

Malicious Software

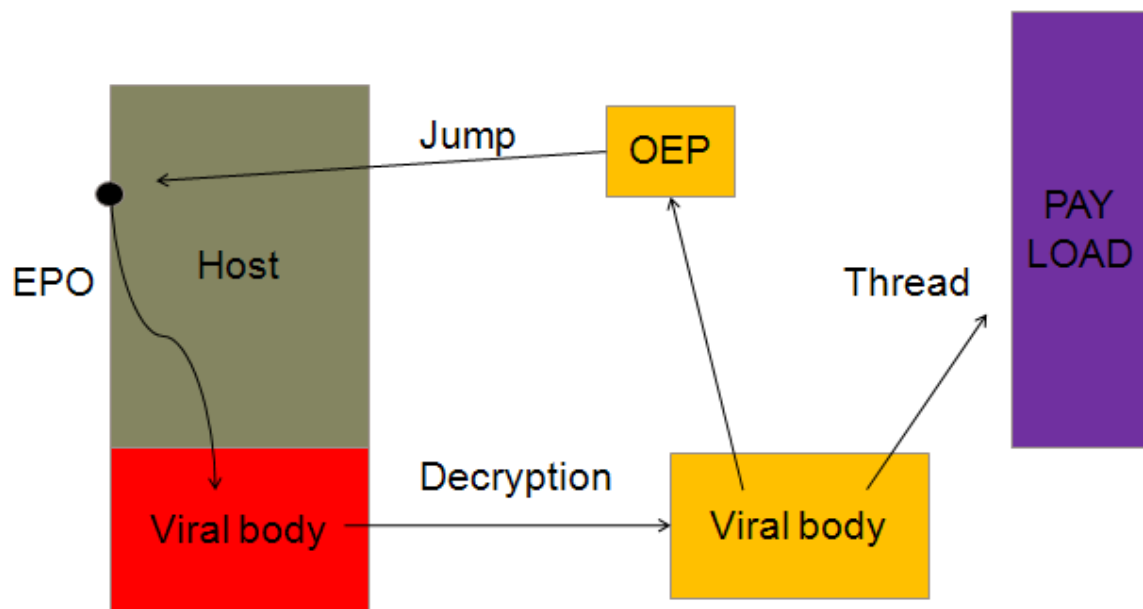
Virus - a piece of software that can “infect” other programs by modifying them in the file system; the modification includes injecting the original program with a routine to make copies of the virus program, which can then go on to infect other programs.

Virus code can be prepended or appended to an executable program, or it can be embedded in some other fashion. The key to its operation is that the infected program, when invoked, will first execute the virus code and then execute the original code of the program.

Example: sality PE infector

http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/sality_peer_to_peer_viral_network.pdf

Encrypted virus: A portion of the virus creates a random encryption key and encrypts the remainder of the virus. The key is stored with the virus. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

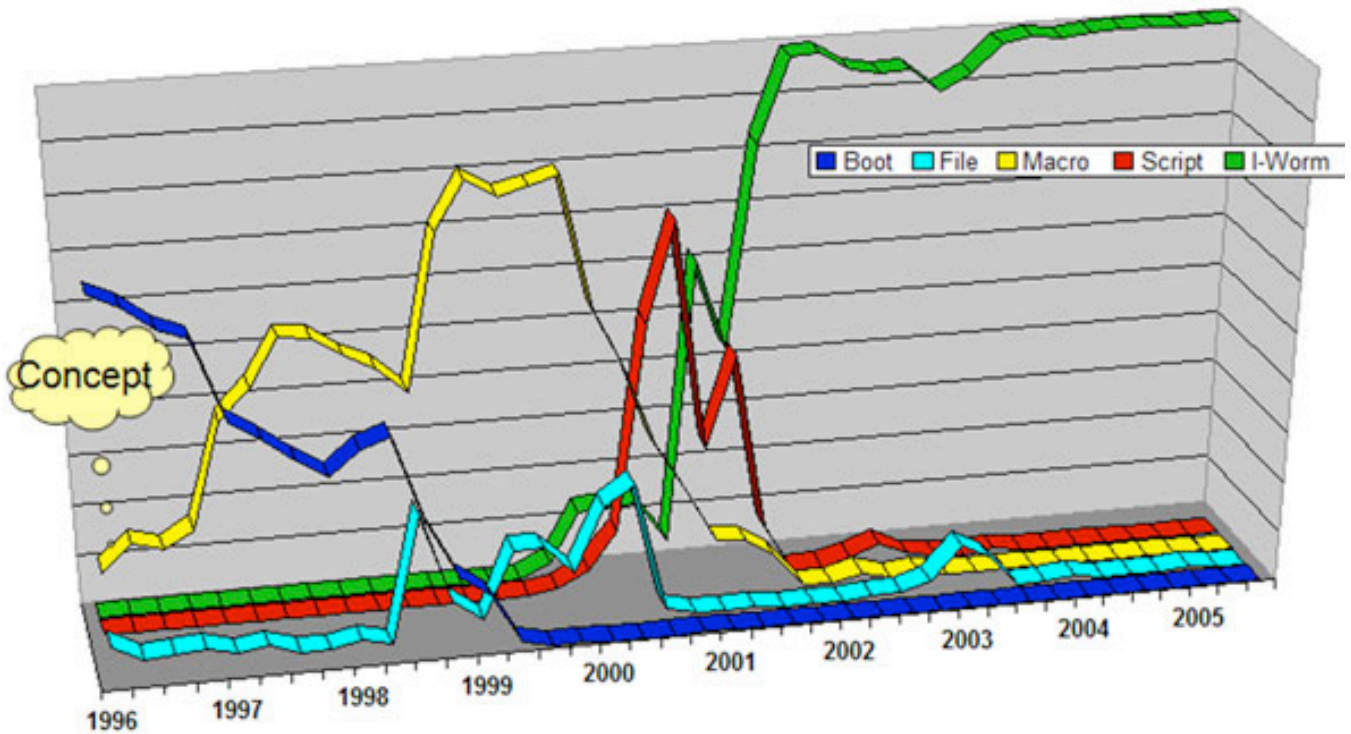


Execution Flow of an Infected File

Types of Viruses:

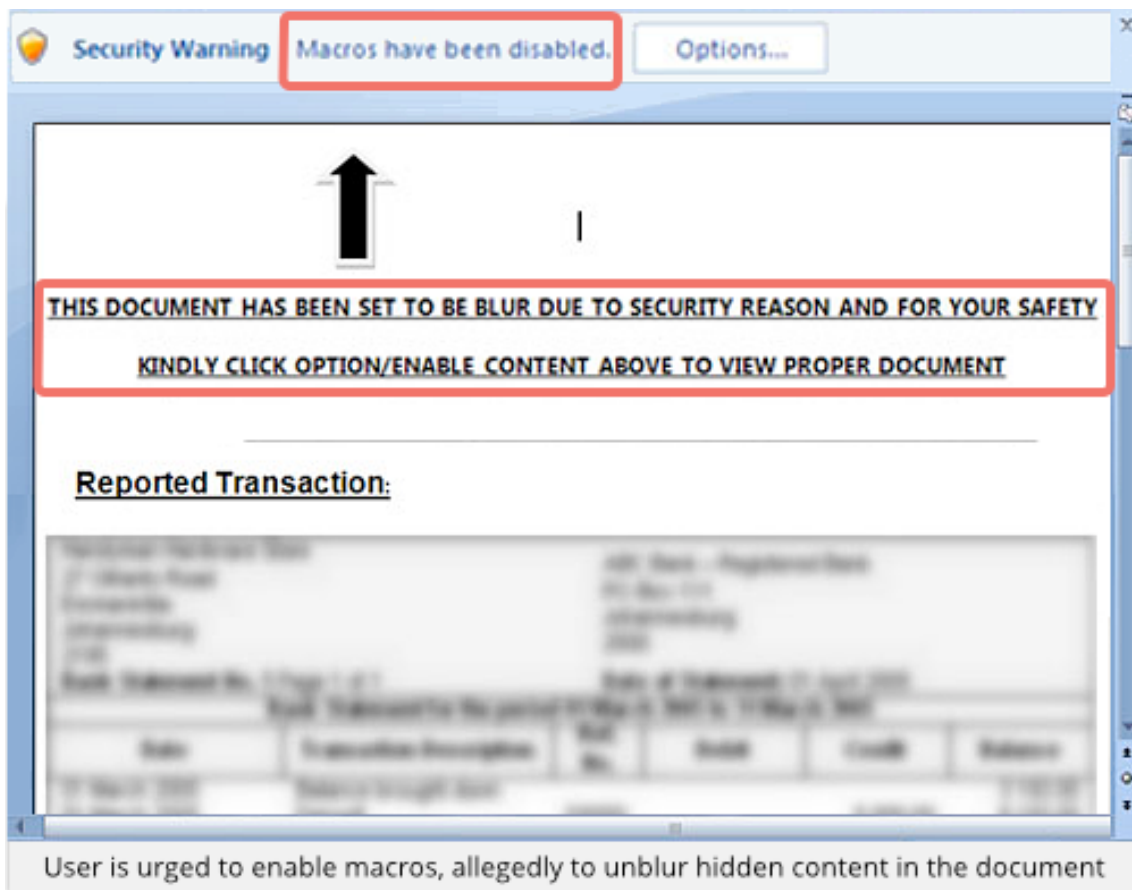
- **Boot sector infector:** Infects a master boot record or boot record and spreads when a system is booted from the disk containing the virus. Example: stoned, Michelangelo, mebroot, rovnix
- **File infector:** Infects files that the operating system or shell consider to be executable.
- **Macro virus:** Infects files with macro code that is interpreted by an application.
Example: Melissa, iloveyou

Macro viruses were popular until about the year 2000 when Windows executables became the prominent form.



