

Intrusion Detection

Vitaly Shmatikov

Reading Assignment

- ◆ Optional: “Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection” by Ptacek and Newman
 - [Linked from the course website \(reference section\)](#)

What Should Be Detected?

- ◆ Attempted and successful break-ins
- ◆ Attacks by legitimate users
 - For example, illegitimate use of root privileges
 - Unauthorized access to resources and data
- ◆ Trojan horses
- ◆ Viruses and worms
- ◆ Denial of service attacks

Where Are IDS Deployed?

◆ Host-based

- Monitor activity on a single host
- Advantage: better visibility into behavior of individual applications running on the host

◆ Network-based (NIDS)

- Often placed on a router or firewall
- Monitor traffic, examine packet headers and payloads
- Advantage: single NIDS can protect many hosts and look for global patterns

Intrusion Detection Techniques

◆ Misuse detection

- Use attack “signatures” (need a model of the attack)
 - Sequences of system calls, patterns of network traffic, etc.
- Must know in advance what attacker will do (how?)
- Can only detect known attacks

◆ Anomaly detection

- Using a model of normal system behavior, try to detect deviations and abnormalities
 - E.g., raise an alarm when a statistically rare event(s) occurs
- Can potentially detect unknown attacks

◆ Which is harder to do?

Misuse vs. Anomaly

- ◆ Password file modified Misuse
- ◆ Four failed login attempts Anomaly
- ◆ Failed connection attempts on 50 sequential ports Anomaly
- ◆ User who usually logs in around 10am from a UT dorm logs in at 4:30am from a Russian IP address Anomaly
- ◆ UDP packet to port 1434 Misuse
- ◆ "DEBUG" in the body of an SMTP message Not an attack!
(most likely)

Misuse Detection (Signature-Based)

- ◆ Set of **rules** defining a behavioral signature likely to be associated with attack of a certain type
 - Example: **buffer overflow**
 - A setuid program spawns a shell with certain arguments
 - A network packet has lots of NOPs in it
 - Very long argument to a string function
 - Example: **SYN flooding (denial of service)**
 - Large number of SYN packets without ACKs coming back
 - ...or is this simply a poor network connection?
- ◆ Attack signatures are usually very specific and may miss variants of known attacks
 - **Why not make signatures more general?**

Extracting Misuse Signatures

- ◆ Use **invariant** characteristics of known attacks
 - Bodies of known viruses and worms, port numbers of applications with known buffer overflows, RET addresses of overflow exploits
 - Hard to handle mutations
 - Polymorphic viruses: each copy has a different body
- ◆ Big research challenge: fast, automatic extraction of signatures of new attacks
- ◆ **Honeypots** are useful for signature extraction
 - Try to attract malicious activity, be an early target

Anomaly Detection

- ◆ Define a **profile** describing “normal” behavior
 - Works best for “small”, well-defined systems (single program rather than huge multi-user OS)
- ◆ Profile may be statistical
 - Build it manually (this is hard)
 - Use machine learning and data mining techniques
 - Log system activities for a while, then “train” IDS to recognize normal and abnormal patterns
 - Risk: attacker trains IDS to accept his activity as normal
 - Daily low-volume port scan may train IDS to accept port scans
- ◆ IDS flags deviations from the “normal” profile

What's a "Profile?"

◆ Login and session activity

- Login and location frequency; last login; password fails; session elapsed time; session output, CPU, I/O

◆ Command and program execution

- Execution frequency; program CPU, I/O, other resources (watch for exhaustion); denied executions

◆ File access activity

- Read/write/create/delete frequency; records read/written; failed reads, writes, creates, deletes; resource exhaustion

◆ How to make all this auditing scalable?

Host-Based IDS

- ◆ Use OS auditing and monitoring mechanisms to find applications taken over by attacker
 - Log all relevant system events (e.g., file accesses)
 - Monitor shell commands and system calls executed by user applications and system programs
 - Pay a price in performance if every system call is filtered
- ◆ **Con:** need an IDS for every machine
- ◆ **Con:** if attacker takes over machine, can tamper with IDS binaries and modify audit logs
- ◆ **Con:** only local view of the attack

Level of Monitoring

◆ Which types of events to monitor?

