Lecture 24: Detection
Announcements
Firewalls
Secure External Access to Inside Machines

- Often need to provide secure remote access to a network protected by a firewall
  - Remote access, telecommuting, branch offices, …
- Create secure channel (Virtual Private Network, or VPN) to tunnel traffic from outside host/network to inside network
  - May allow bypassing the firewall, reducing firewall effectiveness
  - Try it yourself at http://www.net.berkeley.edu/vpn/
Why Have Firewalls Been Successful?

- Central control – easy administration and update
  - Single point of control: update one config to change security policies
  - Potentially allows rapid response
- Easy to deploy – transparent to end users
  - Easy incremental/total deployment to protect 1000’s
- Addresses an important problem
  - Security vulnerabilities in network services are rampant
  - Easier to use firewall than to directly secure code …
Think like an attacker

• Suppose you wanted to attack a company protected by a firewall. What attacks might you try?

• Share your ideas on chat (mark it visible to everyone)
Firewall Disadvantages

• Functionality loss – less connectivity, less risk
  • May reduce network’s usefulness
  • Some applications don’t work with firewalls
    • Two peer-to-peer users behind different firewalls

• The malicious insider problem
  • Assume insiders are trusted
    • Malicious insider (or anyone gaining control of internal machine) can wreak havoc

• Firewalls establish a security perimeter
  • Like Eskimo Pies: “hard crunchy exterior, soft creamy center”
  • Threat from travelers with laptops, cell phones, …
Lateral Movement

- Common attack: compromise an internal machine, then use that to attack other internal machines
- From there, you can now exploit internal systems directly
  - Bypassing the primary firewall
- That is the shortcoming of firewalls: A **single** breach of the perimeter by an attacker and you can no longer make **any** assertions about subsequent internal state
Takeaways on Firewalls

• Firewalls: Reference monitors and access control all over again, but at the network level
• Attack surface reduction
• Centralized control
Detection
Structure of FooCorp Web Services

Internet

FooCorp's border router

Remote client

Internet

FooCorp's border router

FooCorp Servers

Front-end web server

bin/amazeme -p xxx

2. GET /amazeme.exe?profile=xxx

8. 200 OK
Output of bin/amazeme
Network Intrusion Detection

• Approach #1: look at the network traffic
  • (a “NIDS”: rhymes with “kids”)
  • Scan HTTP requests
  • Look for “/etc/passwd” and/or “../../” in requests
    • Indicates attempts to get files that the web server shouldn't provide
Structure of FooCorp Web Services

1. Remote client
2. GET /amazeme.exe?profile=xxx
8. 200 OK
   Output of bin/amazeme

Monitor sees a copy of incoming/outgoing HTTP traffic

FooCorp’s border router

FooCorp Servers

Front-end web server

bin/amazeme -p xxx

Internet

FooCorp Web Services

Computer Science 161 Spring 2020
Popa and Wagner
Network Intrusion Detection

- Approach #1: look at the network traffic
  - (a “NIDS”: rhymes with “kids”)
  - Scan HTTP requests
  - Look for “/etc/passwd” and/or “..../..”

- Pros:
  - No need to touch or trust end systems
    - Can “bolt on” security
  - Cheap: cover many systems w/ single monitor
  - Cheap: centralized management
Inside the NIDS

HTTP Request
URL = /fubar/
Host = ....

HTTP Request
URL = /baz/?id=...
ID = 1f413

Sendmail
From = someguy@
To = otherguy@...
Network Intrusion Detection (NIDS)

- NIDS has a table of all active connections, and maintains state for each
  - e.g., has it seen a partial match of /etc/passwd?
- What do you do when you see a new packet not associated with any known connection?
  - Create a new connection: when NIDS starts it doesn’t know what connections might be existing
- New hotness: Network monitoring
  - Goal is not to detect attacks but just to understand everything.
Evasion

• What should NIDS do if it sees a RST packet?
  
  • Assume RST will be received?
  • Assume RST won’t be received?
  • Other (please specify)
Evasion

• What should NIDS do if it sees this?
  • Alert – it’s an attack
  • No alert – it’s all good
  • Other (please specify)
Evasion

- Evasion attacks arise when you have “double parsing”

- \textit{Inconsistency} - interpreted differently between the monitor and the end system

- \textit{Ambiguity} - information needed to interpret correctly is missing
Evasion Attacks (High-Level View)

- Some evasions reflect incomplete analysis
  - In our FooCorp example, hex escapes or “.//./../” alias
  - In principle, can deal with these with implementation care (make sure we fully understand the spec)
    - Of course, in practice things inevitably fall through the cracks!

