

12. Malicious Software


Computer Security Courses @ POLIMI

What is “malware”?

- A *portmanteau* of “MALicious softWARE”
 - Meaning: **code that is intentionally written to violate a security policy.**
- Several "categories", or "features":
 - **Viruses:** code that **self-propagates** by infecting executables or other files (but also documents with macros, boot loader). They are not standalone programs (i.e., executables).
 - **Worms:** programs that **self-propagate**, even remotely, often by exploiting host vulnerabilities, or by social engineering (e.g., mail worms).
 - **Trojan horses:** apparently benign program that hide a malicious functionality and allow remote control.

A Brief History of Malware...Viruses

1971 – **Creeper** is first self-replicating program on PDP-10



IN THE CREEPER. CATCH ME IF YOU CAN!

1981 – First outbreak of **Elk Cloner** on Apple II floppy disks

1983 – The first documented experimental virus (Fred Cohen's pioneering work; name coined by Len Adleman)

Theory of Computer Viruses

- Fred Cohen ('83), theorized the existence and produced the first examples of viruses
 - From a theoretical computer science point of view, interesting concept of **self modifying** and **self propagating code**.
 - Soon, the security challenges were understood.
- It is **impossible** to build the **perfect virus detector** (i.e., propagation detector)
 - Let **P** be a perfect detection program
 - We can build a virus **V** with **P** as a subroutine:
 - if **P(V)** = True $\sim\rightarrow$ **V** halts (does not spread) \rightarrow **V** is not a virus
 - if **P(V)** = False $\sim\rightarrow$ **V** spreads \rightarrow **V** is a virus

The Malicious Code Lifecycle

Reproduce, infect, stay hidden, run **payload**

Reproduction and Infection phase

- Balance **infection versus detection** possibility
- Need to identify a suitable **propagation vector** (may be social engineering, vulnerability exploits)
 - Need to infect files (viruses only) and propagate to other machines
- **Note:** most modern malware does not self-propagate at all (most 2bots and trojans).

Variety of techniques to stay hidden.

Payload: Sometimes harmful...others..not, simply annoying

- ping-pong: https://www.youtube.com/watch?v=v1-jNkzBx_M
- Yerg: <https://youtu.be/966O0sXubxw>
- Cascade: <https://www.youtube.com/watch?v=gN7JKHOVnJ0>
- YAAI: <https://youtu.be/LSgk7ctw1HY?t=54>

Infection Techniques (Virus)

- **Boot viruses**

- Master Boot Record (MBR) of hard disk (first sector on disk) or boot sector of partitions
 - e.g., Brain, nowadays Mebroot/Torpig
- Rather old, but interest is growing again
 - diskless workstations, virtual machines (SubVirt)

- **File infectors**

- simple overwrite virus (damages original program)
- parasitic virus
 - append code and modify program entry point
- (multi)cavity virus
 - inject code in unused region(s) of program code

A Brief History of Malware...Viruses

1971 – **Creeper** is first self-replicating program on PDP-10

1981 – First outbreak of **Elk Cloner** on Apple II floppy disks

1983 – The first documented experimental virus (Fred Cohen's pioneering work; name coined by Len Adleman)

1987 – **Christmas worm** (mass mailer) hit IBM Mainframes (500,000 replications/hour) -> paralyzed many networks



1988 – **Internet worm** (November 2, 1988): created by Robert **Morris** Jr. - birth of CERT

Worms

- How 99 lines of code brought down the Internet (ARPANET actually) in Nov. 1988
- Robert Morris Jr. (at the time a Ph.D student at Cornell), wrote a program that could:
 - Connect to another computer, and find and use one of several vulnerabilities (e.g., buffer overflow in fingerd, password cracking) to copy itself to that second computer.
 - Begin to run the copy of itself at the new location.
 - Both the original code and the copy would then repeat these actions in an infinite loop to other computers on the ARPANET (**mistake!**)

A Brief History of Malware...Viruses

1971 – **Creeper** is first self-replicating program on PDP-10

1981 – First outbreak of **Elk Cloner** on Apple II floppy disks

1983 – The first documented experimental virus (Fred Cohen's pioneering work; name coined by Len Adleman)

1987 – **Christmas worm** (mass mailer) hit IBM Mainframes (500,000 replications/hour) -> paralyzed many networks

1988 – **Internet worm** (November 2, 1988): created by Robert **Morris** Jr. - birth of CERT

1998 – Back Orifice, the **trojan** for the IRC masses -> to demonstrate the lack of security in MS systems

1999 – [Melissa](#) virus: large-scale email macro-virus (the **first concept macro virus** was in 1995)

“Macro” Viruses

- Data files traditionally safe from viruses
- Macro functionalities add code to data files
- **Example:** spreadsheet macros can
 - Modify a spreadsheet
 - Modify other spreadsheets
 - Access address book
 - Send email
 - ...
- *Successful example:* the Melissa virus.
- Difficult to remove

...Turning Into an History of Worms

1999 – First DDoS attacks via trojaned machines (zombies)

1999 – Kernel Rootkits become public (Knark, modification of system call table)

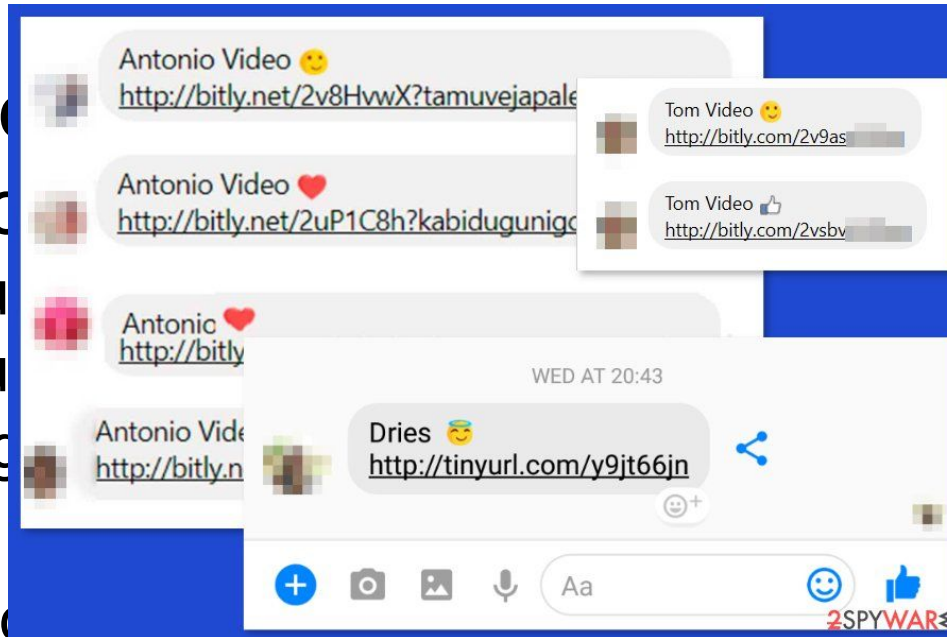
2000 – [ILOVEYOU](#) (large-scale email worm) - Social Engineering

Mass-mailers: Rebirth of the Worms

- Email software started allowing attached files, including:
 - Executables (dancing bears)
 - Executables masquerading as data
 - E.g. “LOVE-LETTER-FOR-YOU.txt.vbs”
- Spread by emailing itself to others
 - Use address book to look more trustworthy
- Modern variations ?

Mass-mailers: Rebirth of the Worms

- Email so... attached
 - files, inc...
 - Execu...
 - Execu...
 - E.g...
- Spread...
 - Use a...
- Modern variations include social networks to spread (e.g., ever received a suspicious-looking Twitter message/ facebook post from a friend?)



