricking of hardware*

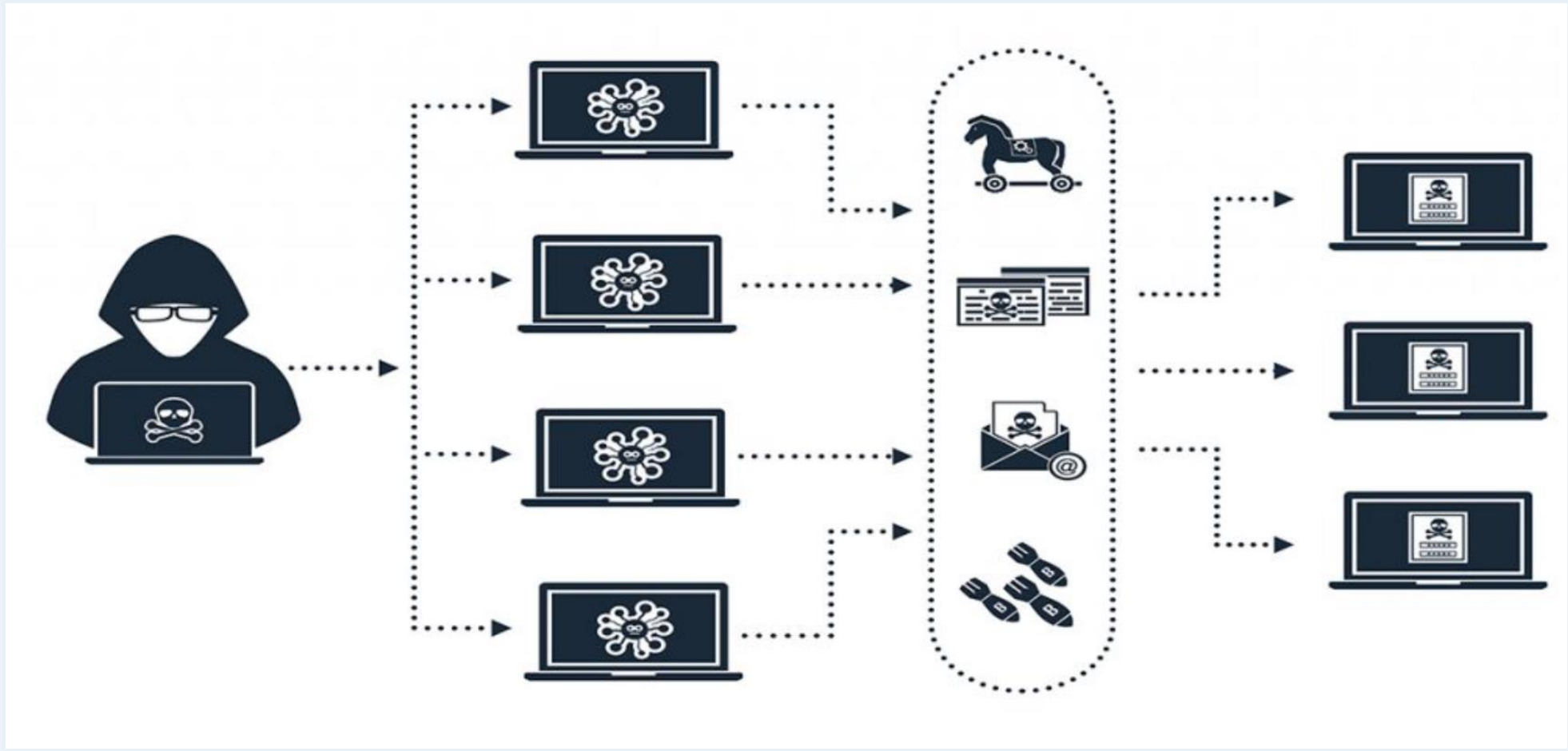
The worm difference

- Worms propagate automatically and continuously,

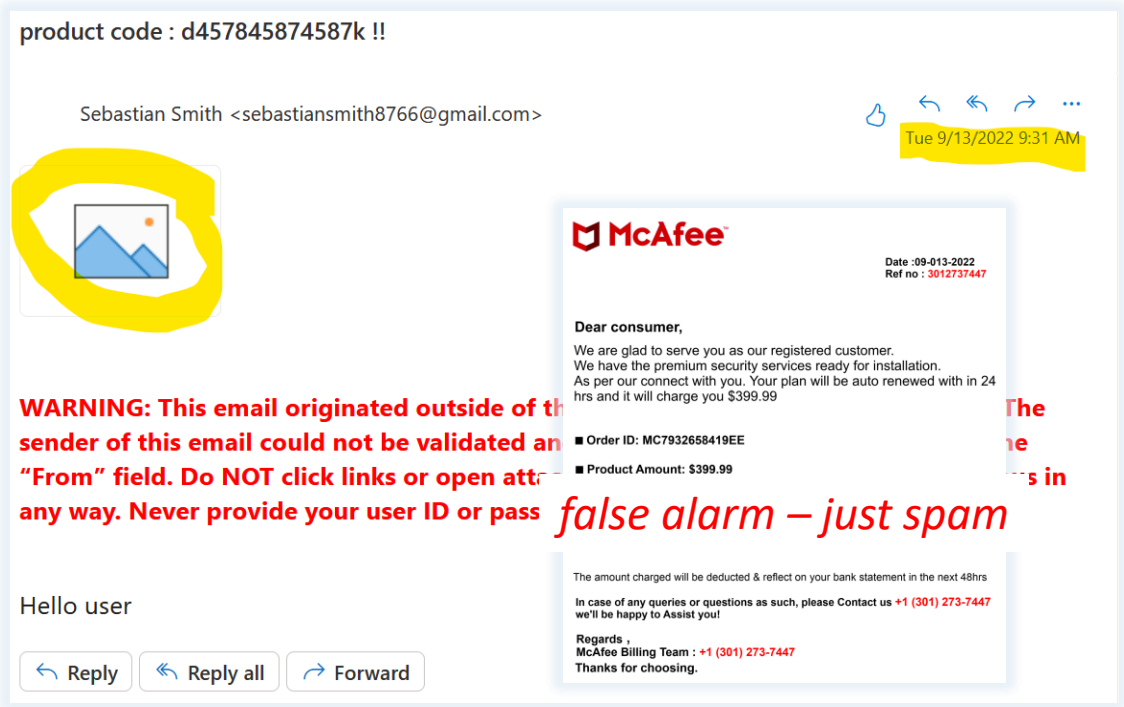
  without user interaction.

- Worms spread across machines over networks, leveraging network protocols and network daemons
  - » *rather than infecting host programs beforehand as viruses do*
- Worms exploit software vulnerabilities
  - » *e.g., buffer overflows*
  - » *viruses tend to abuse software features or use social engineering*
- Aka, *network worms/network viruses*
  - » *uncontrolled spread may cause DoS, e.g., 1988 Morris worm*

THE BUSINESS OF MALWARE – BOTNETS

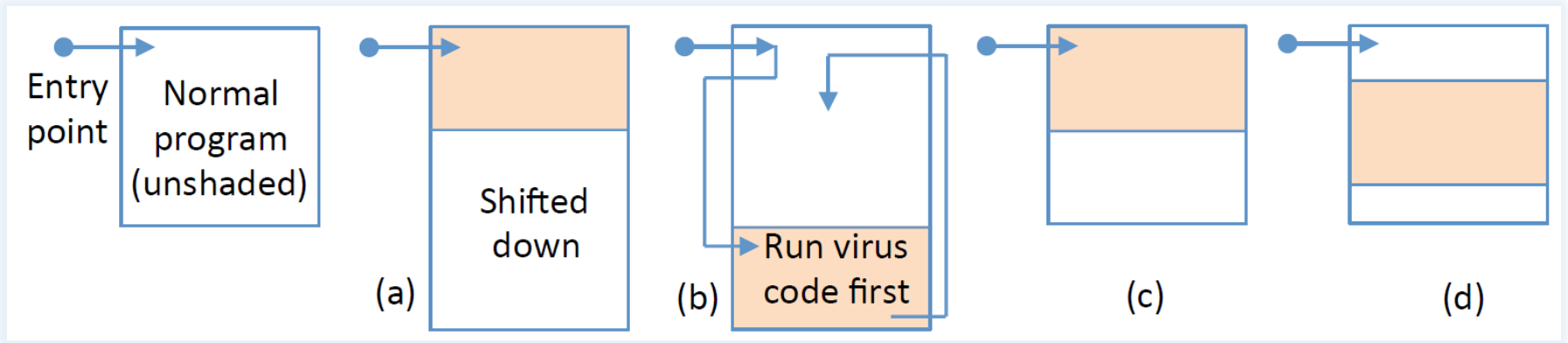


EMAIL-BASED MALWARE



- Spreads through email-related file infection, attachments, and features of clients and infrastructure (often enabled by default)
- It typically requires a user action
  - » *e.g., opening an email message*
  - » *and may involve social engineering*
- A common tactic is to extract next-targets from the mail client's address book.
  - » *since email allows long recipient lists, spreading is one-to-many*

PROGRAM FILE VIRUSES—INFECTION STRATEGIES



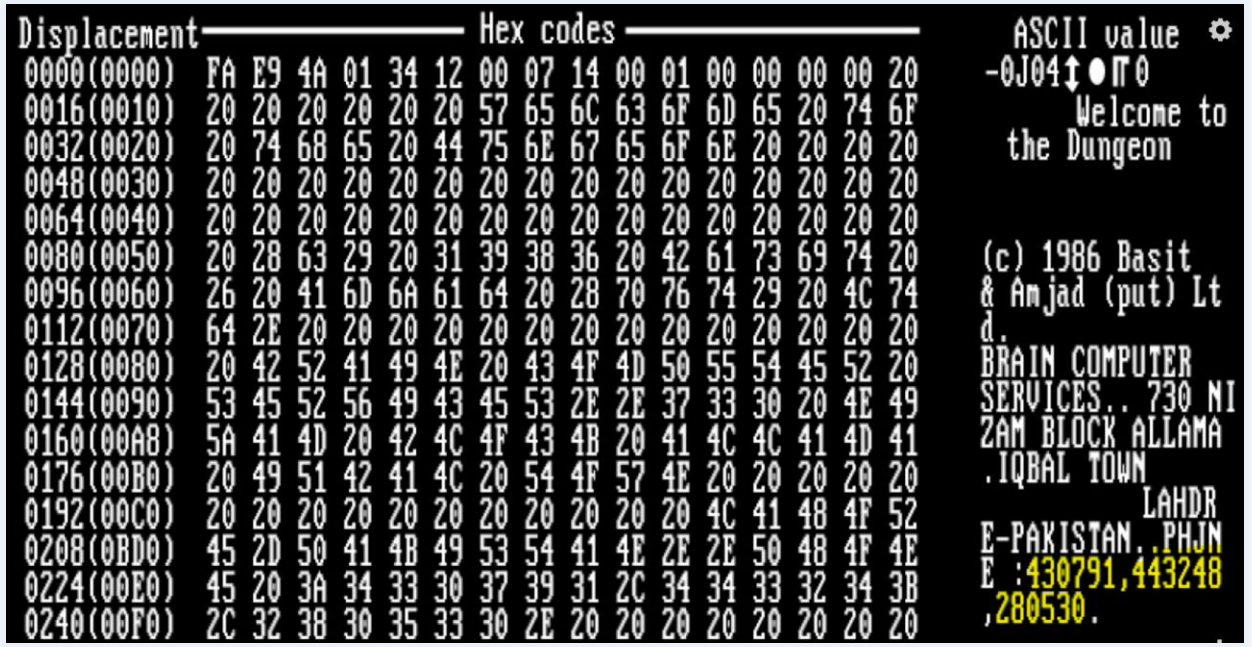
[CREDIT: Oorschot]

a) Shift & prepend
b) Append to the end
c) Overwrite host (beginning)
d) Overwrite host (interior)

Note: variations include using temp file, *shell script viruses*, etc.

