Overview

- Access control lists
- Capability lists
- Locks and keys
- Rings-based access control
- Propagated access control lists

Access Control Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

ACLs:

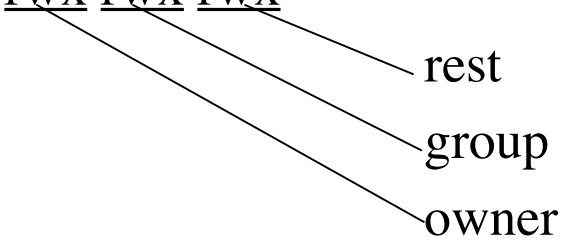
- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }

Default Permissions

- Normal: if not named, *no* rights over file
  - Principle of Fail-Safe Defaults
- If many subjects, may use groups or wildcards in ACL
  - UNICOS: entries are (*user*, *group*, *rights*)
    - If *user* is in *group*, has rights over file
    - '*' is wildcard for *user*, *group*
      - (holly, *, r): holly can read file regardless of her group
      - (*, gleep, w): anyone in group gleep can write file

Abbreviations

- ACLs can be long … so combine users
  - UNIX: 3 classes of users: owner, group, rest
  - rwx rwx rwx



  - Ownership assigned based on creating process
    - Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)

ACLs + Abbreviations

- Augment abbreviated lists with ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies
- Example: IBM AIX
  - Base permissions are abbreviations, extended permissions are ACLs with user, group
  - ACL entries can add rights, but on deny, access is denied

Permissions in IBM AIX

```
attributes:
base permissions
   owner(bishop):     rw-
   group(sys):        r—
   others:            —-
extended permissions enabled
   specify            rw-    u:holly
   permit             -w-    u:heidi, g=sys
   permit             rw-    u:matt
   deny               -w-    u:holly, g=faculty
```

ACL Modification

- Who can do this?
  - Creator is given *own* right that allows this
  - System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
    - Transferring right to another modifies ACL

Privileged Users

- Do ACLs apply to privileged users (*root*)?
  - Solaris: abbreviated lists do not, but full-blown ACL entries do
  - Other vendors: varies

Groups and Wildcards

- Classic form: no; in practice, usually
  - AIX: base perms gave group sys read only

    ```
    permit     -w-    u:heidi, g=sys
    ```

    line adds write permission for heidi when in that group
  - UNICOS:
    - holly : gleep : r
      - user holly in group gleep can read file
    - holly : * : r
      - user holly in any group can read file
    - * : gleep : r
      - any user in group gleep can read file

Conflicts

- Deny access if any entry would deny access
  - AIX: if any entry denies access, *regardless or rights given so far*, access is denied
- Apply first entry matching subject
  - Cisco routers: run packet through access control rules (ACL entries) in order; on a match, stop, and forward the packet; if no matches, deny
    - Note default is deny so honors principle of fail-safe defaults

Handling Default Permissions

- Apply ACL entry, and if none use defaults
  - Cisco router: apply matching access control rule, if any; otherwise, use default rule (deny)
- Augment defaults with those in the appropriate ACL entry
  - AIX: extended permissions augment base permissions

Revocation Question

- How do you remove subject's rights to a file?
  - Owner deletes subject's entries from ACL, or rights from subject's entry in ACL
- What if ownership not involved?
  - Depends on system
  - System R: restore protection state to what it was before right was given
    - May mean deleting descendent rights too …

Windows NT ACLs

- Different sets of rights
  - Basic: read, write, execute, delete, change permission, take ownership
  - Generic: no access, read (read/execute), change (read/write/execute/delete), full control (all), special access (assign any of the basics)
  - Directory: no access, read (read/execute files in directory), list, add, add and read, change (create, add, read, execute, write files; delete subdirectories), full control, special access

Accessing Files

- User not in file's ACL nor in any group named in file's ACL: deny access
- ACL entry denies user access: deny access
- Take union of rights of all ACL entries giving user access: user has this set of rights over file

Capability Lists

- Rows of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

C-Lists:

- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

Semantics

- Like a bus ticket
  - Mere possession indicates rights that subject has over object
  - Object identified by capability (as part of the token)
    - Name may be a reference, location, or something else
  - Architectural construct in capability-based addressing; this just focuses on protection aspects
- Must prevent process from altering capabilities
  - Otherwise subject could change rights encoded in capability or object to which they refer

Implementation

- Tagged architecture
  - Bits protect individual words
    - B5700: tag was 3 bits and indicated how word was to be treated (pointer, type, descriptor, *etc*.)
- Paging/segmentation protections
  - Like tags, but put capabilities in a read-only segment or page
    - CAP system did this
  - Programs must refer to them by pointers
    - Otherwise, program could use a copy of the capability—which it could modify

Implementation (*con't*)

- Cryptography
  - Associate with each capability a cryptographic checksum enciphered using a key known to OS
  - When process presents capability, OS validates checksum
  - Example: Amoeba, a distributed capability-based system
    - Capability is (*name*, *creating_server*, *rights*, *check_field*) and is given to owner of object
    - *check_field* is 48-bit random number; also stored in table corresponding to *creating_server*
    - To validate, system compares *check_field* of capability with that stored in *creating_server* table
    - ***Vulnerable if capability disclosed to another process***

