CSE331: Fundamentals of Security

Spring 2015

Radu Sion

Intrusion Detection

© 2005-15
portions © Matt Bishop

Intrusion Detection

- Basics
- Models of Intrusion Detection
- Architecture of an IDS
- Organization
- Incident Response

Goals of IDS

- Detect wide variety of intrusions
  – Previously known and unknown attacks
  – Suggests need to learn/adapt to new attacks or changes in behavior
- Detect intrusions in timely fashion
  – Often needs to be be real-time (especially when system responds to intrusion)
    - Problem: analysis may impact response time of system

Goals of IDS

- Present analysis:
  - simple, easy-to-understand format
  - Ideally a binary indicator
  - Usually more complex, allowing analyst to examine suspected attack
  - User interface critical, especially when monitoring many systems
- Be accurate (performance-overhead trade-off !)
  - Minimize false positives, false negatives
  - Minimize time spent verifying/looking for attacks

Models of Intrusion Detection

- Anomaly detection
  - Know: what is usual
  - **Bad:** what is **unusual**
- Misuse detection
  - Know: what is **bad**
  - Good: what is not bad
- Specification-based detection
  - Know: what is good
  - **Bad:** what is **not good**

Anomaly Detection

- Analyzes a set of characteristics of system, and compares their values with expected values; report when computed statistics do not match expected statistics
  - Threshold metrics
  - Statistical moments
  - Markov model

Threshold Metrics

- Counts number of events that occur
  - Between m and n events expected to occur
  - If number falls outside this range, anomalous
- Example
  - Windows: lock user out after k failed sequential login attempts. Range is $(0, k–1)$.
    - k or more failed logins deemed anomalous
- Problems
  - Appropriate threshold may depend on non-obvious factors
    - Typing skill of users
    - If keyboards are US keyboards, and most users are French, typing errors very common

Statistical Moments

- Analyzer computes standard deviation (first two moments), other measures of correlation (higher moments)
  - If measured values fall outside expected interval for particular moments, anomalous
- Problems
  - Profile may evolve over time; solution is to weigh data appropriately or alter rules to take changes into account
  - Assumes behavior of processes and users can be modeled statistically
    - Ideal: matches a known distribution such as Gaussian or normal
    - Otherwise, must use techniques like clustering to determine moments, characteristics that show anomalies, etc.
  - Large overheads in real-time computation

Example: IDES

- Developed at SRI International (Denning's model)
  - Represent users, login session, other entities as ordered sequence of statistics $<q_{0,j}, \ldots, q_{n,j}>$
  - $q_{i,j}$ (statistic i for day j) is count or time interval
  - Weighting favors recent behavior over past behavior
    - $A_{k,j}$ sum of counts making up metric of kth statistic on jth day
    - $q_{k,l+1} = A_{k,l+1} - A_{k,l} + 2^{-rt} q_{k,l}$ where t is number of log entries/total time since start, r factor determined through experience

Markov Model

- Past state affects current transition
- Anomalies based upon *sequences* of events, and not on occurrence of single event
- Problem: need to train system for valid sequences
  - Use known, training data that is not anomalous
  - The more training data, the better the model
  - Training data should cover *all* possible normal uses

System Call Traces

- Define normal behavior in terms of <u>sequences</u> of system calls (*traces*)
- Hofmeyr: experiments show it distinguishes *sendmail* and *lpd* from other programs
- (1) Training trace is:

  `open read write open mmap write fchmod close`
- (2) Produces trace database
- (3) **Later** observe trace:

  `open read read open mmap write fchmod close`
- (4) Deploy distance metric from trace database

Derivation of Statistics

- Problem: assuming Gaussian (or other) distribution of events is not always ok !
- Clustering – try to understand behavior
  - Does not assume *a priori* distribution of data
  - Obtain data, group into subsets (*clusters*) based on some property (*feature*)
  - Analyze the clusters, not individual data points

Example: Clustering (based on "percent")

proc	**user**	**value**	**percent**	**clus#1**	**clus#2**
p_1	matt	359	100%	4	2
p_2	holly	10	3%	1	1
p_3	heidi	263	73%	3	2
p_4	steven	68	19%	1	1
p_5	david	133	37%	2	1
p_6	mike	195	54%	3	2

- Scheme 1: break into 4 groups (>0%, >25%, >50%, >75%); 2, 4 may be anomalous (1 entry each)
- Scheme 2: break into 2 groups (>0%, >50%)

