Lecture 28
December 5, 2025

ECS 235A, Computer and Information Security

Administrative Stuff

- Friday, December 5 is the last day of class
- Please get all homework and extra credit in by then
  - I must submit grades 3 days after the final exam date
- Today's office hour is moved to Friday at 11:00am to 11:50am
  - Like normal Wednesday office hours, it will be on Zoom
  - Please use the usual Zoom link

No work, including homework, extra credit, or projects, can be accepted after the final exam date, December 9, 2025

Anonymity Itself

- Some purposes for anonymity
  - Removes personalities from debate, or with appropriate choice of pseudonym, shape course of debate by implication
  - Prevent retaliation
  - Protect privacy
- Are these benefits or drawbacks?
  - Depends on society, and who is involved

Pseudonyms

- Names of authors of documents used to imply something about the document
- Example: *U.S. Federalist Papers*
  - These argued for the states adopting the U.S. Constitution
  - Real authors were Alexander Hamilton, James Madison, John Jay, all Federalists who wanted the Constitution adopted
  - But using alias “Publius” hid their names
    - Debate could focus on content of the *Federalist Papers*, not the authors or their personalities
    - Roman Publius seen as a model governor, implying the *Papers* represented responsible political philosophy, legislation

Whistleblowers

- Criticism of powerholders often fall into disfavor; powerholders retaliate, but anonymity protects these critics
  - Example: Anonymous sources spoke to Woodward and Bernstein, during U.S. Watergate scandal in 1970s; one important source, called “Deep Throat”, provided guidance that helped uncover a pattern of activity leading to impeachment articles against President Nixon and his resignation
    - “Deep Throat” later revealed as an assistant director of Federal Bureau of Investigation; had this been known, he would have been fired and might have been prosecuted
  - Example: Galileo openly held Copernican theory of the earth circling the sun; brought before the Inquisition and forced to recant

Undesirable Uses

- Attacking systems or people
  - Anonymity hides the perpetrator(s), making responding directly against the attackers difficult
- Example: "Politician U. N. Owen is a creep who stole money from people who needed it for a life-saving operation"
  - And that's it – how do folks determine the motivation of the attacker, to see if there is bias influencing the accuracy of the claim

Undesirable Uses

- Another example: attack on Sony
  - It looked like it came from attackers within the USA
  - It really involved primarily North Korean attackers
- People often say things anonymously that they would not say in person
  - Internet "flame wars"
  - Fear of being sued for slander or libel

Privacy

- Anonymity protects privacy by obstructing amalgamation of individual records
- Important, because amalgamation poses 3 risks:
  - Incorrect conclusions from misinterpreted data
  - Harm from erroneous information
  - Not being let alone
- Also hinders monitoring to deter or prevent crime
- Conclusion: anonymity can be used for good or ill
  - Right to remain anonymous entails responsibility to use that right wisely

Basis of Intrusion Detection Systems

- Hypothesis: exploiting vulnerabilities requires abnormal use of normal commands or instructions
  - Includes deviation from usual actions
  - Includes execution of actions leading to break-ins
  - Includes actions inconsistent with specifications of privileged programs

Models of Intrusion Detection

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

Types of Learning

- *Supervised learning methods*: begin with data that has already been classified, split it into “training data”, “test data”; use first to train classifier, second to see how good the classifier is
- *Unsupervised learning methods*: no pre-classified data, so learn by working on real data; implicit assumption that anomalous data is small part of data
- Measures used to evaluate methods based on:
  - TP: true positives (correctly identify anomalous data)
  - TN: true negatives (correctly identify non-anomalous data)
  - FP: false positives (identify non-anomalous data as anomalous)
  - FN: false negatives (identify anomalous data as non-anomalous)

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

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

Measuring Effectiveness

- *Accuracy*: percentage (or fraction) of events classified correctly
  - $((TP + TN) / (TP + TN + FP + FN)) \times 100\%$
- *Detection rate*: percentage (or fraction) of reported attack events that are real attack events
  - $(TP / (TP + FN)) \times 100\%$
  - Also called the *true positive rate*
- *False alarm rate*: percentage (or fraction) of non-attack events reported as attack events
  - $(FP / (FP + TN)) \times 100\%$
  - Also called the *false positive rate*

