

Internet Worms & Buffer Overflow Attacks

A Quick Overview

Original slides from Sumitha Bhandarkar

History and Introduction

The Very Beginning

- 1982 : The term “worm” was coined by Shoch and Hupp of Xerox PARC ¹
- The name was inspired by the “tapeworm” program described in John Brunner’s 1972 novel, “The Shockwave Rider”
- The “worm” programs were basically benign distributed programs capable of self replication performing maintenance tasks
- Research in using “worm” programs were abandoned because the consequences of malfunction could be dire.

¹ J. F. Shoch and J. A. Hupp, “The ‘Worm’ Programs: Early Experience with a Distributed Computation,” Communications of the ACM, vol. 25, no. 3, pp. 172–180, March 1982.

History and Introduction

The Morris Worm of 1988

- 1988 : First “worm” program to “infect” the Internet.
 - Released by Robert T Morris of Cornell Univ. Released from Berkeley.
 - Affected DEC’s VAX and Sun Microsystems’s Sun 3 systems.
 - About 6000 victims i.e.,5-10% of the Internet at that time.
 - More machines had to be disconnected from the net to avoid infection
 - Propagated faster than intended.
 - Some estimate loss to be \$98 million. Other reports claim it was less than \$1 million.
 - Triggered the creation of CERT
-

History and Introduction

The Morris Worm (Tech Details^{2,3})

- Host addresses obtained by examining the system tables `/etc/hosts.equiv` and `/.rhosts`, user files like `.forward` and `.rhosts`, dynamic routing information produced by the `netstat` program, and randomly generated host addresses on local network.
- Multi-vector propagation : exploited `rsh/rexec` (guess weak passwords), `fingerd` (buffer overflow) and `sendmail` (bug in the “debug mode”).
- Worm masqueraded as “`sh`”. Killed `argv` array so a “`ps`” would not show it. Fork to infect new m/c while parent continued scanning.
- No harmful payload. 99 line program for boot strap. If successful, followed by binary object file.

² Donn Seeley, “The tour of a worm”. <http://vx.netlux.org/lib/ase01.html>

³ Bob Page “A Report On The Internet Worm”. <http://www.ee.ryerson.ca:8080/~elf/hack/iworm.html>

History and Introduction

Buffer Overflow

- Fill a buffer with more than it can hold, causing the overflow to overwrite the contents of the stack.
- By carefully constructing the what “overflows”, malicious code can be executed
- Possible in languages like c/c++ with no bounds checking
- Ex:

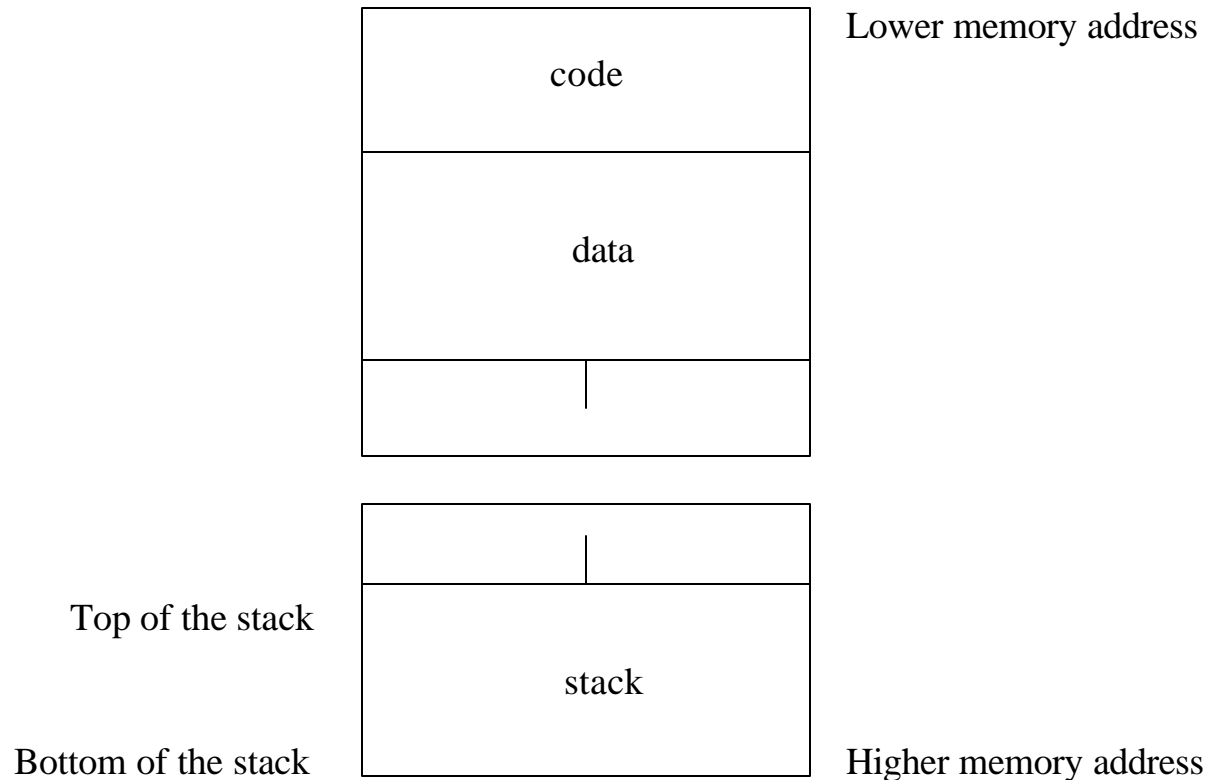
```
void foo(char *str)
{
    char buffer[10];
    strcpy(buffer, str);
}

void main()
{
    char *longStr = "This is too large a string to fit in a buffer of 10 bytes";
    foo(longStr);
}
```

