Announcements

• Project 2 is on the web.
  – Due: March 15th
  – Send groups to Jeff Vaughan (vaughan2@seas) today.

• Plan for today:
  – Automatic Signature Extraction
  – Other kinds of Intrusion Detection Tools
Naïve Content Sifting

- \text{ProcessTraffic}(\text{packet}, \text{srcIP}, \text{dstIP}) \{ \\
  \text{count[packet]}++; \\
  \text{Insert(srcIP, dispersion[packet].sources);} \\
  \text{Insert(dstIP, dispersion[packet].dests);} \\
  \text{if (count[packet] > countThresh} \\
    \text{&& size(dispersion[packet].sources) > srcThresh} \\
    \text{&& size(dispersion[packet].dests) > dstThresh)} \{ \\
      \text{Alarm(packet)} \\
    \} \\
\}

- Tables \text{count} and \text{dispersion} are indexed by entire packet content.
Practical Content Sifting

- Reduce size of count table by:
  - Hashing the packet content to a fixed size (not cryptographic hashes)
  - Hash collisions may lead to false positives
  - So, do multiple different hashes (say 3) -- worm content is flagged only if counts along all hashes exceed a threshold

- Include the destination port in the hash of the packet content
  - Current worms target specific vulnerabilities, so they usually aim for a particular port.

- To check for substring matches they propose to use a Rabin fingerprint
  - Probabilistic, incrementally computable hash of substrings of a fixed length.
Rabin Fingerprints

Given string of length n
- Write as sequence of bytes: \( t_0 \ t_1 \ t_2 \ldots \ t_n \)

Check all possible substrings of length k

Choose constants p (a prime) and M (modulus)

Fingerprint for substrings are:
- \( F_1 = (t_0 * p^{(k-1)} + t_1 * p^{(k-2)} + \ldots + t_k) \mod M \)
- \( F_2 = (t_1 * p^{(k-1)} + t_2 * p^{(k-2)} + \ldots + t_{k+1}) \mod M \)
  \( = (F_1 * p + t_{k+1} - t_0 * p^k) \mod M \)
- \( F_3 = (F_2 * p + t_{k+2} - t_1 * p^k) \mod M \)
- \( F_i = (F_{i-1} * p + t_{k+i-1} - t_{i-1} * p^k) \mod M \)

For efficiency, precompute table of \( x * p^k \)
Multistage Filters, Pictorially

Field Extraction

Hash 1

Increment

Counters

Stage 1

Comparator

Hash 2

Stage 2

Comparator

Hash 3

Stage 3

Comparator

Alert!
If all counters above threshold
Tracking Address Dispersion

- In this case, we care about the number of distinct source (or destination) addresses in packets that contain suspected worm data.

- Could easily keep an exact count by using a hash table, but that becomes too time and memory intensive.
  - In the limit, need one bit per address to mark whether it has been seen or not.

- Instead: Keep an *approximate* count
- Scalable bitmap counters
  - Reduce memory requirements by 5x
Scalable Bitmap Counters

• Suppose there are 64 possible addresses and you want to use only 32 bits to keep track of them.

• High-level idea:
  – Hash the address into a value between 0 and 63
  – Use only the lower 5 bits (yielding 32)
  – To estimate actual number of addresses, multiply the number of bits set in the bitmap by 2.
Multiple Bitmaps, Pictorially

- Recycle bitmaps after they fill up
- Adjust the scale factors on the counts accordingly

![Diagram of multiple bitmaps and hash space](image)
Results

• Earlybird successfully detects and extracts virus signatures from every known recent worm (CodeRed, MyDoom, Sasser, Kibvu.B,…)

• Tool generates content filter rules suitable for use with Snort
Analysis

• False Positives:
  – SPAM
    • No solution yet
  – BitTorrent (35% of Internet traffic?!) 
    • Replicates packets, so it actually looks like worm traffic
  – Common protocol headers 
    • HTTP and SMTP
    • Some P2P system headers
    • Solution: whitelist by hand

• False Negatives:
  – Hard (impossible?) to prove absence of worms
  – Over 8 months Earlybird detected all worm outbreaks reported on security mailing lists
Attacks

• What about violating the assumptions?
  – Invariant content
  – Worm propagates randomly
  – Worm propagates quickly
Polymorphic Viruses/Worms

• Virus/worm writers know that signatures are the most effective way to detect such malicious code.

• Polymorphic viruses mutate themselves during replication to prevent detection
  – Virus should be capable of generating many different descendants
  – Simply embedding random numbers into virus code is not enough
Strategies for Polymorphic Viruses

- Change data:
  - Use different subject lines in e-mail

- Encrypt most of the virus with a random key
  - Virus first decrypts main body using random key
  - Jumps to the code it decrypted
  - When replicating, generate a new key and encrypt the main part of the replica

- Still possible to detect decryption portion of the virus using virus signatures
  - This part of the code remains unchanged
  - Worm writer could use a standard self-decompressing executable format (like ZIP executables) to cause confusion (many false positives)
Advanced Evasion Techniques

- Randomly modify the code of the virus/worm by:
  - Inserting no-op instructions: subtract 0, move value to itself
  - Reordering independent instructions
  - Using different variable/register names
  - Using equivalent instruction sequences:
    \[ y = x + x \quad \text{vs.} \quad y = 2 \times x \]
  - These viruses are sometimes called "metamorphic" viruses in the literature.

- There exist C++ libraries that, when linked against an appropriate executable, automatically turn it into a metamorphic program.