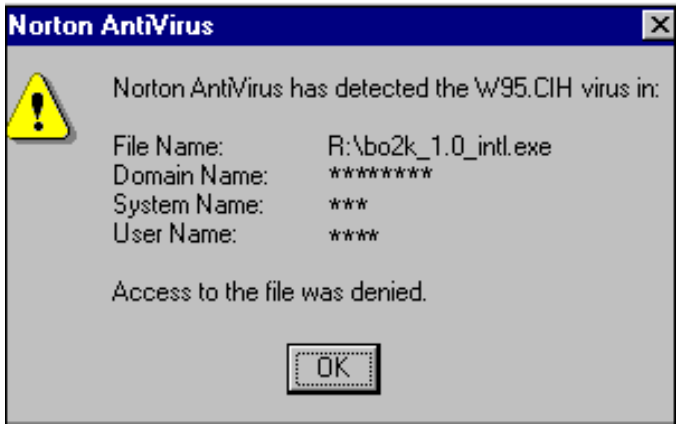
BRAIN VIRUS (1986)

- Considered first PC virus
- Bootkit – boot sector virus
  - » *ensured that it ran first, before OS*
- Occasionally destroyed FAT
  - » *resulting in data los*
- Stealthy
  - » *interrupt hooking ensured that MBR looked normal*
    - on an infected system
- Honest self-attribution!!



[CREDIT: [Wikipedia]]

CIH Chernobyl virus (1998-2000)



[CREDIT: [Wikipedia]]

- Targets Windows 95/98/ME
- Destructive
  - » *overwrites partition table*
  - » *FAT*
  - » *attempts BIOS overwrite (!)*
    - succeeding on some systems
- Used filesystem slack space for storages
  - » *i.e., does not overwrite file but lives in the padding*
  - » *spreads across multiple files, if needed*
- At peak, widespread damage, especially in Asia
  - » *cannot spread on Windows NT-based systems*

Data file viruses

- Macro/scripting viruses
  - » *document format supports code execution*
    - e.g., MS Word, Adobe PDF
  - » *capability meant to bring convenience*
  - » *in practice, an attack vector that is difficult to secure*
- Data used to trigger software bugs
  - » *document needs to be interpreted*
  - » *which may lead to a variety of buffer overflow/memory management attacks*

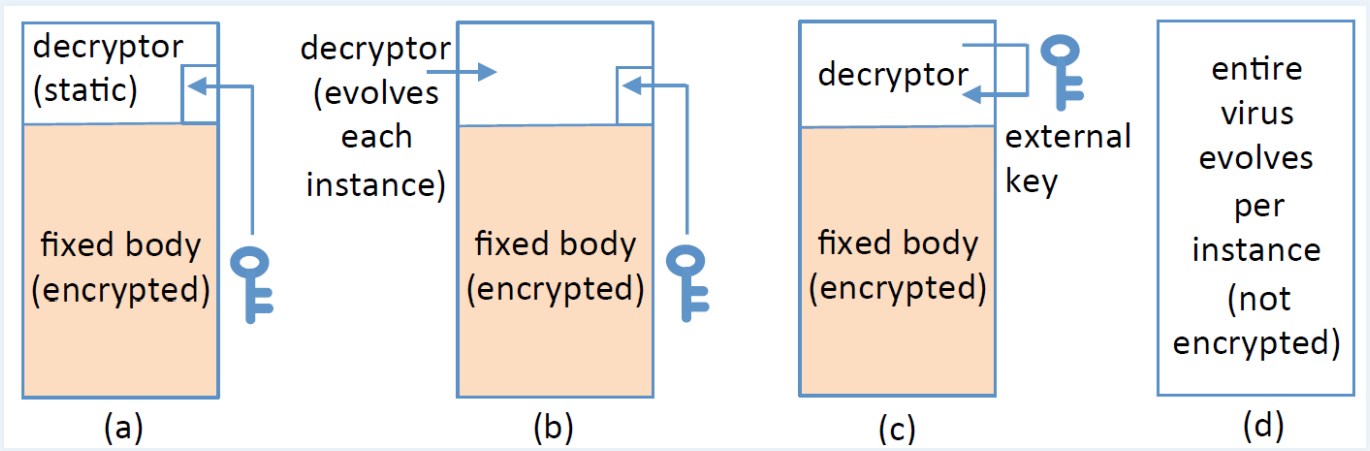
VIRUS DETECTION & THE *HALTING PROBLEM*

- "There exists a Turing machine whose halting problem is recursively unsolvable."
- That is,
  - *it is provably impossible for one program to be able to determine if another program will terminate*
- Corollaries
  - *we cannot hope to have **the** antivirus program*
  - *we can only hope to solve **known** special cases (known malware)*
  - *this leads to a **move-countermove game** with increasing complexity*
    - paradoxically, better AV leads to better malware!

PRACTICAL MALWARE DETECTION

- Byte sequence signatures
  - » *bread-and-butter method*
  - » *fast, but only works for known malware*
  - » *easy to foil*
- Integrity verification
  - » *focuses on ensuring the integrity of known-good executables*
  - » *many techniques, some with hardware support*
  - » *attempt to close off entire attack vectors*
- Behavioral signature
  - » *rationale: malware behaves differently*
    - e.g., unusual system call sequences
  - » *problems*
    - eliciting the behavior, differentiating from normal behavior, false positives

VIRUS ANTI-DETECTION TECHNIQUES



[CREDIT: Oorschot]

a) Packing/encryption
   - » *simple: XOR scheme w/ fixed key*
   - » *complex: partial JIT decryption, random checksums, dummy code, etc.*
   - » *see* ref/Silver Needle in the Skype -- BlackHat EU (Biondi, 2006).pdf

b) Polymorhism
   - » *self-mutation + decryptor*

c) External decryption key
   - » *e.g., filesystem, network*

d) Metamorphism
   - » *mutate source and recompile*

SPREADING TECHNIQUES

- Auto-rooter
  - » *scans for vulnerable targets*
  - » *immediately attacks & and installs backdoor, rootkit, etc.*
- Localized scanning
  - » *e.g., Code Red II (2001) – prefer LAN targets*
    - presumably, they are more likely to be similarly vulnerable
- Context-aware scanning
  - » *use local info to pick targets*

Fast spread techniques

- Hit-list
  - » *rationale: identify global vulnerable population*
    - and maximize early spread
- Permutation scanning
  - » *split target address list among running instances*
    - avoid reconnecting with an infected host
- Internet hit-list
  - » *compile global list of vulnerable hosts*
    - e.g., google dorks

The 1988 Internet (Morris) worm

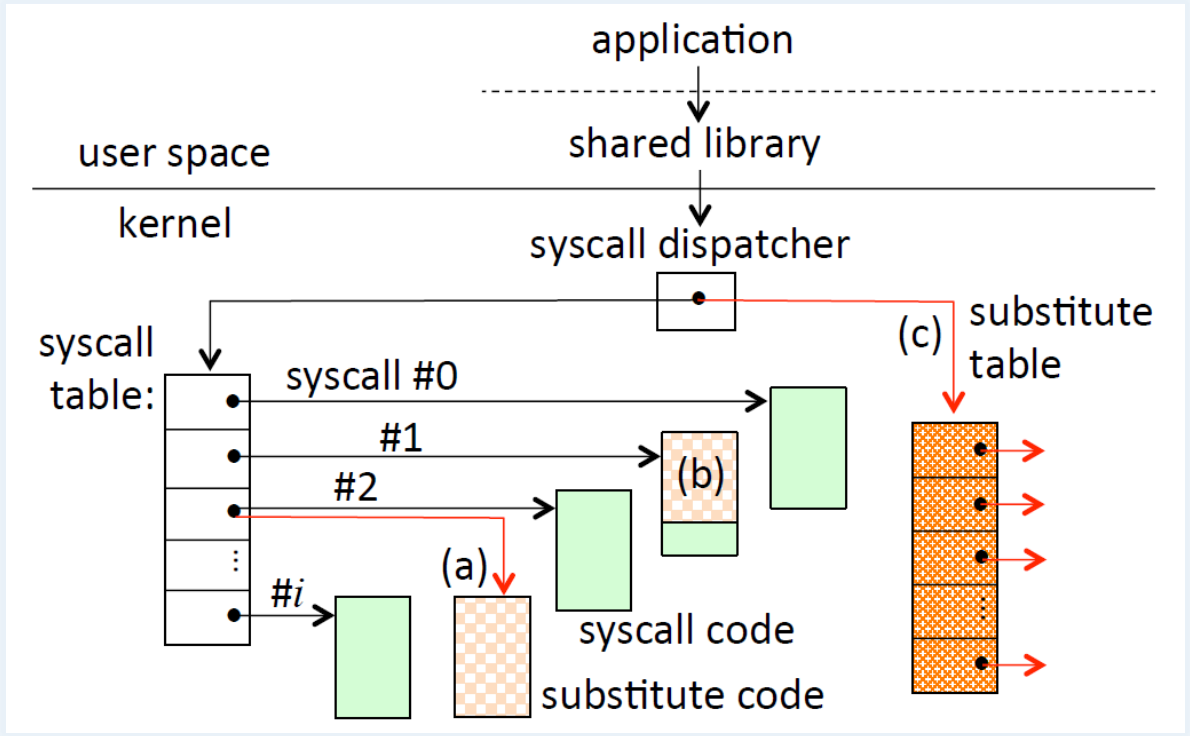
- No malicious payload
  - » *spread infected 10% of the internet & caused DoS*
- Vulnerabilities exploited
  - » *stack buffer overrun in* fingerd
  - » *backdoor-like* DEBUG *command in* sendmail
  - » *password-guessing attack using* /etc/passwd
    - + rexec with acquired credentials
  - » *abuse of trusted remote logins through* /etc/hosts.equiv *using* rsh
- Upon success, downloaded
  - » *binaries*
  - » *source code*
    - to be compiled on the victim machine

Stealth & deception

- Trojan
  - » *appearance does not match content*
- Backdoor
  - » *authorization bypass, arbitrary code execution*
- Rootkit (user/kernel)
  - » *controls host, conceals installed malware*
  - » *typical functions*
    - backdoor
    - cloaking (via system call hooking)
    - surveillance – files, keylogging, camera, mic, etc.