Finding Features

- Which features best show anomalies?
  - CPU use may not, but I/O use may
- Use training data
  - Anomalous data marked
  - **Feature selection** program picks features, clusters that best detects anomalies in data
  - That is why data mining is important ☺

Misuse Modeling

- Determines whether executed sequence is known to violate the policy ("known: what is **bad**; "good"= what is not bad)
  - Known/potential exploits grouped into *rule sets*
  - Match data against rule sets to detect attack
- Cannot detect unknown attacks ;(

- Anomaly detection
  - Know: what is usual
  - **Bad:** what is **unusual**
- Misuse detection
  - Know: what is **bad**
  - Good: what is not bad
- Specification-based detection
  - Know: what is good
  - **Bad:** what is **not good**

Example: Network Flight Recorder

- Built to ease adding new rules by users
- Architecture:
  - Packet sucker: read packets from network
  - Decision engine: “filters” extract information
  - Backend: write data to disk, discard matching packet
    - Query backend allows administrators to extract raw, post-processed data from this file
    - Query backend is separate from NFR process
- Problem: need to know what to look for

N-Code Language for Filters

- Example: ignore traffic **not** for 2 web servers:

```
# list of my web servers
my_web_servers = [ 10.237.100.189 10.237.55.93 ] ;
# we assume all HTTP traffic is on port 80
filter watch tcp ( client, dport:80 )
{
    if (ip.dest != my_web_servers) return;

# now process the packet; we just write out packet info
    record system.time, ip.src, ip.dest to www_list;
}
www_list = recorder("log")
```

Specification Modeling

- Determines whether execution of sequence of instructions violates specification (known: what is good; **Bad:** what is **not good**)
- Only check programs that alter protection state
  - Need to identify and specify expected behavior of these

- Anomaly detection
  - Know: what is usual
  - **Bad:** what is **unusual**
- Misuse detection
  - Know: what is **bad**
  - Good: what is not bad
- Specification-based detection
  - Know: what is good
  - **Bad:** what is **not good**

IDS Architecture

- *Agent* gathers data for analysis
- *Director* analyzes data obtained from the agents according to its internal rules (often centralized)
- *Notifier* gets results from director, takes action
    - notifies security officer
    - reconfigures agents, director to alter collection, analysis methods
    - activates response mechanism
        - thwarts intrusion
        - The Empire Strikes back ?!

Agent

- Runs where information is to be collected
- Messages the director
- May:
  - pre-process information (extract relevant parts)
  - delete un-needed information
- Director may request agent to send additional data

Example

- IDS uses failed login attempts in its analysis
- Agent scans login log every 5 minutes, sends director for each new login attempt:
  - Time of failed login
  - Account name and entered password
- Director requests all records of login (failed or not) for particular user
  - Suspecting a brute-force cracking attempt

Host-Based Agent

- Obtain information from logs
  - May use many logs as sources
  - May be security-related or not
  - May be virtual logs if agent is part of the kernel
    - Very non-portable
- Agent generates its information
  - Scans information needed by IDS, turns it into equivalent of log record
  - Typically, check policy; may be very complex

Network-Based Agents

- Detects network-oriented attacks
  - Denial of service attack introduced by flooding a network
- Monitor traffic for a large number of hosts
- Examine the contents of the traffic itself
- Agent must have same view of traffic as destination
  - TTL tricks, fragmentation may obscure this
- End-to-end encryption defeats content monitoring
  - Can do traffic analysis, though

Network Issues

- Network architecture dictates agent placement
  - Ethernet or broadcast medium: one agent per subnet
  - Point-to-point medium: one agent per connection, or agent at distribution/routing point
- Focus is usually on intruders entering network
  - If few entry points, place network agents behind them
  - Does not help if inside attacks to be monitored

Aggregation of Information

- Agents produce information at multiple layers of abstraction
  - Application-monitoring agents provide one view (usually one line) of an event
  - System-monitoring agents provide a different view (usually many lines) of an event
  - Network-monitoring agents provide yet another view (e.g., many network packets) of an event

Director

- Reduces information from agents
  - Eliminates unnecessary, redundant records
- Analyzes remaining information to determine if attack under way
  - Analysis engine can use a number of techniques, discussed before, to do this
- Usually run on separate system
  - Does not impact performance of monitored systems
  - Rules, profiles not available to ordinary users

Example: multi-agent IDS

- Jane logs in to:
  - daytime: to perform system maintenance
  - at night: to write reports
- One night she begins recompiling the kernel
- Agent #1 reports logins and logouts
- Agent #2 reports commands executed
  - Neither agent spots discrepancy
  - Director correlates log, spots it at once