Virus prevalence by type

However, in 2014, there were some new macro viruses created and social engineering was used to turn off security controls which disabled macros by default:



<http://blog.dynamoo.com/show-macros>

2014 Macro Virus in a MS Office document used Social Engineering to get macros turned on to distribute the Napolar virus. Other attacks used a macro to download trojans such as Zeus password stealer or Dridex infostealer.

- **Stealth virus:** A form of virus explicitly designed to hide itself from detection by antivirus software. Thus, the entire virus, not just a payload is hidden in the file system, registry and process list
- **Self garbling virus** - modifies itself to elude detection by pattern recognition
- **Polymorphic virus:** A virus that mutates with every infection, making detection by the “signature” of the virus very difficult. A Polymorphic virus copies itself and then changes the copy using a mutation engine.
- **Metamorphic virus:** As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses may change their behavior as well as their appearance.
- **Blended malware** - uses several methods to spread. A virus that is able to infect both boot sectors and program files, it reproduces in more than one way.
- **Multipartite virus** – used to indicate a virus that was able to infect or reproduce in more than one way. Example: Anthrax, One_half
Sality – multipartite file infector
- **Mobile Code** – programs that can be distributed unchanged to a variety of platforms
Viruses, worms and trojans can use mobile code as a payload or can take advantage of vulnerabilities in the code interpreter (Javascript, Java, ActiveX, VBScript) to spread

[Sality](#) employs polymorphic and entry-point obscuring (EPO) techniques to infect files:

- The entry-point address of the host is unchanged.
- The code at the entry-point is changed, and replaced by a variable stub, generated by Sality polymorphic code generator (dubbed “Simple Poly Engine v1.1a (c) Sector”).
- This stub jumps to the main virus body, appended to the last section of the host file.

The initial code of this body is also polymorphic and contains junk instructions to thwart emulation strategies used by anti-virus. This stub eventually decrypts and executes a secondary region, which is the loader itself.

- The loader is run in a separate thread in the infected process. Its role is to load and execute Sality itself (hereby referred to as payload). If another copy of Sality is running on the system, it will wait.
- Meanwhile, the original entry-point code (OEP) of the host is restored and the host is executed

Worm

A worm is a program that can replicate itself and send copies from computer to computer across network connections. Upon arrival, the worm may be activated to replicate and propagate again. In addition to propagation, the worm usually performs some unwanted function such as remote access.

An e-mail virus has some of the characteristics of a worm because it propagates itself from system to system. However, we classify it as a virus because it uses a document modified to contain viral macro content or application exploit and requires human action to propagate. A worm actively seeks out more machines to infect and each machine that is infected serves as an automated launching pad for attacks on other machines.

Examples: MyDoom, Code Red, nimda, slammer, conficker, Morto, QNAP shellshock worm

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following:

- **Electronic mail facility:** A worm mails a copy of itself to other systems, so that its code is run when the e-mail or an attachment is received or viewed. MyDoom used email to spread
- **Remote execution capability:** A worm executes a copy of itself on another system, either using an explicit remote execution facility or by exploiting a program flaw in a network service to take-over its operations. Code Red, nimda, conficker and QNAP shellshock worm used exploits to spread.

Zero-day exploit: to achieve maximum surprise and distribution, a worm exploits an unknown vulnerability that is only discovered as the worm is launched and analyzed.

- **Remote login capability:** A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other, where it then executes. Morto use remote desktop login to spread.

More Malware types

- **Trojan horse** – Program that appears desirable, but actually contains code that is harmful.
Example: Vundo, Zeus, Cidox
- **Spyware** – Hidden application injected through poor browser security to collect private data, send advertising, etc
- **Companion Virus** (spawning virus) - Some viral programs do not physically touch the target file at all. An example is having an infected .com file with the same name and in the same directory of as an .exe file. This is also how DLL hijacking works.
- **Transient virus** - virus only runs when host application is launched
- **Resident virus** - virus loads when host is started and stays running when application is closed. This is also called establishing persistence.
- **Usermode rootkit** - runs in application layer, ring 3, alters applications the user might run
netstat, passwd, logind
Example: betabot

- **Kernel rootkit** - runs on kernel layer, alters the system calls in the kernel
Example: TDSS4, Rovnix, Grayfish
- **Ransomware** - encrypts the user's data and demands payment in order to access the key needed to recover the information
 - PC Cyborg Trojan (1989)
 - Gpcode Trojan (2006)
 - Reveton (2012)
 - CryptoLocker (2013)
 - TorrentLocker, CryptoDefense, CryptoWall (2014)

Rootkit – application designed to hide directories, files, registry keys, network connections, windows and processes from discovery so an attacker can maintain *root* access undetected.

Most rootkits install as a system service or system driver that hides itself from the services or driver list. It also removes itself from a file listing.

When the rootkit is active:

- Its EXE will not show in Explorer, File Manager, or in the output of the DIR command run in a DOS window.
- Its registry keys will not show in Regedit under HKLM\Services
- The port used by the backdoor will not show in **netstat** output.
- The rootkit processes and backdoor process will not appear in the task list when using **Task Manager** or the **ps** command

In effect, the OS starts lying about what is on the hard drive, what processes are running and what network ports are in use.

How do you detect such information hiding programs?

- Scan using port scanner (Nmap, superscan, etc.) from another computer to see there is some backdoor running.
- Scan hard drive from a liveCD or USB boot of a virus scanner like Norton Power Eraser
- Use a rootkit detector to find hidden processes (RKDetector, Rootkit Buster, RootKitRevealer, F-Secure Blacklight, TDSSKiller, Hitman Pro)

Some rootkits don't run in Safe Mode and can be detected by antivirus scans from Safe Mode. Other rootkits, like Hacker Defender, do run in Safe Mode and can only be detected with an off-line scan or using a rootkit detection utility. Hacker Defender installed both a system service and a system driver.

Example Rootkits and Malware

Mebroot – 2007

MBR-based rootkit often used to hide Torpig backdoor

Mebratix – 2008

Malware family based on MBR infection.

Mebroot v2 – 2009

The evolved version of the Mebroot malware.

Stoned Bootkit – 2009

Another example of MBR-based bootkit infection.

Olmarik (TDL4) – 2010/11

The first 64-bit bootkit in the wild.

Stoned Bootkit x64 – 2011

MBR-based bootkit supporting the infection of 64-bit operating systems.

Olmasco (TDL4 modification) – 2011

The first (volume boot record) VBR-based bootkit infection.

DeepBoot – 2011

Used interesting tricks to switch from real-mode to protected mode.

Rovnix – 2011

The evolution of VBR-based infection with polymorphic code.

Mebromi – 2011

The first exploration of the concept of BIOSkits seen in the wild.

VGA Bootkit – 2012

VGA-based bootkit concept.