- OS system calls
- Command line
- Network data (e.g., from routers and firewalls)
- Processes
- Keystrokes
- File and device accesses

Host-Based Anomaly Detection

- ◆ Compute statistics of certain system activities
- ◆ Report an alert if statistics outside range
- ◆ Example: **IDES** (Denning, mid-1980s)
 - For each user, store daily count of certain activities
 - For example, fraction of hours spent reading email
 - Maintain list of counts for several days
 - Report anomaly if count is outside weighted norm

Problem: most unpredictable user is the most important

Tripwire



◆ File integrity checker

- Records hashes of critical files and binaries
 - Recorded hashes must be in read-only memory (why?)
- Periodically checks that files have not been modified, verifies sizes, dates, permission

◆ Good for detecting rootkits

◆ Can be subverted by a clever rootkit

- Install backdoor inside a continuously running system process (no changes on disk!)
- Copy old files back into place before Tripwire runs

◆ How to detect modifications to running process?

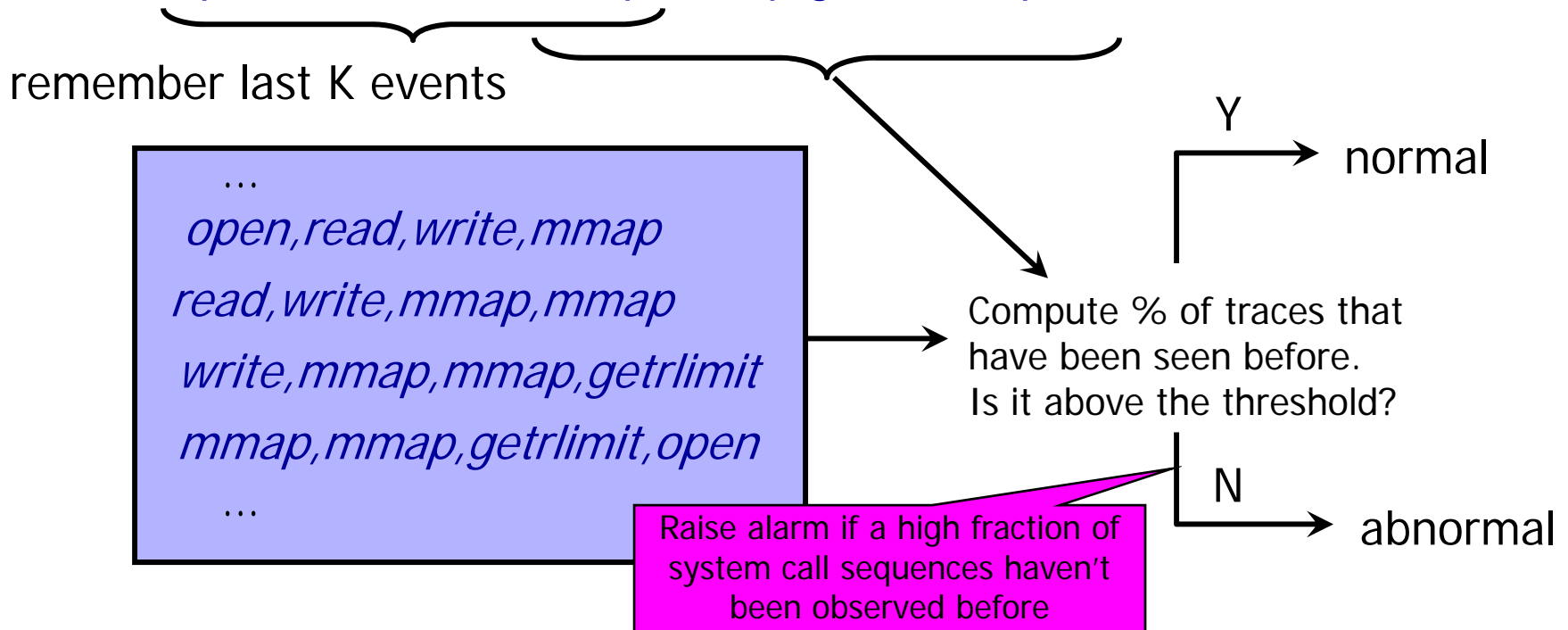
"Self-Immunology" Approach

[Forrest]