...Turning Into an History of Worms

1999 – First DDoS attacks via trojaned machines (zombies)

1999 – Kernel Rootkits become public (Knark, modification of system call table)

2000 – [ILOVEYOU](#) (large-scale email worm) - Social Engineering

2001 – **Code Red** ([large-scale](#), exploit-based worm)

2003 – **SQL Slammer** worm (extremely fast propagation through UDP)

Modern Worms: Mass Scanners

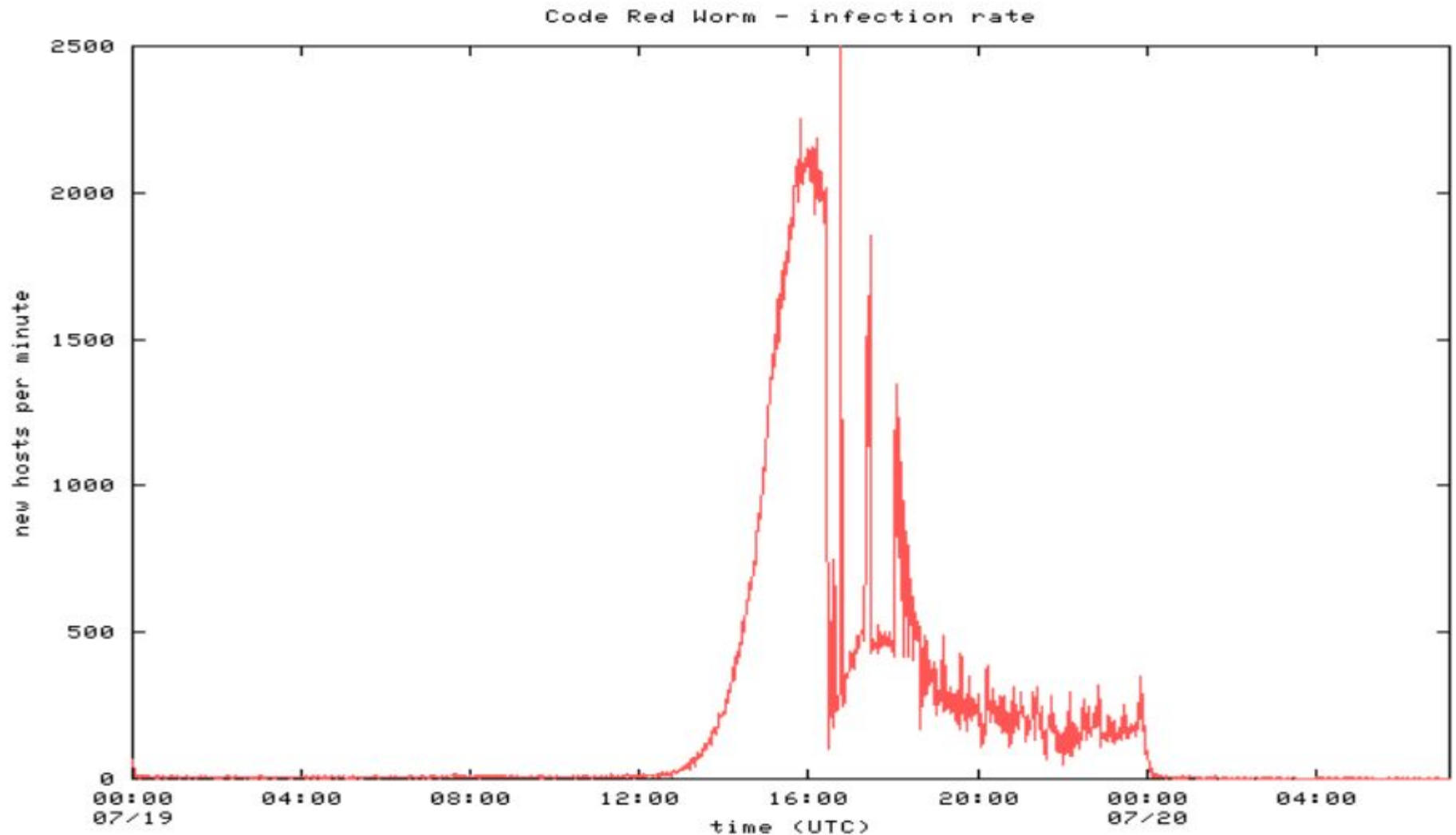
- Basic pattern the same:
 - Infect a computer and seek out new targets.
- Faster spread (minutes), larger scale (hundreds of thousands of hosts)
- Scanning
 - Select random address
 - Local preference: more scans towards local network
 - Permutation scanning (divide up IP address space)
 - Hit list scanning
 - Warhol worm: Hit list + permutation

Paper "How to Own the Internet in your spare time",
<http://www.icir.org/vern/papers/cdc-usenix-sec02/>)

Worm history

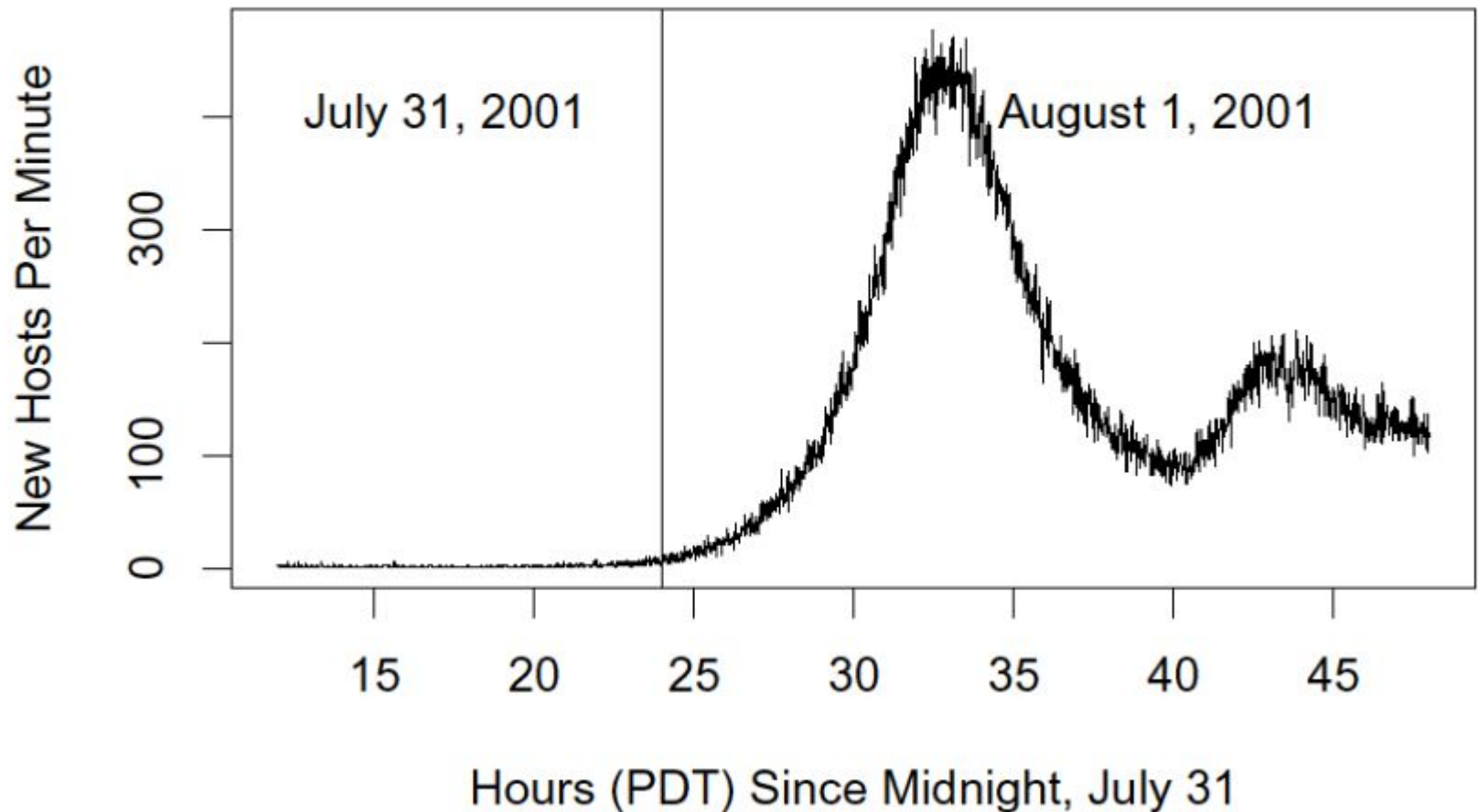
Worm	Date	Distinction
Morris	11/88	Used multiple vulnerabilities, first one :)
ADM	5/98	Random scanning of IP address space
Lion	3/01	Stealthy, rootkit worm
Cheese	6/01	Vigilante worm that secured vulnerable systems
Code Red	7/01	Completely memory resident
Walk	8/01	Recompiled source code locally
Nimda	9/01	Windows worm: client-to-server, c-to-c, s-to-s
Scalper	6/02	11 days after announcement of vulnerability; peer-to-peer network of compromised systems
Slammer	1/03	Used a single UDP packet for explosive growth