Usefulness of Measurement

- Data at installation should be similar to that used to measure effectiveness
- Example: military, academic network traffic different
  - KDD-CUP-99 dataset derived from unclassified and classified network traffic on an Air Force Base
  - Network data captured at Florida Institute of Technology
- FIT data showed anomalies not in KDD-CUP-99
  - FIT data: TCP ACK field nonzero when ACK flag not set
  - KDD-CUP-99 data: HTTP requests all regular, all used GET, version 1.0; in FIT data, HTTP requests showed inconsistencies, some commands not GET, versions 1.0, 1.1
- Conclusion: using KDD-CUP-99 data would show some techniques performing better than they would on the FIT data

Clustering

- 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

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

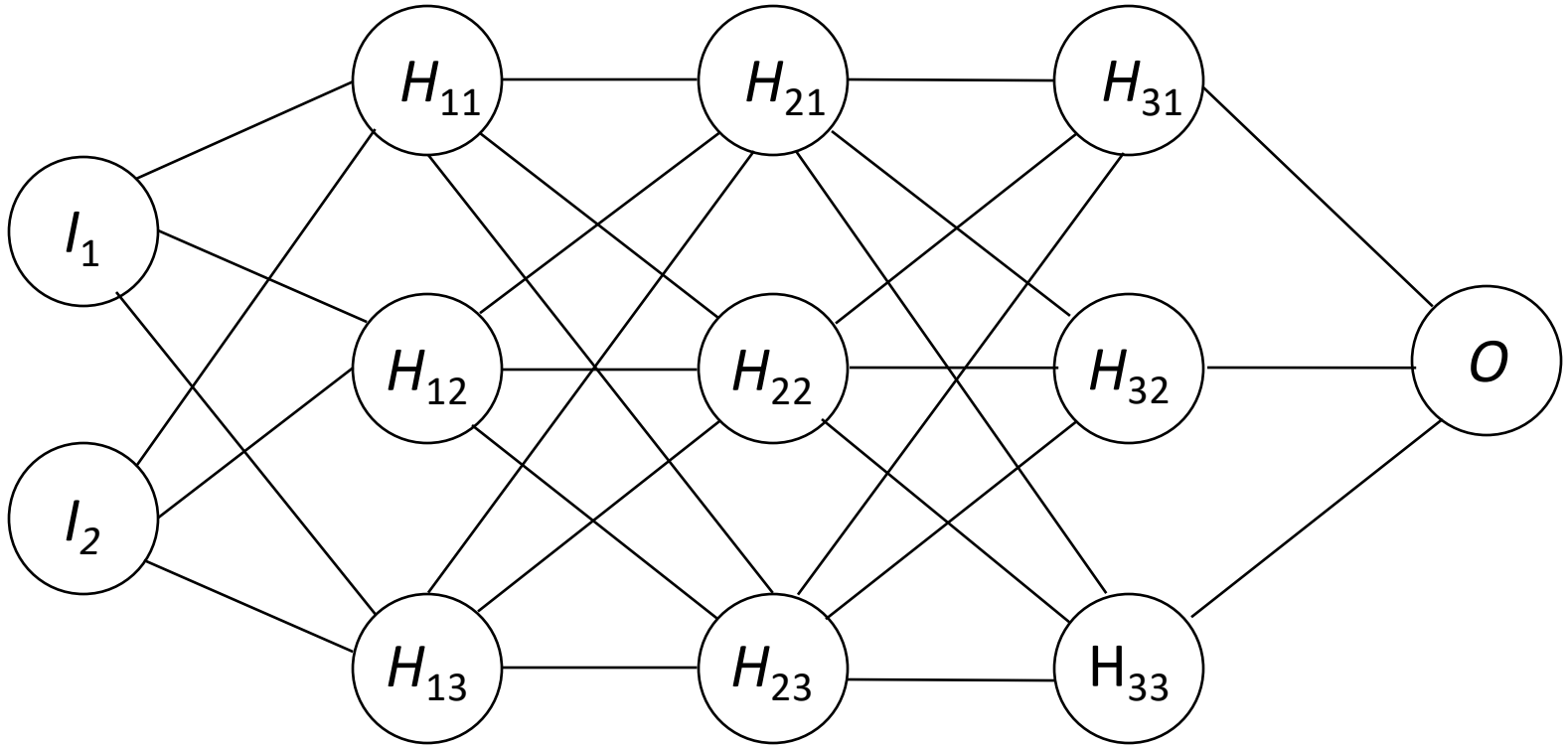
- If following features used:
  - Length of time of connection
  - Number of packets from destination
  - Number of data bytes from source