◆ Normal profile: short sequences of system calls

- Use strace on UNIX

... *open,read,write,mmap,mmap,getrlimit,open,close* ...



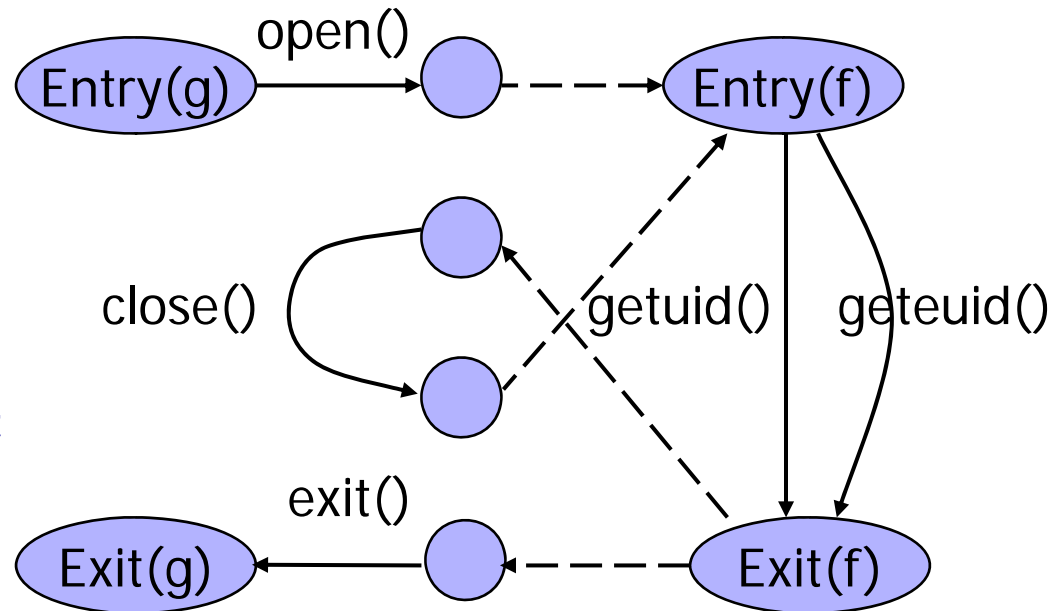
Better System Call Monitoring

[Wagner and Dean]

- ◆ Use static analysis of source code to find out what a normal system call sequence looks like
 - Build a finite-state automaton of expected system calls
- ◆ Monitor system calls from each program
- ◆ System call automaton is conservative
 - No false positives!

Wagner-Dean Example

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



If code behavior is inconsistent with this automaton, something is wrong

Network-Based IDS

- ◆ Inspect network traffic
 - For example, use `tcpdump` to sniff packets on a router
 - Passive (unlike firewalls)
 - Default action: let traffic pass (unlike firewalls)
- ◆ Watch for protocol violations, unusual connection patterns, attack strings in packet payloads
 - Check packets against rule sets
- ◆ **Con:** can't inspect encrypted traffic (IPsec, VPNs)
- ◆ **Con:** not all attacks arrive from the network
- ◆ **Con:** record and process huge amount of traffic

U. of Toronto, 2004 (from David Lie)

◆ Date: Fri, 19 Mar 2004

◆ Quote from email:

"The campus switches have been bombarded with these packets [...] and apparently 3Com switches reset when they get these packets. This has caused the campus backbone to be up and down most of yesterday. The attack seems to start with connection attempts to port 1025 (Active Directory logon, which fails), then 6129 (DameWare backdoor, which fails), then 80 (which works as the 3Com's support a web server, which can't be disabled as far as we know). The HTTP command starts with 'SEARCH /\x90\x02\xb1\x02' [...] then goes off into a continual pattern of '\x90' "