Code Red: Propagation



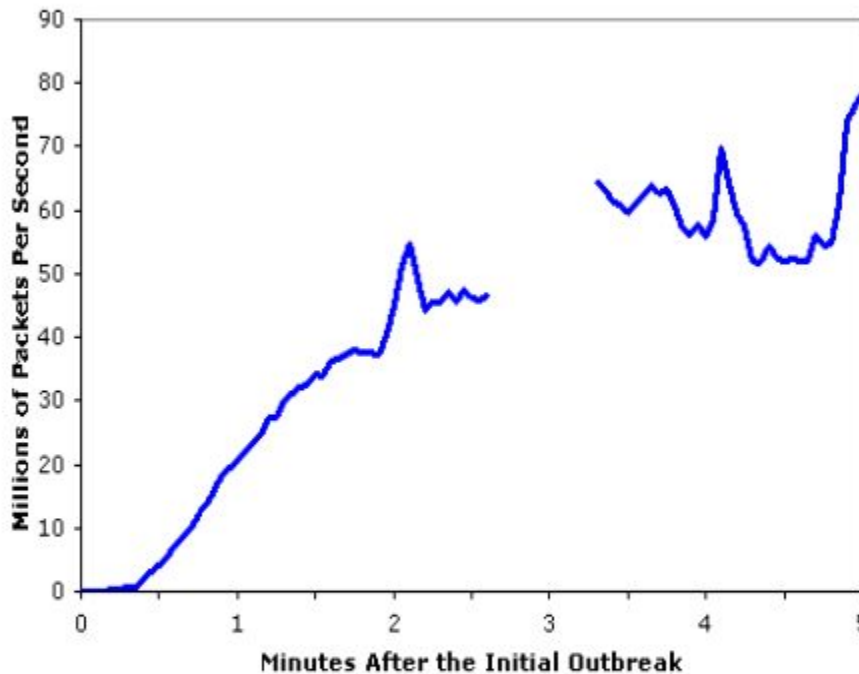
Code Red: Reactivation

Return of Code Red Worm

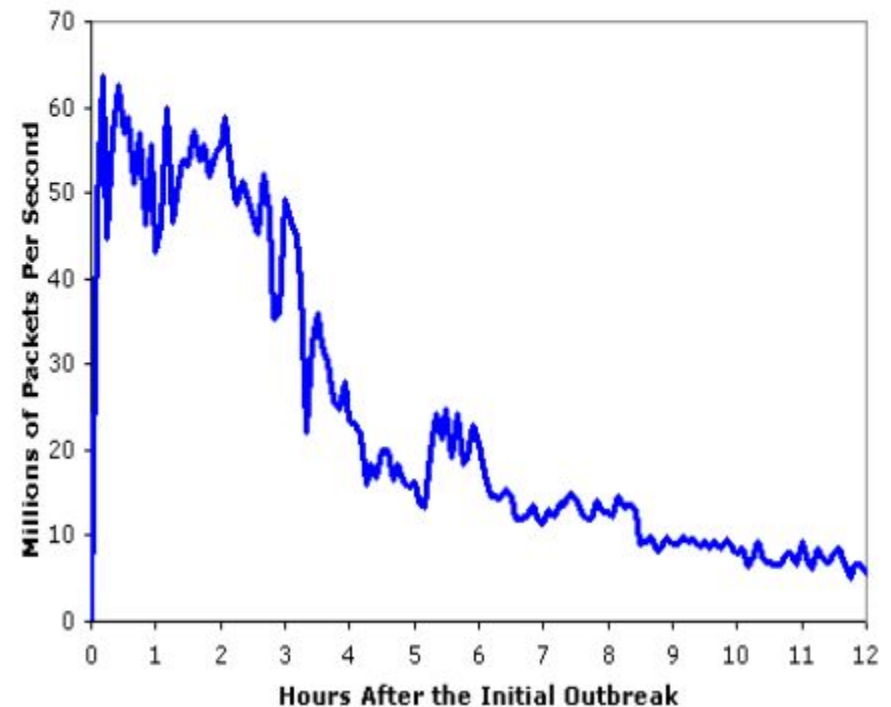


Slammer: UDP Propagation Disaster

Aggregate Scans/Second in the first 5 minutes based on Incoming Connections To the WAIL Tarpit



Aggregate Scans/Second in the 12 Hours After the Initial Outbreak



Silence on the Wires

- We all thought that the Internet would get “wormier”
- Since 2004, “silence on the wires”. No new “major” worm outbreaks
 - Weaponizable vulnerabilities were there, we even collectively braced for impact a couple of times :-)

Where did the worm writers disappear?

Related Questions...

- Why no worm has ever targeted the Internet infrastructure?
 - Traditional worm for propagation + specialized payload for infrastructure damage
- Windows of opportunity were there:
 - June 2003: MS03-026, RPC-DCOM Vulnerability (Blaster) + Cisco IOS Interface Blocked by IPv4 pkts
 - April 2004: MS04-011, LSASS Vulnerability (Sasser) + TCP resets on multiple Cisco IOS products
- So why **/bin/laden** did not strike?

Similar Questions...

- Why no worm has ever targeted the Internet infrastructure?
 - Traditional worm for propagation + specialized payload for infrastructure damage
- Windows of opportunity were there:
 - June 2003: MS03-026, RPC-DCOM Vulnerability (Blaster) + Cisco IOS Interface Blocked by IPv4 pkts
 - April 2004: MS04-011, LSASS Vulnerability (Sasser) + TCP resets on multiple Cisco IOS products
- So why **/bin/laden** did not strike?
 - Answer: **lack of motivation, attackers need the infrastructure to be up to do their stuff.**

...similar answer!

- The attackers are now interested in **monetizing** their malware
- **Direct** monetization (e.g., abuse of credit cards, connection to premium numbers)
- **Indirect** monetization
 - Information gathering
 - abuse of computing resources
 - rent or sell botnet infrastructures

All of this created a growing **underground economy**.

The Cybercrime Ecosystem

- Organized groups
- Various "activities"
 - exploit development and procurement
 - site infection
 - victim monitoring
 - selling "exploit kits"
 - ...they also offer support to their clients.
- Further reading

"Manufacturing Compromise: The Emergence of Exploit-as-a-Service"

<http://cseweb.ucsd.edu/~voelker/pubs/eaas-ccs12.pdf>

...Turning Into an History of Worms

1999 – First DDoS attacks via trojaned machines (zombies)

1999 – Kernel Rootkits become public (Knark, modification of system call table)

2000 – [ILOVEYOU](#) (large-scale email worm) - Social Engineering

2001 – **Code Red** ([large-scale](#), exploit-based worm)

2003 – **SQL Slammer** worm (extremely fast propagation through UDP)

2004+ – **Malware that create botnet infrastructures**
(e.g., **Storm Worm**, **Torpig**, **Koobface**, **Conficker**, **Stuxnet**)

Introducing Bots!