History and Introduction

Buffer Overflow (Tech Details⁴)

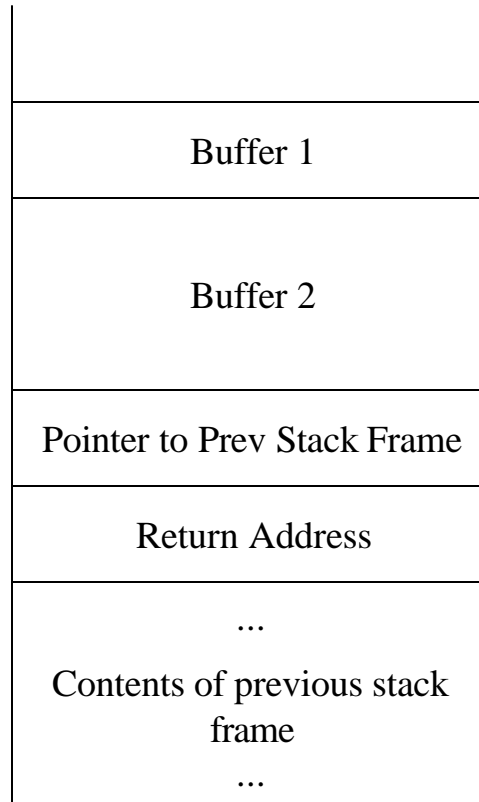
Process Memory Regions



⁴ Aleph One, "Smashing The Stack For Fun And Profit" from Phrack 49

History and Introduction

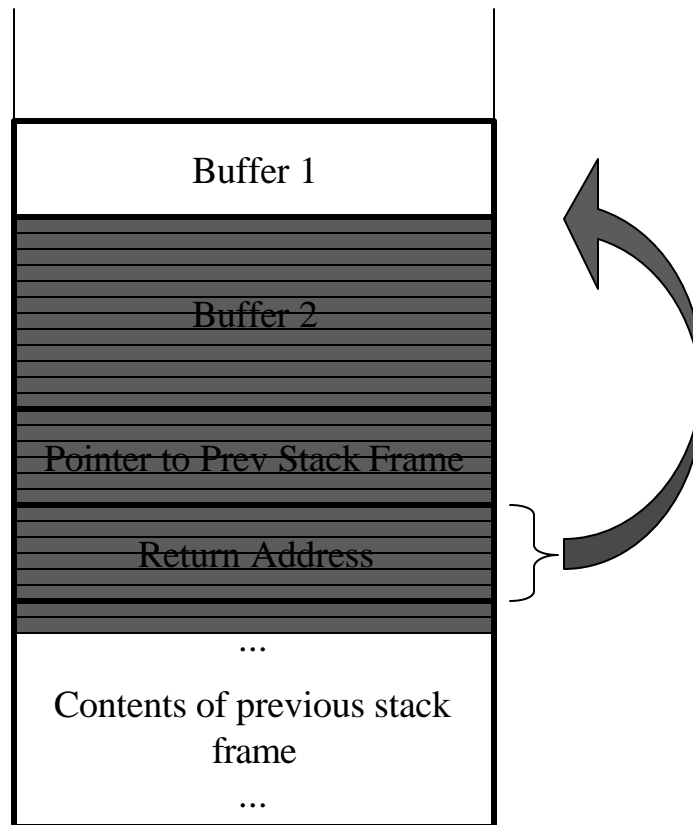
Buffer Overflow (Tech Details⁴)