Popular NIDS



◆ Snort (popular open-source tool)

- Large rule sets for known vulnerabilities
 - **2009-03-31:** A programming error in MySQL Server may allow a remote attacker to cause a Denial of Service (DoS) against a vulnerable machine.
 - **2009-03-27:** Microsoft Windows GDI Buffer Overflow: A programming error in the Microsoft Windows kernel may allow a remote attacker to execute code with system level privileges. This may be exploited when specially crafted EMF files are viewed using Microsoft Internet Explorer.

◆ Bro (developed by Vern Paxson)



- Separates data collection and security decisions
 - **Event Engine** distills the packet stream into high-level events describing what's happening on the network
 - **Policy Script Interpreter** uses a script defining the network's security policy to decide what to do in response

Irony and NIDS

Sourcefire Snort Remote Buffer Overflow

- ◆ Notification Type: IBM Internet Security Systems Protection Advisory
- ◆ Notification Date: Feb 19, 2007
- ◆ Description: **Snort IDS** and Sourcefire Intrusion Sensor IDS/IPS are **vulnerable to a stack-based buffer overflow**, which can result in remote code execution.

... patched since then (phew!)

Witty Worm

- ◆ Exploits buffer overflow in the ICQ filtering module of ISS BlackICE/RealSecure intrusion detectors
 - Single UDP packet to port 4000, standard stack smash
 - Deletes randomly chosen sectors of hard drive
 - Payload contains “(^.^ insert witty message here ^.^)”
- ◆ Chronology of Witty
 - Mar 8, 2004: vulnerability discovered by EEye
 - Mar 18, 2004: high-level description published
 - 36 hrs later: worm released through 100 infectees
 - Accelerate the slow initial phase ⇒ very fast propagation!
 - 45 mins later: all 12,000 vulnerable machines infected!

Port Scanning

- ◆ Many vulnerabilities are OS-specific
 - Bugs in specific implementations, default configuration
- ◆ Port scan is often a prelude to an attack
 - Attacker tries many ports on many IP addresses
 - For example, looking for an old version of some daemon with an unpatched buffer overflow
 - If characteristic behavior detected, mount attack
 - Example: SGI IRIX responds on TCPMUX port (TCP port 1); if response detected, IRIX vulnerabilities can be used to break in
 - “The Art of Intrusion”: virtually every attack involves port scanning and password cracking

Scanning Defense

- ◆ **Scan suppression**: block traffic from addresses that previously produced too many failed connection attempts
 - Goal: detect port scans from attacker-controlled hosts
 - Requires network filtering and maintaining state
 - Can be subverted by slow scanning; does not work very well if the origin of scan is far away (why?)
- ◆ False positives are common, too
 - Website load balancers, stale IP caches
 - E.g., dynamically get an IP address that was used by P2P host