  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

Neural Nets

- Structure with input layer, output layer, at least 1 layer between them
- Each node (neuron) in layer connected to all nodes in previous, following layer
  - Nodes have an internal function transforming inputs into outputs
  - Each connection has associated weight
- Net given training data as input
  - Compare resulting outputs to ideal outputs
  - Adjust weights according to a function that takes into account the discrepancies between actual, ideal outputs
  - Iterate until actual output matches ideal output
  - Called “back propagation"

Neural Net



Neural net:

- 2 inputs I_1, I_2
- 3 hidden layers of 3 neurons each (H_{ij})
- 1 output O

Note neurons in a layer are not connected

Weights on connections not shown

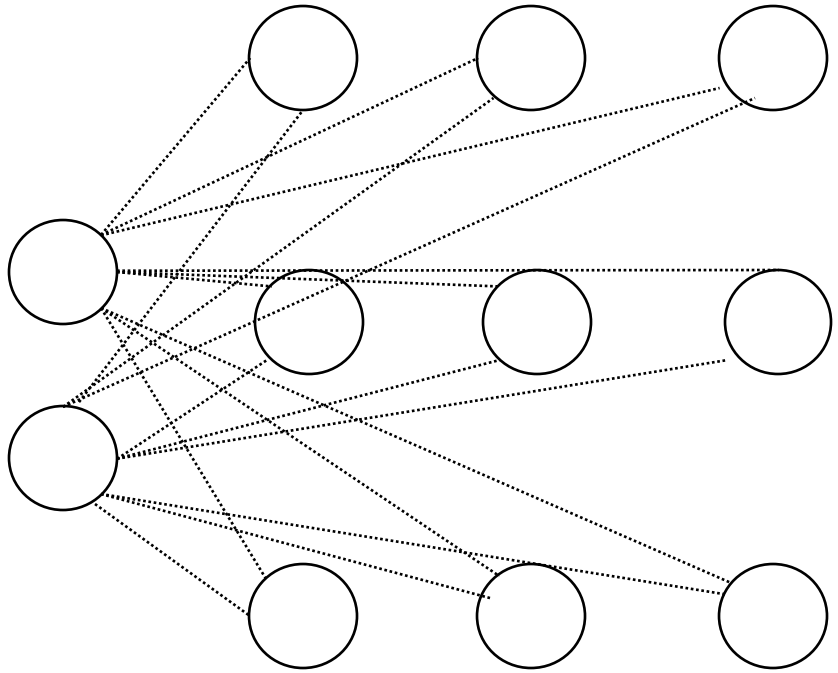
Example

- Neural nets used to analyze KDD-CUP-99 dataset
  - Dataset had 41 features, so neural nets had 41 inputs,, 1 output
  - Split into training data (7312 elements), test data (6980 elements)
- Net 1: 3 hidden layers of 20 neurons each; accuracy, 99.05%
- Net 2: 2 hidden layers of 40 neurons each; accuracy, 99.5%
- Net 3: 2 hidden layers, one of 25 neurons, other of 20 neurons; accuracy, 99%

Self-Organizing Maps

- Unsupervised learning methods that map nonlinear statistical relationships between data points to geometric relationships between points in a 2-dimensional map
- Set of neurons arranged in lattice, each input neuron connected to every neuron in lattice
  - Each lattice neuron given vector of *n* weights, 1 for each of *n* input features
- Input fed to each neuron
  - Neuron with highest output is “winner”; input classified as belonging to that neuron
  - Neuron adjusts weights on incoming edges so weights on edges with weaker values move to edges with stronger signals

Self-Organizing Maps



Self-organizing map with 2 inputs

Each input connected to each neuron in lattice

No lattice neurons connected to any other lattice neuron

Example

- SOM used to examining DNS, HTTP traffic on academic network
- For DNS, lattice of 19 x 25 neurons initialized using 8857 sample DNS connections
  - Tested using set of DNS traffic with known exploit injected, and exploit correctly identified
- For HTTP, lattice of 16 x 27 neurons initialized using 7194 HTTP connections
  - Tested using HTTP traffic with an HTTP tunnel through which *telnet* was run, and the commands setting up tunnel were identified as anomalous

