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
  - May need to be be real-time, especially when system responds to intrusion
    - Problem: analyzing commands may impact response time of system
  - May suffice to report intrusion occurred a few minutes or hours ago

Goals of IDS

- Present analysis in 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
  - Minimize false positives, false negatives
  - Minimize time spent verifying attacks, looking for them

Models of Intrusion Detection

- Anomaly detection
  - What is usual, is known
  - What is unusual, is bad
- Misuse detection
  - What is bad is known
- Specification-based detection
  - We know what is good
  - What is not good is bad

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 (inclusive) 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

Difficulties

- 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
    - Dvorak vs. non-Dvorak within the US

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
- Potential problem
  - Profile may evolve over time; solution is to weigh data appropriately or alter rules to take changes into account

Example: IDES

- Developed at SRI International to test 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
    - $A_{k,j}$ is 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

Example: Haystack

- Let A_n be nth count or time interval statistic
- Defines bounds T_L and T_U such that 90% of values for A_is lie between T_L and T_U
- Haystack computes A_{n+1}
  - Then checks that $T_L \leq A_{n+1} \leq T_U$
  - If false, anomalous
- Thresholds updated
  - A_i can change rapidly; as long as thresholds met, all is well

Potential Problems

- 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.
- Real-time computation a problem too

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 to establish 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 of system

Example: TIM

- Time-based Inductive Learning
- Sequence of events is *abcdedeabcabc*
- TIM derives following rules:

  R_1: $ab \rightarrow c$ (1.0) R_2: $c \rightarrow d$ (0.5) R_3: $c \rightarrow a$ (0.5)
  R_4: $d \rightarrow e$ (1.0) R_5: $e \rightarrow a$ (0.5) R_6: $e \rightarrow d$ (0.5)
- Seen: *abd*; triggers alert
  - *c* always follows *ab* in rule set
- Seen: *acf*; no alert as multiple events can follow *c*
  - May add rule R_7: $c \rightarrow f$ (0.33); adjust R_2, R_3

Sequences of System Calls

- Forrest: define normal behavior in terms of sequences of system calls (*traces*)
- Experiments show it distinguishes *sendmail* and *lpd* from other programs
- Training trace is:

  `open read write open mmap write fchmod close`
- Produces following database:

Traces

open	read	write	open
open	mmap	write	fchmod
read	write	open	mmap
write	open	mmap	write
write	fchmod	close	
mmap	write	fchmod	close
fchmod	close		
close			

- Trace is:

  `open read read open mmap write fchmod close`

Analysis

- Differs in 5 places:
  - Second read should be write (first open line)
  - Second read should be write (read line)
  - Second open should be write (read line)
  - mmap should be write (read line)
  - write should be mmap (read line)
- 18 possible places of difference
  - Mismatch rate 5/18 ≈ 28%

Derivation of Statistics

- IDES assumes Gaussian distribution of events
  - Experience indicates not right distribution
- Clustering
  - 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

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

- Clus#1: break into 4 groups (25% each); 2, 4 may be anomalous (1 entry each)
- Clus#2: break into 2 groups (50% each)

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 reflects anomalous data

Example

- Analysis of network traffic for features enabling classification as anomalous
- 7 features
  - Index number
  - Length of time of connection
  - Packet count from source to destination
  - Packet count from destination to source
  - Number of data bytes from source to destination
  - Number of data bytes from destination to source
  - Expert system warning of how likely an attack

Feature Selection

- 3 types of algorithms used to select best feature set
  - Backwards sequential search: assume full set, delete features until error rate minimized
    - Best: all features except index (error rate 0.011%)
  - Beam search: order possible clusters from best to worst, then search from best
  - Random sequential search: begin with random feature set, add and delete features
    - Slowest
    - Produced same results as other two

Results

- If following features used:
  - Length of time of connection
  - Number of packets from destination
  - Number of data bytes from source

  classification error less than 0.02%
- Identifying type of connection (like SMTP)
  - Best feature set omitted index, number of data bytes from destination (error rate 0.007%)
  - Other types of connections done similarly, but used different sets

Misuse Modeling

- Determines whether a sequence of instructions being executed is known to violate the site security policy
  - Descriptions of known or potential exploits grouped into *rule sets*
  - IDS matches data against rule sets; on success, potential attack found
- Cannot detect attacks unknown to developers of rule sets
  - No rules to cover them