Attacking and Evading NIDS

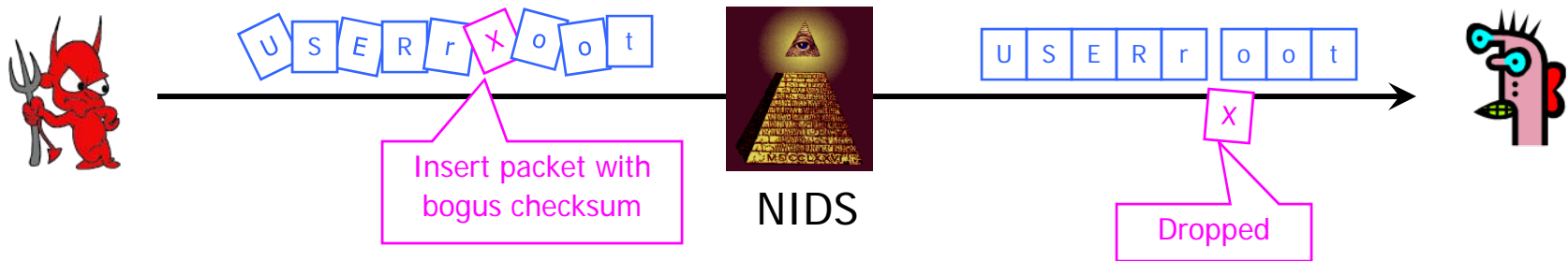
- ◆ Overload NIDS with huge data streams, then attempt the intrusion
 - Bro solution: watchdog timer
 - Check that all packets are processed by Bro within T seconds; if not, terminate Bro, use tcpdump to log all subsequent traffic
- ◆ Use encryption to hide packet contents
- ◆ Split malicious data into multiple packets
 - NIDS does not have full TCP state and does not always understand every command of receiving application
 - Simple example: send "ROB<BS><BS>OT", receiving application may reassemble to "ROOT"

Detecting Attack Strings Is Hard

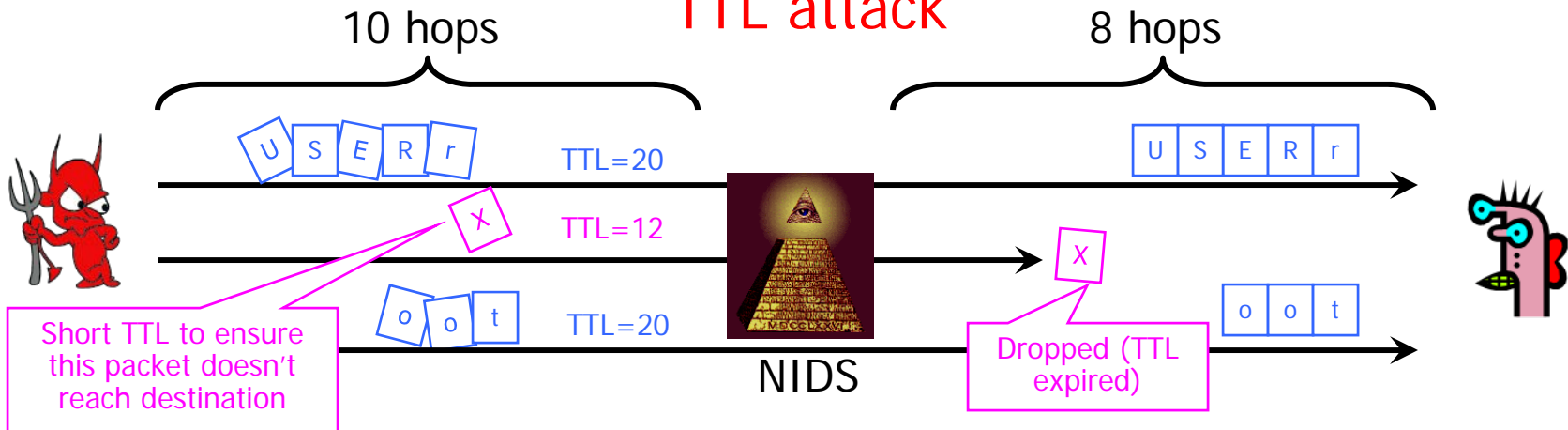
- ◆ Want to detect “USER root” in packet stream
- ◆ Scanning for it in every packet is not enough
 - Attacker can split attack string into several packets; this will defeat stateless NIDS
- ◆ Recording previous packet’s text is not enough
 - Attacker can send packets out of order
- ◆ Full reassembly of TCP state is not enough
 - Attacker can use TCP tricks so that certain packets are seen by NIDS but dropped by the receiving application
 - Manipulate checksums, TTL (time-to-live), fragmentation

TCP Attacks on NIDS

Insertion attack



TTL attack



Anomaly Detection with NIDS

- ◆ Advantage: can recognize new attacks and new versions of old attacks
- ◆ Disadvantages
 - High false positive rate
 - Must be trained on known good data
 - Training is hard because network traffic is very diverse
 - Protocols are finite-state machines, but current state of a connection is difficult to see from the network
 - Definition of “normal” constantly evolves
 - What’s the difference between a flash crowd and a denial of service attack?