⁴ Aleph One, "Smashing The Stack For Fun And Profit" from Phrack 49

History and Introduction

Buffer Overflow (Tech Details⁴)



⁴ Aleph One, "Smashing The Stack For Fun And Profit" from Phrack 49

One More Example

- ```
void function (int a, int b, int c)
{
char buffer1[5];
char buffer2[10];
}
int main()
{
function(4,5,6);
}
```

|                             |
|-----------------------------|
|                             |
| Buffer 1                    |
| Buffer 2                    |
| Pointer to Prev Stack Frame |
| Return Address              |
| 4                           |
| 5                           |
| 6                           |

# Overflow attack Code

- Overflow string could contain both attack code and address of code (overwrite RA)
- May have to guess the location of the RA
  - When we don't have access to source code
  - Overflow string can start with a bunch of NOPs, followed by attack code
    - Don't need to know exactly where the attack code will be on the stack (for RA)

# Buffer Overflow Protection

- Avoid using C library functions that don't use bounds checking: scanf, strcpy...
- Rewrite all C library functions to do bounds checking
- Employ a new library Libsafe that does bounds checking
  - Use these routines in place of not-so-safe library
- Employ safer languages such as Java
- <http://www.mcs.csuhayward.edu/~simon/security/boflo.html> -- easy to read

# Buffer Overflow Protection

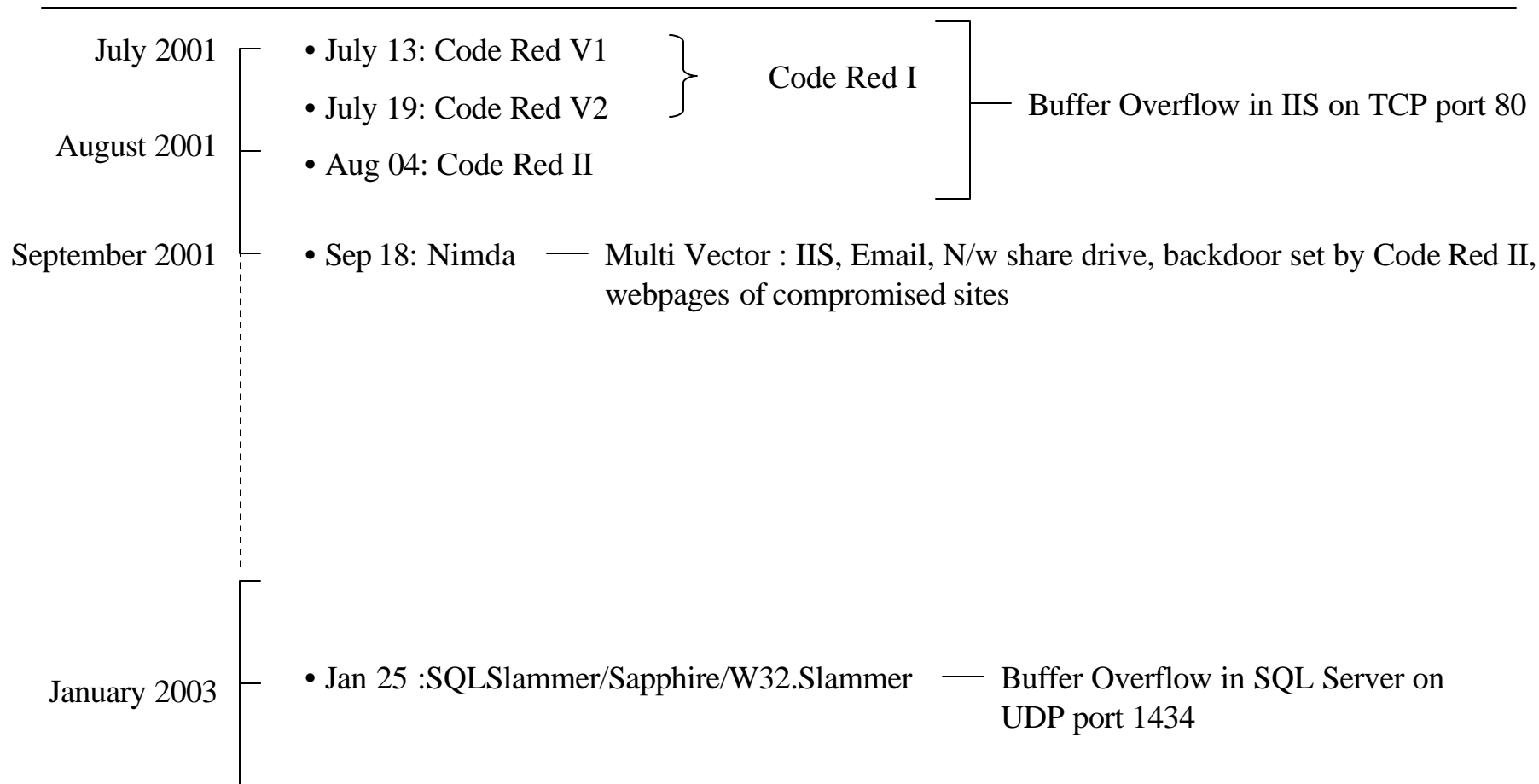
- Do not allow Execution from Stack
- Some programs need these mechanisms
- StackGuard
  - Write a random word between the stack pointer and the Return address
  - Check this random word before a function returns
  - If the word changed =>Buffer Overflow