Gapz – 2012

The next evolution of VBR infection

DreamBoot – 2013

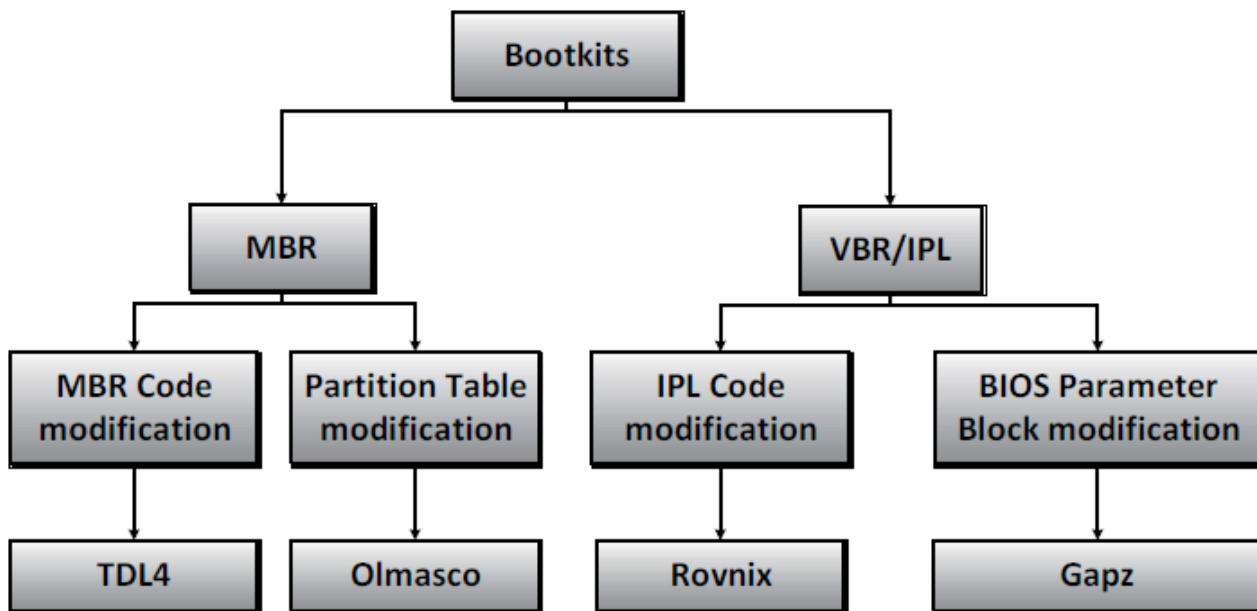
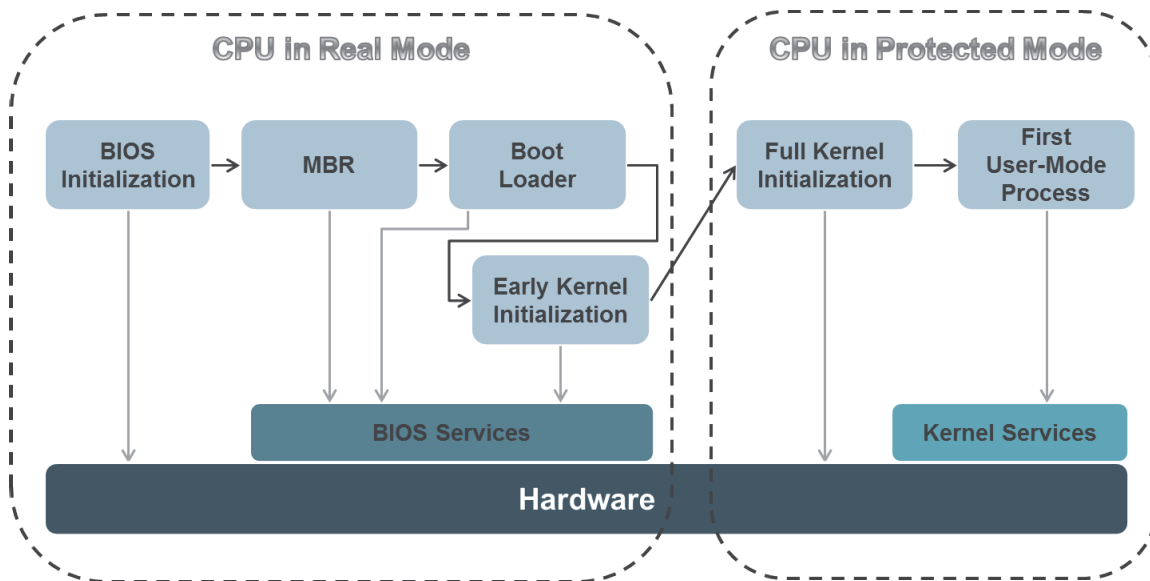
The first public concept of UEFI bootkit.

OldBoot - 2014

The first bootkit for the Android operating system in the wild.

GrayFish – 2015

Bootkit and Rootkit able to run on Windows NT4, 2000, XP, 7, 8, 8.1 32-bit and 64-bit versions.



Modern Rootkits are loaded by several different boot methods

How do Windows rootkits work?

Traditional process hiding techniques

There are many techniques for hiding processes in the system:

DLL infection (file replacement) – new DLL (NTDLL.DLL for e.g.) is provided with some functions patched (like *ZwQuerySystemInformation*). This is easily detectable by file system integrity checkers (provided they are not using DLL, but direct *int 2eh* calls) and hashing.

API hooking – almost the same as above, but no file replacement is needed. Infection is done by means of function calls to *OpenProcess()*, *WriteProcessMemory()*, etc...to alter the API table

An example rootkit of this kind is HackerDefender. Detection is by running comparing the API table with a known clean system of the same version. There is also another method of

disclosing all hidden processes by this and above methods. The Administrator can use a small program which will call *ZwQuerySystemInfo()*, not through the NTDLL, but directly through *int 2eh*. This is how all userland rootkits can be detected. Userland rootkits try to hide some finite list of objects. They can always be uncovered by accessing those objects or scanning for them directly.

Thread injection – in this technique there is no additional process creation. Rootkit chooses appropriate “legal” process, and inserts a new thread into this process. This thread is doing something useful for the attacker. The problem here is the flexibility. Although some tasks could be implemented this way, it would be probably impossible to have a running hidden console and the possibility to start new different, arbitrary programs from this console. It is difficult for even medium-complexity programs to run this way (as a thread).

Kernel service hooking - This is a classic approach, which involves hooking some well known kernel services (such as *svchost.exe* which is running with SYSTEM privileges) and cheating about the results they return. It can be implemented in several variants:

1. **Service Table hooking (ST)**: probably the most popular approach, which is also easy detectable by comparing Service Table with the copy from the “clear” system.
2. **Interrupt Descriptor Table (IDT) hooking**: similar to the above, similar disadvantages allowing it to be detected.
3. **Raw kernel code change**: instead of changing pointers to code, we just changing the code itself, by inserting for e.g. some ‘*jmp*’ instructions.
4. **“Strange” pointers change**: change some uncommon code pointer (in a kernel memory data structure, in contrast to ST or IDT), in order to change the execution path of some kernel services.
There are many such pointers, for example *pServiceDescriptorTable* in *_KTHREAD* object.

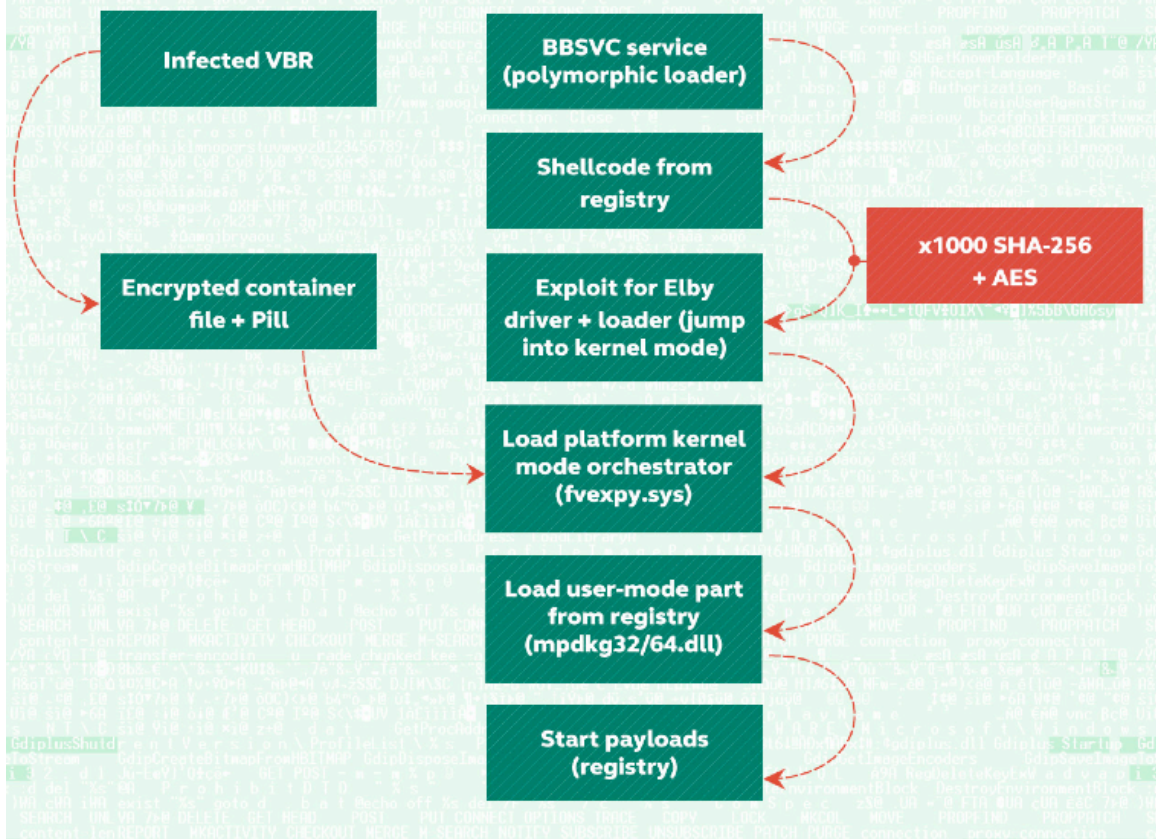
Rootkits implementing techniques 1-3, are relatively easy to detect by calculating hashes on kernel memory. The 4th approach is little more problematic, since it is probably impossible to spot all the valuable (from the attacker’s point of view) pointers in the kernel, which should be checked for changes.