Detecting Backdoors with NIDS

- ◆ Look for telltale signs of sniffer and rootkit activity
- ◆ Entrap sniffers into revealing themselves
 - Use bogus IP addresses and username/password pairs; open bogus TCP connections, then measure ping times
 - Sniffer may try a reverse DNS query on the planted address; rootkit may try to log in with the planted username
 - If sniffer is active, latency will increase
 - Clever sniffer can use these to detect NIDS presence!
- ◆ Detect attacker returning to his backdoor
 - Small packets with large inter-arrival times
 - Simply search for root shell prompt "# " (!!)

Intrusion Detection Problems

- ◆ Lack of training data with real attacks
 - But lots of “normal” network traffic, system call data
- ◆ Data drift
 - Statistical methods detect changes in behavior
 - Attacker can attack gradually and incrementally
- ◆ Main characteristics not well understood
 - By many measures, attack may be within bounds of “normal” range of activities
- ◆ False identifications are very costly
 - Sysadm will spend many hours examining evidence

Intrusion Detection Errors

- ◆ **False negatives:** attack is not detected
 - Big problem in signature-based misuse detection
- ◆ **False positives:** harmless behavior is classified as an attack
 - Big problem in statistical anomaly detection
- ◆ Both types of IDS suffer from both error types
- ◆ Which is a bigger problem?
 - Attacks are fairly rare events
 - IDS often suffer from base-rate fallacy

Conditional Probability

- ◆ Suppose two events A and B occur with probability $\Pr(A)$ and $\Pr(B)$, respectively
- ◆ Let $\Pr(AB)$ be probability that both A and B occur
- ◆ What is the **conditional probability** that A occurs assuming B has occurred?

$$\Pr(A \mid B) = \frac{\Pr(AB)}{\Pr(B)}$$

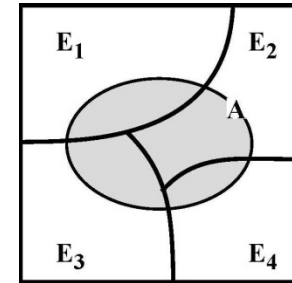
Bayes' Theorem



- ◆ Suppose mutually exclusive events E_1, \dots, E_n together cover the entire set of possibilities
- ◆ Then probability of any event A occurring is

$$\Pr(A) = \sum_{1 \leq i \leq n} \Pr(A | E_i) \cdot \Pr(E_i)$$

- Intuition: since E_1, \dots, E_n cover entire probability space, whenever A occurs, some event E_i must have occurred