Distance to Neighbor

- Anomalies defined by distance from neighborhood elements
  - Different measured used for this
- Example: *k* nearest neighbor algorithm uses clustering algorithm to partition data into disjoint subsets
  - Then computes upper, lower bounds for distances of elements in each partition
  - Determine which partitions are likely to contain outliers

Example

- Experiment looked at system call data from processes
  - Training data used to construct matrix with rows representing system calls, columns representing processes
  - Elements calculated using weighting taking into account system call frequency over all processes, and compensates for some processes using fewer system calls than others
- New process tested using a similarity function to compute distance to processes
  - *k* closest selected; average distance computed and compared to threshold
- Experiment tested values of *k* between 5, 25
  - *k* = 10, threshold value of 0.72 detected all attacks, false positive rate 0.44%
- Conclusion: this method could detect attacks with acceptably low false positive rate

Support Vector Machines

- Works best when data can be divided into 2 distinct classes
- Dataset has *n* features, so map each data point into *n*-dimensional space, each dimension representing a feature
- SVM is supervised machine learning method that splits dataset in 2
  - Technically, it derives a hyperplane bisecting the space
- Use *kernel function* to derive similarity of points
  - Common one: Gaussian radial base function (RBF), $e^{-\gamma\|x-y\|^2}$ where *x*, *y* are points, γ constant function, $||x–y||2 = \sum_{i=1}^{n}(x_i - y_i)^2$
- New data mapped into space, and so falls into either class

Example

- SVM used to analyze KDD-CUP-99 dataset
  - Dataset had 41 features, so used 41-dimensional space
  - Split into training data (7312 elements), test data (6980 elements)
- Accuracy of 99.5%
- SVM training time 18 seconds; neural net training time 18 minutes

