Table of Contents

Malware Malicious Software

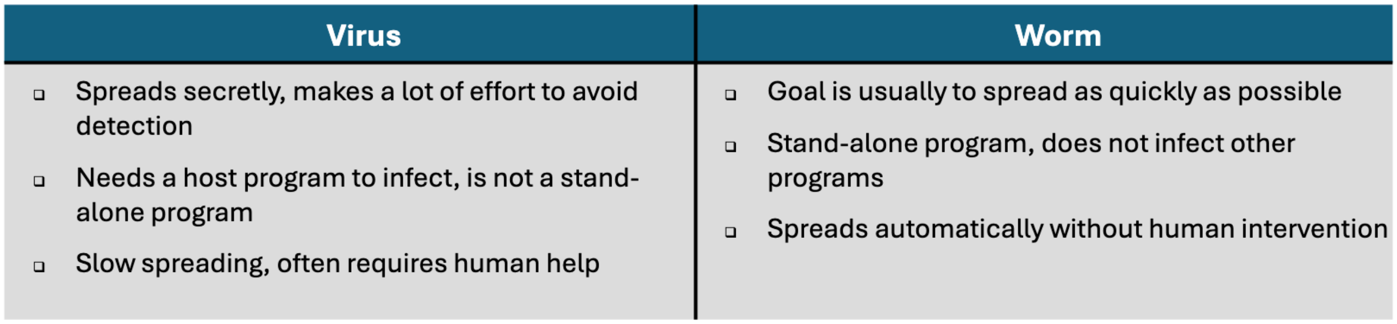
- hostile or intrusive SW designed to infiltrate/damage a computer system, installed w/o owner's informed consent
    - trojan, backdoor, spyware

Virus

- Both **virus** and **worms** replicate automatically and consume system resources

Virus Goals

- hide presence
- replicate themselves

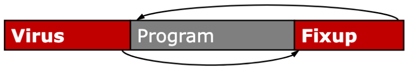


Virus	Worm
❑ Spreads secretly, makes a lot of effort to avoid detection ❑ Needs a host program to infect, is not a stand-alone program ❑ Slow spreading, often requires human help	❑ Goal is usually to spread as quickly as possible ❑ Stand-alone program, does not infect other programs ❑ Spreads automatically without human intervention

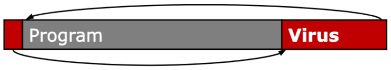
- Inserts own instructions into existing programs
    - virus is executed when the infected program is run
    - on execution, virus may propagate to other programs
- Early virus often infected *disk boot sector*
    - virus loads in memory when disk

Insertion Points

- **Beginning of program** - overwrites start of program, insert fix-up code to replicate code it overwrote
  - virus length is limited as it can't overwrite too much of the program before causing problems



- **End of program** - overwrites first instruction of program with a jump to virus code, then jumps back to beginning of program
  - overwrites one instruction at the start of program, length is not limited



Virus Detection

- virus scanners look for **signatures**, which are strings of bits corresponding to instructions found in known viruses
    - malware is analyzed manually to build signatures
    - signatures are sent to customers periodically and the virus scanner looks for the existence of these signatures in programs on the customer's machines

Designing Signatures

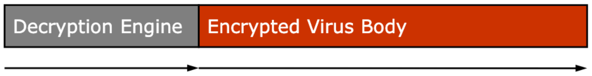
- **long enough** so legitimate code not marked as virus
- **too long** → misses variants of viruses (one signature per variant)
- more signatures, better accuracy

EICAR Anti-Virus Test

- EICAR standardized a common test signature that all anti-virus programs (should) recognize if they are properly working
- When put into a file, should trigger a virus alert

Polymorphic Viruses

- small decryption engine decrypts the rest of the virus
    - simple scheme (XOR) to **defeat signature based scan**
    - encryption changed whenever virus propagates to a new file (encryption body varies)
- vary the virus payload from which a signature is generated



- decryption engine short and simple; hard to build signature that doesn't create numerous false-positives
- **Defeating Polymorphic Viruses**
    - advanced virus scanners will run files they suspect infected in **emulator**
    - as decrypter is run, virus body will be decrypted; signature scanner detects
    - scanner runs for a short time (beginning of program)
    - if no signature matches, virus scanner declares the file clean