- ◆ Can rewrite this formula as

$$\Pr(E_i | A) = \frac{\Pr(A | E_i) \cdot \Pr(E_i)}{\Pr(A)}$$

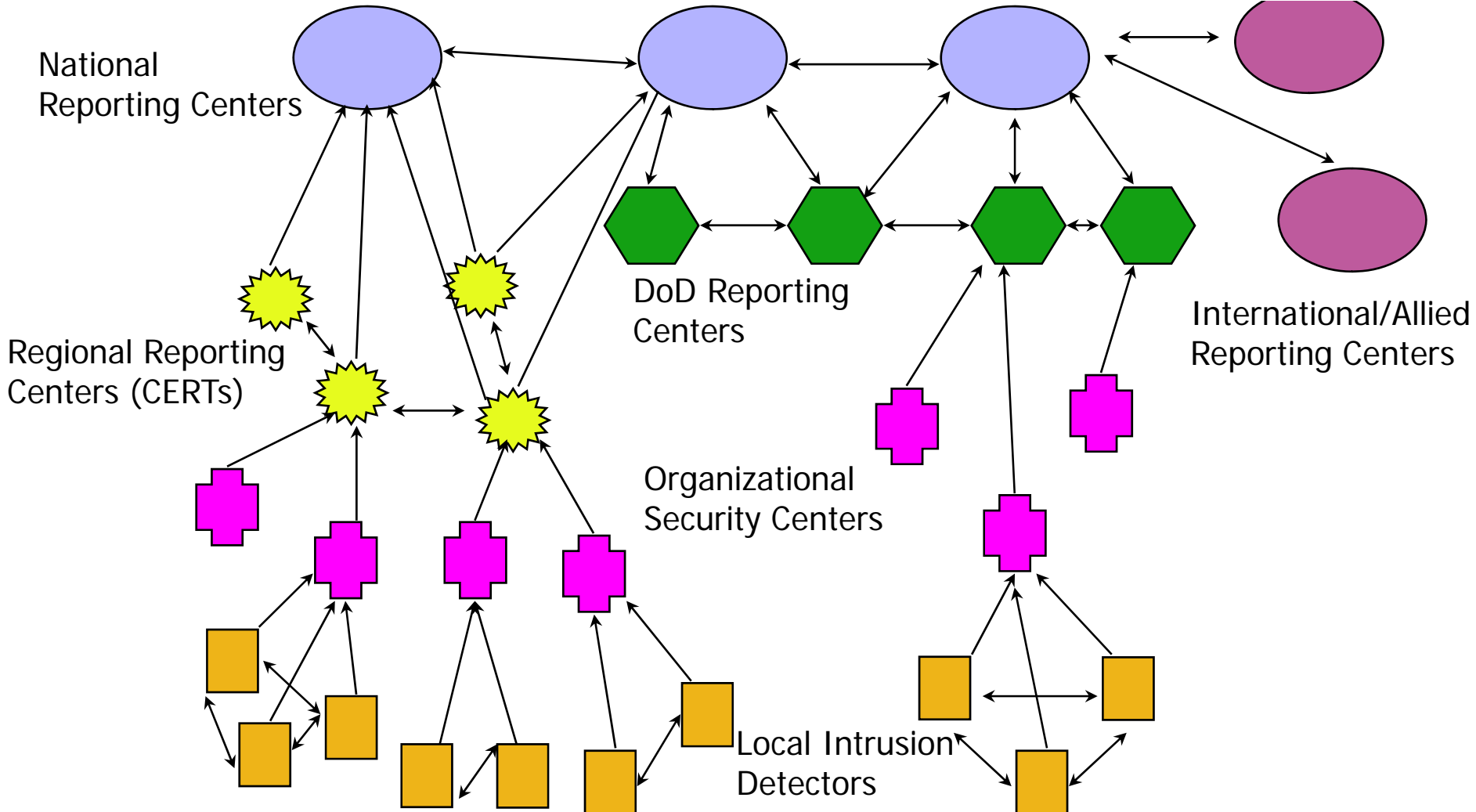
Base-Rate Fallacy

- ◆ 1% of traffic is SYN floods; IDS accuracy is 90%
 - IDS classifies a SYN flood as attack with prob. 90%, classifies a valid connection as attack with prob. 10%
- ◆ What is the probability that a connection flagged by IDS as a SYN flood is actually valid?

$$\begin{aligned} \Pr(\text{valid} \mid \text{alarm}) &= \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm})} \\ &= \frac{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid})}{\Pr(\text{alarm} \mid \text{valid}) \cdot \Pr(\text{valid}) + \Pr(\text{alarm} \mid \text{SYN flood}) \cdot \Pr(\text{SYN flood})} \\ &= \frac{0.10 \cdot 0.99}{0.10 \cdot 0.99 + 0.90 \cdot 0.01} = 92\% \text{ chance raised alarm} \\ &\qquad\qquad\qquad \text{is false!!!} \end{aligned}$$

Strategic Intrusion Assessment

[Lunt]



Strategic Intrusion Assessment

[Lunt]