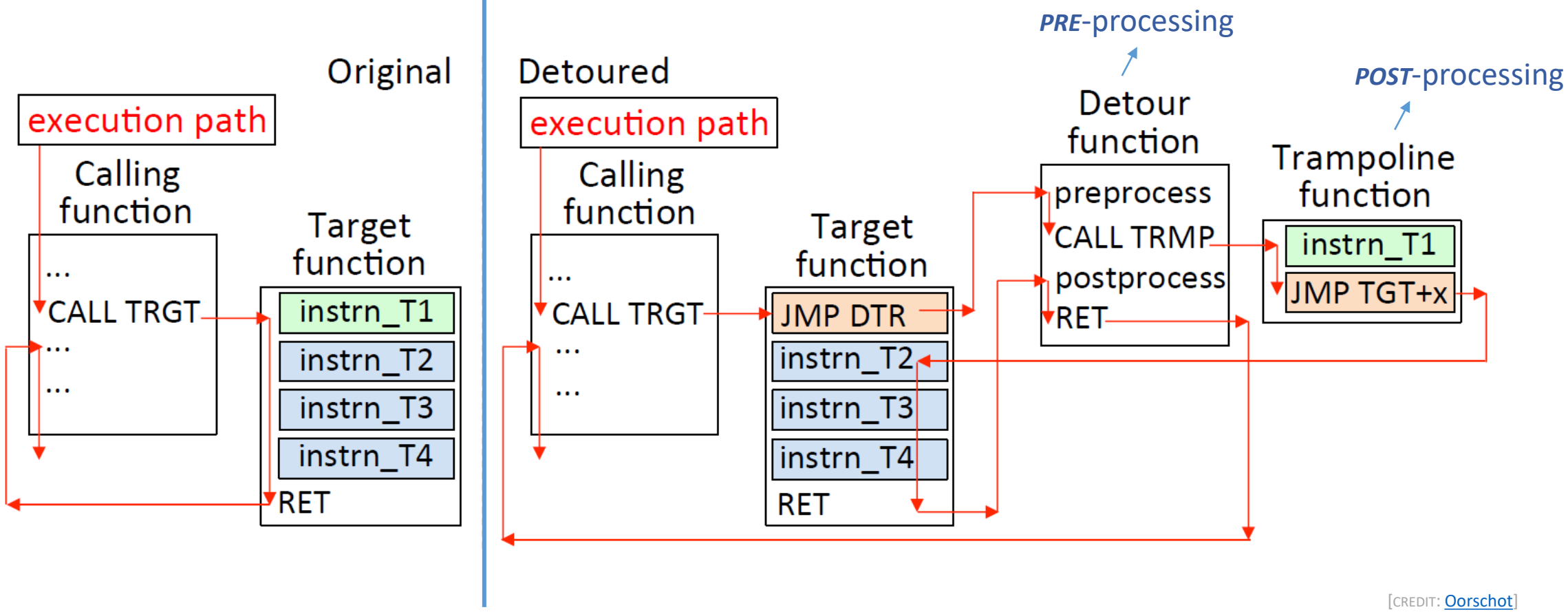
Rootkits

- System call hijacking
  a) hooking individual system call
  b) overwriting individual system call
  c) hooking the entire syscall table by using a substitute table



[CREDIT: Oorschot]

INLINE HOOKING

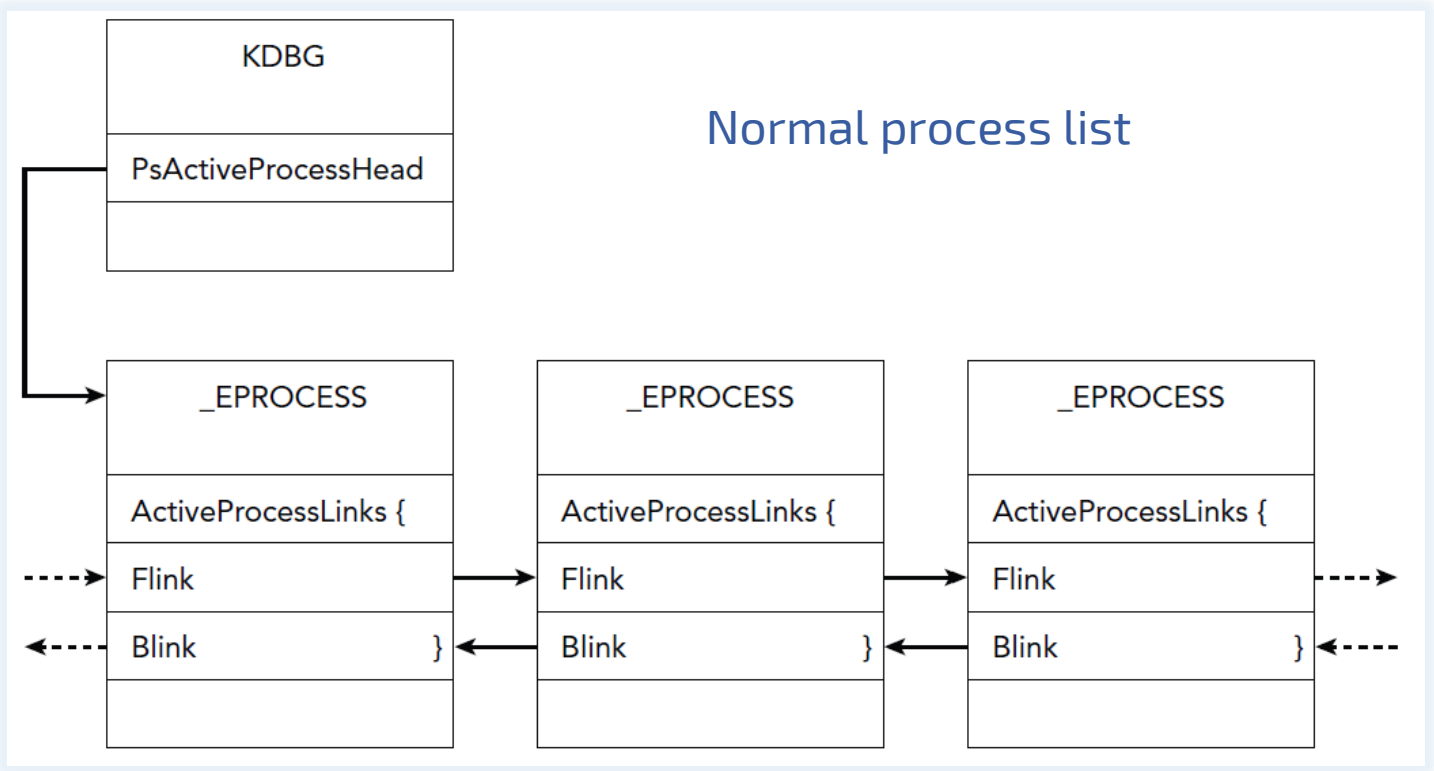


[CREDIT: Oorschot]

DIRECT KERNEL OBJECT MANIPULATION (DKOM)

- Modify kernel data structure (usually) to hide something
  - » *e.g., hiding a process (Windows)*

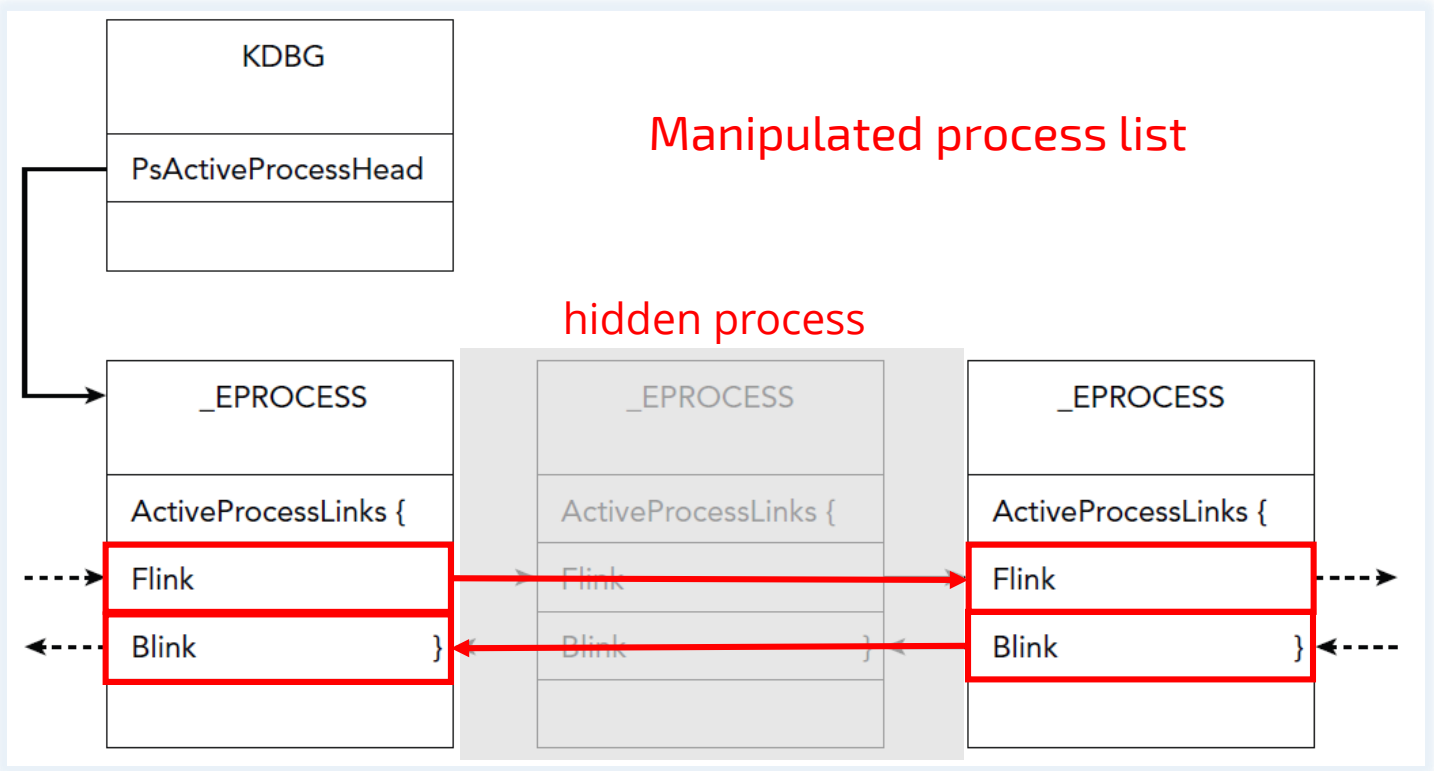
```
'_EPROCESS' (1232 bytes)
0x0 : Pcb ['_KPROCESS']
0x168 : CreateTime ['WinTimeStamp', {'is_utc': True}]
0x170 : ExitTime ['WinTimeStamp', {'is_utc': True}]
0x178 : RundownProtect ['_EX_RUNDOWN_REF']
0x180 : UniqueProcessId ['unsigned int']
0x188 : ActiveProcessLinks ['_LIST_ENTRY']
0x1e0 : SessionProcessLinks ['_LIST_ENTRY']
0x200 : ObjectTable ['pointer64', ['_HANDLE_TABLE']]
0x208 : Token ['_EX_FAST_REF']
0x290 : InheritedFromUniqueProcessId ['unsigned int']
0x2d8 : Session ['pointer64', ['void']]
0x2e0 : ImageFileName ['String', {'length': 16}]
0x308 : ThreadListHead ['_LIST_ENTRY']
0x328 : ActiveThreads ['unsigned long']
0x338 : Peb ['pointer64', ['_PEB']]
0x448 : VadRoot ['_MM_AVL_TABLE']
```



DIRECT KERNEL OBJECT MANIPULATION (DKOM)

- Modify kernel data structure (usually) to hide something
  - » *e.g., hiding a process (Windows)*

```
'_EPROCESS' (1232 bytes)
0x0 : Pcb ['_KPROCESS']
0x168 : CreateTime ['WinTimeStamp', {'is_utc': True}]
0x170 : ExitTime ['WinTimeStamp', {'is_utc': True}]
0x178 : RundownProtect ['_EX_RUNDOWN_REF']
0x180 : UniqueProcessId ['unsigned int']
0x188 : ActiveProcessLinks ['_LIST_ENTRY']
0x1e0 : SessionProcessLinks ['_LIST_ENTRY']
0x200 : ObjectTable ['pointer64', ['_HANDLE_TABLE']]
0x208 : Token ['_EX_FAST_REF']
0x290 : InheritedFromUniqueProcessId ['unsigned int']
0x2d8 : Session ['pointer64', ['void']]
0x2e0 : ImageFileName ['String', {'length': 16}]
0x308 : ThreadListHead ['_LIST_ENTRY']
0x328 : ActiveThreads ['unsigned long']
0x338 : Peb ['pointer64', ['_PEB']]
0x448 : VadRoot ['_MM_AVL_TABLE']
```