Grayfish Rootkit

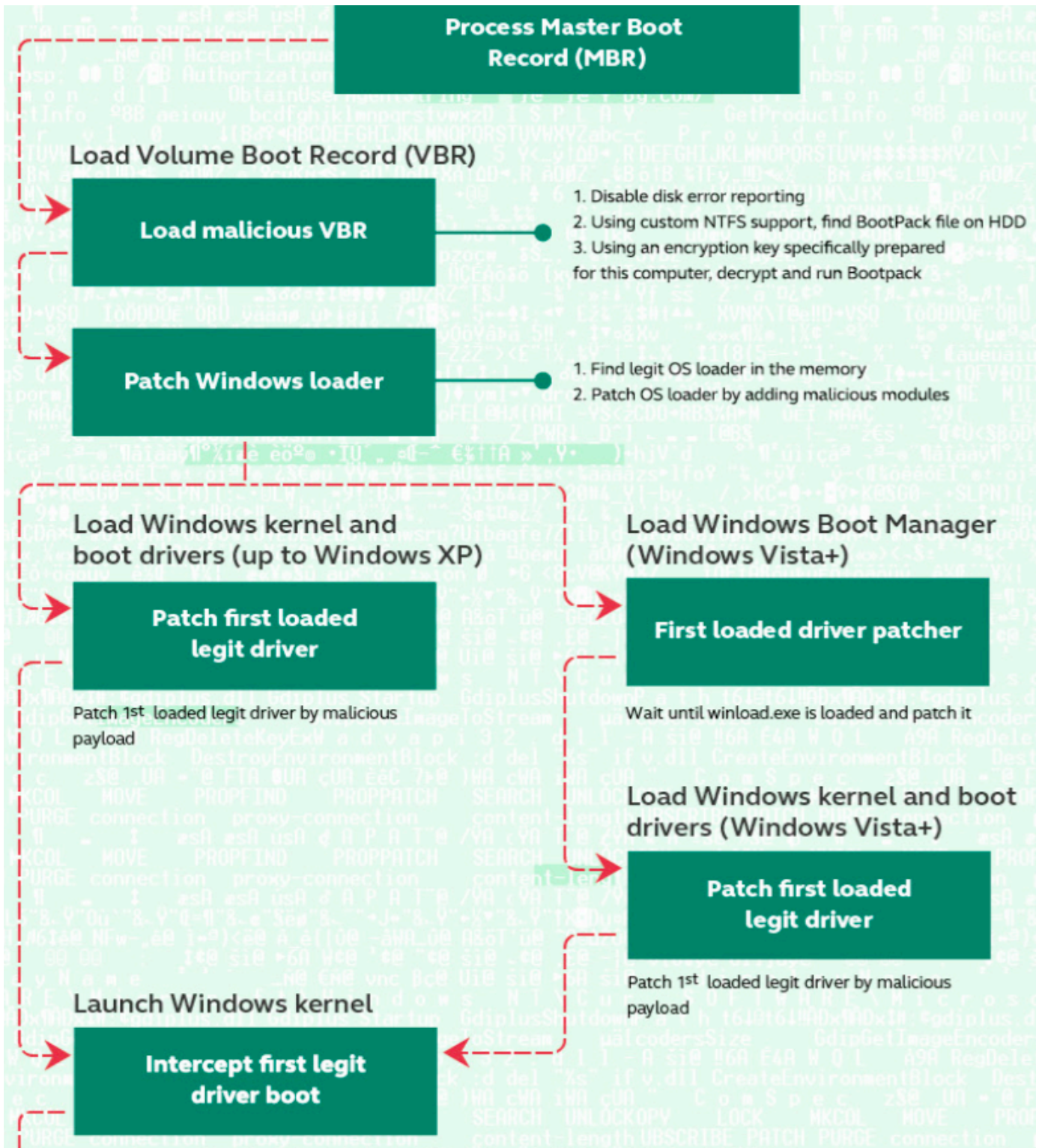
GRAYFISH is the most modern and sophisticated malware implant from the Equation group. It is designed to provide an effective (almost “invisible”) persistence mechanism, hidden storage and malicious command execution inside the Windows operating system.

Developed between 2008 and 2013 Grayfish is compatible with all modern versions of Microsoft’s operating systems, including Windows NT 4.0, Windows 2000, Windows XP, Windows Vista, Windows 7 and 8 – both 32-bit and 64-bit versions.

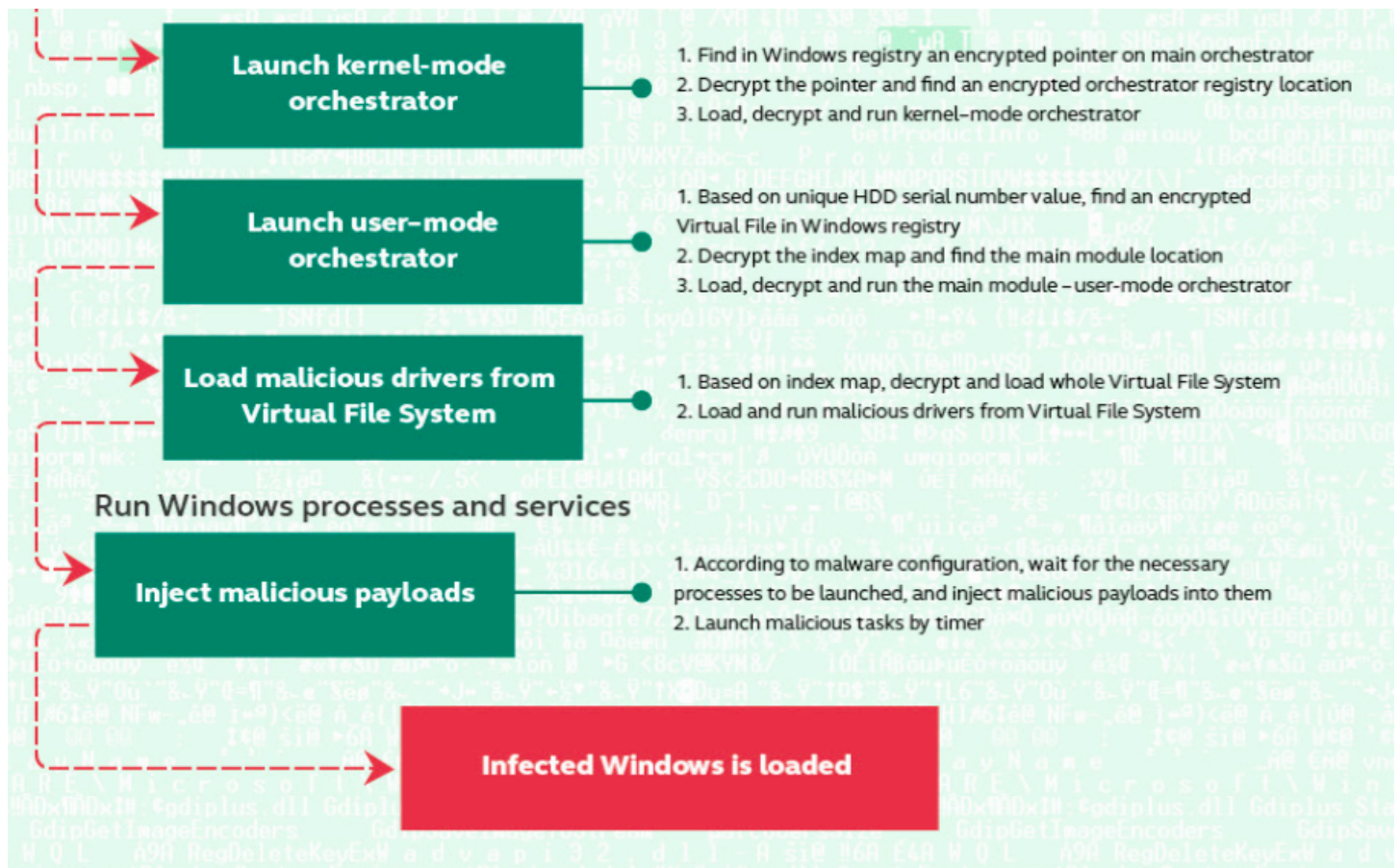
GrayFish architecture



When the computer starts, GrayFish hijacks the OS loading mechanisms by injecting its code into the boot record. This allows it to control the launching of Windows at each stage and bypassing of Windows secure boot controls.