- Some are due to imperfect observability
  - For instance, if what NIDS sees doesn’t exactly match what arrives at the destination
  - E.g., two copies of the “same” packet, which are actually different and with different TTLs
Network-Based Detection

- Issues:
  - Scan for "/etc/passwd"?
    - What about other sensitive files?
  - Scan for "../../"?
    - Sometimes seen in legit. requests (= false positive)
    - What about "%2e%2e%2f%2e%2e%2f"? (= evasion)
      - Okay, need to do full HTTP parsing
    - What about "..////.///.///.///.///.///.///.///.///.///."?
      - Okay, need to understand Unix filename semantics too!
  - What if it’s HTTPS and not HTTP?
    - Need access to decrypted text / session key – yuck!
Host-based Intrusion Detection

• Approach #2: instrument the web server
• Host-based IDS (sometimes called “HIDS”)
• Scan arguments sent to back-end programs
  • Look for “/etc/passwd” and/or “../..”
Structure of FooCorp Web Services

Internet

FooCorp’s border router

FooCorp Servers

Remote client

Front-end web server

```
4. amazeme.exe? profile=xxx
```

```
6. Output of bin/amazeme sent back
```

```
bin/amazeme -p xxx
```

HIDS instrumentation added inside here
Host-based Intrusion Detection

• Approach #2: instrument the web server
  • Host-based IDS (sometimes called “HIDS”)
  • Scan arguments sent to back-end programs
    • Look for “/etc/passwd” and/or “../..../”

• Pros:
  • No problems with HTTP complexities like %-escapes
  • Works for encrypted HTTPS!

• Issues:
  • Have to add code to each (possibly different) web server
    • And that effort only helps with detecting web server attacks
  • Still have to consider Unix filename semantics (“. . . / . . . “)
  • Still have to consider other sensitive files
Log Analysis

• Approach #3: each night, script runs to analyze log files generated by web servers

• Again scan arguments sent to back-end programs
Structure of FooCorp Web Services

Internet

FooCorp’s border router

FooCorp Servers

Remote client

Run Nightly Analysis Of Logs Here

Front-end web server

bin/amazeme -p xxx
Log Analysis:
Aka "Log It All and let Splunk Sort It Out"

- Approach #3: each night, script runs to analyze log files generated by web servers
  - Again scan arguments sent to back-end programs
- Pros:
  - Cheap: web servers generally already have such logging facilities built into them
  - No problems like %-escapes, encrypted HTTPS
- Issues:
  - Again must consider filename tricks, other sensitive files
  - Can’t block attacks & prevent from happening
  - Detection delayed, so attack damage may compound
  - If the attack is a compromise, then malware might be able to alter the logs before they’re analyzed
    - (Not a problem for directory traversal information leak example)
    - Also can be mitigated by using a separate log server
System Call Monitoring (HIDS)

• Approach #4: monitor system call activity of backend processes
  • Look for access to /etc/passwd
Structure of FooCorp Web Services

Internet

FooCorp’s border router

Real-time monitoring of system calls accessing files

FooCorp Servers

Front-end web server

5. bin/amazeme -p xxx

Remote client
System Call Monitoring (HIDS)

• Approach #4: monitor system call activity of backend processes
  • Look for access to /etc/passwd

• Pros:
  • No issues with any HTTP complexities
  • May avoid issues with filename tricks
  • Attack only leads to an “alert” if attack succeeded
    • Sensitive file was indeed accessed

• Issues:
  • Maybe other processes make legit accesses to the sensitive files (false positives)
  • Maybe we’d like to detect attempts even if they fail?
    • “situational awareness”
Detection Accuracy

- Two types of detector errors:
  - False positive (FP): alerting about a problem when in fact there was no problem
  - False negative (FN): failing to alert about a problem when in fact there was a problem

- Detector accuracy is often assessed in terms of rates at which these occur:
  - Define $I$ to be the event of an instance of intrusive behavior occurring (something we want to detect)
  - Define $A$ to be the event of detector generating alarm

- Define:
  - False positive rate = $P[A|\neg I]$
  - False negative rate = $P[\neg A| I]$
Perfect Detection

• Is it possible to build a detector for our example with a false negative rate of 0%?

• Algorithm to detect bad URLs with 0% FN rate:
  ```c
  void my_detector_that_never_misses(char *URL)
  {
    printf("yep, it's an attack!\n");
  }
  ```

• In fact, it works for detecting any bad activity with no false negatives! Woo-hoo!

• Wow, so what about a detector for bad URLs that has no false positives?
  • `printf("nope, not an attack\n")`;