Misuse Detection

- 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

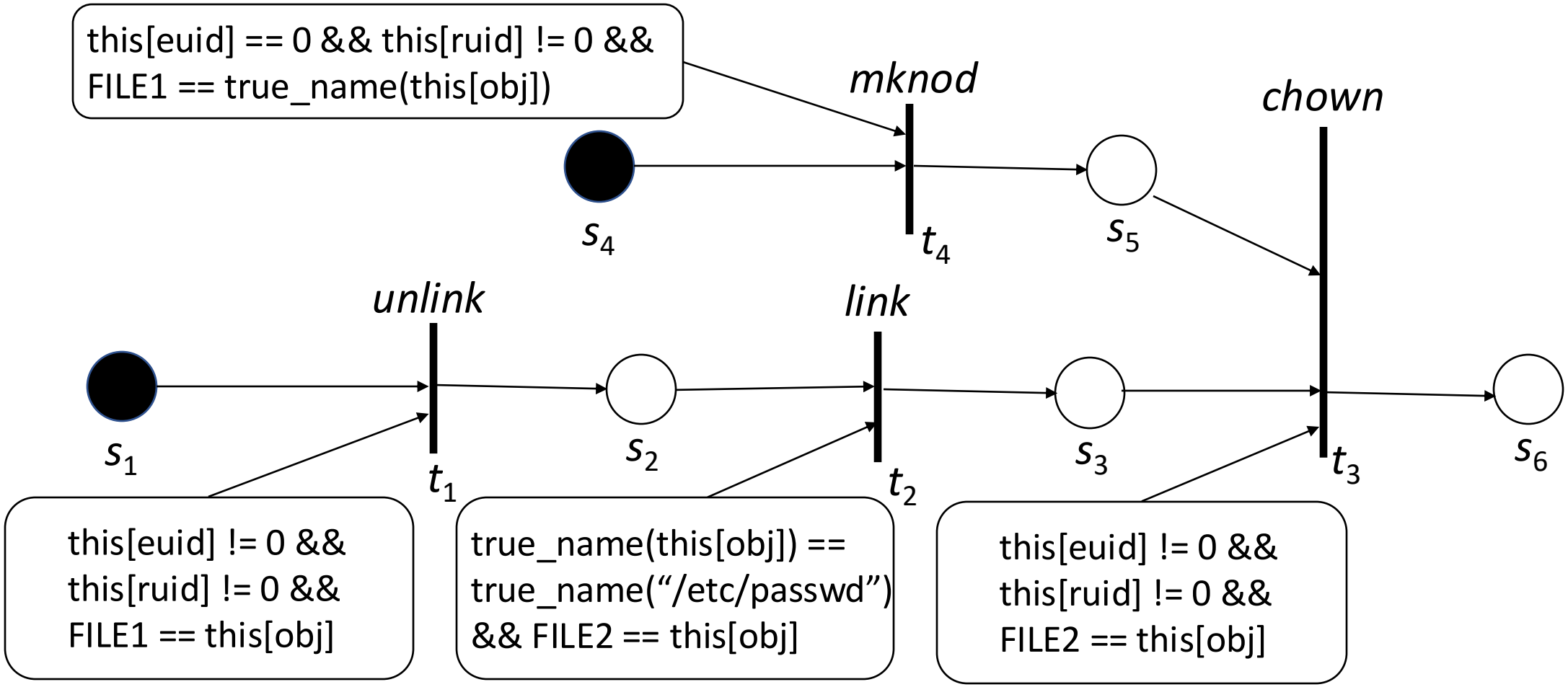
Example: IDIOT

- 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

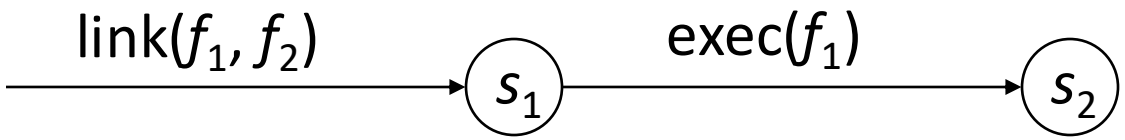
- New signatures can be added dynamically
  - Partially matched signatures need not be cleared and rematched
- Ordering the CPAs allows you to order the checking for attack signatures
  - Useful when you want a priority ordering
  - Can order initial branches of CPA to find sequences known to occur often

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)

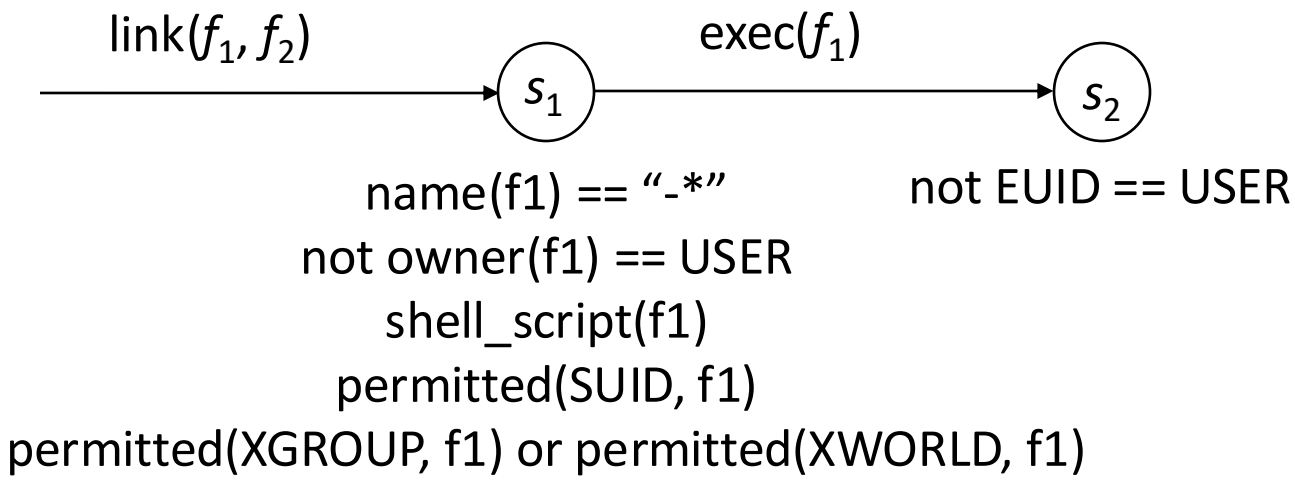
```
ln target ./-s
-s
```

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

now in s_1

	s_1	s_2
1		
2	X	

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