Installation methods

- Via standard kernel module mechanism
  - *get user to approve (social engineering)*
  - *use stolen key to sign the code*
  - *loadable kernel modules (LKM)*
- Exploit vulnerability
  - *e.g. in a daemon with root privs*
- Hijack boot process
  - *might even attack BIOS*
- Attack swapped out kernel memory
- Direct memory access (DMA)
  - *Firewire, Thunderbolt, USB*

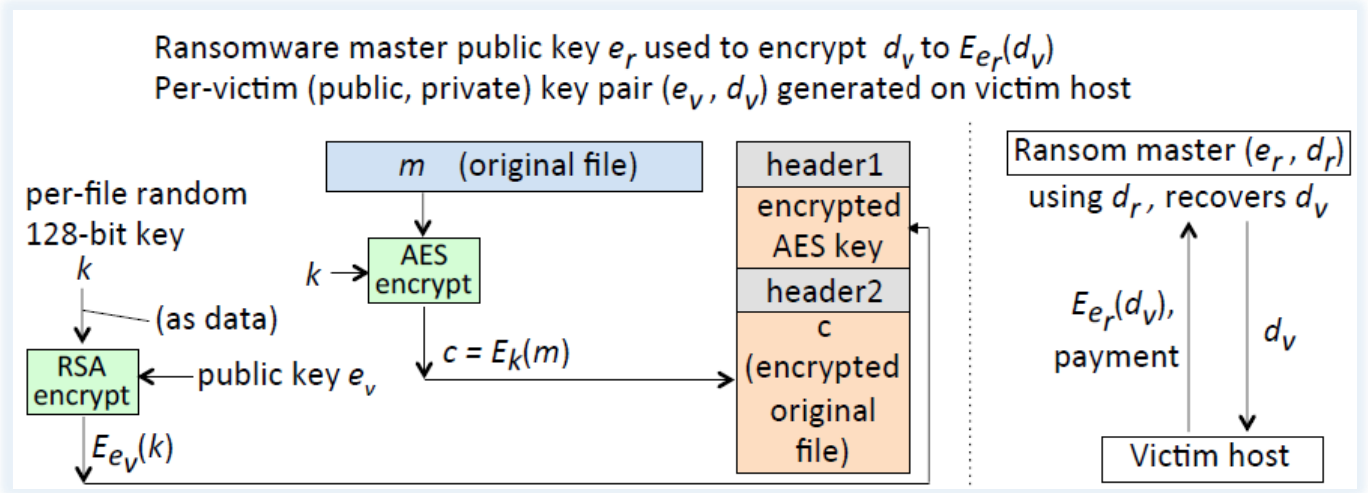
RANSOMWARE

- History
  - *1989 emergence*
    - AIDS trojan
  - *2005-09 the early (simple) years*
    - unsophisticated use of cryptography
  - *2009-13*
    - from scareware to real crypto
  - *2013-15*
    - ransomware dominance
  - *2016-18*
    - RaaS
    - NotPetya
  - *2018-19*
    - integration w/ other malware
  - *2019-*
    - leak sites (doxing)

[CREDIT: Oorschot]

RANSOMWARE LOCKOUT

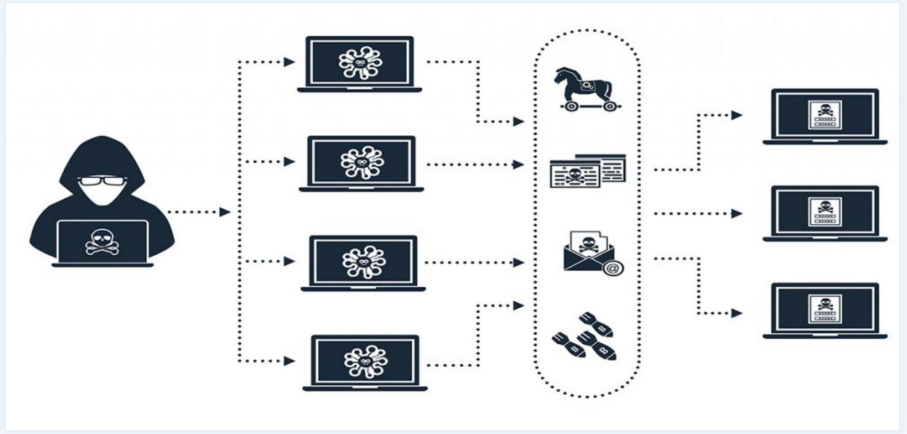
- Modern crypto use
  - » *public + private encryption*
  - » *per-machine key*
- Non-encryption methods
  - » *standard AC mechanisms*
  - » *disabling of OS functions*
  - » *rootkit*
  - » *exfiltration of files*
    - threat of doxing



WannaCry file-locking mechanism

[CREDIT: Oorschot]

Botnets



- Compromised host known as
  - » *bots, or zombies*
  - » *can be commanded to perform arbitrary action*
- Provide an infrastructure
  - » *to be rented out*
- Command & control (C2)
  - » *centralized (e.g., IRC chat)*
    - vulnerable to takedown
  - » *peer-to-peer (p2p)*
    - multi-tiered, more resilient
    - compromised DNS resolution, DNS fastflux

Zero-day exploits

- Zero-day
  - » *an attack that exploits a previously unknown (to the public) vulnerability*
    - i.e., victim has had zero days of notice to remedy the vulnerability
  - » *highly valued/expensive*
    - a grey/black market is known to exist
- Sophisticated attack may employ multiple of these
  - » *e.g., STUXNET used four zero-days*
- Also employed for forensic purposes
  - » *e.g., mobile phone data acquisition*

Social engineering

- Convince/deceive users into installing malware
  - » *e.g.: Happy99 → run attached executable*
- Often misuse usability features, e.g.:
  - » *hiding of file extension*
  - » *double-click to open*
  - » *preview*
    - could trigger downloads and/or client vulnerability exploits
- Also, lesser-known features
  - » *e.g., self-extracting zip file*

Malware classification

- By objective
  - *damage to host and/or its data*
    - DoS, cyberweapons
  - *data theft*
    - espionage, doxing
  - *direct financial gain*
    - e.g., sell fake products/services
  - *long-term surveillance*
  - *malware spread*
  - *resource control*
    - e.g. Bitcoin mining

MALWARE CLASSIFICATION [2]

- By technical attributes
  - *self-replication/breeding*
  - *need for host executable*
  - *stealth*
  - *attack vectors*
  - *insider help*
  - *level of persistency*

Category name	Property (blank denotes: no)			
	BREEDS†	HOSTED	STEALTHY	VECTOR
virus	✓	✓		U
worm	✓			N
Trojan horse		✓	✓	E or S
backdoor		maybe	✓	T or S
rootkit, keylogger			✓	T or S
ransomware				T
drive-by download	★		✓	S

[CREDIT: Oorschot]

CHASING MALWARE IN MEMORY

appendix

ATTACKS ON ENVIRONMENT VARIABLES

- Search order hijacking via PATH
  - PATH=C:\windows;C:\windows\system32
  - PATH=C:\Users\HR101\.tmp;C:\windows;C:\windows\system32
- via PATHEXT
  - PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE
  - PATHEXT=.ZZZ;.COM;.EXE;.BAT;.CMD;.VBS;.VBE

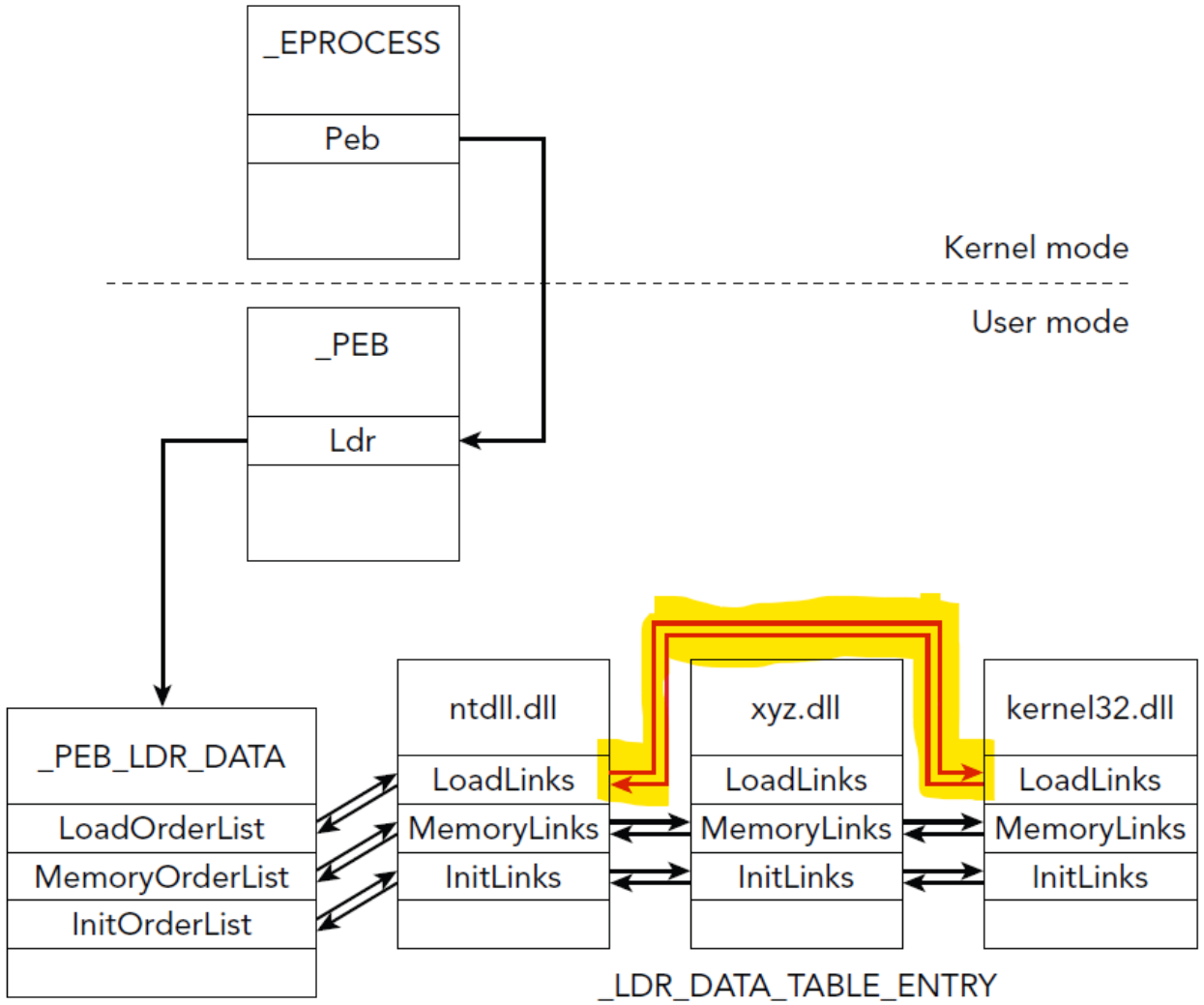
EX: *COREFLOOD* PRESENCE MARKING

```
> vol -f coreflood.vmem --profile WinXPSP3x86 envars -p 2044
Volatility Foundation Volatility Framework 2.6
Pid      Process              Block      Variable                       Value
-------- -------------------- ---------- ------------------------------ -----
    2044 IEXPLORE.EXE         0x00010000 ALLUSERSPROFILE                C:\Documents and Settings\All Users
    2044 IEXPLORE.EXE         0x00010000 APPDATA                        C:\Documents and Settings\Administrator\App
    2044 IEXPLORE.EXE         0x00010000 CLIENTNAME                     Console
    2044 IEXPLORE.EXE         0x00010000 CommonProgramFiles             C:\Program Files\Common Files
    2044 IEXPLORE.EXE         0x00010000 COMPUTERNAME                   BILLY-DB5B96DD3
    2044 IEXPLORE.EXE         0x00010000 ComSpec                        C:\WINDOWS\system32\cmd.exe
    2044 IEXPLORE.EXE         0x00010000 FP_NO_HOST_CHECK               NO
    2044 IEXPLORE.EXE         0x00010000 GIEVMXDVLMISML                 EWONSYG
    2044 IEXPLORE.EXE         0x00010000 HOMEDRIVE                      C:
    2044 IEXPLORE.EXE         0x00010000 HOMEPATH                       \Documents and Settings\Administrator
    2044 IEXPLORE.EXE         0x00010000 LOGONSERVER                    \\BILLY-DB5B96DD3
    2044 IEXPLORE.EXE         0x00010000 NUMBER_OF_PROCESSORS           1
    2044 IEXPLORE.EXE         0x00010000 OS                             Windows_NT
    2044 IEXPLORE.EXE         0x00010000 Path                           C:\Program Files\Internet Explorer;;C:\WIND
    2044 IEXPLORE.EXE         0x00010000 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF
    2044 IEXPLORE.EXE         0x00010000 PROCESSOR_ARCHITECTURE         x86
    2044 IEXPLORE.EXE         0x00010000 PROCESSOR_IDENTIFIER           x86 Family 6 Model 23 Stepping 10, GenuineI
    2044 IEXPLORE.EXE         0x00010000 PROCESSOR_LEVEL                6
    2044 IEXPLORE.EXE         0x00010000 PROCESSOR_REVISION             170a
    2044 IEXPLORE.EXE         0x00010000 ProgramFiles                   C:\Program Files
    2044 IEXPLORE.EXE         0x00010000 SESSIONNAME                    Console
    2044 IEXPLORE.EXE         0x00010000 SystemDrive                    C:
    2044 IEXPLORE.EXE         0x00010000 SystemRoot                     C:\WINDOWS
    2044 IEXPLORE.EXE         0x00010000 TEMP                           C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp
    2044 IEXPLORE.EXE         0x00010000 TMP                            C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp
    2044 IEXPLORE.EXE         0x00010000 USERDOMAIN                     BILLY-DB5B96DD3
    2044 IEXPLORE.EXE         0x00010000 USERNAME                       Administrator
    2044 IEXPLORE.EXE         0x00010000 USERPROFILE                    C:\Documents and Settings\Administrator
    2044 IEXPLORE.EXE         0x00010000 windir                         C:\WINDOWS
```