Grayfish Boot Steps

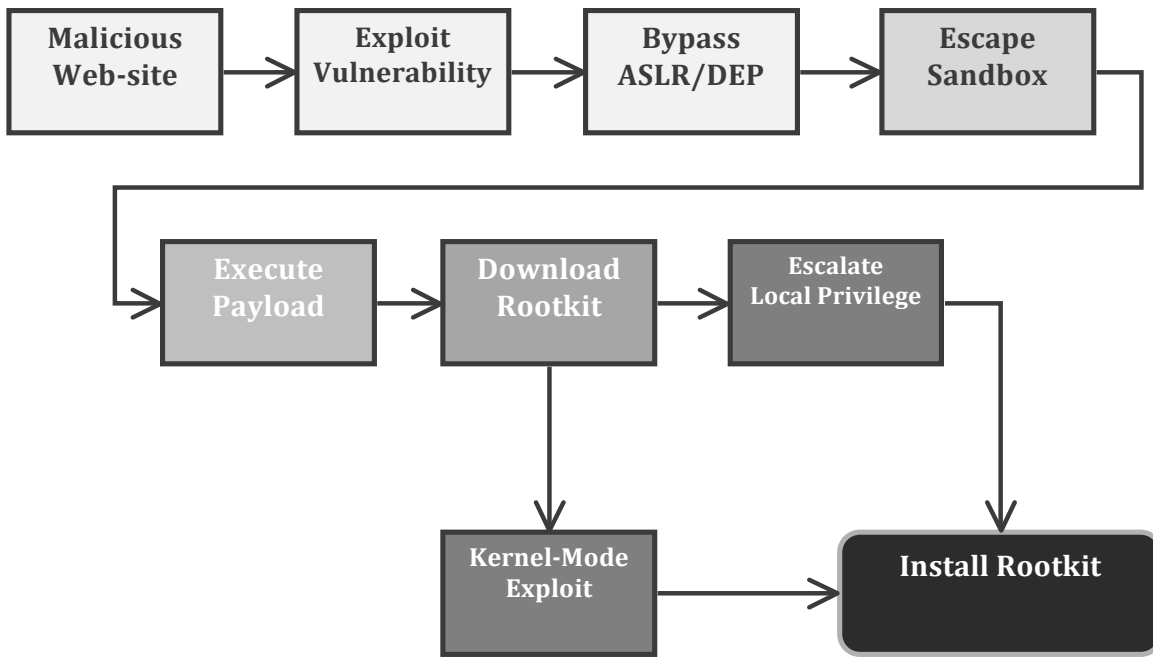


GrayFish Boot Steps – continued.

To store stolen information, as well as its own auxiliary information, GrayFish implements its own encrypted Virtual File System (VFS) inside the Windows registry.

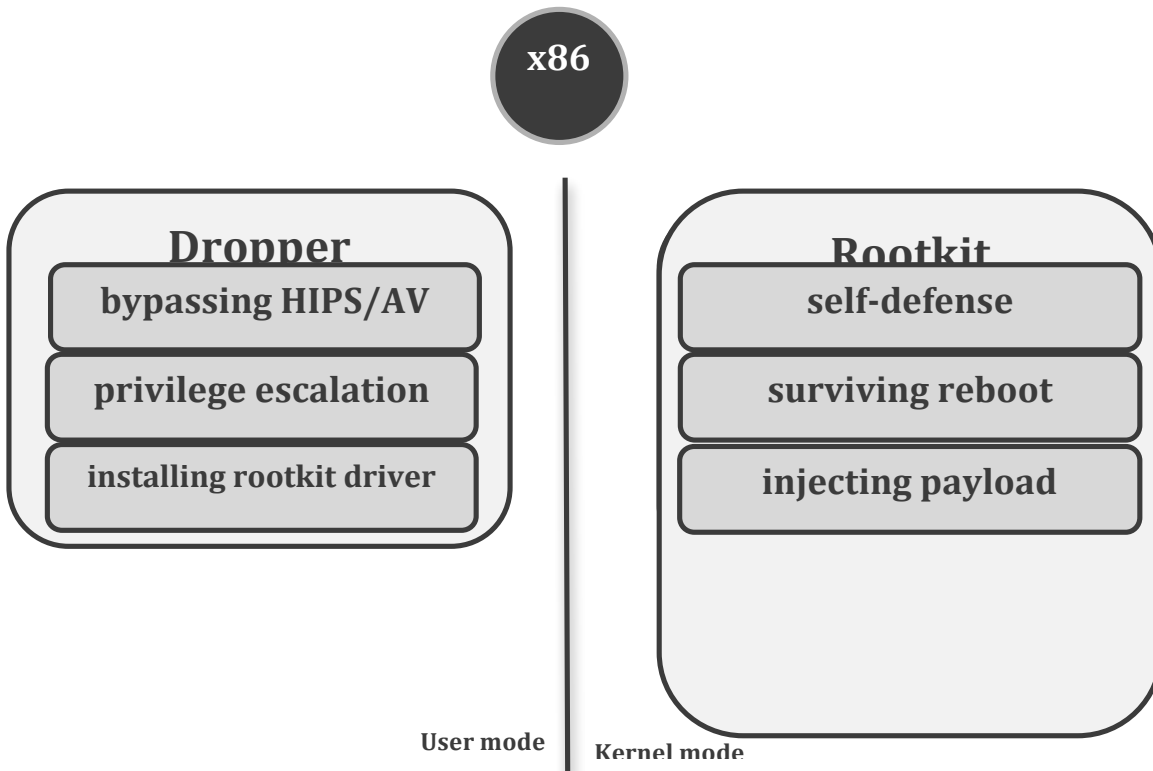
To bypass modern OS security mechanisms that block the execution of untrusted code in kernel mode, GrayFish exploits several legitimate drivers, including one from the CloneCD program. This driver (ElbyCDIO.sys) contains a vulnerability which GrayFish exploits to achieve kernel-level code execution. Despite the fact that the vulnerability was discovered in 2009, the digital signature has not yet been revoked.

Grayfish has an implanter module for flashing the firmware of certain models of Seagate, Toshiba, Samsung, Western Digital, Micron, Maxtor, IBM, and Hitachi hard drives. Thus, infecting the firmware that controls the PC boot process before Windows starts. The module rewrites the firmware and can even capture encryption keys of encrypted hard drives.

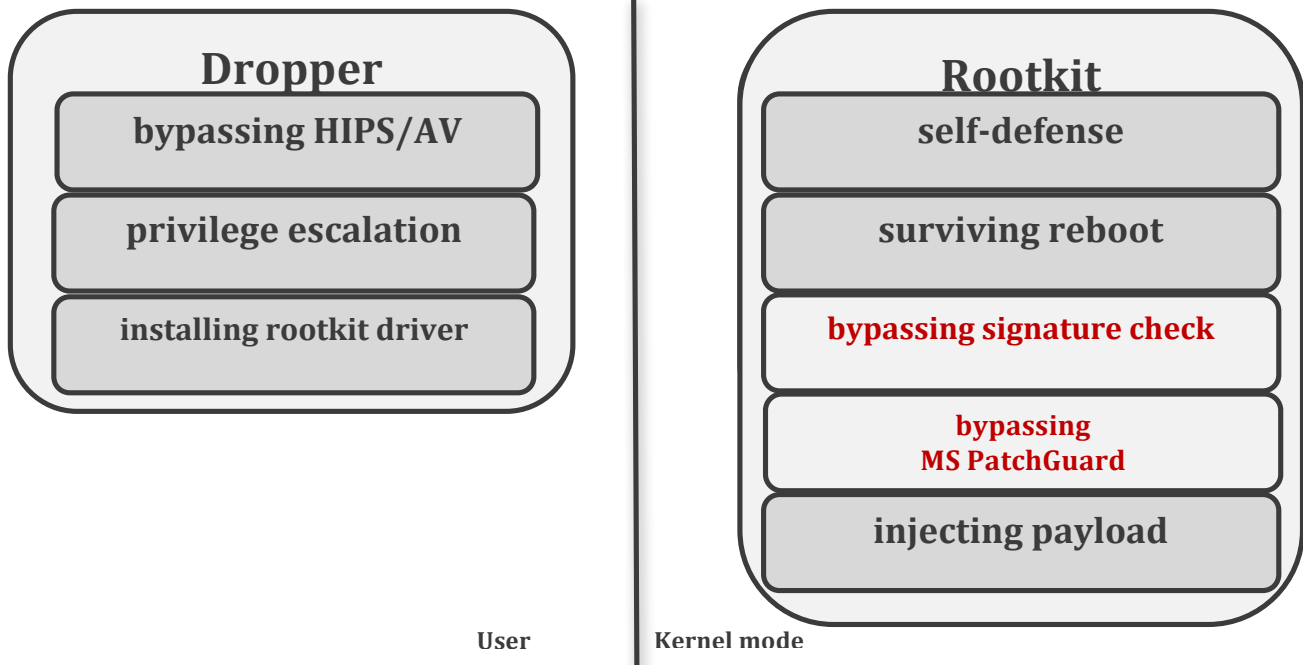


Installation of a Rootkit

Evolution of Rootkit Features



x64



OS Features to make 64-bit Rootkits more Difficult:

- Kernel-Mode Code Signing Policy:
 - ✓ It is “difficult” to load unsigned kernel-mode driver
- Kernel-Mode Patch Protection (Patch Guard):
 - ✓ SSDT (System Service Dispatch Table)
 - ✓ IDT (Interrupt Descriptor Table)
 - ✓ GDT (Global Descriptor Table)
 - ✓ MSRs (Model Specific Registers)

Bypassing Kernel-Mode Code Signing Policy

- Abusing vulnerable signed legitimate kernel-mode driver
- Switching off kernel-mode code signing checks by altering BCD (boot configuration data) data:
 - ✓ abusing WinPe Mode
 - ✓ disabling signing check
 - ✓ enabling test signing
- Patching Bootmgr and OS loader (modify MBR or VBR)
 - ✓ Patch Bootmgr and OS loader (*winload.exe*) to disable KMCSP

UEFI Firmware may make bootkits a thing of the past

If the system's firmware is compliant with the Unified Extensible Firmware Interface, UEFI, specification, the boot process is handled differently to the way in which BIOS firmware handles it now. When the system starts up, the firmware stored in NVRAM reads BCD which is also located in NVRAM, and based on the available options, proceeds to execute winload.exe or winresume.exe. With UEFI, the MBR code is not executed at all, while BCD is read from nonvolatile RAM but not from the disk, so the bootkit fails to load on systems with such firmware.

Other Windows Rootkits:

Mebroot - http://about-threats.trendmicro.com/ArchiveMalware.aspx?language=us&name=TROJ_MEBROOT.SMC

Infests MBR

Also installs a print processor to run as the spoolsv.exe service with system privileges

Starting around the last part of 2008 year, the *Mebroot* rootkit began infiltrating computers' master boot records, the part of the hard drive that loads the operating system, by installing itself from untrusted Web sites. The virus installs a keylogger that is triggered when the infected computer is used to visit any of 900 financial sites, stealing usernames and passwords any time they're typed (and thus eventually stealing your money or identity). This rootkit hides from most virus protection software quite well.

Usermode Rootkits

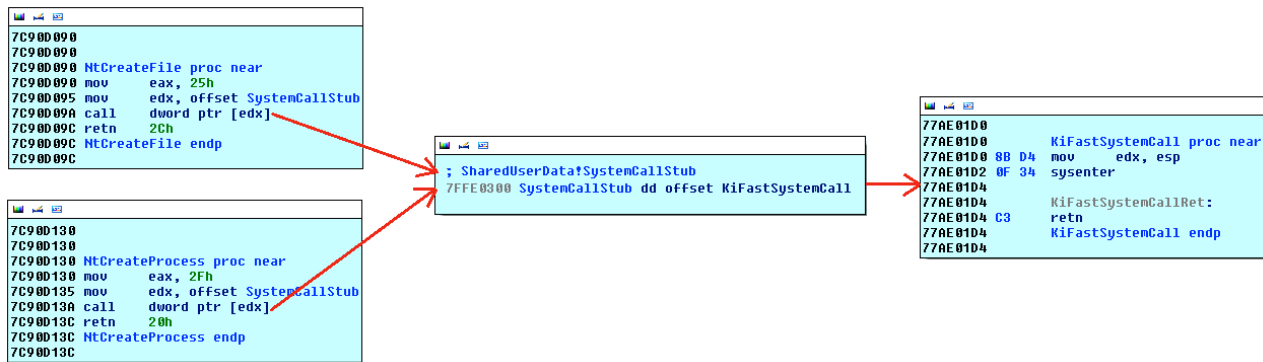
Usermode Hooking – hiding malware without kernel mode access

Usermode System Call hooking

Win32/64 System Calls

System call is a term used to describe functions that do not execute code in usermode, instead they transfer execution to the kernel where the actual work is done. A good example of these is the native API (Ex: NtCreateFile\ZwCreateFile). None of the functions beginning with Nt or Zw actually do their work in usermode, they simply call into the kernel and allow the kernel mode function with the same name to do their work (ntdll!NtCreateFile calls ntoskrnl!NtCreateFile).

Before entering the kernel, all native functions execute some common code, this is known as KiFastSystemCall on 32-bit windows and WOW32Reserved under WOW64 (32-bit process on 64-bit windows).



Native function call path in user mode under Windows 32-bit



Native function call path in user mode under WOW64

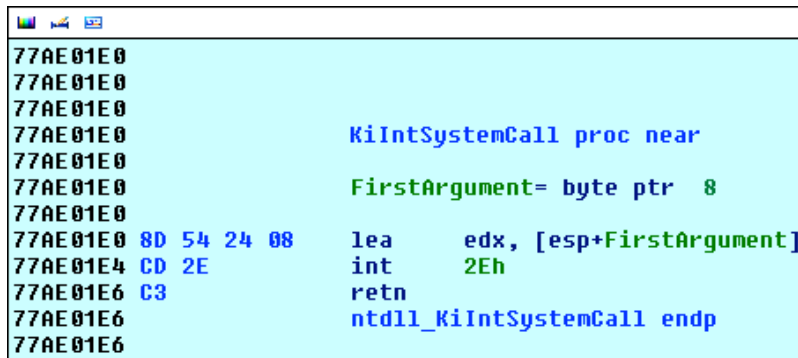
As is evident in both examples: Nt* functions make a call via a 32-bit pointer to KiFastSystemCall (x86) or X86SwitchTo64BitMode (WOW64). Theoretically we could just replace the pointer at SharedUserData!SystemCallStub and WOW32Reserved with a pointer to our code; However, in practice this doesn't work.

SharedUserData is a shared page mapped into every process by the kernel, thus it's only writable from kernel mode. On the other hand WOW32Reserved is writable from user mode, but it exists inside the thread environment block (TEB), so in order to hook it we'd have to modify the TEB for every running thread.

KiFastSystemCall Hook

Because SharedUserData is non-writable, the only other place we can target is KiFastSystemCall which is 5 byte (enough space for a 32-bit jump). Sadly that actually turned out not to be the case because the last byte, 0xC3 (retn), is needed by KiFastSystemCallRet and cannot be modified, which leaves only 4 writable bytes.

The sysenter instruction is supported by all modern CPUs and is the fastest way to enter the kernel. On ancient CPUs (before sysenter was invented) an interrupt was used (int 0x2E), for compatibility it was kept in all subsequent versions of windows.



```
77AE01E0
77AE01E0
77AE01E0
77AE01E0          KiIntSystemCall proc near
77AE01E0
77AE01E0          FirstArgument= byte ptr 8
77AE01E0
77AE01E0 8D 54 24 08      lea    edx, [esp+FirstArgument]
77AE01E4 CD 2E            int    2Eh
77AE01E6 C3              retn
77AE01E6          ntdll_KiIntSystemCall endp
77AE01E6
```

The now obsolete KiIntSystemCall

If we disassemble the first 5 bytes of any native function it will always be "mov eax, XX", this value is the ordinal of the function within the System Service Dispatch Table (SSDT). Once the call enters the kernel, a function will use this number to identify which entry in the SSDT to call, then call it (meaning each function has a unique number). When our hook is called, the SSDT ordinal will still be in the eax register, all we need to do is gather the SSDT ordinals for all the functions we need (by disassembling the first 5 bytes), then we can compare the number in eax with the ordinal for the function we wish to intercept calls for: if it's equal we process the call, if not we just call the original code.

Comparing the function ordinal with the one we want to hook could be messy, especially if we're hooking multiple functions.

```
cmp eax, [ntcreatefile_ordinal]
je ntcreatefile_hook
cmp eax, [ntcreateprocess_ordinal]
je ntcreateprocess_hook
[...]
jmp original_code
```

This code is going to get very long and inefficient the more functions are hooked (because every kernel call is passing through this code, the system could slow down), but there's a better way.

We can build an array of DWORDs in memory (assuming we just want to hook NtCreateFile & NtCreateProcess, let's say the NtCreateFile ordinal is 0x02 and NtCreateProcess ordinal is 0x04), the array would look like this:

```
my_array+0x00 = (DWORD)NULL
my_array+0x04 = (DWORD)NULL
my_array+0x08 = (DWORD)ntcreatefile_hook_address
```



```
my_array+0x0C = (DWORD)NULL
my_array+0x10 = (DWORD)ntcreateprocess_hook_address
[...]
```

Then we could do something as simple as:

```
lea ecx, [my_array]
lea edx, [4*eax+ecx] ;edx will be &my_array[eax]
cmp [edx], 0
je original_code
call [edx] ;call the address pointed to by edx
```

This is pretty much what the kernel code for calling the SSDT function by its ordinal does.