# Overflow strings

- As program input (C library)
- Environment variable
- A read on network port (fingerd)
- Some worms employ multiple strategies

# Current “State of Art”

## Overview of recent worms



# Current “State of Art”

Code Red <sup>5,6</sup>

---

## CRv1

- 99 threads for finding machines to infect, 1 thread for checking if the IIS is running on English (US) Windows NT/2000 system. If so, thread used to deface web pages, else thread used for spreading worm.
- Each of the threads checks the date
  - if between 1st and 19th, generate list of random IP addresses. Try to infect machines on the list.
  - if between 20th and 28th, launch DDoS attack on 198.137.240.91 (once was `www.whitehouse.gov`)
  - rest of the days, stay dormant.

---

<sup>5</sup> David Moore, Colleen Shannon and Jeffery Brown, "Code-Red: a case study on the spread and victims of an Internet worm" , in Proceedings of the Second the ACM Internet Measurement Workshop, 2002.

<sup>6</sup> [http://www.sans.org/rr/malicious/code\\_red5.php](http://www.sans.org/rr/malicious/code_red5.php)

# Current “State of Art”

Code Red <sup>5,6</sup>

---

## CRv1

- The seed for the random number generator was static => same IP addresses were scanned by all infected machines => slow spread.
- Worm was memory resident (even the webpage used for defacing the website), so rebooting would get rid of infection, but the chance of re-infection was high.
- eEye Digital security disassembled and analyzed the code
- Whitehouse webpage was relocated to avoid the DDoS attack.

---

<sup>5</sup> David Moore, Colleen Shannon and Jeffery Brown, "Code-Red: a case study on the spread and victims of an Internet worm" , in Proceedings of the Second the ACM Internet Measurement Workshop, 2002.

<sup>6</sup> [http://www.sans.org/rr/malicious/code\\_red5.php](http://www.sans.org/rr/malicious/code_red5.php)



# Current “State of Art”

Code Red <sup>5,6</sup>

---

## CR II

- Even though completely different payload, got the name “code red II” since the worm code had a comment calling the worm “code red II”
- Upon infection, creates a back door process on the infected machine by copying a command shell ‘CMD.exe’ to a externally accessible location. Also leaves a trojan ‘explorer.exe’ on the root directory. Becomes dormant for 24 hours, and then reboots machine.
- After rebooting, worm begins to spread. If the system is a Chinese IIS server, the worm creates 600 threads. If the system is a non-Chinese IIS, the worm creates 300 threads.

---

<sup>5</sup> David Moore, Colleen Shannon and Jeffery Brown, "Code-Red: a case study on the spread and victims of an Internet worm" , in Proceedings of the Second the ACM Internet Measurement Workshop, 2002.