The Jargon File, version 4.4.7:

```
bot: n  [common on IRC, MUD and among gamers; from  
        "robot"]
```

```
1. An IRC or MUD user who is actually a program. On IRC,  
typically the robot provides some useful service. Examples  
are NickServ, which tries to prevent random users from  
adopting nicks already claimed by others, and MsgServ,  
which allows one to send asynchronous messages to be  
delivered when the recipient signs on.
```

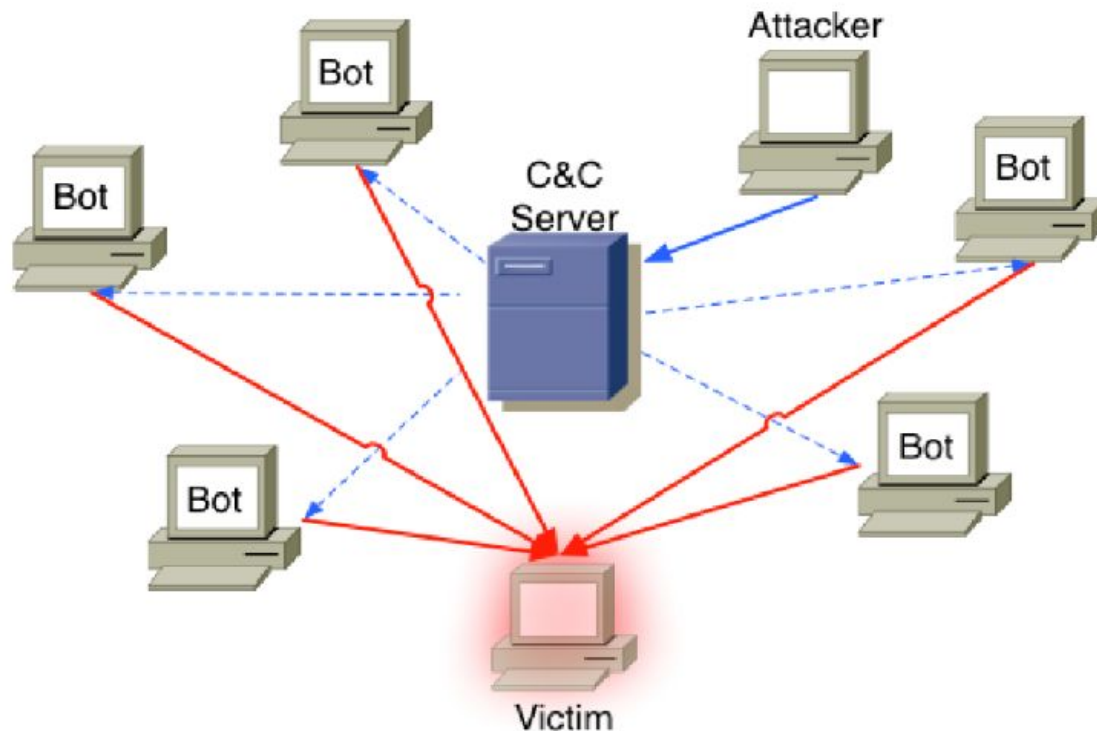
```
[...]
```

Rise of the Bots

- Abuse of IRC bots (IRCwars):
 - IRCwars: one of the first documented DDoS attacks
- 1999: trino "DDoS attack tool"
 - originally ran on Solaris (later ported to Windows)
 - setup of the botnet was mostly manual
 - August 1999: DDoS attack against a server at University of Minnesota using at least 227 bots (<http://staff.washington.edu/dittrich/misc/trino.analysis>)
- 2000s: DDoS attacks against high profile websites (Amazon, CNN, eBay) got huge media coverage

Botnets

A **botnet** is a network that consists of several malicious bots that are controlled by a commander, commonly known as botmaster (botherder).



Examples of commands: Phatbot

- harvest email addresses from host
- log all keypresses
- sniff network traffic
- take screenshots
- start an http server that allows to browse C:
- kill a hard-coded list of processes
 - AV programs
 - rival malware
- steal windows CD keys
 - also keys to popular games
- Socks proxy
 - sets up a proxy to be used as a "stepping stone" for SPAM
- download file at an url
- run a shell command
- Update
 - allows to change the available commands!

Threats posed by bot(net)s

- **For the infected host:** information harvesting
 - Identity data
 - Financial data
 - Private data
 - E-mail address books
 - Any other type of data that may be present on the host of the victim.
- **For the rest of the Internet**
 - Spamming
 - DDoS
 - Propagation (network or email worm)
 - Support infrastructure for illegal internet activity (the botnet itself, phishing sites, drive-by-download sites)

Further Reading

To read more on botnets, C&C strategies and botnet tracking or takedowns

- Your botnet is my botnet: taking over the Torpig botnet
<https://seclab.cs.ucsb.edu/academic/projects/projects/your-botnet-my-botnet/>
- Tracking and Characterizing Botnets Using Automatically Generated Domains (Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, Stefano Zanero)
<http://arxiv.org/abs/1311.5612>

...Turning Into an History of Worms

1999 – First DDoS attacks via trojaned machines (zombies)

1999 – Kernel Rootkits become public (Knark, modification of system call table)

2000 – [ILOVEYOU](#) (large-scale email worm) - Social Engineering

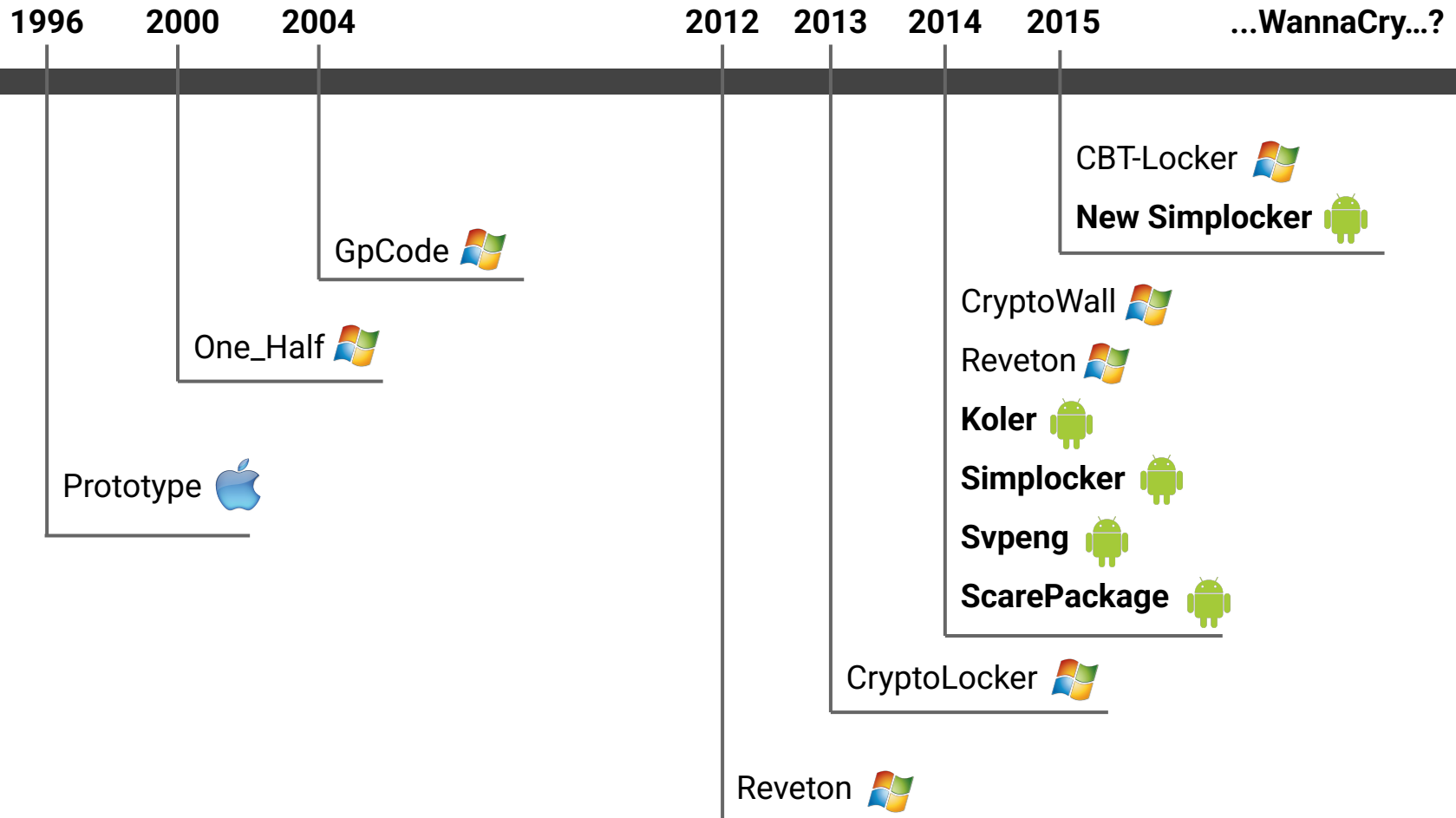
2001 – **Code Red** ([large-scale](#), exploit-based worm)