Adaptive Directors

- Modify profiles, rule sets to adapt their analysis to changes in system
  - Usually use machine learning or planning to determine how to do this
- Example: use neural nets to analyze logs
  - Network adapted to users' behavior over time
  - Used learning techniques to improve classification of events as anomalous
    - Reduced number of false alarms

Notifier

- Accepts information from director
- Takes appropriate action
  - Notify system security officer
  - Respond to attack
- Often GUIs
  - use visualization to convey information

Organization of an NIDS

- Monitoring network traffic for intrusions
  - NSM system
- Combining host and network monitoring
  - DIDS
- Making the agents autonomous
  - AAFID system

Network Security Monitor

- Develops profile of expected usage of network, compares current usage
- Has 3-D matrix for data
  - Axes are source, destination, service
  - Each connection has unique *connection ID*
  - Contents are number of packets sent over that connection for a period of time, and sum of data
  - NSM generates expected connection data
  - Expected data masks data in matrix, and anything left over is reported as an anomaly

Problem

$$\begin{tikzcd}
 & S_1 \arrow[dl, dash] \arrow[dr, dash] & \\
(S_1, D_1) \arrow[d, dash] & & (S_1, D_2) \arrow[d, dash] \\
\begin{array}{c}(S_1, D_1, \text{SMTP}) \\ (S_1, D_1, \text{FTP}) \\ \ldots\end{array} & & \begin{array}{c}(S_1, D_2, \text{SMTP}) \\ (S_1, D_2, \text{FTP}) \\ \ldots\end{array}
\end{tikzcd}$$

- Too much data!
  - Solution: arrange data hierarchically into groups
    - Construct by folding axes of matrix
  - Analyst could expand any group flagged as anomalous

Signatures

- Analyst can write rule to look for specific occurrences in matrix
  - Repeated telnet connections lasting only as long as set-up indicates failed login attempt
- Analyst can write rules to match against network traffic
  - Used to look for excessive logins, attempt to communicate with non-existent host, single host communicating with 15 or more hosts

Distributed IDS

- Neither network-based nor host-based monitoring sufficient to detect some attacks
  - Attacker tries to telnet into system several times using different account names (network-based ID)
  - Attacker tries to log into system using an account without password (host-based ID)
- DIDS uses
  - agents on hosts being monitored
  - network monitor
  - DIDS director uses expert system to analyze data

Attackers Moving in Network

- Intruder breaks into system A as *alice*
- Intruder goes from A to system B, and breaks into B's account *bob*
- Host-based mechanisms cannot correlate these
- DIDS director can see *bob* logged in over *alice*'s connection; expert system infers it is the same user
  - assigns *network identification number* NID (per user)

Handling Distributed Data

- Agent analyzes logs to extract entries of interest
  - uses signatures to look for attacks
    - Summaries sent to Director
  - Other events forwarded directly to director
- DIDS model has agents report:
  - Events (information in log entries)
  - Action (e.g., “create”), domain (e.g., “system”)

Incident Prevention

- Identify attack *before* it completes
- Prevent it from completing
- *Jails* useful for this (remember “Honeypots”)
  - Attacker placed in a confined environment that looks like a full, unrestricted environment
  - Attacker may download files, but gets bogus ones
  - Can imitate a slow system, or an unreliable one
  - Useful to figure out what attacker wants

Somayaji and Forrest (2000)

- "Automated Response using Call Delays"
- IDS records anomalous system calls
  - When number of calls exceeded threshold, system delayed evaluation of system calls (only)
  - If second threshold exceeded, process cannot spawn child
- Performance impact should be minimal on legitimate programs
  - System calls small part of runtime of most programs

Implementation

- Implemented in kernel of Linux system
- Test #1: *ssh* daemon
  - Detected attempt to use global password installed as back door in daemon
  - Connection slowed down significantly
  - When second threshold set to 1, attacker could not obtain login shell
- Test #2: *sendmail* daemon
  - Detected attempts to break in
  - Delays grew quickly to 2 hours per system call

Intrusion Handling

- Restoring system to satisfy site security policy
- Six phases
  - *Preparation* for attack (before attack detected)
  - *Identification* of attack
  - *Containment* of attack (confinement)
  - *Eradication* of attack (stop attack)
  - *Recovery* from attack (restore system to secure state)
  - *Follow-up* to attack (analysis and other actions)

Containment Phase

- Goal: limit access of attacker to system resources
- Two methods
  - Passive monitoring
  - Constraining access