<sup>6</sup> [http://www.sans.org/rr/malicious/code\\_red5.php](http://www.sans.org/rr/malicious/code_red5.php)

# Current “State of Art”

Code Red <sup>5,6</sup>

---

## CR II

- Used localized scanning for locating hosts to infect.
  - Probes machines in the same /8 network with probability 1/2
  - Probes machines in the same /16 network with probability 3/8
  - Probes random machines with probability 1/8

---

<sup>5</sup> David Moore, Colleen Shannon and Jeffery Brown, "Code-Red: a case study on the spread and victims of an Internet worm" , in Proceedings of the Second the ACM Internet Measurement Workshop, 2002.

<sup>6</sup> [http://www.sans.org/rr/malicious/code\\_red5.php](http://www.sans.org/rr/malicious/code_red5.php)

# Current “State of Art”

Nimda<sup>7</sup>

---

- Multiple mechanisms for propagation
  - from client to client via email
  - from client to client via open network shares
  - from web server to client via browsing of compromised web sites
  - from client to web server via active scanning for and exploitation of various Microsoft IIS 4.0 / 5.0 directory traversal vulnerabilities
  - from client to web server via scanning for the back doors left behind by the "Code Red II" and "sadmind/IIS" worms

---

<sup>7</sup> <http://www.cert.org/advisories/CA-2001-26.html>

# Current “State of Art”

## Nimda

---

- 450,000 unique IP addresses spreading Nimda on Sep 19th<sup>§</sup>
- 50% of the previously infected machines stopped spreading the infection within 24 hours (either patched, taken offline or overwhelmed) <sup>§</sup>
- Other interesting facts about nimda -
  - The text in the subject line of the mail message appears to be variable <sup>‡</sup>
  - There seem to be many slight variations in the attached binary file, causing the MD5 checksum to be different. However, the file length of the attachment appears to consistently be 57344 bytes <sup>‡</sup>
  - A later version of the virus/worm modified the attachment name from Readme.EXE to Sample.EXE

---

<sup>‡</sup> <http://www.cert.org/advisories/CA-2001-26.html>

<sup>§</sup> <http://www.caida.org/dynamic/analysis/security/nimda/>

# Current “State of Art”

SQLSlammer/Sapphire/W32.Slammer

---

- Exploited buffer overflow vulnerability in "Server Resolution" service of the SQL Server on UDP port 1434
- Patch was available for the vulnerability for the past 6 months !
- Payload only 376 bytes => memory resident worm
- Payload not harmful. Damage mainly due to DoS resulting from exceedingly large number of UDP packets pumped out by infected hosts
- Mitigation : firewall incoming UDP requests on port 1434 and reboot the system.
- Also, the worm had a fixed MD5 checksum i.e, it was not polymorphic (A0AA4A74B70CBCA5A03960DF1A3DC878)<sup>2</sup>

---

<sup>2</sup> McAfee Security : [http://vil.nai.com/vil/content/v\\_99992.htm](http://vil.nai.com/vil/content/v_99992.htm)

# Current “State of Art”

SQLSlammer/Sapphire/W32.Slammer<sup>10</sup>

---

- At the beginning of infection, doubling time of 8.5 seconds
- 90 percent of vulnerable hosts infected within 10 minutes
- Total infected machines = 75,000
- Full scanning rate (over 55 million scans per second) achieved in 3 minutes
- Two orders of magnitude faster than Code Red.
- Used random scanning for finding the IP address. Flaw in the implementation of random generation algorithm (PRNG) limited the scanned addresses to significantly smaller than the actual Internet address space

---

<sup>10</sup> David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford and Nicholas Weaver, “The Spread of the Sapphire/Slammer Worm”, CAIDA Report.

# Current “State of Art”

SQLSlammer/Sapphire/W32.Slammer<sup>10</sup>

---

- Slammer was bandwidth limited as opposed to being latency limited like Code Red.
- Single thread in the worm that continuously generated scans. (Since UDP was used “connection establishment” was not a limiting factor)
- In principle, an infected machine with a 100 Mb/s connection to the Internet could produce over 30,000 scans/second.
- In practice, due to bandwidth limitations and the per-packet overhead, the largest probe rate directly observed was 26,000 scans/second
- Internet-wide average was approximately 4,000 scans/second per worm during the early phase of growth.

---

<sup>10</sup> David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford and Nicholas Weaver, “The Spread of the Sapphire/Slammer Worm”, CAIDA Report.

# Current “State of Art”

SQLSlammer/Sapphire/W32.Slammer<sup>10</sup>

---

## Implications !

- First high speed worm released in the wild. High speed worms no longer just a possibility but a reality !
- Worm writers can now use less popular s/w as breeding grounds for worms. A small population of machines on high speed connections can slow down the traffic on the Internet !
- Worm defense needs to be automatic. Manual response is not enough to contain the worms !
- a more sophisticated worm might have stopped scanning once the entire susceptible population was infected, leaving itself dormant on over 75,000 machines to do harm at some future point !

