UMSSIA
INTRUSION DETECTION
INTRUSION DETECTION

Sensor1, Event1, Event2...

SensorN, Event1, Event2...

Monitor

No intrusion

Alarm!

Danger!
### IDS CHARACTERISTICS

Characteristics an IDS can be classified/evaluated by:

<table>
<thead>
<tr>
<th>Type of Sensors?</th>
<th>Host IDS / Network IDS / etc...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Search?</td>
<td>Look for known intrusions vs Look for unknown behavior</td>
</tr>
<tr>
<td></td>
<td>Specified vs Learned</td>
</tr>
<tr>
<td>Error Rates?</td>
<td>Against “typical” attacks?</td>
</tr>
<tr>
<td>Evasion?</td>
<td>Errors against targeted attacks?</td>
</tr>
</tbody>
</table>
A **False Positive** error occurs when a non-intrusion causes an alarm, e.g.
- Squirrel chews through sensor cable
- Printer driver scans subnet for printer
- User mistypes password 3x

A **False Negative** error occurs when an intrusion does not result in an alarm.

- **False Positive Rate (FPR)** = \( \frac{\#\text{False Positives}}{\#\text{Normal Events}} \)
- **False Negative Rate (FNR)** = \( \frac{\#\text{False Negatives}}{\#\text{Intrusions}} \)
BASE RATE FALLACY

• Suppose the U of M has
  – 10M network flows each day
  – 100 flows are genuine intrusions

• Suppose we use an IDS with a False positive rate of 0.1%. Then:
  – There are 10K false alarms.
  – So 99.9% of alarms are false positives

• We would need a FPR of just 0.001% to make 50% of alarms real.
SIGNATURE MATCHING IDS

• The “misuse detection” problem is to find behavior that “looks like” an intrusion.

• The basic form of such systems is:
  1. Collect information about known attack methods and types, such as SrcAddr, SrcPort, DstAddr, DstPort, Protocol, Packet contents
  2. Create and look for signatures: a Code red packet, a port scan, etc...
SIGNATURE DETECTION

- Language to specify intrusion patterns
  - 4-tuple/protocol values for potential intrusion
    * Eg External host -> file server (port 110, 135, ...)
    * Eg Internal workstation -> external P2P host
  - Packet contents: Could be single or multiple packets (stream reconstruction)
  - Possibly model of protocol/app finite state machine

- Lots of state in pattern matching engine

- Example rule:
  - alert tcp any any -> myip 21 (content:"site exec"; content:"%"; msg:"site exec buffer overflow attempt";)

SIGNATURE MATCHING

• Advantages:
  – very low false positive rate
  – Automated extraction is possible

• Disadvantages
  – Only detects already known attacks
  – Simple changes to an attack can defeat detection, for example:
    – scan even ports, then odd ports.
    – “rm –rf /*” -> “rm –rf ../../../../../../../*”
ANOMALY DETECTION

1. Try to identify what is normal, e.g. “normal” command sequence or common 4-tuple/protocol, session length, intervals, etc...

2. Look for major deviations (outliers), e.g. unusual target port, source addr, or port sequence (scan)

Sometimes (but not always): Apply AI/Machine Learning to “learn” what is normal.

Advantage: more robust to “altered” attacks.
ANOMALOUS PROBLEMS

- High false positive rate
- Attacks might not be obvious until too late
- Attacks can hide in “normal” traffic
- Require training on “known good” data
- Problems when what’s “normal” changes
  - Eg, flash crowds, new users, new applications…
HOST-BASED IDS

... monitor a single host to detect intrusions. Typical “sensors” include:
- Disk & memory contents
- User input and commands
- System calls

... Typically equate “intrusion” with privilege escalation, eg remote ! local ! root.

... Are vulnerable to various attacks, e.g.: replace HIDS process/binaries, “Mimic” attack, context problems, subvirtion.
EXAMPLE: TRIPWIRE

• A “standard attack” might go like this:
  – Gain user access to system
  – Gain root access
  – Replace system binaries to set up backdoor
  – Use backdoor for future activities

• **Tripwire** is a simple anomaly-based HIDS that monitors system binaries:
  – Compute hash of key system binaries
  – Compare current hash to earlier stored hash
  – Report problem if hash is different
  – Store reference hash codes on read-only medium

• Example attack: “rootkit” replaces binaries with new versions **with the same hash.**
IDES was a HIDS developed in the 1980s by Dorothy Denning, SRI. It worked as follows:

- For each user, store daily count of certain activities
  - E.g., Fraction of hours spent reading email
- Maintain a list of counts for several days
- Report an anomaly if the current count is outside the weighted norm

The main vulnerability of IDES was the ability to “train” the system to accept intrusive behaviour.