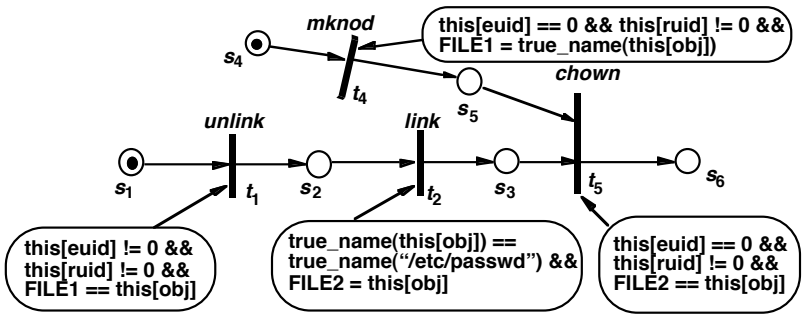
Example: IDIOT

- Event is a single action, or a series of actions resulting in a single record
- Five features of attacks:
  - Existence: attack creates file or other entity
  - Sequence: attack causes several events sequentially
  - Partial order: attack causes 2 or more sequences of events, and events form partial order under temporal relation
  - Duration: something exists for interval of time
  - Interval: events occur exactly *n* units of time apart

IDIOT Representation

- Sequences of events may be interlaced
- Use colored Petri nets to capture this
  - Each signature corresponds to a particular CPA
  - Nodes are tokens; edges, transitions
  - Final state of signature is compromised state
- Example: *mkdir* attack
  - Edges protected by guards (expressions)
  - Tokens move from node to node as guards satisfied

IDIOT Analysis



IDIOT Features

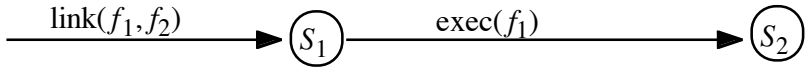
- New signatures can be added dynamically
  - Partially matched signatures need not be cleared and rematched
- Ordering the CPAs allows you to order the checking for attack signatures
  - Useful when you want a priority ordering
  - Can order initial branches of CPA to find sequences known to occur often

Example: STAT

- Analyzes state transitions
  - Need keep only data relevant to security
  - Example: look at process gaining *root* privileges; how did it get them?
- Example: attack giving setuid to *root* shell

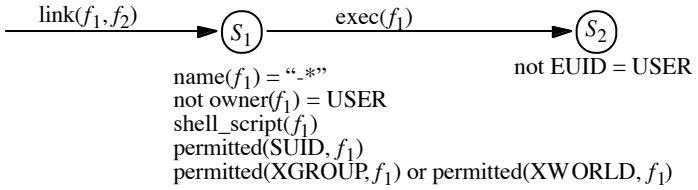
```
ln target ./—s
—s
```


State Transition Diagram



- Now add postconditions for attack under the appropriate state

Final State Diagram



- Conditions met when system enters states s_1 and s_2; USER is effective UID of process
- Note final postcondition is USER is no longer effective UID; usually done with new EUID of 0 (*root*) but works with any EUID

USTAT

- USTAT is prototype STAT system
  - Uses BSM to get system records
  - Preprocessor gets events of interest, maps them into USTAT's internal representation
    - Failed system calls ignored as they do not change state
- Inference engine determines when compromising transition occurs

How Inference Engine Works

- Constructs series of state table entries corresponding to transitions
- Example: rule base has single rule above
  - Initial table has 1 row, 2 columns (corresponding to s1 and s2)
  - Transition moves system into s1
  - Engine adds second row, with "X" in first column as in state s1
  - Transition moves system into s2
  - Rule fires as in compromised transition
    - Does not clear row until conditions of that state false

State Table

		s_1	s_2
	1		
now in s_1	2	X	

Example: NFR

- Built to make adding new rules easily
- Architecture:
  - Packet sucker: read packets from network
  - Decision engine: uses filters to extract information
  - Backend: write data generated by filters to disk
    - Query backend allows administrators to extract raw, postprocessed data from this file
    - Query backend is separate from NFR process

N-Code Language

- Filters written in this language
- Example: ignore all traffic not intended 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
- Only need to check programs that alter protection state of system
- System traces, or sequences of events t_1, … t_i, t_{i+1}, …, are basis of this
  - Event t_i occurs at time $C(t_i)$
  - Events in a system trace are totally ordered

System Traces

- Notion of *subtrace* (subsequence of a trace) allows you to handle threads of a process, process of a system
- Notion of *merge of traces U, V* when trace U and trace V merged into single trace
- *Filter p* maps trace T to subtrace T' such that, for all events $t_i \in T'$, $p(t_i)$ is true

