Firewall & Network-based Intrusion Detection

Dawn Song

dawnsong@cs.berkeley.edu

Review

- **Reference Monitor**
  - **Software Fault Isolation**
  - **System call interposition**

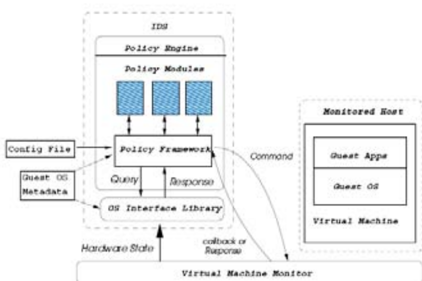
How to Protect Policy Checker?

- **In different user process or in kernel**
- **Relying on the trust to kernel**
- **Can we do better?**

Virtual Machine Monitors

- **Virtual machine: execution envrionment that gives the illusion of a real machine**
- **VMM**
  - **sits below OS**
  - **Much smaller than OS, easier to verify/get right**
  - **Natual place to enforce security policies**
  - **Policy checker does not need to rely on OS**
- **Examples**
  - **VMWare**
  - **Xen**

Virtual Machine Introspection based Policy Enforcement



Sample Security Policies

- **Enforce memory access**
- **Enforce NIC access: e.g., prevent promiscuous mode**
- **Raw socket detector**
- **Signature detector**
- **Program integrity checker**
- **Lie detector for rootkits**

Summary of VMM-based Enforcement

- VMM is much smaller, easier for correctness
- See entire system state, powerful policies
- Much higher performance overhead

Moving to yet another level

- Inline reference monitor
- System call interposition
- VMM introspection
- Can we move it to yet another level?

Network-level Security Policy Enforcement

- Firewalls
  - Peremiter defense
  - Btw internet & intranet
  - Block traffic violating security policy
  - Central chokepoint uses single place to easily enforce a security policy on 1,000's of machines
    - Similar to airport security – few entrances

Packet Filters

- **Simplest kind of firewall is a *packet filter***
  - Router with list of access control rules
  - Router checks each received packet against security rules to decide to forward or drop it
  - Each rule specifies which packets it applies to based on a packet's header fields
    - » Specify source and destination IP addrs, port numbers, and protocol names, or wild cards
    - » Each rule also specifies an action for matching packets: ALLOW or DROP
    - » `<ACTION> <PRTCL> <SRC:PT> -> <DEST:PT>`
  - List of rules is examined one-by-one
    - » First matching rule determines how packet will be handled

Security Policy based on IP Header

- **A TCP service is specified by machine's IP address and TCP port number on it**
  - Web server `www.cs.berkeley.edu` at `169.229.60.105`, port `80`
  - Mail service at `169.229.60.93`, port `25`
  - UDP services similarly identified
- **Identify each svc with triplet *(m,r,p)*:**
  - *m* is machine's IP addr (`A.B.C.D/[MASK]`)
  - *r* is a TCP/UDP protocol identifier
  - *p* is the port number
  - Example: official web servers on subnet `1.2.3.x` -> add(`1.2.3.0/24`, TCP, `80`) to allowed list

Example Ruleset

- **What does this ruleset do?**
  - `drop  tcp *:* -> *:23`
  - `allow  *  *:* -> *:*`
- **Answer:**
  - Blocks all TCP pkts destined to port 23 (telnet)
    - » Telnet uses cleartext passwords!
  - Forwards all other traffic
- **Problems?**
- **No notion of a connection, or of inbound vs outbound connections**
  - Drops outbound telnet connections from inside users
  - This is a default-allow policy!!

Another Example

- **Want to allow:**
  - **Inbound mail connections to our mail server (`1.2.3.4:25`)**
  - **All outbound connections**
  - **Nothing else**
  - **Consider this ruleset:**
    - » `allow tcp *:* -> 1.2.3.4:25`
    - » `allow tcp {int_hosts}:* -> *:*`
    - » `drop   *  *:* -> *:*`
- **This policy doesn't work…**
  - **TCP connections are bidirectional**
  - **3-way handshake: send SYN, receive SYN|ACK, send ACK, send DATA w/ACK bit**

Problem: Outbound Connections Fail

- **Inside host opens TCP connection to port 80 on external machine:**
  - **Initial SYN packet passed through by rule 2**
  - **SYN|ACK packet coming back is dropped**
    - » **Fails rule 1 (not destined for port 25)**
    - » **Fails rule 2 (source not inside host)**
    - » **Matches rule 3 -> DROP**
- **Distinguish between 2 kinds of inbound pkts**
  - **Allow inbound packets associated with an outbound connection to pass**
  - **Restrict inbound packets associated with an inbound connection**