2003 – **SQL Slammer** worm (extremely fast propagation through UDP)

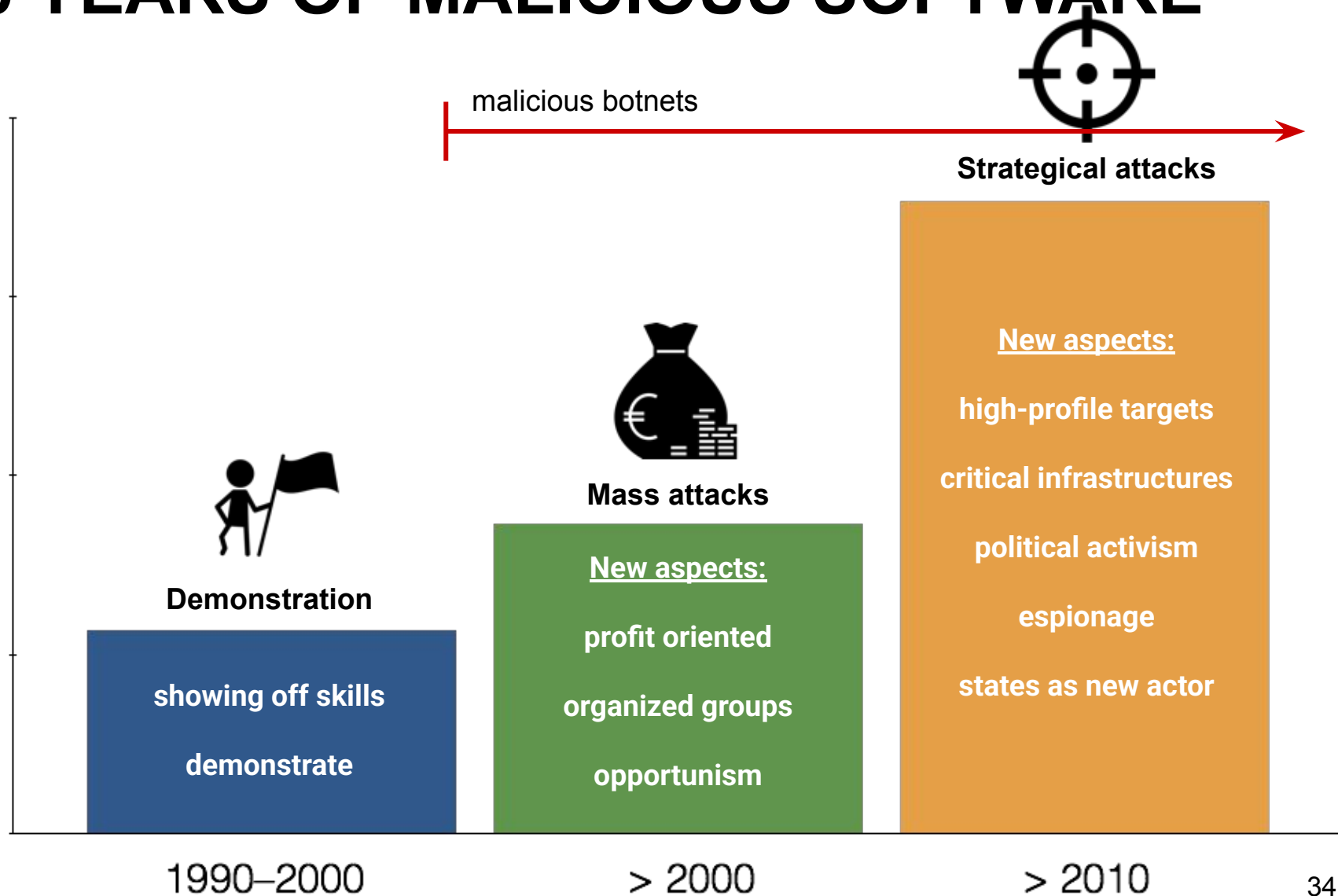
2004+ – **Malware that create botnet infrastructures (e.g., Storm Worm, Torpig, Koobface, Conficker, Stuxnet)**

2010+ – Scareware, Ransomware and State-sponsored malware

BRIEF HISTORY OF RANSOMWARE



30 YEARS OF MALICIOUS SOFTWARE



Defending Against Malware

- **Patches**

- Most worms exploit known vulnerabilities

- **Detect Malware**

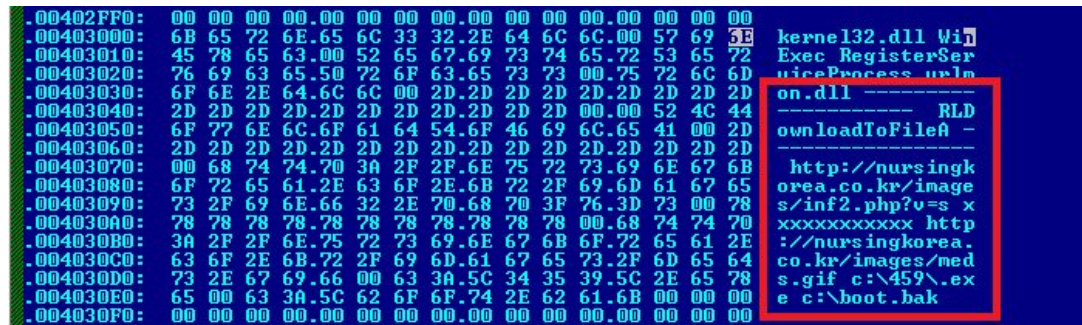
- Signatures (What the malware is)
 - Known unique patterns or sequence of code.
 - Must be developed automatically
 - Worms operate too quickly for human response
 - Behaviors (What the malware does)
 - Known actions performed on a system (e.g., stealing data, modifying files, spying on users, spreading, or disrupting operations)
 - Intrusion or anomaly detection (Effect on infected system/network)
 - Notice fast spreading, suspicious activity.
 - Can be a driver to automated signature generation

Useless against zero-day worms

Antivirus and Anti-malware

Software designed to detect, block, and remove malicious software (malware) from computers and networks.