IDES proposed a solution: include signatures of known intrusive acts and raise an alarm if the profile matches the attack profile.
• In the 1990s
• Build traces during normal run of program
  – Example program behavior (sys calls)
    open read write open mmap write fchmod close
  – Store all 4-call sequences in a file:
    open read write open
    read write open mmap
    write open mmap write
    open mmap write fchmod
    mmap write fchmod close
• Use an “Order 4 Markov Model” to compute the probability of a new syscall sequence...
  – Compute # of mismatches to get mismatch rate
  – For example, report an anomaly if you see:
    open read read open mmap write fchmod close
• Vulnerability: “Mimic attack”
Example: Wagner & Dean, IEEE S&P ’01:

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

Idea: Build NFA for each application, from source code.
A network-based IDS monitors network traffic for intrusions, e.g.:

- DoS
- Exploiting bugs in protocol, application, OS
- Install worm, virus, bot, spyware...

Challenges for NIDS: fragmentation, data volume, “low and slow” attacks...
• A typical attack is “port scanning” to find out what services a host is running.
• For example, the Nmap tool: (http://www.insecure.org/nmap/)
  – Determines OS/hostname/device type via “service fingerprinting” (eg, IRIX listens on TCP port 1)
  – Determines what service is listening on a port and can determine application name, version
  – Operates in optional obfuscation mode
• Afterwards, attackers can exploit known vulnerabilities in the OS/applications found via Nmap.
EXAMPLE: SNORT

• ... is a signature-based, portable, opensource NIDS with over 2 million downloads, and 100K active users.
• ... scans a tcpdump log, and finds connections matching attack signatures.
• ... uses regular expressions that match known attacks.
• Example sig:
  – alert tcp any any -> [a.b.0.0/16,c.d.e.0/24] 80
    ( msg:"WEB-ATTACKS conf/httpd.conf attempt"; nocase; sid:1373;
      flow:to_server,established;
      content:"conf/httpd.conf"; [...] )
EXAMPLE: BRO

• ... is a high-speed “policy-based” NIDS. It uses scripts that monitor connections, and check for conformance to “expected” behavior based on protocol.

• ... logs, for all TCP connections (via SYN/FIN/RST packets): start time, duration, service, addresses, sizes, etc. It also identifies the application based on these packets.

• ... supports many standard applications: DNS, FTP, HTTP, SMTP, NTP, ...

• ... on Telnet and Rlogin generates the following events:
  – login_successful, login_failure, activating_encryption, login_confused, login_input_line, login_output_line

• In its first five months of operation, Bro found 120 UCB break-ins (60 root compromises)
DISTRIBUTED IDS

- Idea: combine data from many sensors
  - HIDS data from many hosts
  - NIDS data from multiple segments
- Positives: can detect “stealthy” scans, possibly lower false alarm rate, etc.
- Negatives: much more data to scan, exchange, etc.
A **honeypot** is a closely monitored network decoy or decoy(s). Honeypots:

- Distract adversaries from more valuable machines on a network (?)
- Provide early warning about new attack and exploitation trends
- Frequently consist of multiple VMs that use up the “empty” IP address space on a network.

A **tarpit** is essentially a slow honeypot that accepts connections, but prevents resets.
INTRUSION PREVENTION

• Idea: respond to detected intrusion
  – Reset connection
  – Block host(s) at firewall
  – Etc...

• Big problem: self-DoS.
  – Attacker makes it look like potentially legitimate connections are intrusions
  – Hosts get blocked at firewall
ATTACKS ON THE IDS

• Witty worm: buffer overflow in commercial IDS systems
• DoS
  – On NIDS: crash system or overwhelm monitor
  – Using NIDS: see IPS
• Subterfuge:
  – Retransmits, NULs in packets, TTLs, checksums, etc..
  – “rob<DEL><BS><BS>ot”