Inbound versus Outbound Connections

- **Key idea: use a feature of TCP!**
  - **ACK bit set on all packets except first one**
  - **Recipients discard any TCP packet with ACK bit set, if packet is not associated with an existing TCP connection**
- **Solution ruleset?**
  - `allow tcp *:* -> 1.2.3.4:25`
  - `allow tcp {int_hosts}:* -> *:*`
  - `allow tcp *:* -> {int_hosts}:* (if ACK bit set)`
  - `drop   *  *:* -> *:*`
  - **Rules 1 and 3 allow inbound connections to port 25 on machine `1.2.3.4`**
  - **Rules 2 and 3 allow outbound connections to any port**

Example Using This Ruleset

- Outside attacker trying to exploit finger service (TCP port 79) vulnerability
  - Tries to open an inbound TCP connection to our finger server
- Attempt #1:Sends SYN pkt to int. machine
  - Pkt doesn't have ACK bit set, so fw rule drops it
- Attempt #2: Sends SYN|ACK pkt to internal machine
  - FW permits pkt, then dropped by TCP stack (ACK bit set but isn't part of existing connection)
- We can specify policies restricting inbound connections arbitrarily

IP Spoofing: Another Security Hole

- IP protocol doesn't prevent attacker from sending pkt with wrong (*spoofed*) src addr
  - Most routers ignore src addrs
- Suppose `1.2.3.7` is an internal host
  - Attacker sends spoofed TCP SYN packet
    - Src addr `1.2.3.7`, dest addr target internal machine, dest port 79 – rule 2 allows
  - Target replies with SYN|ACK pkt to `1.2.3.7` and waits for ACK (to finish 3-way handshake)
  - Attacker sends spoofed TCP ACK packet
  - Attacker then sends data packet

Attack Analysis

- Attack allows connections to internal hosts
  - Violates of our security policy
  - Allows attacker to exploit any security holes
    - Ex: finger service vulnerability
  - Caveat:
    - Attacker has to "guess" Initial Sequence Number set by target in SYN|ACK packet sent to `1.2.3.7` (many ways to guess…)
- Modified Solution
  - Packet filter marks each packet with incoming interface ID, and rules match IDs
    - Recall: Router has 2+ interfaces, forwards packets from one to another

New Solution

- **New ruleset**
  - **Int. interface: `in`, ext. interface: `out`**
  - `allow tcp *:*/out -> 1.2.3.4:25/in`
  - `allow tcp *:*/in  -> *:*/out`
  - `allow tcp *:*/out -> *:*/in  (if ACK bit set)`
  - `drop   *  *:*     -> *:*`
  - **Allows inbound packets only if destined to `1.2.3.4:25` (rule 1), or, if ACK bit set (rule 3)**
  - **Drops all other inbound packets**
- **Clean solution: defeats IP spoofing threat**
  - **Simplifies ruleset admin (no hardcode internal hosts list)**

Other Kinds of Firewalls

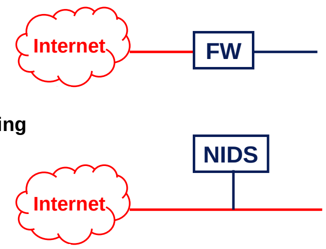
- **Packet filters are quite crude firewalls**
  - **Network level using TCP, UDP, and IP headers**
- **Alternative: examine data field contents**
  - **Application-layer firewalls (application firewalls)**
    - **Can enforce more restrictive security policies and transform data on the fly**
- **For more information on firewalls, read:**
  - **Cheswick, Bellovin, and Rubin: *Firewalls and Internet Security: Repelling the Wily Hacker*.**
- **Packet filtering sw available for many OS's:**
  - **Linux `iptables`, OpenBSD/FreeBSD PF, and Windows XP SP2 firewall**

Administravia

- **Expect emails soon from John regarding milestone #2 feedback**
- **Most groups did well**
- **Need to follow interface specs**

Firewall vs. NIDS

- **Firewall**
  - **Active filtering**
  - **Fail-close**
- **Network IDS**
  - **Passive monitoring**
  - **Fail-open**



Network-based Intrusion Detection

- **Often stateful, deep-packet inspection**
  - **Full stream re-assembly**
- **Examples**
  - **Snort**
  - **Bro**
  - **Commercial appliances**
- **Detection methods**
  - **Misuse detection (signature-based)**
    - » **E.g., snort rules**
  - **anomaly detection (specification-based or statistical-based)**
    - » **E.g., port-scanning detection**
- **Often much more complex than packet filters**

Attacks on NIDS

- **Algorithmic complexity attacks**
- **Evasion attacks**
- **Stealthy port scanning**

Algorithmic Complexity Attacks

- DoS attacks not only serious for denying service, but can be more severe by using it as a component of an attack
- DoS attack on IDS enables other attacks to remain undetected
- "Denial of Service via Algorithmic Complexity Attacks" by Crosby and Wallach

Complexity Attack on Hash Table

- On average, a hash table has O(n) overhead to insert n elements
- In the worst case, a hash table may have $O(n^2)$ overhead to insert n elements!
- Attack against Perl hash table:
  - 90K inserts
    - » Random: < 2 sec
    - » Worse case: > 6500 sec

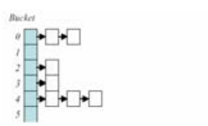


Figure 1: Normal operation of a hash table.