- Basic strategy: **signature-based detection**
 - database of byte-level or instruction-level signatures that match known malware



```
.00402FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.00403000: 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 57 69 3E kernel32.dll Wi
.00403010: 45 78 65 63 00 52 65 67 69 73 74 65 72 53 65 72 Exec RegisterSer
.00403020: 76 69 63 65 50 72 6F 63 65 73 73 00 75 72 6C 6D uiceProcess urln
.00403030: 6F 6E 2E 64 6C 6C 00 2D 2D 2D 2D 2D 2D 2D 2D 2D on.dll
.00403040: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
.00403050: 6F 77 6E 6C 6F 61 64 54 6F 46 69 6C 65 41 00 2D
.00403060: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
.00403070: 00 68 74 74 70 3A 2F 2F 6E 75 72 73 69 6E 67 6B
.00403080: 6F 72 65 61 2E 63 6F 2E 6B 72 2F 69 6D 61 67 65
.00403090: 73 2F 69 6E 66 32 2E 70 68 70 3F 76 3D 73 00 78
.004030A0: 78 78 78 78 78 78 78 78 78 78 00 68 74 74 70
.004030B0: 3A 2F 2F 6E 75 72 73 69 6E 67 6B 6F 72 65 61 2E
.004030C0: 63 6F 2E 6B 72 2F 69 6D 61 67 65 73 2F 6D 65 64
.004030D0: 73 2E 67 69 66 00 63 3A 5C 34 35 39 5C 2E 65 78
.004030E0: 65 00 63 3A 5C 62 6F 6F 74 2E 62 61 6B 00 00 00
.004030F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.00403100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

<i>Virus Name</i>	<i>String Pattern (Signature)</i>
Accom.128 0	89C3 B440 8A2E 2004 8A0E 2104 BA00 05CD 21E8 D500 BF50 04CD
Die.448	B440 B9E8 0133 D2CD 2172 1126 8955 15B4 40B9 0500 BA5A 01CD
Xany.979	8B96 0906 B000 E85C FF8B D5B9 D303 E864 FFC6 8602 0401 F8C3

- wildcards can be used, regular expressions common

Antivirus and Anti-malware

Software designed to detect, block, and remove malicious software (malware) from computers and networks.

- Basic strategy: **signature-based detection**
 - database of byte-level or instruction-level signatures that match known malware
 - wildcards can be used, regular expressions common
- **Heuristics** (check for signs of infection)
 - code execution starts in last section
 - incorrect header size in PE header
 - suspicious code section name
 - patched import address table
- **Behavioral Detection**
 - detect signs (behavior) of known malware
 - detect “common behaviors”/ actions of malware

Malware Analysis and Detection

Ex-post workflow

1. Suspicious executable reported by "someone"
2. Automatically and manually analyzed

Static analysis

- Parsing the binary code without executing it.
- Techniques: disassembly, decompilation, string extraction, control flow analysis.
- Extract Signatures/Heuristics

Dynamic analysis

- Involves executing the malware in a controlled environment to monitor behavior.
- Techniques: sandboxing, hooking, tracing, runtime API monitoring.
- Extract behaviors/Heuristics

3. Antivirus signature and behaviors developed

A Note on Malicious Behaviors

Summary:




Description	Risk
Changes security settings of Internet Explorer: This system alteration could seriously affect safety surfing the World Wide Web.	
Creates files in the Windows system directory: Malware often keeps copies of itself in the Windows directory to stay undetected by users.	
Performs File Modification and Destruction: The executable modifies and destructs files which are not temporary.	
Spawns Processes: The executable produces processes during the execution.	
Performs Registry Activities: The executable reads and modifies registry values. It may also create and monitor registry keys.	

Table of Contents

▼ expand all collapse all ▲

-  General information
-  sample.exe
 -  services.exe
 -  i1ru74n4.exe
 -  Explorer.EXE
 -  i1ru74n4.exe

Recognizing Interesting Behaviors

M. Polino, A. Scorti, F. Maggi, S. Zanero,
*Jackdaw: Towards Automatic Reverse
Engineering of Large Datasets of Binaries*,
DIMVA 2015

Paper [PDF](#)

Slides: [PDF](#)


```

<dataflow_dependency id="19">
  <call api_function="RegOpenKeyExW"
    caller_address="0x0065006e,0x00b0df8f" id="1"
    objects="ObjectAttributes='hku\s-1-5-21-842925246-1425521274-308236825-500\software
      \microsoft\internet explorer\privacy',
    SubKey='software\microsoft\internet explorer\privacy'"/>
  <call api_function="RegQueryValueExW"
    caller_address="0x0065006e,0x00b1cb90" id="2"
    objects="ValueName='cleancookies'"/>
  <call api_function="RegOpenKeyExW"
    caller_address="0x0065006e,0x00b0dfb5,0x00b1ca3a" id="3"
    objects="ObjectAttributes='hku\s-1-5-21-842925246-1425521274-308236825-500\software
      \microsoft\internet explorer\privacy',
    SubKey='software\microsoft\internet explorer\privacy'"/>
  <call api_function="GetProcAddress"
    caller_address="0x0065006e,0x00b0dfb5,0x00b1ca3a">