Passive Monitoring

- Records attacker's actions; do *not* interfere with it
  - Idea is to find out what the attacker is after and/or methods the attacker is using
- Problem: attacked system is vulnerable throughout
  - Attacker can also attack other systems
- Example: type of OS can be derived from settings of TCP and IP packets of incoming connections
  - Analyst draws conclusions about source of attack

Constraining Actions

- Reduce protection domain of attacker
- Problem: if defenders do not know what attacker is after, reduced protection domain may <u>not</u> contain what the attacker is after
- e.g., Stoll created document that attacker downloaded (knew what she was after)
  - Download took several hours, during which the underlying phone call was traced to Germany

Deception

- Deception Tool Kit (DTK)
  - Creates false network interface
  - Can present any network configuration to attackers
  - When probed, can return wide range of vulnerabilities
  - Attacker wastes time attacking non-existent systems while analyst collects and analyzes attacks to determine goals and abilities of attacker
  - Experiments show deception is effective response to keep attackers from targeting real systems

Eradication Phase

- Usual approach: deny or remove access to system, or terminate processes involved in attack
- Use wrappers to implement access control
  - Example: wrap system calls
    - On invocation, wrapper takes control of process
    - Wrapper can log call, deny access, do intrusion detection
    - Experiments focusing on intrusion detection used multiple wrappers to terminate suspicious processes
  - Example: network connections
    - Wrapper around servers log, do access control on, incoming connections and control access to Web-based databases

Firewalls

- Mediate access to organization’s network
  - Also mediate access out to the Internet
- Example: Java applets filtered at firewall
  - Use proxy server to rewrite them
    - Change “<applet>” to something else
  - Discard incoming web files with hex sequence CA FE BA BE (Java class files prefix)
  - Block all files with name ending in “.class” or “.zip”
    - Lots of false positives

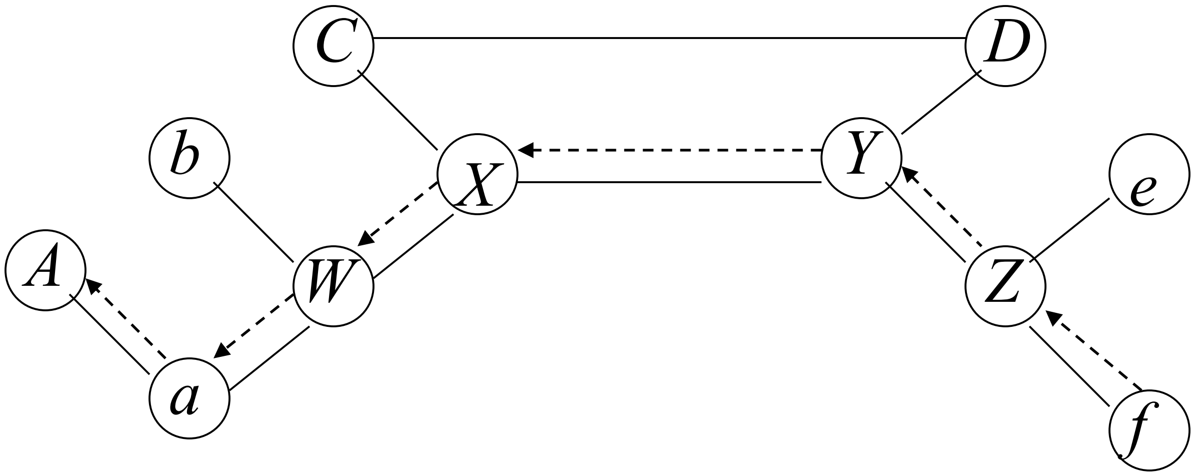
Intrusion Detection and Isolation Protocol

- Coordinates response to attacks
- *Boundary controller* is system that can block connection from entering perimeter
  - Typically firewalls/routers
- *Neighbor* is system directly connected
- *IDIP domain* is set of systems that can send messages to one another without messages passing through boundary controller

Protocol

- IDIP protocol engine monitors connection passing through members of IDIP domains
  - If intrusion observed, engine reports it to neighbors
  - Neighbors propagate information about attack
  - Trace connection, datagrams to boundary controllers
  - Boundary controllers coordinate responses
    - Usually, block attack, notify other controllers to block relevant communications

Example



- *C*, *D*, *W*, *X*, *Y*, *Z* boundary controllers
- *f* launches flooding attack on *A* (saturates links)
- a notifies W which notifies X which notifies Y and C …
- After X suppresses traffic intended for *A*, *W* eliminates supression
- Now *A*, *b*, *a*, and *W* can communicate again
- Problem: paths without IDIP controllers are vulnerable