---

<sup>10</sup> David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford and Nicholas Weaver, “The Spread of the Sapphire/Slammer Worm”, CAIDA Report.

# Current “State of Art”

## Quick Recap

---

- localized scanning (Morris worm, Code Red II)
  - multi-vector propagation (Morris worm, Nimda)
  - bandwidth-limited worms (Slammer)
-

# Future of worm technology

“How to own the Internet in your spare time?” <sup>11</sup>

---

## Hit list scanning

- Before the worm is released, the worm author collects a list of say 10,000 to 50,000 potentially vulnerable machines, ideally ones with good network connections.
- The worm, when released onto an initial machine on this hit-list, begins scanning down the list. When it infects a machine, it divides the hit-list in half, communicating half to the recipient worm, keeping the other half.
- The hit-list itself can be generated generally with little fear of detection using Stealthy scans, Distributed scans, DNS searches, Spiders, Public surveys or Just listening

---

<sup>11</sup> S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time" , in Proceedings of the 11th USENIX Security Symposium, August 2002.

# Future of worm technology

“How to own the Internet in your spare time?” <sup>11</sup>

---

## Permutation scanning

- All worms share a common pseudo random permutation of the IP address space
- When an infected machine is encountered, choose a new random start point
- In effect this is a self-coordinated, comprehensive scan with random probing.
- After seeing several infected machines without discovering new vulnerable targets, stops the scanning process
- After a preset time limit, the worms wake up and start scanning with a new permutation key
- Create a huge base of compromised machines capable of being controlled by a “master” sometime in future.

---

<sup>11</sup> S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time" , in Proceedings of the 11th USENIX Security Symposium, August 2002.

# Future of worm technology

“How to own the Internet in your spare time?” <sup>11</sup>

---

## Warhol Worm

- Capable of attacking most vulnerable targets in well under an hour, possibly less than 15 minutes.
- A combination of hit-list and permutation scanning can create it

## Topological Scanning

- Uses information contained on the victim machine in order to generate the hit list on the fly.
- Similar to the “e-mail viruses”. Ex: if the vulnerability is in a webserver look at all the URLs on the disk. If it is also a peer-to-peer service, attack the peers first and then release the worm on the net.

---

<sup>11</sup> S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time" , in Proceedings of the 11th USENIX Security Symposium, August 2002.

# Future of worm technology

“How to own the Internet in your spare time?” <sup>11</sup>

---

## Flash Worms

- Similar to hit-list scanning, only the hit list is “honed” such that it contains list of machines guaranteed to be vulnerable.
- Hand-pick child nodes, such that they have a high bandwidth connection and hand them a block of addresses from the hit list
- Include some form of redundancy to ensure that mitigation of one child will not eliminate the entire block.
- Requires a lot of effort, but could be used very easily in state-sponsored cyber-warfare.

---

<sup>11</sup> S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time" , in Proceedings of the 11th USENIX Security Symposium, August 2002.

# Future of worm technology

“How to own the Internet in your spare time?” <sup>11</sup>

---

## Stealth worms/Contagion/Surreptitious worms

- Exactly the opposite of the worms discussed so far.
- Spreads very slowly to avoid detection. An infected machine stays dormant for sometime before starting a new infection.
- A pair of vulnerabilities could be used. One for slowly spreading the infection. The other for blowing the cover and launching an all out attack.
- Services like P2P networks, which contain a large number of nodes and run on similar configuration provide the most fertile ground for this kind of worm.

## Distributed Worms !

- Any of the above worm with capability of distributed communication and control

---

<sup>11</sup> S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time" , in Proceedings of the 11th USENIX Security Symposium, August 2002.

# Requirements for Containing Worms

“Internet Quarantine : Requirements for Containing Self-Propagating Code” <sup>12</sup>

---

- Effectiveness of containment mostly depends on -
  - Reaction time : How quickly can the worm be detected and how much time does it take for the reaction to begin ?
  - Containment strategy : On what basis do we contain the worm ? Two strategies considered in this paper are, address blacklisting and content-filtering.
  - Deployment : How widely is the containment system deployed ?

