Lecture 26
December 2, 2024

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

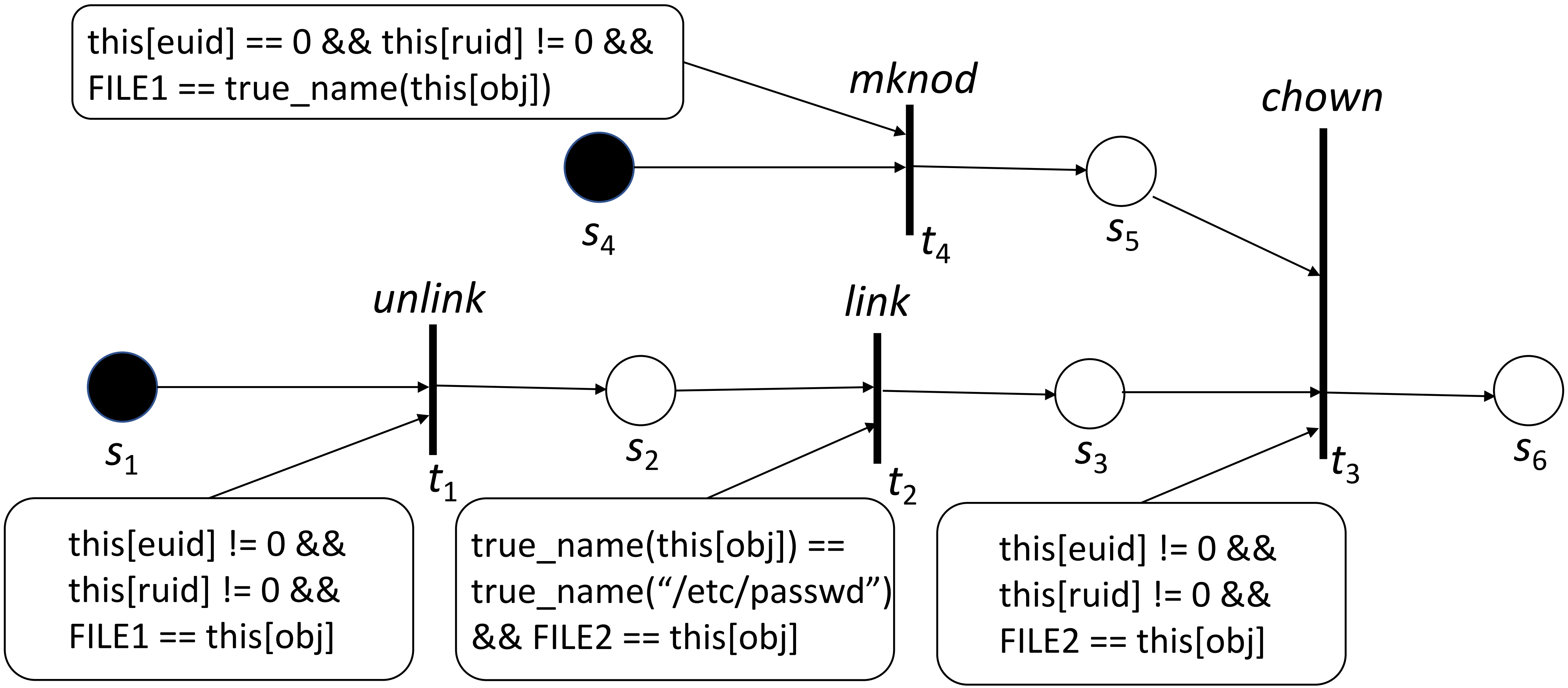
Example: IDIOT

- 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 automata 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

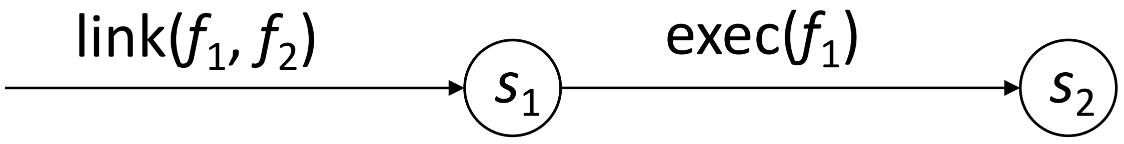
- New signatures can be added dynamically
  - Partially matched signatures need not be cleared and rematched
- Ordering the CPAs allows you to order the checking for attack signatures
  - Useful when you want a priority ordering
  - Can order initial branches of CPA to find sequences known to occur often

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 (here, target is a setuid-to-*root*s shell script)