ANALYZING DLLs

- DLLs contain code & resources that can be shared among multiple processes
  - » *frequently used by malware because*
    - DLLs are designed to run inside a host process,
    - giving them access to all of the process' resources
      - its threads, handles, and full range of process memory.
    - DLLs allow toolkits to be modular and extensible
- Anomalies
  - » *list discrepancies:*
    - unlinking of metadata structures from one or more lists
  - » *unexpected filenames/paths*
    - e.g.: `C:\Windows\system32\sys\kernel32.dll`
  - » *context mismatch*
    - `ws2_32.dll/crypt32.dll/hnetcfg.dll/pstorec.dll` →networking/cryptography/firewall/storage
      - does the process **really** need those?

HIDING DLLs


_EPROCESS
Peb
Kernel mode
User mode
_PEB
Ldr
_PEB_LDR_DATA
LoadOrderList
MemoryOrderList
InitOrderList
ntdll.dll
xyz.dll
kernel32.dll
LoadLinks
MemoryLinks
InitLinks
_LDR_DATA_TABLE_ENTRY

EXAMPLE: dlllist

```
$ python vol.py -f mem.dmp --profile=WinXPSP3x86 dlllist -p 3108

notepad.exe pid:   3108
Command line : "C:\WINDOWS\system32\notepad.exe"
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000    0x14000     0xffff C:\WINDOWS\system32\notepad.exe
0x7c900000    0xb2000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x77dd0000    0x9b000     0xffff C:\WINDOWS\system32\ADVAPI32.dll
0x77fe0000    0x11000     0xffff C:\WINDOWS\system32\Secur32.dll
0x77c10000    0x58000     0xffff C:\WINDOWS\system32\msvcrt.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
0x7e410000    0x91000     0xffff C:\WINDOWS\system32\USER32.dll
0x7c9c0000   0x817000     0xffff C:\WINDOWS\system32\SHELL32.dll
<snip>
0x7e1e0000    0xa2000        0x3 C:\WINDOWS\system32\urlmon.dll
0x771b0000    0xaa000        0x3 C:\WINDOWS\system32\WININET.dll
0x77a80000    0x95000        0x3 C:\WINDOWS\system32\CRYPT32.dll
0x71ab0000    0x17000       0x27 C:\WINDOWS\system32\WS2_32.dll
0x71a50000    0x3f000        0x4 C:\WINDOWS\system32\mswsock.dll
0x662b0000    0x58000        0x1 C:\WINDOWS\system32\hnetcfg.dll
0x76f20000    0x27000        0x1 C:\WINDOWS\system32\DNSAPI.dll
```

DETECTING UNLINKED DLLS

- PE file scanning
  - » *brute force scan for PE headers (MZ)*
  - » *can readily be overwritten*
- VAD cross-referencing → ldrmodules
  - » *looks for large nodes with PAGE_EXECUTE_WRITECOPY protections, VadImageMap type, and the Image control flag set*
  - » *compares the starting addresses from the VAD nodes with the DllBase value from the _LDR_DATA_TABLE_ENTRY structures*
  - » *entries identified through the VAD that are **not** represented in the DLL lists are potentially hidden*

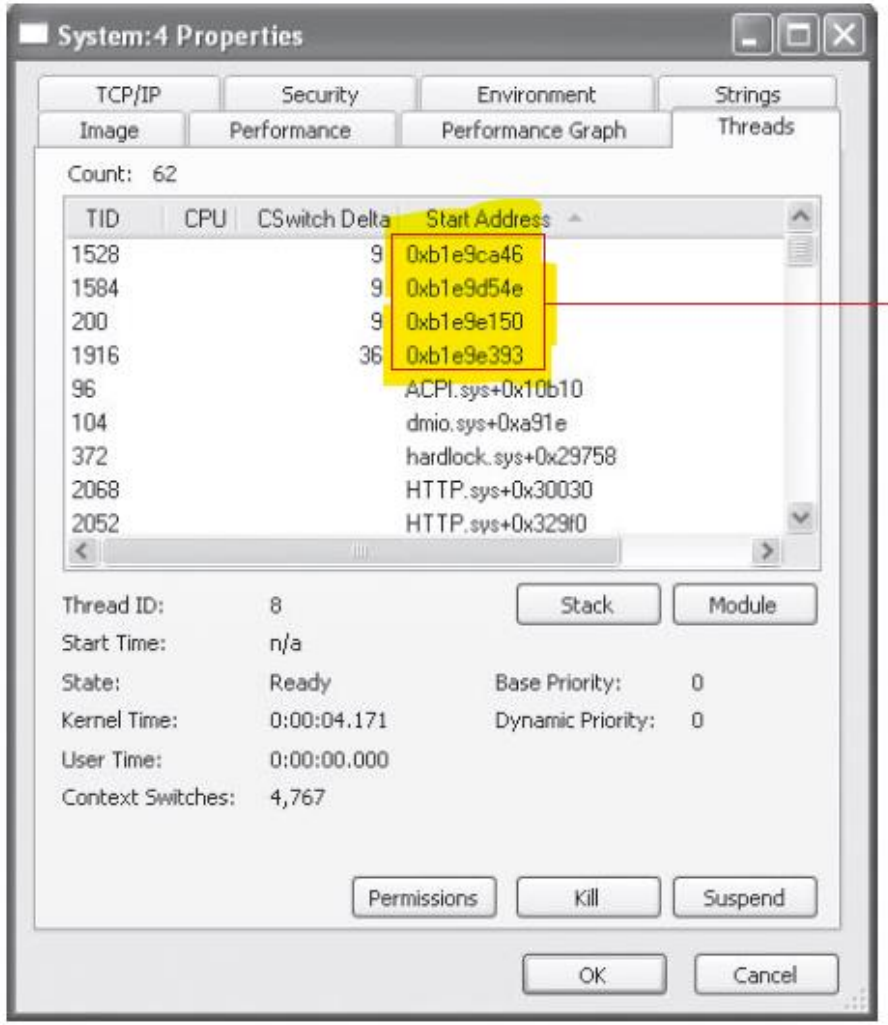
ldrmodules

```
$ python vol.py -f memory.dmp --profile=Win7SP1x64 ldrmodules -p 616
Volatility Foundation Volatility Framework 2.4
Process      Base                 InLoad InInit InMem MappedPath
------------ ------------------ ------ ------ ----- ----------
svchost.exe  0x0000000074340000 True   True   True  \Windows\[snip]\sfc.dll
svchost.exe  0x00000000779a0000 True   True   True  \Windows\[snip]\ntdll.dll
svchost.exe  0x000007feff570000 False  False  False \Windows\[snip]\lpkz2.dll
svchost.exe  0x0000000077780000 True   True   True  \Windows\[snip]\kernel32.dll
svchost.exe  0x000007fefd990000 True   True   True  \Windows\[snip]\msasn1.dll
svchost.exe  0x000007fefbbe0000 True   True   True  \Windows\[snip]\wtsapi32.dll
svchost.exe  0x000007fefdac0000 True   True   True  \Windows\[snip]\KernelBase.dll
svchost.exe  0x000007fefcc00000 True   True   True  \Windows\[snip]\gpapi.dll
svchost.exe  0x000007fefb800000 True   True   True  \Windows\[snip]\ntmarta.dll
svchost.exe  0x000007fefcc20000 True   True   True  \Windows\[snip]\userenv.dll
svchost.exe  0x000007fefbd60000 True   True   True  \Windows\[snip]\xmllite.dll
svchost.exe  0x000007feff460000 True   True   True  \Windows\[snip]\oleaut32.dll
svchost.exe  0x000007fefde70000 True   True   True  \Windows\[snip]\urlmon.dll
svchost.exe  0x000007fef9290000 True   True   True  \Windows\[snip]\wscapi.dll
[snip]
svchost.exe  0x00000000ff720000 True   False  True  \Windows\[snip]\svchost.exe
```

NO PE SIGNATURE – SHELLCODE

```
$ python vol.py -f carberp.mem --profile=WinXPSP3x86 malfind
Volatility Foundation Volatility Framework 2.4
[snip]

Process: svchost.exe Pid: 992 Address: 0x9d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x009d0000  b8 35 00 00 00 e9 8b d1 f3 7b 68 6c 02 00 00 e9   .5.......{hl....
0x009d0010  94 63 f4 7b 8b ff 55 8b ec e9 6c 11 e4 7b 8b ff   .c.{..U...l..{..
0x009d0020  55 8b ec e9 99 2e 84 76 8b ff 55 8b ec e9 74 60   U......v..U...t`
0x009d0030  7f 76 8b ff 55 8b ec e9 8a e9 7f 76 8b ff 55 8b   .v..U......v..U.

0x9d0000 b835000000       MOV EAX, 0x35
0x9d0005 e98bd1f37b       JMP 0x7c90d195
0x9d000a 686c020000       PUSH DWORD 0x26c
0x9d000f e99463f47b       JMP 0x7c9163a8
0x9d0014 8bff             MOV EDI, EDI
0x9d0016 55               PUSH EBP
```