Metamorphic Viruses

- change their code on every infection by rewriting themselves
    - changing register allocations, equivalent instruction sequences, change order of blocks of code
    - Some also integrate themselves into different portions of the infected program, and not just at the beginning
        - they may not be always executed, slowing the infection rate
        - harder to detect presence
- Defeating Metamorphic Viruses
    - run emulator & look for sequences of executed instruction

    - viruses leave **markers** in infected files so they know not to infect them again (look for markers)

Worm

- spread automatically by identifying & exploiting vulnerabilities in hosts
    - after finding a new vulnerable machine, the worm installs itself on the machine and searches for another machine
    - worms can spread fast, high-speed networking ubiquitous

Morris Worm

- 2 parts: server program, bootstrap/vector program
    - server looked for vulnerable remote target machines and tried to exploit a vulnerability on them
    - successful = create shell on target, uploaded the vector, compiled it on target and then ran the vector program; download worm from server, and starts server on newly infected host

Exploited 4 Vulnerabilities...

- vulnerable versions of fingered program = buffer overflow
- sendmail on many systems had been compiled with a DEBUG option. By connecting to the sendmail port and sending DEBUG, attacker received a root shell
- worm try popular passwords, stored in /etc/passwd, readable by any user
- worm connects to hosts in /etc/hosts.equiv. Hosts in this file are other mahcines in which users can log into without a password

Hide itself

- prevent itself from re-infecting already-infected machines
- deletes files after they were loaded in memory, obscured program arguments, and killed unneeded parent processes
    - difficult to analyze/detect the worm

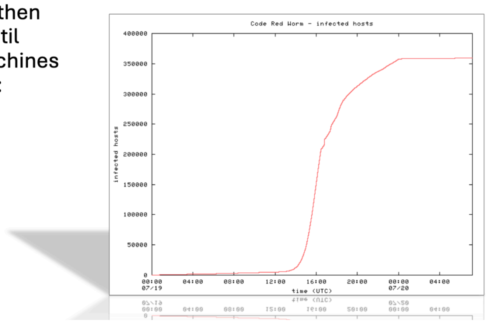
Modern Worms

Increase Speed

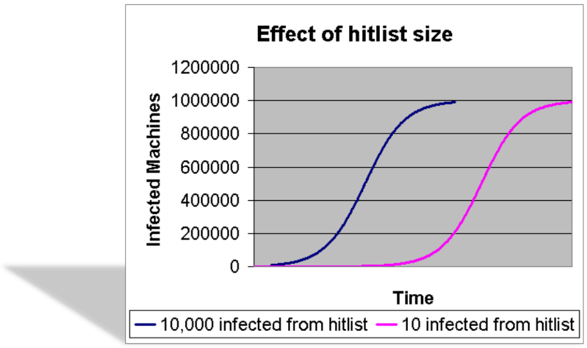
- Improve scanning speed for vulnerable machines
    - hit lists scanning
        - worm builder pre-seeds worms with hosts that are potentially vulnerable and with high BW, so that initially the worm has a lot of scanning bandwidth
    - local scanning v.s. random scanning
        - worm can try infecting local hosts first since connecting to them will be faster
- Use UDP instead of TCP
    - UDP no ACK, send attack packet w/o having to wait for ACK (BW-limited than latency-limited)

Worm Propagation

Worms start slowly, then propagate rapidly until most vulnerable machines have been exploited:



Worm Propagation: Hitlist Seeding



Worm Defenses

- prevent attacks
  - patch systems
  - disable unnecessary services
  - services not externally visible should be firewall-ed
- after worm is released
  - shutdown vulnerable service, so that it cannot be infected
  - create a signature for detected worms & filter them at network layer

Earlybird (UC San Diego)

- As a worm starts spreading, its code will appear in an increasing number of network messages
- Thus worms can be identified by looking for commonly occurring substrings in packets
- To reduce false positives, filter by only flagging substrings that appear in many packets with highly diverse src/dest pairs

Shields (Microsoft Research)

- Vulnerabilities often lie in obscure paths (*i.e.*, some rarely used feature) in network protocols that aren't heavily tested
- Worms exercise these paths so if a protocol is in one of these paths, then block
- Works as a quick fix until a patch can be released so you don't have to shut down vulnerable service

Zero-Day Exploits

- previously-unknown, exploitable security vulnerability is a **zero-day exploit**
  - Stuxnet - 4 0day exploits
    - LNK Vulnerability
    - Windows Printer Sharing
    - Windows Keyboard Driver
    - STEP7 PLC Controller