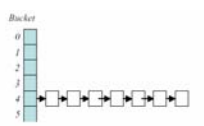


Figure 2: Worst-case hash table collisions.

Complexity Attack Against Bro

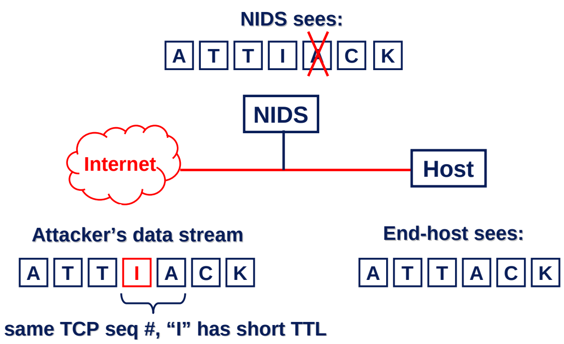
- Bro uses simple xor to "hash" values for hash table
  - Easy to find collisions!
- Example: Bro port scanning detector keeps a hash table of dst IP addresses
  - Keep the list of dst IP addresses for each <src IP, dst port>
- Using source IP spoofing, can exploit this structure to perform DoS attack!

	Attack	Random
Total CPU time	44.50 min	.86 min
Hash table time	43.78 min	.02 min

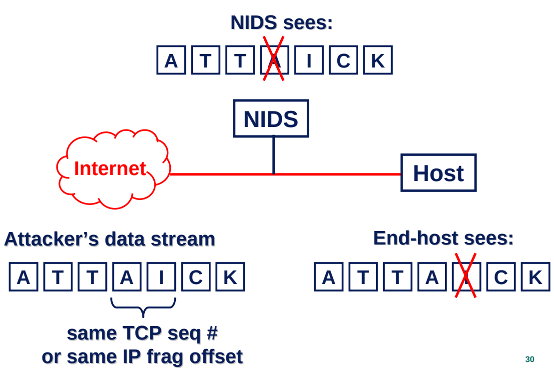
NIDS: Evasion & Normalization

- **Problems**
  - **Complete fragment reassembly necessary to detect certain attacks**
  - **NIDS only has partial knowledge of what traffic the host sees (e.g., TTL expires, MTU)**
  - **Ambiguities in TCP/IP (e.g., Overlapping IP & TCP fragments)**
    - » **Different OS implement standard differently**

Small TTL Attack

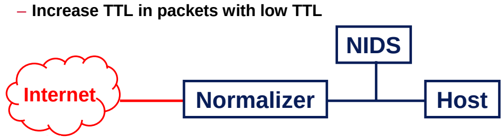


Fragmentation Overlap Attack



Solution: Traffic Normalizer

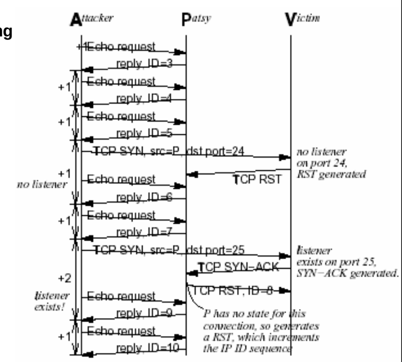
- Introduce “bump in the wire”: traffic normalizer to evade protocol ambiguities
  - Drop overlapping IP/TCP fragments
  - Increase TTL in packets with low TTL



- Other approaches
  - Host-based IDS
  - Detailed Intranet map

Stealth Port Scanning

- IP id field used for stealth port scanning



Principle: Reference Monitor

- SFI, System call interposition, VMM introspection, Firewall/NIDS: one thing in common
- One enforcement mechanism: *reference monitor*
  - Examines every request to access any controlled resource (an object) and determines whether to allow request

Reference Monitor Security Properties

- ***Always invoked***
  - *Complete mediation* property: all security-relevant operations must be mediated by RM
  - RM should be invoked on every operation controlled by access control policy
- ***Tamper-resistant***
  - Maintain RM integrity (no code/state tampering)
- ***Verifiable***
  - Can verify RM correctness (correctly enforces desired access control policy)
    - » Requires extremely simple RM
    - » Can't verify correctness for systems with any appreciable degree of complexity

Conclusion

- VMM introspection
- Firewall/NIDS
- Reference monitor
  - Fundamental security concept
  - Apply at different levels
  - Enforce security policies & limit damage on attacks