THREAD IoC EXAMPLE: **TIGGER**



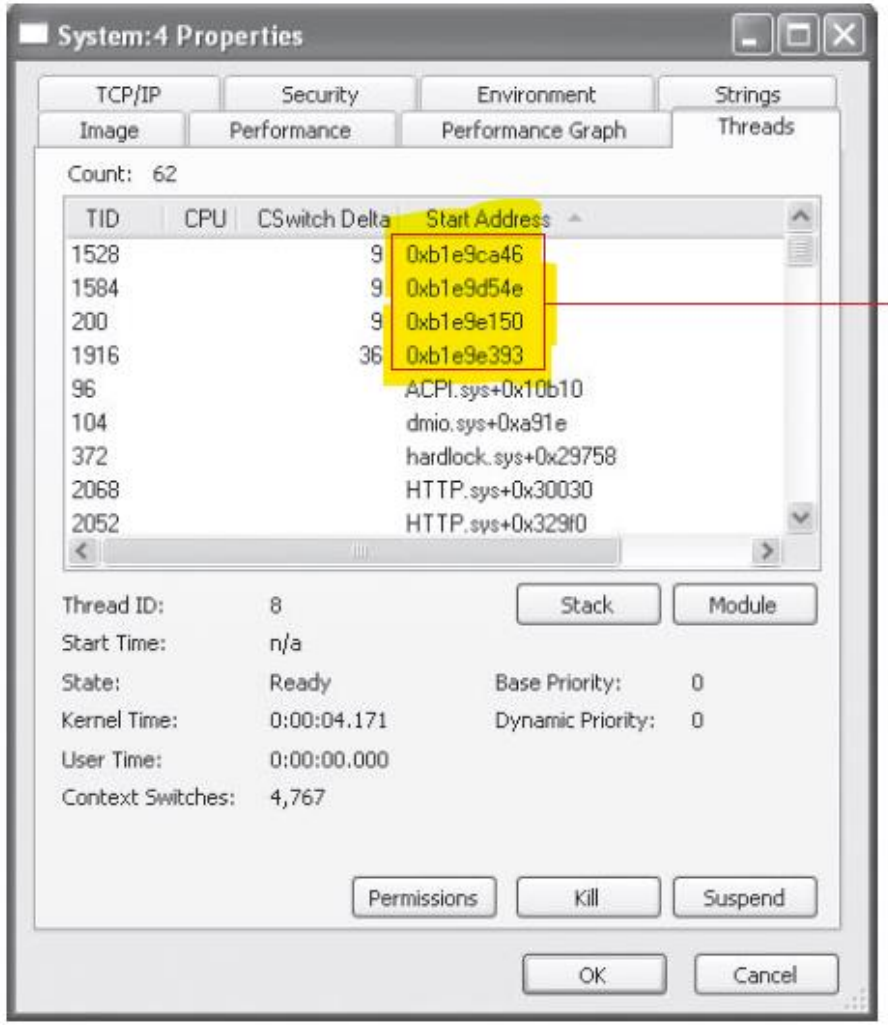
Four new threads
with no known driver

ORPHAN THREAD DETECTION

```
$ python vol.py -f orphan.vmem threads -F OrphanThread
     --profile=WinXPSP3x86
[snip]

ETHREAD: 0xff1f92b0 Pid: 4 Tid: 1648
Tags: OrphanThread,SystemThread
Created: 2010-08-15 19:26:13
Exited: 1970-01-01 00:00:00
Owning Process: System
Attached Process: System
State: Waiting:DelayExecution
BasePriority: 0x8
Priority: 0x8
TEB: 0x00000000
StartAddress: 0xf2edd150 UNKNOWN
ServiceTable: 0x80552180
  [0] 0x80501030
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_SYSTEM
```

- Detection can be bypassed by patching `_ETHREAD.StartAddress`

THREAD IoC EXAMPLE: **TIGGER**



Four new threads
with no known driver

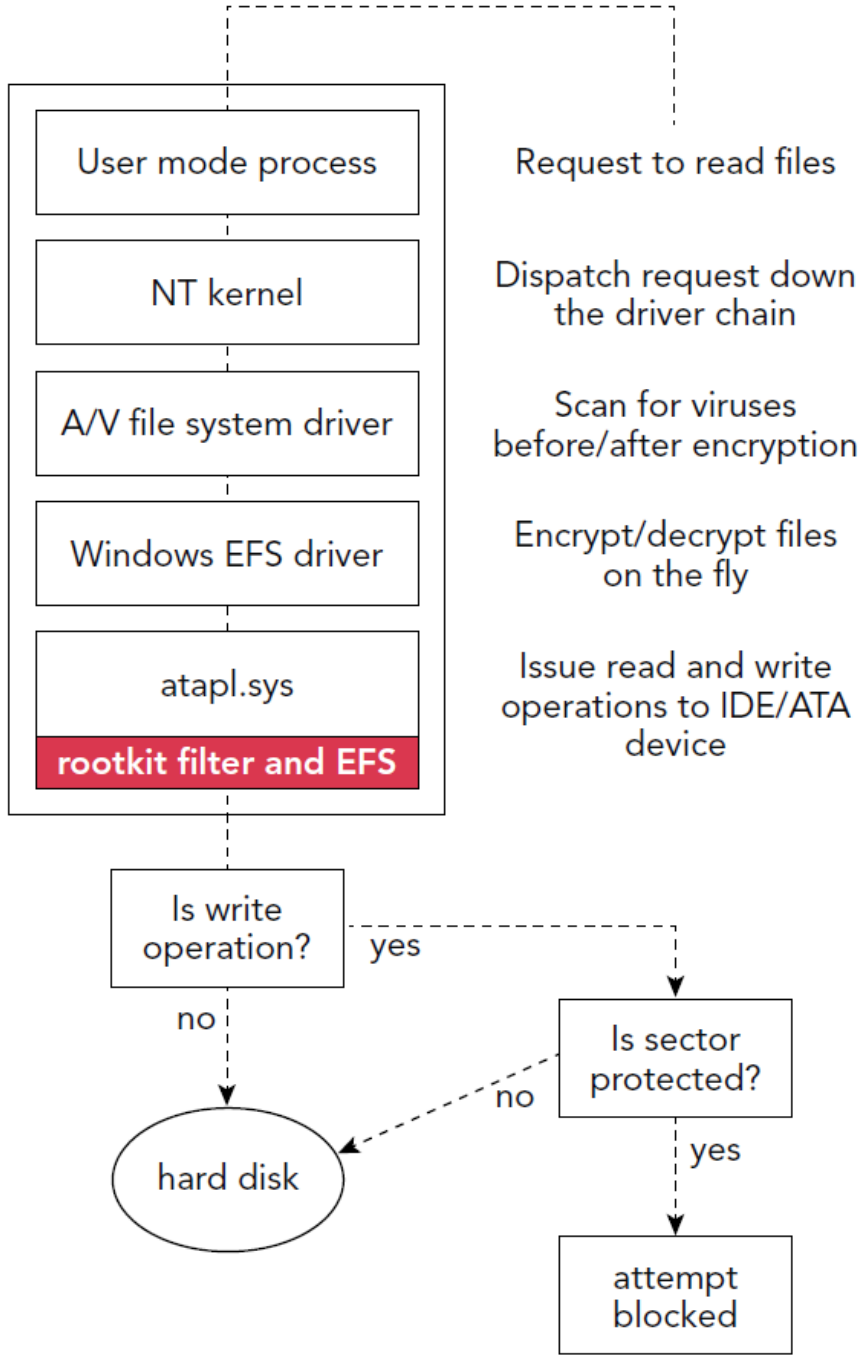
ORPHAN THREAD DETECTION

```
$ python vol.py -f orphan.vmem threads -F OrphanThread
     --profile=WinXPSP3x86
[snip]

ETHREAD: 0xff1f92b0 Pid: 4 Tid: 1648
Tags: OrphanThread,SystemThread
Created: 2010-08-15 19:26:13
Exited: 1970-01-01 00:00:00
Owning Process: System
Attached Process: System
State: Waiting:DelayExecution
BasePriority: 0x8
Priority: 0x8
TEB: 0x00000000
StartAddress: 0xf2edd150 UNKNOWN
ServiceTable: 0x80552180
  [0] 0x80501030
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_SYSTEM
```

- Detection can be bypassed by patching `_ETHREAD.StartAddress`

MISUSE OF DEVICE TREES

- Windows uses a layered (or stacked) architecture for handling I/O requests]
  - » *permits transparent file system archiving and encryption;*
  - » *firewall filtering of network connections*
- Also, opens new avenues for rootkits
  - » *e.g.:* `atapi.sys`