Driver Signing

- windows uses signing mechanism to validate OS files & drivers
  - critical files are signed by author, using public-key signature, to establish the authenticity of a file
  - used to detect changes to file

Rootkit

- software designed to hide the fact that system has been compromised

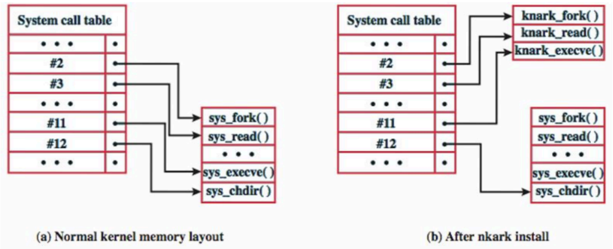
  - subverts the mechanisms that report on processes, files, registry entries, etc
- rootkit may be
  - memory based - does not survive a reboot
  - persistent - stored in config file and runs on each boot

TDL3 & Festi

- extremely prevalent rootkits in 2010
- advanced stealth mechanisms

Kernel System - Call Hooking

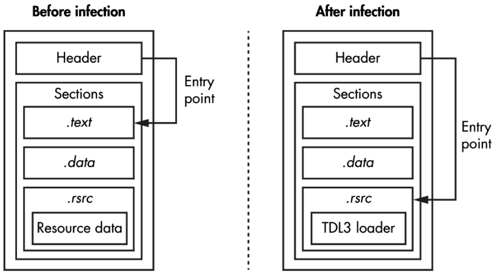
- to maintain stealth, a rootkit typically **intercepts system calls** in the kernel (hooking) to modify returned results and hide its files, registry keys, process info etc.
  - similar to PLT/GOT attack

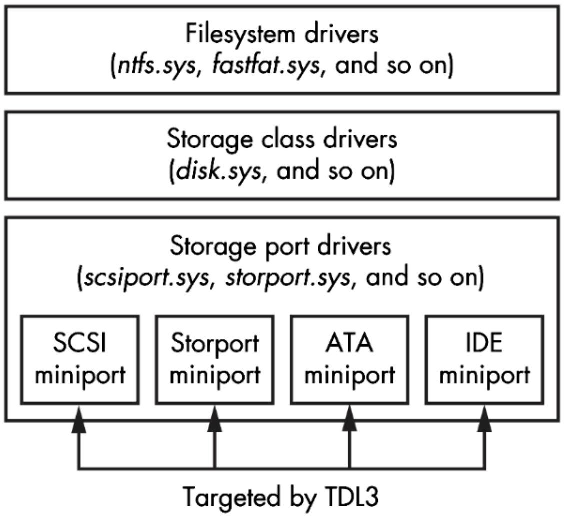


(a) Normal kernel memory layout

(b) After nkark install

- **TDL3** injected malicious code into Windows boot-start drivers (essential component in the loading of the OS)
  - sophisticated hooking - intercepting R & W I/O operations at the very bottom of the storage driver stack
    - intercept all R/W calls
    - hide any references to itself
    - create its own hidden, encrypted FS on disk





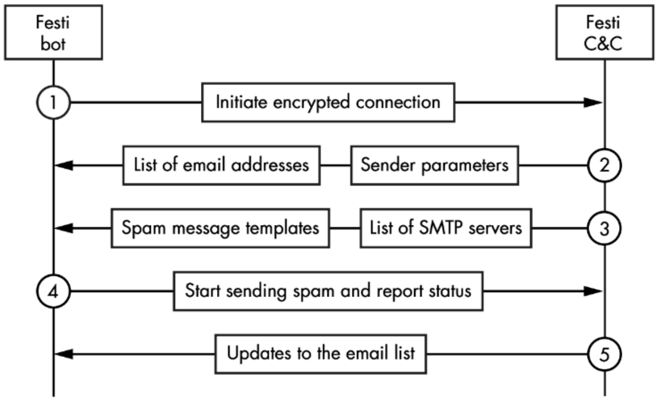
S

Botnets

- collection of compromised machines running malicious software under a common command-and-control infrastructure
  - software typically installed using worms, trojans, backdoors
  - used for DoS, spamming, etc
- **characteristics**
  - remote control facility for coordinating bot machines
    - use IRC, HTTP, covert channels, with optional encryption
    - newer botnets provide GP remote execution
    - remote-control is one feature that distinguishes a botnet from a worm