Hook Analyzer Security Tool

A useful tool for analyzing hooking is Hook Analyzer.

It allows an investigator/analyst to perform “static & run-time / dynamic” analysis of suspicious applications, also gather (analyze & correlated) threat intelligence related information (or data) from various open sources on the Internet.

<http://www.darknet.org.uk/2014/05/hook-analyser-3-1-malware-analysis-tool/>

Linux Rootkits

T0rn – rootkit replaced actual system binaries with the intent to hide information. Replaced programs included critical files such as du, find, ifconfig, login, ls, netstat, ps, sz and top. T0rn was an improvement over previous rootkits because of its increased chances for evading detection. When an administrator looked for anomalies using the swapped binaries, 'controlled' outputs were produced that masked underpinnings of the rootkit from observation. T0rn was still detectable with other UNIX tools.

Adore - Linux LKM based rootkit for Linux v2.x. Features smart PROMISC flag hiding, persistent file and directory hiding (still hidden after reboot), process-hiding, netstat hiding, rootshell-backdoor, and an uninstall routine. It includes a userspace program to control everything.

Phalanx - self-injecting kernel rootkit designed for the Linux 2.6 branch. Features include file hiding, process hiding, socket hiding, a TTY sniffer, a TTY connectback-backdoor, and auto injection on boot. Used in 2011 for SSH-key dictionary attacks. This rootkit uses /dev/mem/ interface to inject hostile code into kernel memory and hijack system calls. Phalanx allows continued privileged access to the compromised system while hiding its presence from administrators by subverting standard OS functionality. Once in place, the rootkit steals other SSH keys and sends them to the attacker to facilitate further attacks.

Darbe-A - 2012 kernel rootkit backdoor based in the /proc file system. Linux Kernel Module(security.ko) is injected into the system, control program(./kontrol fabrika) makes a normal user have root privileges.

<https://isc.sans.edu/diary/SSHD+rootkit+in+the+wild/15229>

WeaponX is a kernel based rootkit for Mac OSX (prior to 10.7) which is roughly based on adore. It runs as a kernel extension, similar to a LKM.

Using the Linux Kernel Module, a rootkit can modify the kernel's syscall table. Doing this the rootkit can replace a system call to point to a program of the rootkit. Another technique which a rootkit can use is to delete a log entry on the system so there will be no log entry of the attacker's activities. Also, to hide the attacker's tracks the rootkit can replace standard UNIX programs such as **ps** to not show the processes which the rootkit is running.

The problem with detecting good rootkits is that you can't even trust the kernel and operating system in which the rootkit is installed on. So this makes it hard to detect them by installing detection software directly on the affected operating system. A better solution is to install a packet sniffer on an unaffected machine to look at the information being sent to and from the machine which might have a rootkit installed on it. Looking at the local log files will not always allow the system administrator to detect an attacker using a rootkit because the rootkit can delete the entries the attacker makes.

Another way to detect rootkits is to boot from a live CD. This allows you to trust the kernel and the software running on the linux CD to investigate the files on the possibly affected computer for rootkits. Also there are programs which try to find rootkits locally like chkrootkit however this program depends on the local **ps** command to find them. As we know a rootkit can change the **ps** command to what it likes. Another problem with this approach is that the rootkit can detect and change the chkrootkit software. If the user finds the rootkit sometimes it is very hard to make sure that it has been removed. Most experts recommend that one should just reformat the system and start over. If using backups make sure that the backed up files don't contain harmful files. There is software which tries to remove rootkits called Rkdetector v2.0 (<http://www.rootkitdetector.com>).

New Linux Malware Detector (Linux Malware Detect) <http://www.rfxn.com/projects/linux-malware-detect/>

Uses pattern matching and IOC (indicators of compromise) from the CYMRU malware hash registry to find Linux malware.

A rootkit could be introduced in source code of Open source software if the distribution site was compromised.

If an attacker is really worried that the system administrator has all things hashed and the rootkit will be detected, they could avoid the file system altogether — perhaps installing a rootkit into memory and never using the drive. One drawback, of course, is that a rootkit stored in volatile memory will vanish if the system reboots.

To take things to an extreme, perhaps a rootkit can install itself into firmware present in the BIOS, PCI card or a flash RAM chip somewhere.

- Bootkit Threats evolution:
 - ✓ Win32/Mebrook (2007)
 - ✓ Win32/Mebratix (2008)
 - ✓ Win32/Mebrook v2 (2009)
 - ✓ Win64/Olmarik (2010/11)
 - ✓ Win64/Rovnix (2011)
 - ✓ Win32/GrayFish (2015)
 - ✓ Win64/GrayFish (2015)

Rootkits have been implemented as Windows services, device drivers and network protocols or network services.

Rootkits have been developed to run in VMWare Virtual machines.

Rootkits are often used with other trojans to hide a program that steals financial information, for example.

<http://arstechnica.com/security/2015/01/worlds-first-known-bootkit-for-os-x-can-permanently-backdoor-macs/>

Keyloggers

Keyloggers – a program to copy information typed on a keyboard and store it for retrieval by an attacker. These are usually written as a device driver with a rootkit feature to hide itself from the drivers list. They are used to steal usernames/passwords and other sensitive information (credit card numbers, SSN's, etc.)

Web Browser Keyloggers – keylogger that records the information typed into forms on web pages. Also call form grabbers since they can store the information enter on web forms used for logins, placing credit card orders, etc.

Video Keylogger – a program to copy typed information, mouse movements and mouse clicks. These are used to steal information type on an onscreen keypad like that used by some online banking websites. The output is stored as a movie file showing the area around the mouse when the mouse buttons are pressed.

Keyloggers store the captured keystrokes on the local disk. Often the file storing the keystrokes is hidden using rootkit techniques. One method for detecting a keylogger is to watch for file that only grows in size as you type. Some keyloggers pre-parse the keystrokes by looking for patterns that are likely to be username and password combinations and only storing the captured credentials. Some keyloggers connect out periodically and post the logged keystrokes to a website or ftp server.

Botnets

Botnet - a collection of software robots, or bots, which run autonomously and automatically. They run on groups of "zombie" or "bot" computers controlled remotely by hackers.

Botnets have been written in Java, Perl, PHP, C and Visual Basic.
Some common Botnet distributions:

- Agobot
- SDBot
- GAOBot
- Spybot
- Reptile
- Zbot
- Conficker
- Flashback
- PerlBot
- Aidra

The command and control server is typically a web server or a modified IRC server. If using an IRC server, each bot client joins one or more chatrooms to receive instructions and commands. Access to the IRC server and entrance into the chatroom can be password protected.

Here are some example Botnet commands:

- **ddos.synflood [host] [time] [delay] [port]**
starts a SYN flood
- **ddos.httpflood [url] [number] [referrer] [recursive = true|false]**
starts a HTTP flood
Fetch websites from a web server.
If "recursive" is set, the bot parses the replies and follows links recursively.
- **scan.addnetrange [255.255.255.255/32] [priority]**
- **http.update**
Executes a file from a given HTTP URL
The same command is also available via FTP
- **cvar.set spam_aol_channel [channel]**
AOL Spam - Channel name
- **spam.setlist**
downloads list with email-addresses to spam them
- **spam.settemplate**
downloads an email template
- **spam.start**
starts spamming
- **cvar.set sniffer_enabled 1/0 – enable packet sniffer**
options to sniff HTTP, FTP and IRC

New functions are added all the time. Many botnets are open source so you can add your own exploit spreader, dictionary attack function, encryption module, key logger, etc.