devicetree PLUGIN

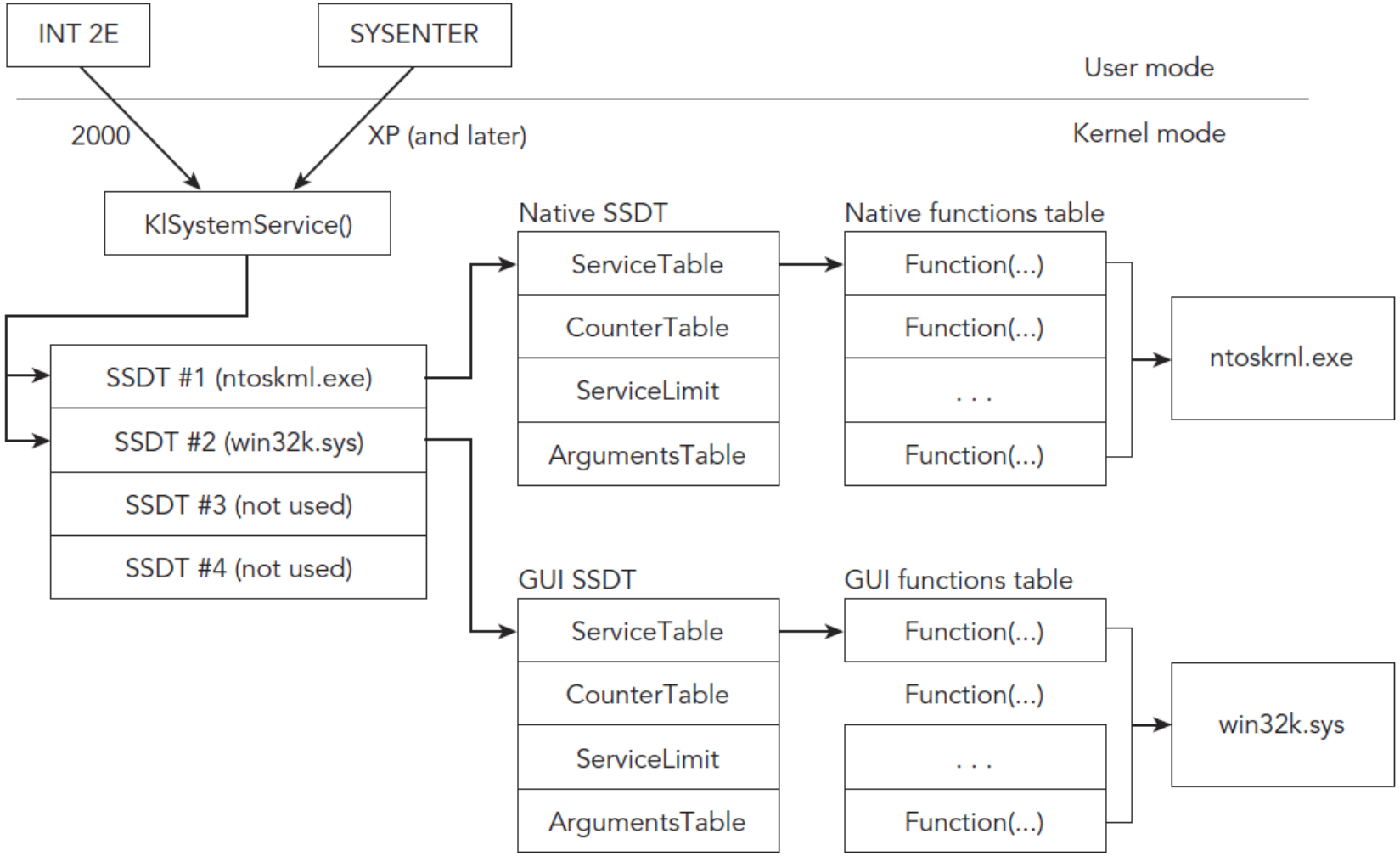
```
vol-xp2 -f stuxnet.vmem devicetree | grep -B 4 MRxNet
```

```
DRV 0x0205e5a8 \FileSystem\vmhgfs
---| DEV 0x820f0030 hgfsInternal UNKNOWN
---| DEV 0x821a1030 HGFS FILE_DEVICE_NETWORK_FILE_SYSTEM
------| ATT 0x81f5d020  - \FileSystem\FltMgr FILE_DEVICE_NETWORK_FILE_SYSTEM
---------| ATT 0x821354b8  - \Driver\MRxNet FILE_DEVICE_NETWORK_FILE_SYSTEM
```

```
DRV 0x0253d180 \FileSystem\Ntfs
---| DEV 0x82166020  FILE_DEVICE_DISK_FILE_SYSTEM
------| ATT 0x8228c6b0  - \FileSystem\sr FILE_DEVICE_DISK_FILE_SYSTEM
---------| ATT 0x81f47020  - \FileSystem\FltMgr FILE_DEVICE_DISK_FILE_SYSTEM
------------| ATT 0x81fb9680  - \Driver\MRxNet FILE_DEVICE_DISK_FILE_SYSTEM
---| DEV 0x8224f790 Ntfs FILE_DEVICE_DISK_FILE_SYSTEM
------| ATT 0x81eecdd0  - \FileSystem\sr FILE_DEVICE_DISK_FILE_SYSTEM
---------| ATT 0x81e859c8  - \FileSystem\FltMgr FILE_DEVICE_DISK_FILE_SYSTEM
------------| ATT 0x81f0ab90  - \Driver\MRxNet FILE_DEVICE_DISK_FILE_SYSTEM
```

```
DRV 0x023ae880 \FileSystem\MRxSmb
---| DEV 0x81da95d0 LanmanDatagramReceiver FILE_DEVICE_NETWORK_BROWSER
---| DEV 0x81ee5030 LanmanRedirector FILE_DEVICE_NETWORK_FILE_SYSTEM
------| ATT 0x81bf1020  - \FileSystem\FltMgr FILE_DEVICE_NETWORK_FILE_SYSTEM
---------| ATT 0x81f0fc58  - \Driver\MRxNet FILE_DEVICE_NETWORK_FILE_SYSTEM
```

SSDT: System Service Descriptor Table



`ssdt` PLUGIN

```
> vol-hw3 ssdt | grep -A 10 'SSDT'
Volatility Foundation Volatility Framework 2.6
[x64] Gathering all referenced SSDTs from KeAddSystemServiceTable...
Finding appropriate address space for tables...
SSDT[0] at fffff800028d4b00 with 401 entries
  Entry 0x0000: 0xfffff80002ce5190 (NtMapUserPhysicalPagesScatter) owned by ntoskrnl.exe
  Entry 0x0001: 0xfffff80002bcba00 (NtWaitForSingleObject) owned by ntoskrnl.exe
  Entry 0x0002: 0xfffff800028cbdd0 (NtCallbackReturn) owned by ntoskrnl.exe
  Entry 0x0003: 0xfffff80002beeb10 (NtReadFile) owned by ntoskrnl.exe
  Entry 0x0004: 0xfffff80002becbb0 (NtDeviceIoControlFile) owned by ntoskrnl.exe
  Entry 0x0005: 0xfffff80002be7ee0 (NtWriteFile) owned by ntoskrnl.exe
  Entry 0x0006: 0xfffff80002b8ddc0 (NtRemoveIoCompletion) owned by ntoskrnl.exe
  Entry 0x0007: 0xfffff80002b8af10 (NtReleaseSemaphore) owned by ntoskrnl.exe
  Entry 0x0008: 0xfffff80002be2da0 (NtReplyWaitReceivePort) owned by ntoskrnl.exe
  Entry 0x0009: 0xfffff80002cb4e20 (NtReplyPort) owned by ntoskrnl.exe
--
SSDT[1] at fffff960001b1c00 with 827 entries
  Entry 0x1000: 0xfffff960001a5580 (NtUserGetThreadState) owned by win32k.sys
  Entry 0x1001: 0xfffff960001a2630 (NtUserPeekMessage) owned by win32k.sys
  Entry 0x1002: 0xfffff960001b3c6c (NtUserCallOneParam) owned by win32k.sys
  Entry 0x1003: 0xfffff960001c1dd0 (NtUserGetKeyState) owned by win32k.sys
  Entry 0x1004: 0xfffff960001bb1ac (NtUserInvalidateRect) owned by win32k.sys
  Entry 0x1005: 0xfffff960001b3e70 (NtUserCallNoParam) owned by win32k.sys
  Entry 0x1006: 0xfffff960001ab5a0 (NtUserGetMessage) owned by win32k.sys
  Entry 0x1007: 0xfffff9600018fbec (NtUserMessageCall) owned by win32k.sys
  Entry 0x1008: 0xfffff960001b56c4 (NtGdiBitBlt) owned by win32k.sys
  Entry 0x1009: 0xfffff960002ad750 (NtGdiGetCharSet) owned by win32k.sys
```

SSDT ATTACKS → POINTER REPLACEMENT

- Overwrite pointer in the SSDT; e.g.:

```
> vol -f laqma.vmem ssdt | egrep -v '(ntoskrnl\.exe|win32k\.sys)'
Volatility Foundation Volatility Framework 2.6
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 80501030 with 284 entries
  Entry 0x0049: 0xfca29884 (NtEnumerateValueKey) owned by lanmandrv.sys
  Entry 0x007a: 0xfca2953e (NtOpenProcess) owned by lanmandrv.sys
  Entry 0x0091: 0xfca29654 (NtQueryDirectoryFile) owned by lanmandrv.sys
  Entry 0x00ad: 0xfca29544 (NtQuerySystemInformation) owned by lanmandrv.sys
```

CODE INJECTION

TYPES OF CODE INJECTION

- Remote DLL injection
  - » *A malicious process forces the target process to load a specified DLL from disk by calling LoadLibrary or the native LdrLoadDll*
- Remote code injection
  - » *A malicious process writes code into the memory space of a target process and forces it to execute.*
    - code can be a block of shellcode (i.e., not a PE file) or it can be a PE file whose import table is preemptively configured for the target process.
- Reflective DLL injection
  - » *A malicious process writes a DLL (as a sequence of bytes) into the memory space of a target process.*
    - DLL handles its own initialization and need not exist on disk
- Hollow process injection
  - » *A malicious process starts a new instance of a legitimate process (e.g.: lsass.exe) in suspended mode*
  - » *Before resuming it, the executable section(s) are freed and reallocated with malicious code*

REMOTE DLL INJECTION [1]

1. Process A enables debug privilege (`SE_DEBUG_PRIVILEGE`) that gives it the right to read and write other process' memory as if it were a debugger.
2. Process A opens a handle to Process B by calling `OpenProcess`.
   - *It must request at least `PROCESS_CREATE_THREAD`, `PROCESS_VM_OPERATION`, and `PROCESS_VM_WRITE`.*
3. Process A allocates memory in Process B using **`VirtualAllocEx`**
   - *protection is typically `PAGE_READWRITE`*
4. Process A transfers a string to Process B's memory by calling **`WriteProcessMemory`**
   - *string identifies the full path on disk to the malicious DLL and it is written at the address allocated in the previous step*

REMOTE DLL INJECTION [2]

5. Process A calls **`CreateRemoteThread`** to start a new thread in Process B that executes the **`LoadLibrary`** function
   - » *The thread's parameter is set to the full path to the malicious DLL, which already exists in Process B's memory*
6. At this point, the injection is complete and Process B has loaded the DLL.
   - » *Process A calls **`VirtualFree`** to free the memory containing the DLL's path.*
7. Process A calls **`CloseHandle`** on Process B's process to clean up.

DETECTION OF REMOTE DLL INJECTION

- Typically, no conclusive evidence
  - » *VAD, PEB lists look normal*
  - » *malicious DLL looks like all others; need specific DLL knowledge*
- Secondary indicators
  - » *if DLL does attempt to hide (by unlinking its _LDR_DATA_TABLE_ENTRY from the ordered list(s)),* **`ldrmodules`** *will flag it*
  - » *if the injected DLL is packed, and the unpacking procedure copies the decompressed code to a new memory region*

  ➔ *detectable with* **`malfind`**

```
> volatility -f stuxnet.vmem --profile=WinXPSP3x86 malfind
```