Follow-Up Phase

- Take action external to system
  - Legal action (issue: span communities)
  - Thumb-printing: trace-back at **connection** level
  - IP header marking: trace-back at **packet** level
  - Counter-attacking ☺

Thumb-printing

- Compares contents of connections to determine which are in a chain of connections
- Characteristic of a good thumbprint
  1. Takes as little space as possible
  2. Low probability of collisions (connections with different contents having same thumbprint)
  3. Minimally affected by common transmission errors
  4. Additive, so two thumbprints over successive intervals can be combined
  5. Cost little to compute, compare

Example: Foxhound

- Thumbprints are linear combinations of character frequencies
  - Experiment used *telnet*, *rlogin* connections
- Computed over normal network traffic
- Control experiment
  - Out of 4000 random pairings, 1 match reported
    - So thumbprints unlikely to match if connections paired randomly
    - Matched pair had identical contents

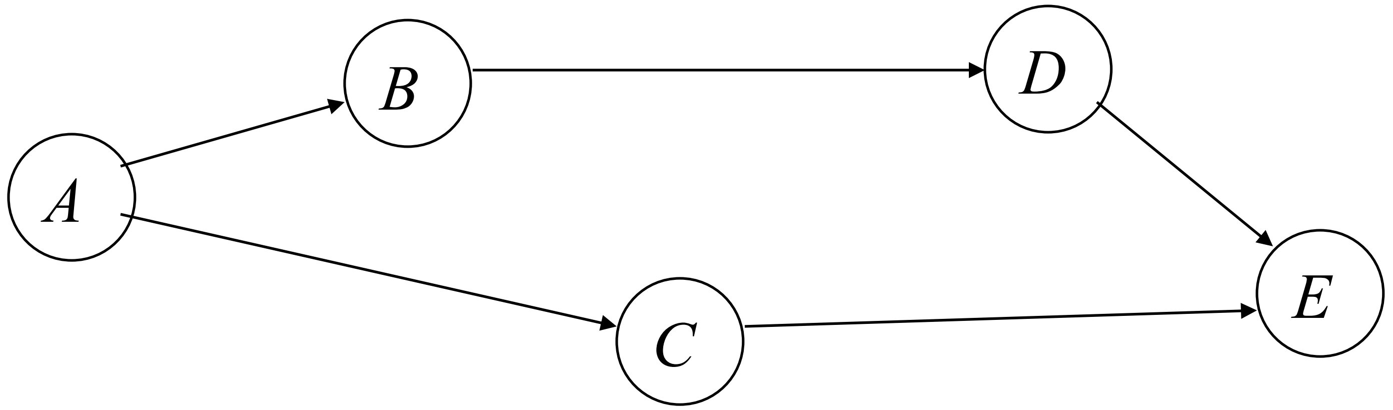
IP Header Marking

- Router places data into each header: path taken
- When do you mark it?
  - Deterministic: always marked
  - Probabilistic: marked with some probability
- How do you mark it?
  - Internal: marking placed in existing header
  - Expansive: header expanded to include extra space

Example 1

- Expand header to have *n* slots for router addresses
- Router address placed in slot *s* with probability *sp*
- Use: suppose SYN flood occurs in network

Use



- *E* SYN flooded; 3150 packets could be result of flood
- E sees packets: 600 (*A*, *B*, *D*); 200 (*A*, *D*); 150 (*B*, *D*); 1500 (*D*); 400 (*A*, *C*); 300 (*C*)
  - *aggregates: A*: 1200; *B*: 750; *C*: 700; *D*: 2450
- Note traffic increases between *B* and *D*: *B* probable culprit

Counter-attacking

- Use legal procedures
  - Collect chain of evidence so legal authorities can establish attack was real
  - Check with lawyers for this
    - Rules of evidence very specific and detailed
    - If you don’t follow them, expect case to be dropped
- Technical attack
  - Goal is to damage attacker seriously enough to stop current attack and deter future attacks

Consequences

1. May harm innocent party
   - Attacker may have broken into source of attack or may be impersonating innocent party
2. May have side effects
   - If counterattack is flooding, may block legitimate use of network
3. Antithetical to shared use of network
   - Counterattack absorbs network resources and makes threats more immediate
4. May be legally actionable

Example: Counter-worm

- Counter-worm given signature of real worm
  - spreads rapidly
  - deletes all occurrences of original worm
- Some issues
  - What if infected system is gathering worms for fun?
  - How do originators of counter-worm know it will not cause problems for any system?
    - And are they legally liable if it does?