---

<sup>12</sup> David Moore, Colleen Shannon, Geoffrey M. Voelker and Stefan Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code" , in IEEE Infocom 2003, San Francisco, CA, USA, April 2003.

# Requirements for Containing Worms

“Internet Quarantine : Requirements for Containing Self-Propagating Code” <sup>12</sup>

---

- Using the mathematical foundation governing the spread of infectious disease

|         |                                                          |
|---------|----------------------------------------------------------|
| $N$     | size of the total vulnerable population                  |
| $S(t)$  | susceptibles at time $t$                                 |
| $I(t)$  | infectives at time $t$                                   |
| $\beta$ | contact rate                                             |
| $s(t)$  | susceptibles ( $S(t)$ ) / population ( $N$ ) at time $t$ |
| $i(t)$  | infectives ( $I(t)$ ) / population( $N$ ) at time $t$    |

TABLE I

COMPONENTS OF THE  $SI$  MODEL.

$$\frac{dI}{dt} = \beta \frac{IS}{N}$$

$$\frac{dS}{dt} = -\beta \frac{IS}{N}$$

$$\frac{di}{dt} = \beta i(1 - i)$$

$$i(t) = \frac{e^{\beta(t-T)}}{1 + e^{\beta(t-T)}}$$

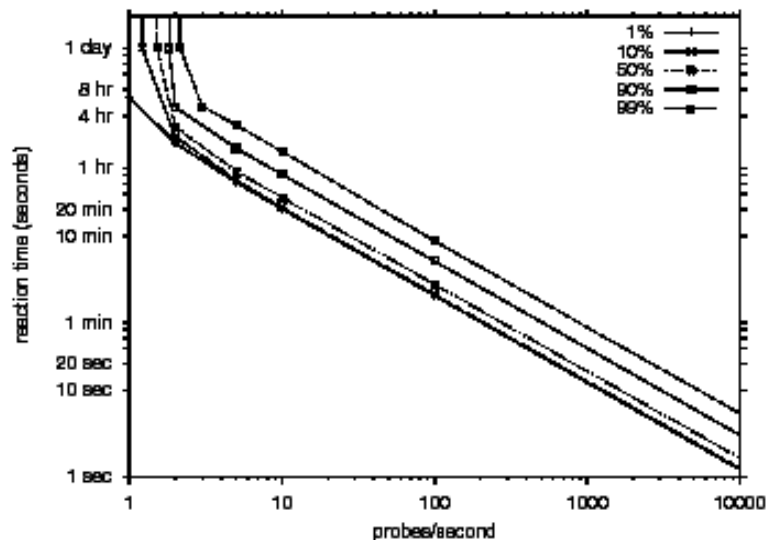
---

<sup>12</sup> David Moore, Colleen Shannon, Geoffrey M. Voelker and Stefan Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code" , in IEEE Infocom 2003, San Francisco, CA, USA, April 2003.

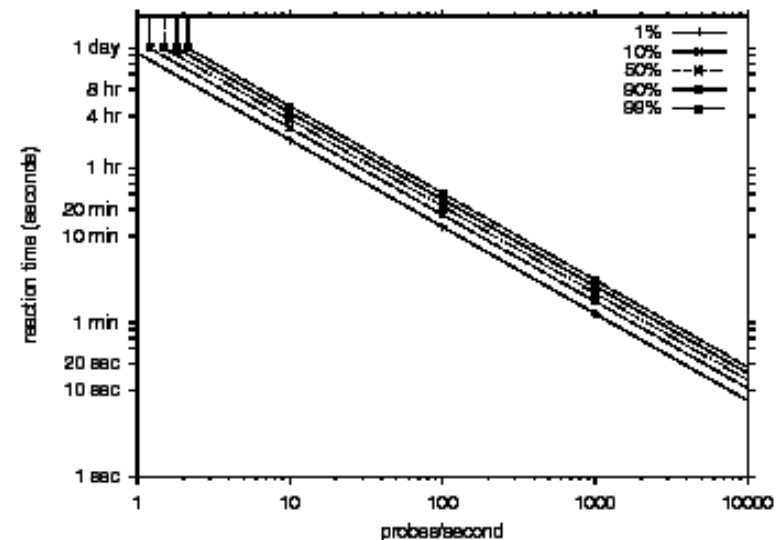
# Requirements for Containing Worms

“Internet Quarantine : Requirements for Containing Self-Propagating Code” <sup>12</sup>

Ideal Situation (All routers deploy containment system)



(a) Address Blacklisting



(b) Content Filtering

Fig. 3. Reaction times necessary for (a) address blacklisting and (b) content filtering to contain worms of various degrees of aggressiveness, as represented by their probe rates. Each curve corresponds to a particular degree of infection (e.g., “10%” refers to 10% of susceptible hosts infected after 24 hours). Note the use of log scales on both axes.