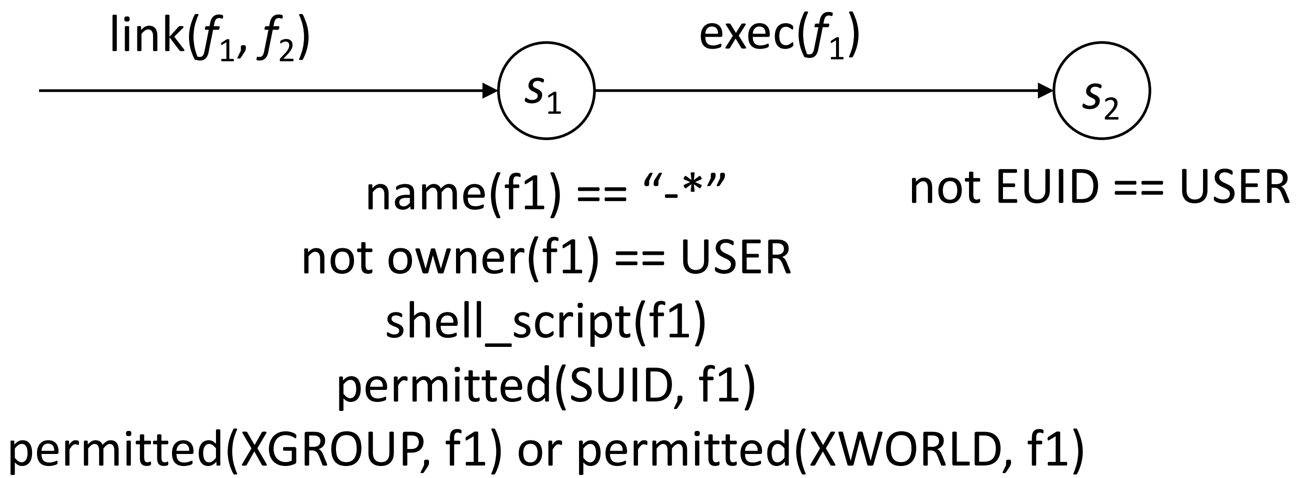
```
ln target ./-s
-s
```

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 that 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 s_1 and s_2)
  - Transition moves system into s_1
  - Engine adds second row, with "X" in first column as in state s_1
  - Transition moves system into s_2
  - Rule fires as in compromised transition
    - Does not clear row until conditions of that state false

State Table

	s_1	s_2
1		
2	X	

now in s_1 (points to row 2)

Example: Bro/Zeke

- Built to make adding new rules easily
- Architecture:
  - Event engine: reads packets from network, processes them, passes results up
    - Uses variety of protocol analyzers to map network flows into events
    - Does *no* evaluation of whether something is good or bad
  - Policy script interpreter evaluates results based on scripts that determine what is bad

Example Script (Detect SSH Servers)

```
# holds a list of SSH servers
global ssh_hosts: set[addr];

event connection_established(c: connection)
{
	local responder = c$id$resp_h;    # address of responder (server)
	local service = c$id$resp_p;      # port on server
	if ( service != 22/tcp )   # SSH port is 22
		return;
	# if you get here, it’s SSH
	if ( responder in ssh_hosts )     # see if we saw this already
		return;
	# we didn’t -- add it to the list and say so
	add ssh_hosts[responder];
	print "New SSH host found", responder;
}
```

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

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

- 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

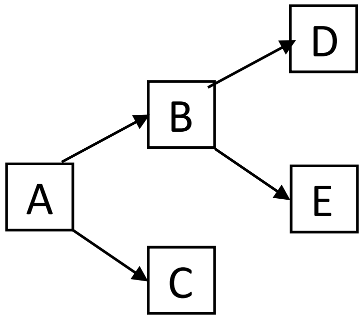
Adaptive Directors

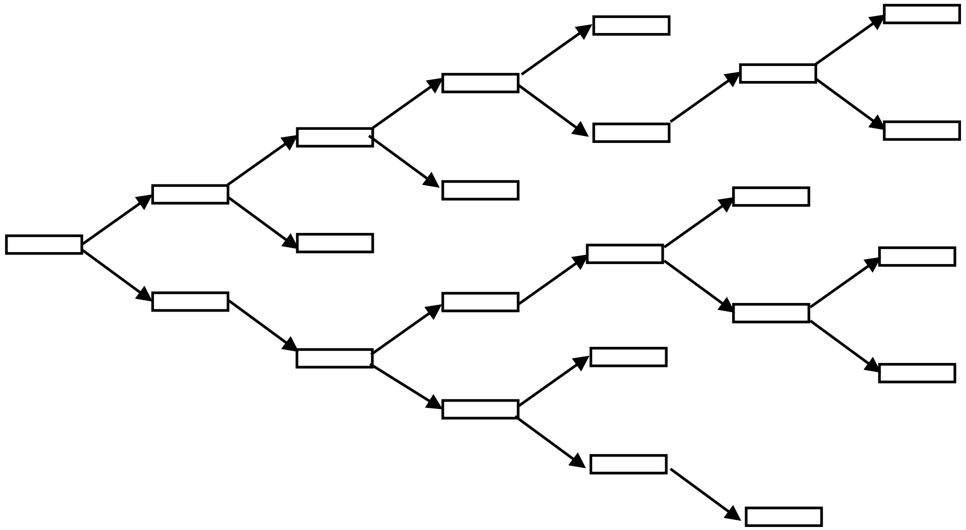
- Modify profiles, rule sets to adapt their analysis to changes in system
  - Usually use machine learning or planning to determine how to do this
- Example: use neural nets to analyze logs
  - Network adapted to users' behavior over time
  - Used learning techniques to improve classification of events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