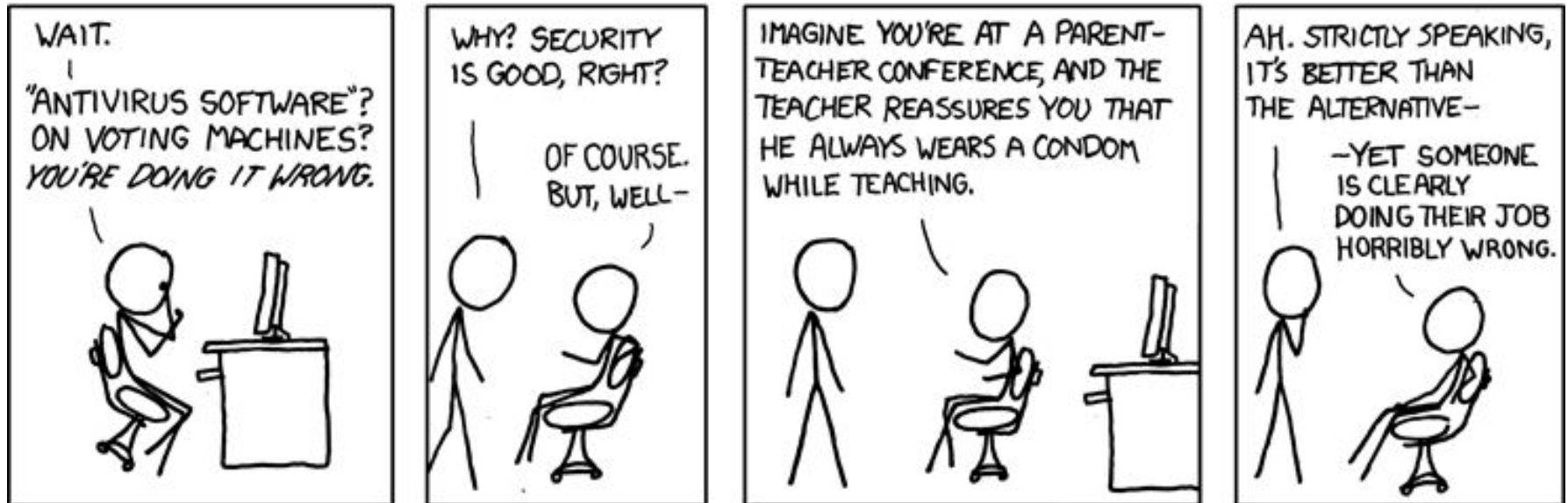
  <function_boundaries begin="0x00b0de6f" blocks="0x00b0dfb5,0x00b0df8f" end="0x00b0e06f"
    function_name="sub_B0DE6F"/>
  <function_boundaries begin="0x00b1cb7d" blocks="0x00b1cb90" end="0x00b1cba6"
    function_name="sub_B1CB7D"/>
  <function_boundaries begin="0x00b1calc" blocks="0x00b1ca3a" end="0x00b1ca6f"
    function_name="sub_B1CA1C"/>
  <path begin="0x00b0de6f" end="0x00b0e06f" function_name="sub_B0DE6F">
</dataflow_dependency>

```

Name normalization

Not everything needs an antivirus

PREMIER ELECTION SOLUTIONS (FORMERLY DIEBOLD)
HAS BLAMED OHIO VOTING MACHINE ERRORS ON PROBLEMS
WITH THE MACHINES' MCAFEE ANTIVIRUS SOFTWARE.



Virus Stealth Techniques

Virus scanners quickly discover viruses by searching around entry point

- **Entry Point Obfuscation**

- Multicavity viruses
- virus **hijacks control later** (after program is launched)
 - overwrite import table addresses (e.g., libraries)
 - overwrite function call instructions

Malware Stealth Techniques: Obfuscation

- **Polymorphism**

- Changes code structure (layout) with each infection.
- Payload remains functionally identical but is encrypted or packed with a different key at each infection:
 - Makes signature-based detection highly ineffective.
 - However, antivirus tools may detect the decryption routine if not obfuscated.

- **Metamorphism**

- Generates entirely new variants of the code for each infection.
- Code appears different (syntax), but preserve the semantic.
 - Harder to detect even with heuristic or behavioral analysis.

Metamorphism: Original Code

5B 00 00 00 00	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
50	push eax
0F 01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
FA	cli
8B 2B	mov ebp, [ebx]

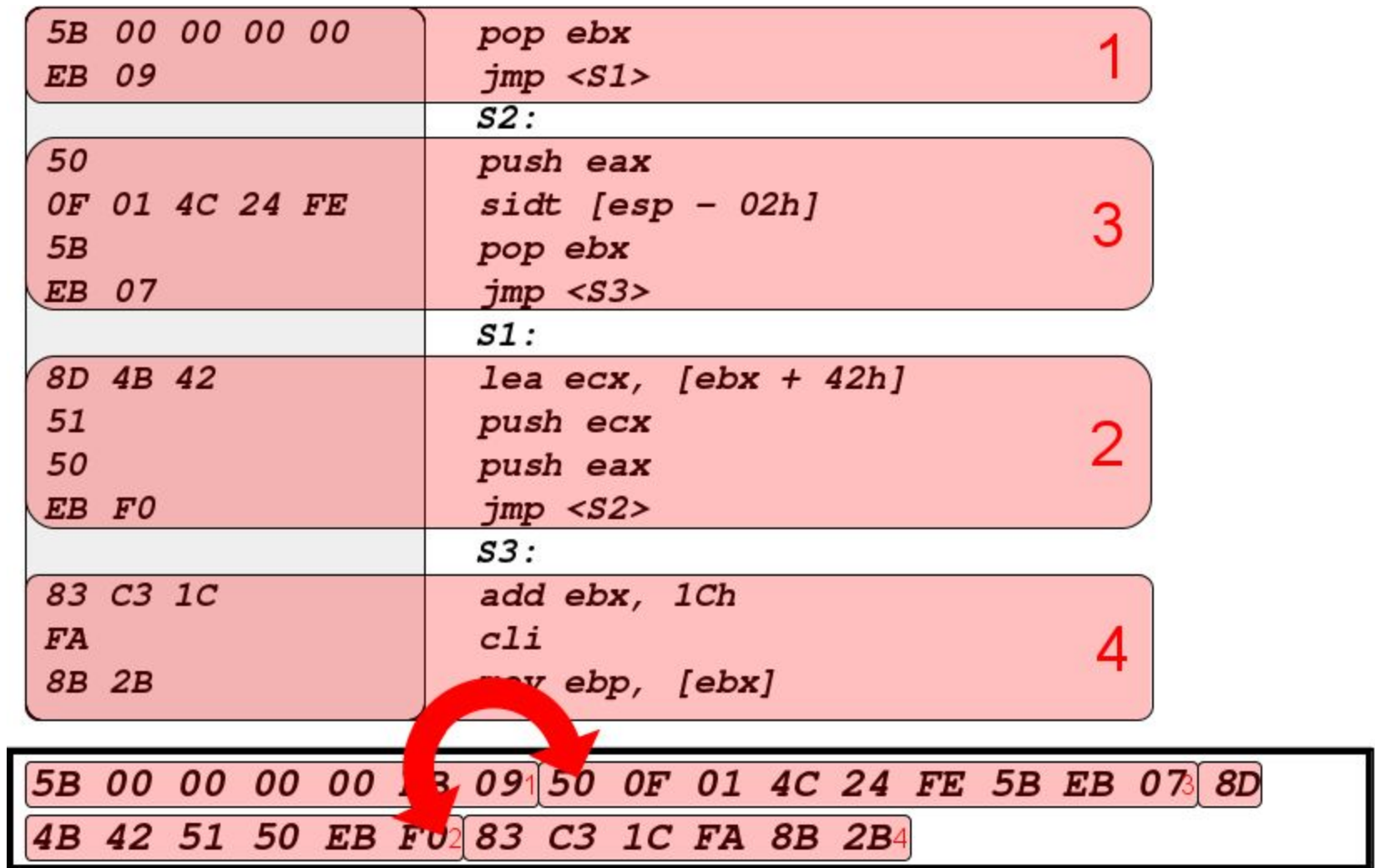
**5B 00 00 00 00 8D 4B 42 51 50 50 0F 01 4C 24 FE 5B
83 C3 1C FA 8B 2B**

Metamorphism: dead code insertion

5B 00 00 00 00	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
90	nop
50	push eax
40	inc eax
0F 01 4C 24 FE	sidt [esp - 02h]
48	dec eax
5B	pop ebx
83 C3 1C	add ebx, 1Ch
FA	cli
8B 2B	mov ebp, [ebx]

5B 00 00 00 00 8D 4B 42 51 50 90 50 40 0F 01 4C 24 FE 48 5B 83 C3 1C FA 8B 2B

Metamorphism: instruction reorder



Malware general stealth techniques

- **Dormant period**
 - During which no malicious behavior is exhibited
- **Event-triggered payload**
 - Often: C&C channel
 - ["Identifying Dormant Functionality in Malware Programs"](#)
(access from POLIMI network)
- **Anti-virtualization techniques**
- **Encryption / Packing**
 - Similar to polymorphism but more advanced techniques are available in more complex malware
- **Rootkit techniques**

Anti-virtualization techniques

Why **malware detects virtualization**: If a program is not run natively on a machine (bare-metal hardware), it may be:

- Under **analysis** (e.g., in a security lab)
- Being **scanned** (e.g., inside an antivirus sandbox)
- **Debugged** (e.g., by a researcher)

Modern malware checks the **execution environment** to evade detection and hinder analysis:

- **Virtual Machine (VM)**
 - Easily detectable via timing analysis or environment artifacts.
- **Hardware-Assisted VM** (e.g., Intel VT-x, AMD-V)
 - Requires more advanced checks, but still often detectable.
- **Emulator**
 - Hardest to detect in theory, but in practice vulnerable to:
 - Timing discrepancies
 - Incomplete instruction set implementations
 - Behavioral deviations among emulators

Packing

- **Encrypt** malicious payload to evade detection.
- Use **small a encryption/decryption routine** changing the key at each execution to avoid signature matching.

Typical Packing Functions

- Compression / Decompression
- Encryption / Decryption
- Inclusion of metamorphic components
- Anti-debugging techniques
- Anti-VM techniques
- Code virtualization to hide real instructions

Further Readings

Tal Garfinkel, Keith Adams, Andrew Warfield, Jason Franklin, *“Compatibility is Not Transparency: VMM Detection Myths and Realities”*

http://static.usenix.org/legacy/events/hotos07/tech/full_papers/garfinkel/garfinkel_html/

“Lines of Malicious Code: Insights Into the Malicious Software Industry”

https://publik.tuwien.ac.at/files/PubDat_212802.pdf

Malware Analysis vs Stealth Techniques

Static Analysis

Pros:

- Full code visibility and coverage, including dormant or unreachable code.
- Safe — no risk of code execution.

Cons:

- Obfuscation-resistant techniques (e.g., metamorphism, encryption, packing) can hinder analysis.
- No visibility into runtime behavior or environment-dependent logic.

Dynamic Analysis

Pros:

- Bypasses obfuscation: focuses on real behavior.
- Reveals runtime artifacts (e.g., dropped files, connections, registry changes).

Cons:

- Limited code coverage: dormant branches may not execute.
- Risk of environment detection by malware (anti-VM, anti-debugging).

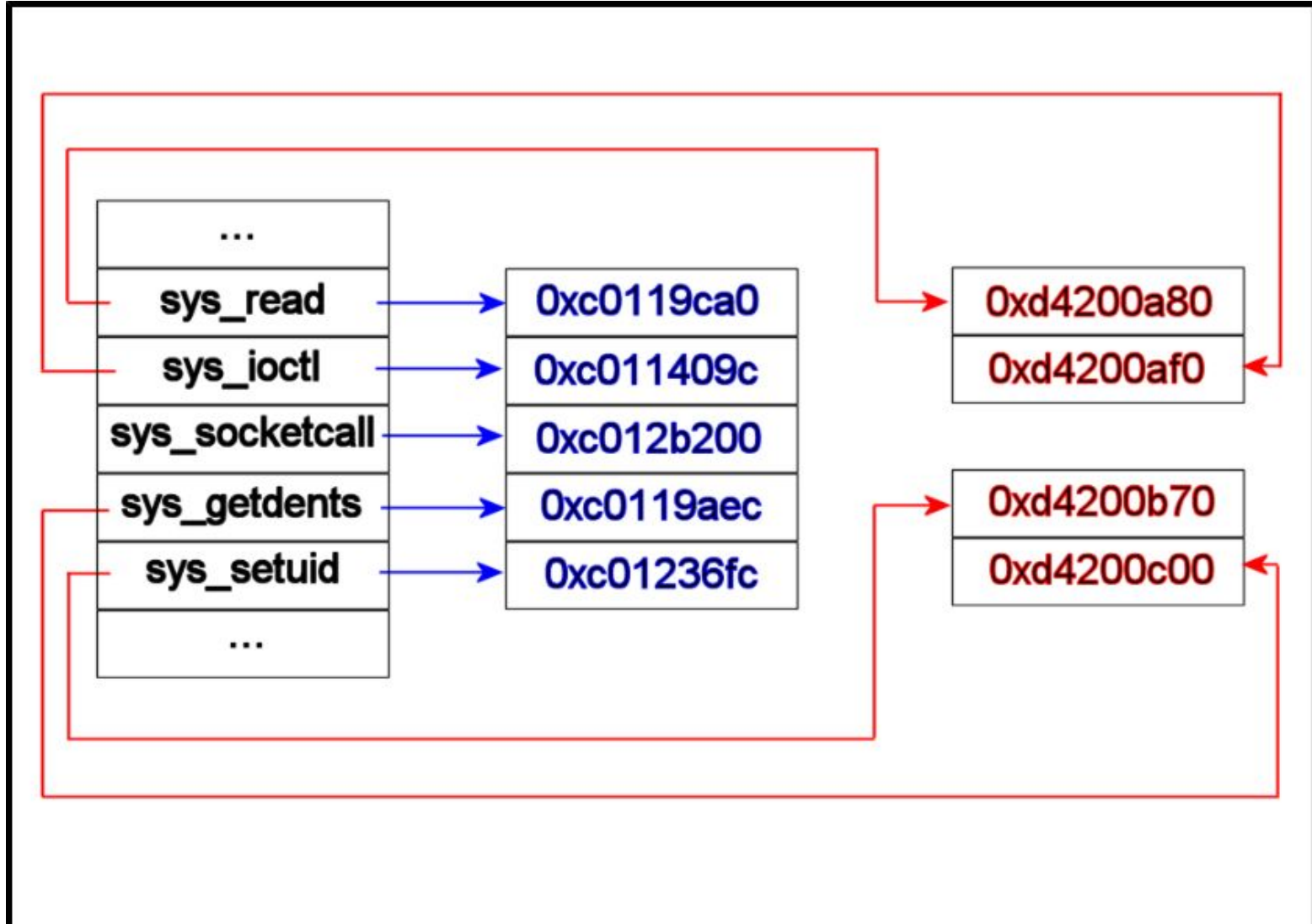
What are Rootkits?

- History: you become **root** on a machine, and you planted your **kit** to remain root
 - Make files, processes, user and directories disappear
 - Make the attacker **invisible**
- Can be either **userland** or **kernel-space**
 - Linux userland rootkit example:
 - Backdoored login, sshd, passwd
 - Trojanized utilities to hide: ps, netstat, ls, find, du, who, w, finger, ifconfig.
 - Windows userland rootkit targets:
 - Task Manager, Process Explorer, Netstat, ipconfig.

From Userland to Kernel Space

- Userland rootkit
 - “easier” to build, but often incomplete
 - easier to detect (cross layer examination, use of non-trojaned tools)
- Kernel space rootkit
 - More difficult to build, but can hide artifacts completely
 - Can only be detected via post-mortem analysis
 - Concept was born on 1997, Phrack 50, HalfLife “Abuse of the Linux Kernel for Fun and Profit”
 - First implementation of syscall hijacking
 - 1998, plaguez, “Weakening the Linux Kernel”, first complete LKM rootkit

Syscall hijacking



Methods for Hijacking Those Calls

- Methods

- Hook SYS_CALL Table, Interrupt Descriptor Table, o Global Descriptor Table
 - After in kernel 2.6 SYS_CALL table was hidden, scanning the IDT looking for a FAR JMP
*0x<syscall table address>[eax]
- Detour Patching
- Directly patch through /dev/mem or /dev/kmem (Silvio Cesare showed it possible even with monolithic kernel)

- Exercise or self-research: How to detect?

Methods for Recognizing Rootkits

- Intuition (“Hmmmm...that’s funny...”)
- Post-mortem on different system
- Trusted computing base / tripwire / etc.
- Cross-layer examination

The screenshot shows a Windows XP desktop with two windows open. The left window is a Command Prompt titled 'Command Prompt' showing the execution of commands to check the directory of C:\hackerdefender and the file hxdef100.exe. The right window is RootkitRevealer by Sysinternals, displaying a list of files and their properties.

Command Prompt Output:

```
C:\hackerdefender>dir
Volume in drive C has no label.
Volume Serial Number is 1482-981C

Directory of C:\hackerdefender

03/15/2005  01:27 PM    <DIR>          .
03/15/2005  01:27 PM    <DIR>          ..
12/31/2003  12:15 PM             70,144  hxdef100.exe
12/31/2003  12:17 PM             3,872  hxdef100.ini
               2 File(s)          74,016 bytes
               2 Dir(s)  1,537,146,880 bytes free

C:\hackerdefender>hxdef100.exe

C:\hackerdefender>dir
Volume in drive C has no label.
Volume Serial Number is 1482-981C

Directory of C:\hackerdefender

03/15/2005  01:30 PM    <DIR>          .
03/15/2005  01:30 PM    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  1,537,134,592 bytes free

C:\hackerdefender>_
```

RootkitRevealer - Sysinternals: www.sysinternals.com

Path	Timestamp	Size	Description
HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Minimal\HackerDefender100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Network\HackerDefender100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDER100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_HACKERDEFENDERDRV100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\HackerDefender100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
HKLM\SYSTEM\ControlSet001\Services\HackerDefenderDrv100	3/7/2005 5:57 PM	0 bytes	Hidden from Windows API.
C:\WINDOWS\hxdef100.2.ini	3/7/2005 5:57 PM	3.61 KB	Hidden from Windows API.
C:\WINDOWS\hxdef100.exe	3/7/2005 5:57 PM	68.50 KB	Hidden from Windows API.
C:\WINDOWS\hxdef100.ini	3/7/2005 5:57 PM	3.78 KB	Hidden from Windows API.
C:\WINDOWS\hxdefdrv.sys	3/7/2005 5:57 PM	3.26 KB	Hidden from Windows API.
C:\WINDOWS\Prefetch\HXDEF100.EXE-1BF5F48A.pf	3/7/2005 5:57 PM	6.14 KB	Hidden from Windows API.

Scan complete: 11 discrepancies found. [Scan]

It can get even more complex

- **Rootkit in BIOS**
 - In ACPI, John Heasman
 - CMOS, eEye bootloader
 - Bootkit which is not even in the BIOS (Brossard)
- **Rootkit on firmware** of NIC or Video Card
- **Rootkits in virtualization systems** (how do you recognize a rootkit which acts as an hypervisor?)