- ◆ Test over two-week period by Air Force Information Warfare Center
 - Intrusion detectors at 100 Air Force bases alarmed on 2,000,000 sessions
 - Manual review identified 12,000 suspicious events
 - Further manual review => four actual incidents
- ◆ Conclusion
 - Most alarms are false positives
 - Most true positives are trivial incidents
 - Of the significant incidents, most are isolated attacks to be dealt with locally

Network Telescopes and Honeypots

- ◆ Monitor a cross-section of Internet address space
 - Especially useful if includes unused “dark space”
- ◆ Attacks in far corners of the Internet may produce traffic directed at your addresses
 - “Backscatter”: responses of DoS victims to randomly spoofed IP addresses
 - Random scanning by worms
- ◆ Can combine with “honeypots”
 - Any outbound connection from a “honeypot” behind an otherwise unused IP address means infection (why?)
 - Can use this to extract worm signatures (how?)

Case Study: Witty Worm

[Kumar, Paxson, Weaver]

- ◆ Remember Witty worm?
 - Targeted ISS BlackICE/RealSecure intrusion detectors
 - Propagated by sending copies of itself by single-packet UDP to randomly generated IP addresses
- ◆ Network telescope
 - Observes large fraction of IP address space (e.g., /8)
- ◆ UCSD telescope recorded all Witty packets it saw
 - In the best case, it saw approximately 4 out of every 1000 packets sent by each Witty infectee (why?)

Pseudocode of Witty

[Kumar, Paxson, Weaver]

1. `srand(get_tick_count())` ← Seed pseudo-random generator
(Witty uses linear congruential PRNG:
 $X_{i+1} = a X_i + b \text{ mod } m$)
2. `for(i=0; i<20,000; i++)`
3. `destIP ← rand() [0..15] | rand() [0..15]` Each Witty packet contains bits from 4 consecutive pseudo-random numbers
4. `destPort ← rand() [0..15]`
5. `packetSize ← 768 + rand() [0..8]`
6. `packetContents ← top of stack`
7. `send packet to destIP/destPort`
8. `if(open(physicaldisk,rand() [13..15]))`
`write(rand() [0..14] || 0x4E20); goto 1;`
This is enough to recover the state of infectee's PRNG by brute force in 2^{16} attempts (from a single Witty packet!)
Use this to estimate infectee's bandwidth (how?)
9. `else goto 2` ← Answer:

What does it mean if telescope observes consecutive packets that are "far apart" in the pseudo-random sequence? re-seeding of infectee's PRNG caused by failed disk access

Bug in Witty's PRNG

[Kumar, Paxson, Weaver]

- ◆ Witty implements a permutation PRNG
 - Every 32-bit integer is generated exactly once
- ◆ ... but only uses 16 highest bits of each number
 - Misinterprets Knuth's advice that the higher-order bits of linear congruential PRNGs are more "random"
- ◆ Result: misses approx. 10% of IP address space
- ◆ ... but telescope data indicates that some hosts in the "missed" space still got infected
 - Maybe multi-homed or NAT'ed hosts infected via a different (scanned) IP address?

Witty's Hitlist

[Kumar, Paxson, Weaver]

- ◆ Some hosts in the unscanned space got infected very early in the outbreak
 - Many of the infected hosts are in the same /24
 - Witty's PRNG would have generated too few packets into that space to account for the speed of infection
 - They were not infected by random scanning!
 - Attacker had the hitlist of initial infectees
- ◆ Prevalent /16 = U.S. military base
 - Likely explanation: attacker (ISS insider?) knew of ISS software installation at the base
 - Worm released 36 hours after vulnerability disclosure

Patient Zero

[Kumar, Paxson, Weaver]

- ◆ A peculiar “infectee” shows up in the telescope observation data early in the Witty outbreak
 - Sending packets with destination IP addresses that could not have been generated by Witty’s PRNG
 - It was not infected by Witty, but running different code to generate target addresses!
 - Each packet contains Witty infection, but payload size not randomized; also, this scan did not infect anyone
 - Initial infectees came from the hitlist, not from this scan
- ◆ Probably the source of the Witty outbreak
 - IP address belongs to a European retail ISP; information passed to law enforcement