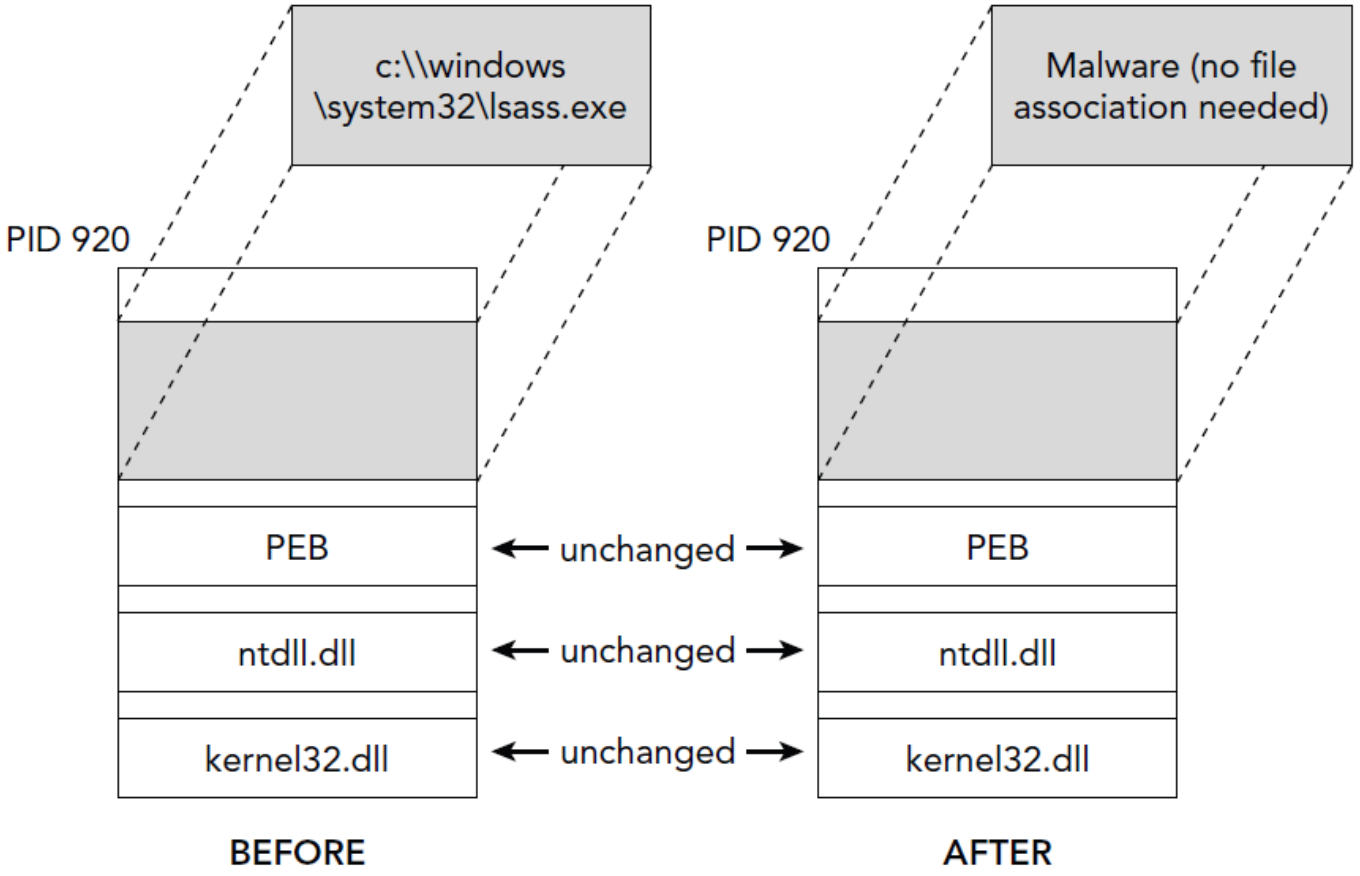
```
Process: services.exe Pid: 668 Address: 0x13f0000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x013f0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x013f0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x013f0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x013f0030  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00   ................

0x013f0000 4d              DEC EBP
0x013f0001 5a              POP EDX
0x013f0002 90              NOP
0x013f0003 0003            ADD [EBX], AL
0x013f0005 0000            ADD [EAX], AL
0x013f0007 000400          ADD [EAX+EAX], AL
0x013f000a 0000            ADD [EAX], AL
```

NO PE SIGNATURE – SHELLCODE

```
$ python vol.py -f carberp.mem --profile=WinXPSP3x86 malfind
Volatility Foundation Volatility Framework 2.4
[snip]

Process: svchost.exe Pid: 992 Address: 0x9d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x009d0000  b8 35 00 00 00 e9 8b d1 f3 7b 68 6c 02 00 00 e9   .5.......{hl....
0x009d0010  94 63 f4 7b 8b ff 55 8b ec e9 6c 11 e4 7b 8b ff   .c.{..U...l..{..
0x009d0020  55 8b ec e9 99 2e 84 76 8b ff 55 8b ec e9 74 60   U......v..U...t`
0x009d0030  7f 76 8b ff 55 8b ec e9 8a e9 7f 76 8b ff 55 8b   .v..U......v..U.

0x9d0000 b835000000       MOV EAX, 0x35
0x9d0005 e98bd1f37b       JMP 0x7c90d195
0x9d000a 686c020000       PUSH DWORD 0x26c
0x9d000f e99463f47b       JMP 0x7c9163a8
0x9d0014 8bff             MOV EDI, EDI
0x9d0016 55               PUSH EBP
```

HOLLOW PROCESS INJECTION

- Original code is replaced with malicious code



HOLLOW PROCESS INJECTION STEPS [1]

1. Start a new instance of a legitimate process (e.g., `lsass.exe`),
   - » *but with its first thread suspended*
2. Acquire the contents for the malicious replacement code
   - » *disk file, memory buffer, network download*
3. Determine the base address (`ImageBase`) of the `lsass.exe` process, and then free/ unmap the containing memory section
   - » *the process becomes an empty container (the DLLs, heaps, stacks, and open handles are still intact, but no process executable).*
4. Allocate a new memory segment in `lsass.exe` and make sure that the memory can be read, written, and executed.
   - » *reuse the same `ImageBase` or a use different one*

HOLLOW PROCESS INJECTION STEPS [2]

5. Copy the PE header for the malicious process into the newly allocated memory in `lsass.exe`
6. Copy each PE section for the malicious process into the proper virtual address in `lsass.exe`
7. Set the start address for the first (suspended) thread to point at the malicious process' `AddressOfEntryPoint` value
8. Resume the thread
   - » *the malicious process begins executing within the container created for `lsass.exe`.*
     - `ImagePathName` in the PEB still points to C:\windows\system32\lsass.exe

DETECTION [1] – SAME PE INDICATED

```
> vol-stux pslist | grep lsass
Volatility Foundation Volatility Framework 2.6
0x81e70020 lsass.exe                680    624     19      342      0      0 2010-10-29 17:08:54 UTC+0000
0x81c498c8 lsass.exe                868    668      2       23      0      0 2011-06-03 04:26:55 UTC+0000
0x81c47c00 lsass.exe               1928    668      4       65      0      0 2011-06-03 04:26:55 UTC+0000
```

```
> vol-stux dlllist -p 680,868,1928 | grep lsass
Volatility Foundation Volatility Framework 2.6
lsass.exe pid:    680
Command line : C:\WINDOWS\system32\lsass.exe
0x01000000     0x6000     0xffff C:\WINDOWS\system32\lsass.exe
lsass.exe pid:    868
Command line : "C:\WINDOWS\\system32\\lsass.exe"
0x01000000     0x6000     0xffff C:\WINDOWS\system32\lsass.exe
lsass.exe pid:   1928
Command line : "C:\WINDOWS\\system32\\lsass.exe"
0x01000000     0x6000     0xffff C:\WINDOWS\system32\lsass.exe
```

DETECTION [2] – DIFFERENT VAD SITUATION

```
> vol-stux vadinfo -p 680 --addr=0x1000000
Volatility Foundation Volatility Framework 2.6
**************************************************************************
Pid:    680
VAD node @ 0x81db03c0 Start 0x01000000 End 0x01005fff Tag Vad
Flags: CommitCharge: 1, ImageMap: 1, Protection: 7
Protection: PAGE_EXECUTE_WRITECOPY
ControlArea @823e4008 Segment e1735398
NumberOfSectionReferences:          3 NumberOfPfnReferences:           4
NumberOfMappedViews:                1 NumberOfUserReferences:          4
Control Flags: Accessed: 1, File: 1, HadUserReference: 1, Image: 1
FileObject @82230120, Name: \Device\HarddiskVolume1\WINDOWS\system32\lsass.exe
First prototype PTE: e17353d8 Last contiguous PTE: fffffffc
Flags2: Inherit: 1
```

```
> vol-stux vadinfo -p 868 --addr=0x1000000
Volatility Foundation Volatility Framework 2.6
**************************************************************************
Pid:    868
VAD node @ 0x81f1ef08 Start 0x01000000 End 0x01005fff Tag Vad
Flags: CommitCharge: 2, Protection: 6
Protection: PAGE_EXECUTE_READWRITE
ControlArea @81fbeee0 Segment e24b4c10
NumberOfSectionReferences:          1 NumberOfPfnReferences:           0
NumberOfMappedViews:                1 NumberOfUserReferences:          2
Control Flags: Commit: 1, HadUserReference: 1
First prototype PTE: e24b4c50 Last contiguous PTE: e24b4c78
Flags2: Inherit: 1
```

DETECTION [3] – **ldrmodules**

```
> vol-stux ldrmodules -p 868
Volatility Foundation Volatility Framework 2.6
Pid      Process              Base       InLoad InInit InMem MappedPath
-------- -------------------- ---------- ------ ------ ----- ----------
     868 lsass.exe            0x00080000 False  False  False
     868 lsass.exe            0x7c900000 True   True   True  \WINDOWS\system32\ntdll.dll
     868 lsass.exe            0x77e70000 True   True   True  \WINDOWS\system32\rpcrt4.dll
     868 lsass.exe            0x7c800000 True   True   True  \WINDOWS\system32\kernel32.dll
     868 lsass.exe            0x77fe0000 True   True   True  \WINDOWS\system32\secur32.dll
     868 lsass.exe            0x7e410000 True   True   True  \WINDOWS\system32\user32.dll
     868 lsass.exe            0x01000000 True   False  True
     868 lsass.exe            0x77f10000 True   True   True  \WINDOWS\system32\gdi32.dll
     868 lsass.exe            0x77dd0000 True   True   True  \WINDOWS\system32\advapi32.dll
```

DETECTING MALWARE PERSISTENCE

- System startup:
  - *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce*
  - *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run*
  - *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*
- User logons:
  - *HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows*
  - *HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows\Run*
  - *HKCU\Software\Microsoft\Windows\CurrentVersion\Run*
  - *HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce*

SERVICES

```
> vol-xp2 -f stuxnet.vmem printkey -K "controlset001\services\mrxnet"
Volatility Foundation Volatility Framework 2.6
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\system
Key name: MRxNet (S)
Last updated: 2011-06-03 04:26:47 UTC+0000

Subkeys:
  (V) Enum

Values:
REG_SZ        Description     : (S) MRXNET
REG_SZ        DisplayName     : (S) MRXNET
REG_DWORD     ErrorControl    : (S) 0
REG_SZ        Group           : (S) Network
REG_SZ        ImagePath       : (S) \??\C:\WINDOWS\system32\Drivers\mrxnet.sys
REG_DWORD     Start           : (S) 1
REG_DWORD     Type            : (S) 1
```

vol-stux userassist | grep BINARY | cut -b 20-

```
RUNPATH:Notepad++.lnk :
RUNPATH:C:\Program Files\Notepad++\notepad++.exe :
RUNPATH:Mozilla Firefox.lnk :
RUNPATH:C:\Program Files\Mozilla Firefox\firefox.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\PyScripter-v1.9.9.7-Setup.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\pywin32-213.win32-py2.5.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\SymbolTypeViewer_v1.0_beta\setup.exe :
RUNPATH:C:\WINDOWS\system32\cmd.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\yara-python-1.4.win32-py2.6.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\Pyrex-0.9.9.win32.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\pydasm-1.5.win32-py2.6.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\ssdeep-2.0-0.1.win32-py2.6.exe :
RUNCPL:SYSDM.CPL :
RUNPATH:C:\WINDOWS\system32\NOTEPAD.EXE :
RUNPIDL:%csidl2%\PyScripter\PyScripter for Python 2.6.lnk :
RUNPIDL:%csidl2%\PyScripter :
RUNPATH:C:\Program Files\PyScripter\PyScripter.exe :
RUNPIDL:%csidl2%\Wireshark.lnk :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\idastdnw_101001_ae2076e448f89f3a93c0fc7
RUNPATH:C:\Documents and Settings\Administrator\Desktop\hexx86w_101001_ae2076e448f89f3a93c0fc73
RUNPATH:C:\Documents and Settings\Administrator\Desktop\pywin32-214.win32-py2.6.exe :
RUNPATH:Immunity Debugger.lnk :
RUNPATH:C:\Program Files\Immunity Inc\Immunity Debugger\ImmunityDebugger.exe :
RUNPATH:VMware Shared Folders.lnk :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\SysinternalsSuite\procexp.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\SysinternalsSuite\Procmon.exe :
RUNPATH:C:\Documents and Settings\Administrator\Desktop\74ddc49a7c121a61b8d06c03f92d0c13.exe :
```