Notifier

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

- Credit card companies alert customers when fraud is believed to have occurred
  - Configured to send email or SMS message to consumer
- 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

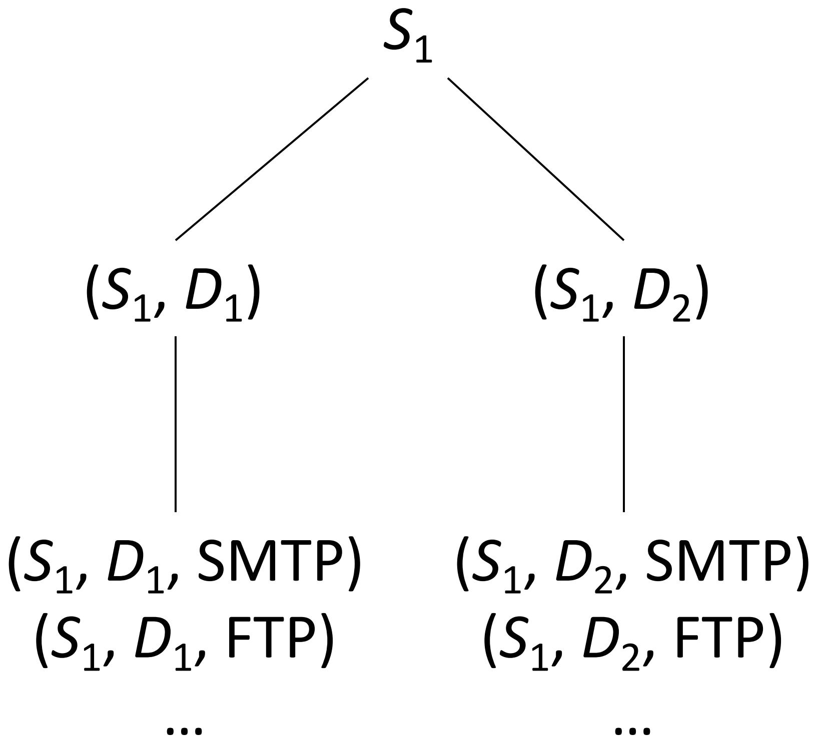
Organization of an IDS

- 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

Combining Sources: DIDS

- Neither network-based nor host-based monitoring sufficient to detect some attacks
  - Attacker tries to telnet into system several times using different account names but is blocked at the router: network-based IDS detects this, but not host-based monitor
  - Attacker tries to log into system using an account without password: host-based IDS detects this, but not network-based monitor
- DIDS uses agents on hosts being monitored, and a 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 could see *bob* logged in over *alice*’s connection; expert system infers they are the same user
  - Assigns *network identification number* NID to this user

Handling Distributed Data

- Agent analyzes logs to extract entries of interest
  - Agent 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, domain

Actions and Domains

- Subjects perform actions
  - session_start, session_end, read, write, execute, terminate, create, delete, move, change_rights, change_user_id
- Domains characterize objects
  - tagged, authentication, audit, network, system, sys_info, user_info, utility, owned, not_owned
  - Objects put into highest domain to which it belongs
    - Tagged, authenticated file is in domain tagged
    - Unowned network object is in domain network

More on Agent Actions

- Entities can be subjects in one view, objects in another
  - Process: subject when changes protection mode of object, object when process is terminated
- Table determines which events sent to DIDS director
  - Based on actions, domains associated with event
  - All NIDS events sent over so director can track view of system
    - Action is *session_start* or *execute*; domain is *network*

Layers of Expert System Model

1. Log records
2. Events (relevant information from log entries)
3. Subject capturing all events associated with a user; NID assigned to this subject
4. Contextual information such as time, proximity to other events
    - Sequence of commands to show who is using the system
    - Series of failed logins follow

Top Layers

5. Network threats (combination of events in context)
   - Abuse (change to protection state)
   - Misuse (violates policy, does not change state)
   - Suspicious act (does not violate policy, but of interest)
6. Score (represents security state of network)
   - Derived from previous layer and from scores associated with rules
     - Analyst can adjust these scores as needed
   - A convenience for user