Examples

- Subject S composed of processes p, q, r, with traces T_p, T_q, T_r has $T_s = T_p \oplus T_q \oplus T_r$
- Filtering function: apply to system trace
  - On process, program, host, user as 4-tuple

    `< ANY, emacs, ANY, bishop >`

    lists events with program "emacs", user "bishop"

    `< ANY, ANY, nobhill, ANY >`

    list events on host "nobhill"

Example: Apply to *rdist*

- Ko, Levitt, Ruschitzka defined PE-grammar to describe accepted behavior of program
- *rdist* creates temp file, copies contents into it, changes protection mask, owner of it, copies it into place
  - Attack: during copy, delete temp file and place symbolic link with same name as temp file
  - *rdist* changes mode, ownership to that of program

Relevant Parts of Spec

```
7.   SE: <rdist>
8.   <rdist> -> <valid_op> <rdist> |.
9.   <valid_op> -> open_r_worldread
                 …
     |  chown
        {        if !(Created(F) and M.newownerid = U)
                 then violation(); fi;                 }
        …
10.  END
```

- Chown of symlink violates this rule as M.newownerid ≠ U (owner of file symlink points to is not owner of file *rdist* is distributing)

Comparison and Contrast

- Misuse detection: if all policy rules known, easy to construct rulesets to detect violations
  - Usual case is that much of policy is unspecified, so rulesets describe attacks, and are not complete
- Anomaly detection: detects unusual events, but these are not necessarily security problems
- Specification-based vs. misuse: spec assumes if specifications followed, policy not violated; misuse assumes if policy as embodied in rulesets followed, policy not violated

IDS Architecture

- Basically, a sophisticated audit system
  - *Agent* like logger; it gathers data for analysis
  - *Director* like analyzer; it analyzes data obtained from the agents according to its internal rules
  - *Notifier* obtains results from director, and takes some action
    - May simply notify security officer
    - May reconfigure agents, director to alter collection, analysis methods
    - May activate response mechanism

Agents

- Obtains information and sends to director
- May put information into another form
  - Preprocessing of records to extract relevant parts
- May delete unneeded information
- Director may request agent send other information

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
  - Not 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 (involving many network packets) of an event

Director

- Reduces information from agents
  - Eliminates unnecessary, redundent 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

- Jane logs in to perform system maintenance during the day
- She logs in 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

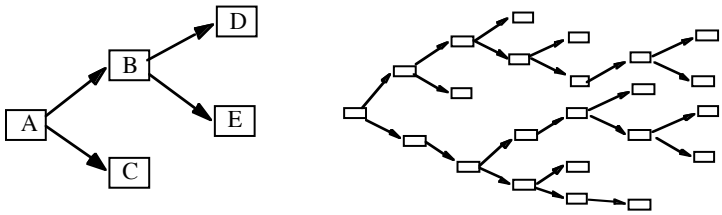
Adaptive Directors

- Modify profiles, rulesets 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
  - Well-designed ones use visualization to convey information

GrIDS GUI



- GrIDS interface showing the progress of a worm as it spreads through network
- Left is early in spread
- Right is later on

Other Examples

- Courtney detected SATAN attacks
  - Added notification to system log
  - Could be configured to send email or paging message to system administrator
- IDIP protocol coordinates IDSes to respond to attack
  - If an IDS detects attack over a network, notifies other IDSes on co-operative firewalls; they can then reject messages from the source

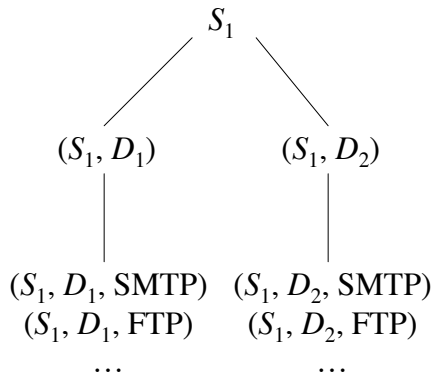
Organization of an IDS

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

Monitoring Networks: NSM

- 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



- 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

Other

- Graphical interface independent of the NSM matrix analyzer
- Detected many attacks
  - But false positives too
- Still in use in some places
  - Signatures have changed, of course
- Also demonstrated intrusion detection on network is feasible
  - Did no content analysis, so would work even with encrypted connections