Festi

- uses Command & Control server to deliver its plug-in payloads (TCP flood, UDP flood, DNS flood...)
- these payloads were stored only in RAM, which made analysis more-difficult
- deployed **Anti-Virtual Machine** techniques to increase difficulty of reverse-engineering
  - use syscall to determine whether in virtual environment (& inform C&C server)
  - if so, C&C server would return alternate plug-ins, to confuse security researchers
- employed **Anti-Debugging** techniques: check for kernel debuggers, and actively disable debugger breakpoints
- used random domain-name generator, so creators could keep rotating domain names and stay ahead of "take down requests"



Defenses

HIPS (Host Instrusion-Prevention Systems) typically analyze at least 3 of the most common OS components that rootkits use for infection

- **System-Event Hooking** hooks OS calls related to process creation (intercept start-up of applications and inject malicious code into userspace memory as they boot)
- **System-call hooking** provides false info and hide the files/keys/existence of the rootkit from the user and malware scanners
- **Intercepting the Object Dispatcher** intercept creation of objects within kernel; these DS represent all resources the OS manages

Rootkit Defenses: Object Dispatcher

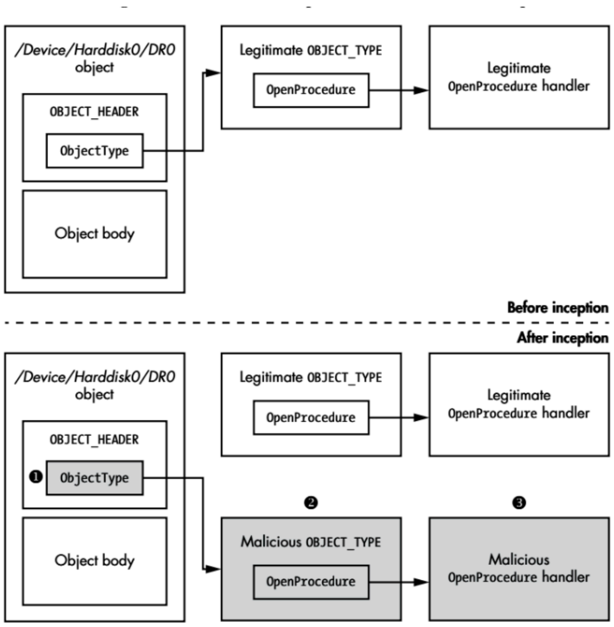


Image Credit: *Rootkits and Bootkits.*

Content-Based Attacks

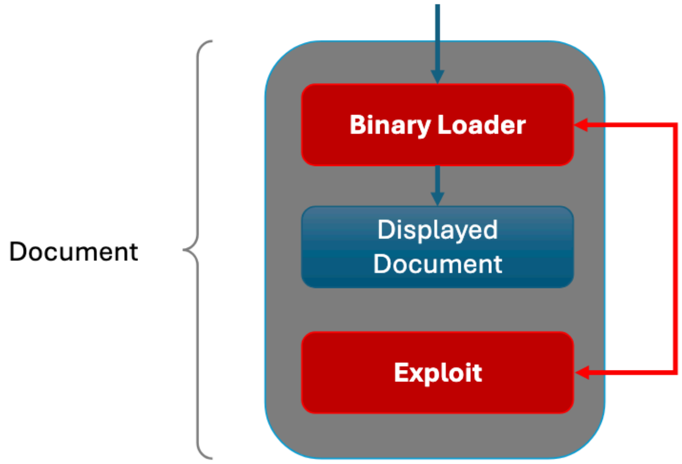
- Vulnerability in Samsung Notes allows attackers to read memory and execute arbitrary code via a malicious JPEG file