<sup>12</sup> David Moore, Colleen Shannon, Geoffrey M. Voelker and Stefan Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code" , in IEEE Infocom 2003, San Francisco, CA, USA, April 2003.

# Requirements for Containing Worms

“Internet Quarantine : Requirements for Containing Self-Propagating Code” <sup>12</sup>

## Realistic Deployment

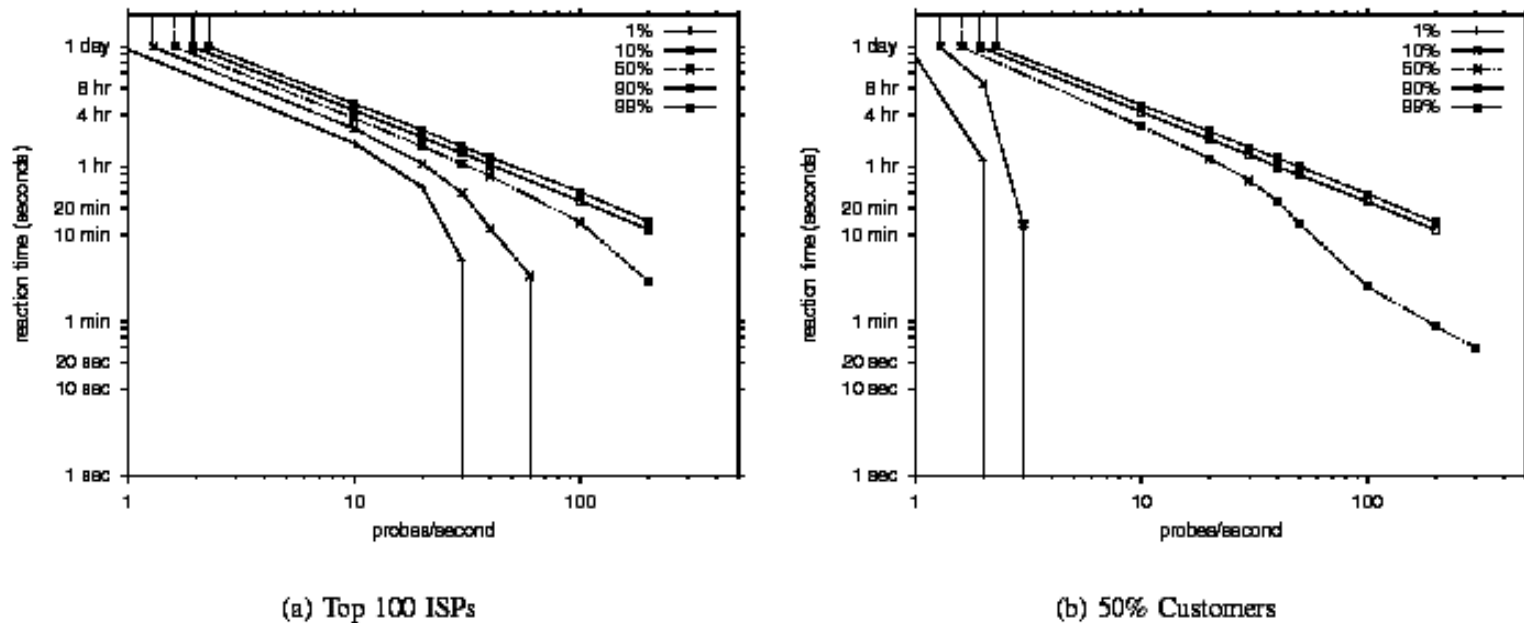


Fig. 5. The reaction times required for effective worm containment for various worm intensities. Shown are graphs for the two deployment scenarios of worm containment: (a) “Top 100 ISPs” and (b) “50% Customers.”

<sup>12</sup> David Moore, Colleen Shannon, Geoffrey M. Voelker and Stefan Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code", in IEEE Infocom 2003, San Francisco, CA, USA, April 2003.

# Summary

- Buffer Overflow attacks –simple strategy for infecting end hosts
- Can write worm programs based on such attacks
- Worms can potentially spread very fast
- Need fast mechanisms to identify and contain them.