See page 7 of Georbot analysis for recent botnet command list:

http://blog.eset.com/wp-content/media_files/ESET_win32georbot_analysis_final.pdf

Also review pages 3 and 4

Mac Botnet used Reddit for communication:

<http://grahamcluley.com/2014/10/mac-malware-botnet-reddit/>

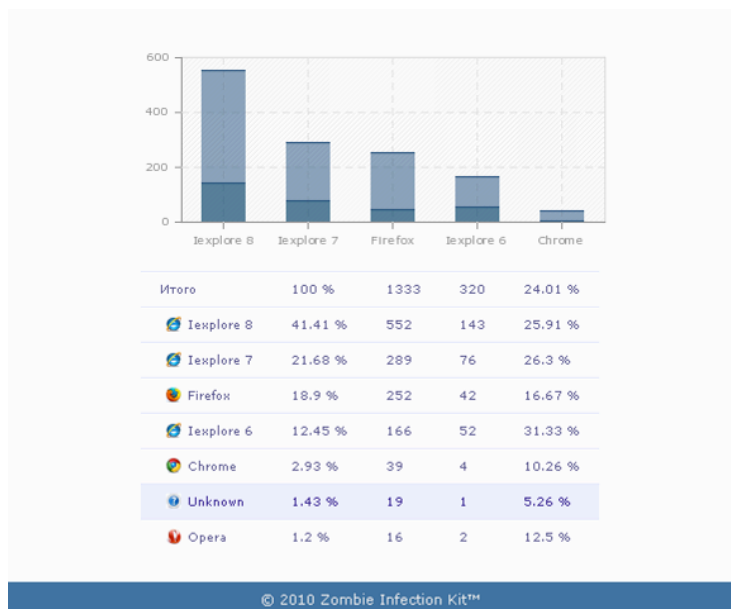
Modern Malware is Mass Produced and Distributed Using Exploit Kits

The number of crimeware application suites has grown in recent years making it easier to produce malicious code, build botnets, create phishing attacks, etc.

Example Crimeware applications are:

- Blackhole Exploit Kit (author was arrested in Nov. 2013)
- CritX exploit kit
- Magnitude exploit kit
- Nuclear exploit kit
- Fiesta exploit kit
- Angler Exploit Kit
- Redkit Exploit Pack
- Styx Exploit Pack
- Astrum Exploit Kit

Zombie Infection Kit:



This screen shot from Zombie Infection Kit Shows the real-time Browser exploitation Statistics.

Note the support for Firefox and Google Chrome.

SEO Sploit Pack:



The screenshot shows the 'SEO SPLOIT PACK' interface with a navigation bar at the top containing 'Total', 'Browser', 'OS', 'Country', 'Referer', 'Exploit', 'Upload', and 'Clear'. Below the navigation bar is a table titled 'Statistics on Exploits' with three columns: 'Exploit', 'Loads', and 'Percent'. The data is as follows:

Exploit	Loads	Percent
Arbitrary	780	15.82
JavaDES1	429	8.7
JavaGSB2	375	7.61
JavaOCon	950	19.27
JavaSMB3	1047	21.24
MDAC	126	2.56
PDFall	1223	24.81

This screen shot from the SEO Sploit Pack shows the Effectiveness of various exploits targeting Java, PDF and Windows.

Note the number of Java exploits



The screenshot shows the 'SEO SPLOIT PACK' interface with a navigation bar at the top containing 'Total', 'Browser', 'OS', 'Country', 'Referer', 'Exploit', 'Upload', and 'Clear'. Below the navigation bar is a table titled 'Statistics on Exploits' with three columns: 'Exploit', 'Loads', and 'Percent'. The data is as follows:

Exploit	Loads	Percent
Arbitrary	990	8.82
JavaBOF	1	0.01
JavaDES1	1105	9.85
JavaGSB2	1233	10.99
JavaOCon	2682	23.9
JavaSMB3	2306	20.55
MDAC	139	1.24
PDFall	2766	24.65
U3dPDF	1	0.01

This screen shot from the SEO Sploit Pack shows the Effectiveness of various exploits targeting Java, PDF and Windows.

Note the number of Java exploits.

Out of date Java clients are very common.

Blackhole Infection Kit:



This screen shot from Blackhole Infection Kit shows the efficiency of various exploits available in the kit.

DENIS >	13285	11505	1187	10.32
default >	4	3	1	0.00

This screen shot from Blackhole Infection Kit shows the percentage of browsers and Operating Systems Infected.

БРАУЗЕРЫ	ХИТЫ	ХОСТЫ	ЗАГРУЗКИ	% ↑
Chrome >	2273	2148	485	22.58
Mozilla >	104	72	11	15.71
Firefox >	5033	4847	581	11.99
Opera >	360	288	22	7.75
MSIE >	4232	3080	77	2.51
Safari >	1287	1102	11	1.00

Note the support for Opera, Safari and Google Chrome.

ОС	ХИТЫ	ХОСТЫ	ЗАГРУЗКИ	% ↑
Windows 2003	21	18	5	27.78
Windows 2000	41	22	4	18.18
Linux	179	143	19	13.48
Windows XP	3838	3206	399	12.48
Windows 7	5059	4490	478	10.66
Windows Vista	3173	2752	264	9.61
Mac OS	978	900	18	2.00

Some bots are Linux and OS X computers.

Most Exploited Applications:

- Java runtime environment
- Adobe Reader
- Flash Player
 - flash for Devices is going away
- SilverLight
- Internet Explorer
- Media Player

A survey found that only 2 percent of Windows systems had no out-of-date programs

Trojan E-mail Attachments

Email attachments remain one of the primary delivery vectors for trojans. Recent trojans have been delivered as email attachments that appear to come from UPS, FedEx or DHL. The Trojan comes in an attachment that purports to be an invoice for an undeliverable package, and then stealthily allows your financial data to be stolen by foreign crooks. UPS rarely sends attachments, so if you receive something suspicious, don't open it. A variant of this is an attachment that seems to come from an airline with the attachment being tickets you purchased. The attachment is a trojan and could be a .ZIP, .PDF or .DOC file.

Chanitor story

<https://www.damballa.com/following-an-advanced-malware-infection-chain/>

Running Antivirus Alone is not a Winning Strategy

- Only 38% of Zeus malware is detected by antivirus software
- A recent study by University of Alabama at Birmingham found that antivirus detection rates averaged around 25% for malware arriving in email.

Malware analysis sites:

<http://wepawet.iseclab.org> – wepawet analyzes suspicious javascript/PDF and Flash files

<http://anubis.iseclab.org> – Anubis analyzes activity of suspicious binaries and reports network pcap, process activity, file system activity and registry activity

<https://malwr.com/> - analyzes suspicious binaries using the cuckoo sandbox and produces report of file systems activity, process activity, network activity and screen shots.

<https://www.virustotal.com> – analyzes files submitted using dozens of virus scanners, also analyzes websites and Android apps to determine if they are malicious

totalhash.com – analyze files submitted using sandbox, multiple virus scanners, allows searching by hash, User-Agent string and other artifacts from files and metadata

www.hybrid-analysis.com - new sandbox service for static and dynamic analysis of submitted files

Malware Analysis Tools

These tools can be useful for analyzing suspect email attachments.

OfficeMalScanner - <http://www.reconstructor.org/code/OfficeMalScanner.zip>

locates shellcode and VBA macros from MS Office (DOC, XLS, and PPT) files.

Officemalscanner.exe evil.ppt scan → use heuristics to locate exploits and shellcode

Officemalscanner.exe evil.ppt info → dump macros into text files

pyOLEScanner.py - <https://github.com/Evilcry/PythonScripts>

can examine and decode some aspects of malicious binary Office files.

Analyzing PDF files

Adobe PDF File Format Notes

A PDF File is comprised of header, objects, cross-reference table (to locate objects), and trailer.

“/OpenAction” and “/AA” (Additional Action) specifies the script or action to run automatically.

“/Names”, “/AcroForm”, “/Action” can also specify and launch scripts or actions.

“/JavaScript” specifies JavaScript to run.

“/GoTo*” changes the view to a specified destination within the PDF or in another PDF file.

“/Launch” launches a program or opens a document.

“/URI” accesses a resource by its URL.

“/SubmitForm” and “/GoToR” can send data to URL.

“/RichMedia” can be used to embed Flash in PDF.

“/ObjStm” can hide objects inside an Object Stream.

Be mindful of obfuscation with hex codes, such as “/JavaScript” vs. “/J#61vaScript”. (See examples at <http://blog.didierstevens.com/2008/04/29/pdf-let-me-count-the-ways/>)

Tools for Analyzing Adobe PDF Files

[PDF Stream Dumper](http://sandsprite.com/blogs/index.php?uid=7&pid=57) combines many PDF analysis tools under a single graphical user interface.
<http://sandsprite.com/blogs/index.php?uid=7&pid=57>

[Jsunpack-n's pdf.py](http://jsunpack.blogspot.com/2009/06/jsunpack-n-updates-for-pdf-decoding.html) extract JavaScript from PDF files.

<http://jsunpack.blogspot.com/2009/06/jsunpack-n-updates-for-pdf-decoding.html>

More info on malware analysis:

Free online malware analysis class

<http://blog.segu-info.com.ar/2014/08/clase-gratuita-sobre-analisis-dinamico.html>

Malware Dynamic Analysis class:

<http://opensecuritytraining.info/MalwareDynamicAnalysis.html>

Coursera has a free course about Malicious Software:

<https://class.coursera.org/malsoftware-001/class/index>

Course Outline:

Week 1: Introduction

Week 2: Static Analysis

Week 3: Dynamic Analysis

Week 4: Mobile Malware

Week 5: Cybercriminal underground economy

Week 6: The cost of cybercrime