File Formats

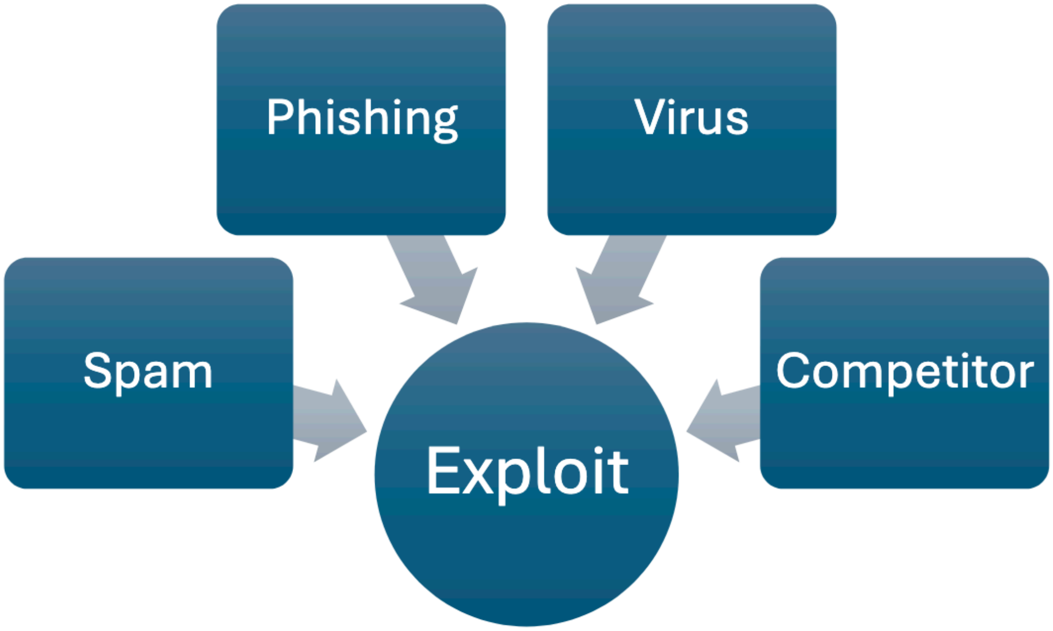
The format specifications for many popular document types (PDF, DOC, etc.) are now extremely complex:

- **PDF spec (v6):** 1,310 pages
- **Microsoft Open XML spec:** 929 pages just to describe how Microsoft has *interpreted* the ISO/IEC 29500 standard that specifies the Open XML document format

Content-Type Attacks



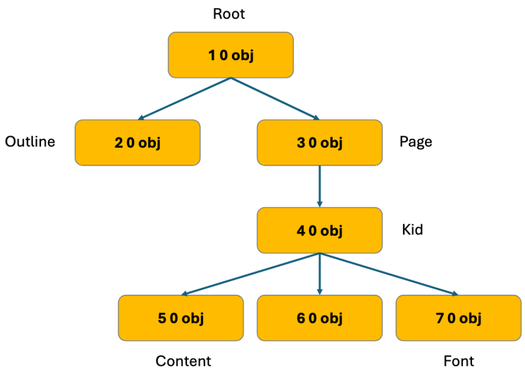
Attack Vectors



Malicious PDF Files

- PDF files start with a line describing the PDF language version
  - `%PDF-1.1`
- Remainder of the file describes a hierarchy of **objects** making up the doc

```
[index #] [version #] obj
<<
[content]
>>
```



- data is uncompressed by default, but often compressed to obfuscate the contents

```
2 0 obj
<< >> stream
BT/default 99 Tf 1 0 0 1 1 715 Tm
(Hello World!) Tj ET
endstream
endobj
```

- Objects can contain malicious javascript, automatically called when document opens

```
7 0 obj
<<
 /Type /Action
 /S /JavaScript
 /JS (app.alert(‘Hello world’);)
>>
endobj
```

Detecting Attacks

- attackers obfuscate their code
- exploits can be triggered through a number of other means (annotations, forms, etc.)
  - contents of those objects may be compressed

Defense

- disable Javascript (breaks some PDF forms)
  - turning off library features
  - virus scanners (reactive not proactive
- recent versions of Adobe Reader for Windows now all ship with DEP (non-executable stack/heap) turned on by default
  - widget-based programming (like ROP) is possible

Social Engineering Trends

Unicode: “Right-to-Left”

Unicode is increasingly replacing ASCII as the standard for encoding text, in order to support everyone’s languages. Many languages aren’t written from left-to-right.

Unicode character (U+202E) is defined as the **Right-to-Left Override** (RLO). It switches the direction that text is displayed in, and is increasingly being used in a variety of social-engineering “phishing” attacks:

```
Resume - John Al[RLO]cod.exe
```

displays as: `Resume - John Alexe.doc`

```
www.payp[RLO]moc.la
```

displays as: `www.paypal.com`