- Sometimes vulnerable software itself offers opportunities for hiding bad code.
  - Example: ssh or SSL vulnerabilities may permit worm to propagate over encrypted channels, making content filtering impossible.
  - If IPSEC becomes popular, similar problems may arise with it.
Other Evasion Techniques

- Observation: worms don't need to scan randomly
  - They won't be caught by internet telescopes

- Meta-server worm: ask server for hosts to infect (e.g., Google for "powered by php")

- Topological worm: fuel the spread with local information from infected hosts (web server logs, email address books, config files, SSH "known hosts")
  - No scanning signature; with rich interconnection topology, potentially very fast.

- Propagate slowly: "trickle" attacks
  - Also a very subtle form of denial of service attacks
Witty Worm

- “Bandwidth-limited” UDP worm like Slammer.
- Vulnerable pop. (12K) attained in 75 minutes.
- Payload: slowly corrupt random disk blocks.
Witty, con’t

• Flaw had been announced the *previous day*.

• Telescope analysis reveals:
  – Initial spread seeded via a *hit-list*.
  – In fact, targeted a U.S. military base.
  – Analysis also reveals “Patient Zero”, a European retail ISP.

• Written by a Pro.
Broader View of Defenses

• Prevention -- *make the monoculture hardier*
  – Get the code right in the first place …
    • … or figure out what’s wrong with it and fix it
  – Lots of active research (static & dynamic methods)
  – Security reviews now taken seriously by industry
    • E.g., ~$200M just to *review* Windows Server 2003
  – But very expensive
  – And very large Installed Base problem

• Prevention -- *diversify the monoculture*
  – Via exploiting existing heterogeneity
  – Via creating artificial heterogeneity
Broader View of Defenses, con’t

• Prevention -- *keep vulnerabilities inaccessible*
  – Cisco’s *Network Admission Control*
    • Examine hosts that try to connect, block if vulnerable
  – Microsoft’s *Shield*
    • Shim-layer blocks network traffic that fits known *vulnerability* (rather than known *exploit*)
Detecting Attacks

- Attacks (against computer systems) usually consist of several stages:
  - Finding software vulnerabilities
  - Exploiting them
  - Hiding/cleaning up the exploit

- Attackers care about finding vulnerabilities:
  - What machines are available?
  - What OS / version / patch level are the machines running?
  - What additional software is running?
  - What is the network topology?

- Attackers care about not getting caught:
  - How detectible will the attack be?
  - How can the attacker cover her tracks?

- Programs can automate the process of finding/exploiting vulnerabilities.
  - Same tools that sys. admins. use to audit their systems…
  - A worm is just an automatic vulnerability finder/exploiter…
Attacker Reconnaissance

• Network Scanning
  – Existence of machines at IP addresses
  – Attempt to determine network topology
  – ping, tracert

• Port scanners
  – Try to detect what processes are running on which ports, which ports are open to connections.
  – Typical machine on the internet gets 10-20 port scans per day!
  – Can be used to find hit lists for flash worms

• Web services
  – Use a browser to search for CGI scripts, Javascript, etc.
Determining OS information

- Gives a lot of information that can help an attacker carry out exploits
  - Exact version of OS code can be correlated with vulnerability databases
- Sadly, often simple to obtain this information:
  - Just try telnet

```
playground~$ telnet hpux.u-aizu.ac.jp
Trying 163.143.103.12 ...
Connected to hpux.u-aizu.ac.jp.
Escape character is '^]'.
HP-UX hpux B.10.01 A 9000/715 (ttyp2)

login:
```
Determining OS

- Or ftp:

```
$ ftp ftp.netscape.com 21
Connected to ftp.gfftp.netscape.com.
220-36
220 ftpnscp.newaol.com FTP server (SunOS 5.8) ready.
Name (ftp.netscape.com:stevez):
331 Password required for stevez.
Password:
530 Login incorrect.
ftp: Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> system
215 UNIX Type: L8 Version: SUNOS
ftp>
```
Determining OS

• Exploit different implementations of protocols
  – Different OS’s have different behavior in some cases
• Consider TCP protocol, there are many flags and options, and some unspecified behavior
  – Reply to bogus FIN request for TCP port (should not reply, but some OS’s do)
  – Handling of invalid flags in TCP packets (some OS’s keep the invalid flags set in reply)
  – Initial values for RWS, pattern in random sequence numbers, etc.
  – Can narrow down the possible OS based on the combination of implementation features
• Tools can automate this process
Auditing: Remote auditing tools

• Several utilities available to “attack” or gather information about services/daemons on a system.
  – SATAN (early 1990’s):
    Security Administrator Tool for Analyzing Networks
  – SAINT - Based on SATAN utility
  – SARA - Also based on SATAN
  – Nessus - Open source vulnerability scanner
    • http://www.nessus.org
  – Nmap

• Commercial:
  – ISS scanner
  – Cybercop
http://www.insecure.org/nmap
http://www.insecure.org/nmap/nmap-fingerprinting-article.html
Kinds of Auditing done

- Nessus web pages:
  - Backdoors
  - CGI abuses
  - Denial of Service
  - Finger abuses
  - Firewalls
  - FTP
  - Gain a shell remotely
  - Gain root remotely
  - Netware
  - NIS

- Port scanners
- Remote file access
- RPC
- Settings
- SMTP problems
- SNMP
- Useless services
- Windows
- Windows : User management

- Doing this kind of auditing by hand is complex and error prone
- These tools aren’t fool proof or complete.