Amplifying

- Allows *temporary* increase of privileges
- Needed for modular programming
  - Module pushes, pops data onto stack

    ```
    module stack … endmodule.
    ```
  - Variable *x* declared of type stack

    ```
    var x: module;
    ```
  - *Only* stack module can alter, read *x*
    - So process doesn't get capability, but needs it when *x* is referenced—a problem!
  - Solution: give process the required capabilities while it is in module

Examples

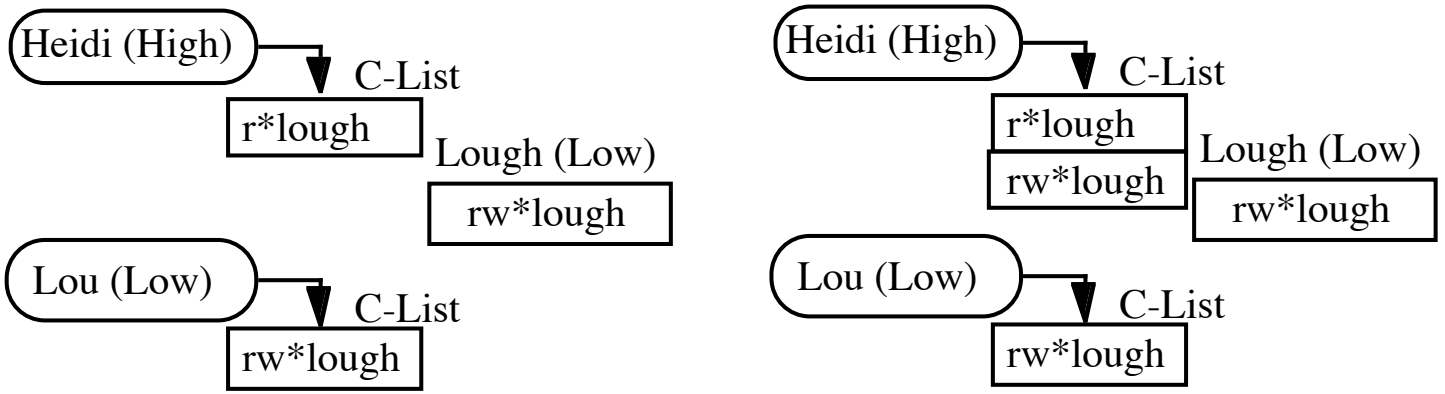
- HYDRA: templates
  - Associated with each procedure, function in module
  - Adds rights to process capability *while the procedure or function is being executed*
  - Rights deleted on exit
- Intel iAPX 432: access descriptors for objects
  - These are really capabilities
  - 1 bit in this controls amplification
  - When ADT constructed, permission bits of type control object set to what procedure needs
  - On call, if amplification bit in this permission is set, the above bits or'ed with rights in access descriptor of object being passed

Revocation

- Scan all C-lists, remove relevant capabilities
  - Far too expensive!
- Use indirection
  - Each object has entry in a global object table
  - Names in capabilities name the entry, not the object
    - To revoke, zap the entry in the table
    - Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object
  - Example: Amoeba: owner requests server change random number in server table
    - All capabilities for that object now invalid

Limits

- Problems if you don't control copying of capabilities



The capability to write file *lough* is Low, and Heidi is High so she reads (copies) the capability; now she can write to a Low file, violating the *-property!

Remedies

- Label capability itself
  - Rights in capability depends on relation between its compartment and that of object to which it refers
    - In example, as as capability copied to High, and High dominates object compartment (Low), write right removed
- Check to see if passing capability violates security properties
  - In example, it does, so copying refused
- Distinguish between "read" and "copy capability"
  - Take-Grant Protection Model does this ("read", "take")

ACLs vs. Capabilities

- Both theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
  - ACLs answer second easily; C-Lists, first
- Suggested that the second question, which in the past has been of most interest, is the reason ACL-based systems more common than capability-based systems
  - As first question becomes more important (in incident response, for example), this may change

Locks and Keys

- Associate information (*lock*) with object, information (*key*) with subject
  - Latter controls what the subject can access and how
  - Subject presents key; if it corresponds to any of the locks on the object, access granted
- This can be dynamic
  - ACLs, C-Lists static and must be manually changed
  - Locks and keys can change based on system constraints, other factors (not necessarily manual)

Cryptographic Implementation

- Enciphering key is lock; deciphering key is key
  - Encipher object o; store $E_k(o)$
  - Use subject's key k' to compute $D_{k'}(E_k(o))$
  - Any of n can access o: store
$$o' = (E_1(o), \ldots, E_n(o))$$
  - Requires consent of all n to access o: store
$$o' = (E_1(E_2(\ldots(E_n(o))\ldots))$$

Example: IBM

- IBM 370: process gets access key; pages get storage key and fetch bit
  - Fetch bit clear: read access only
  - Fetch bit set, access key 0: process can write to (any) page
  - Fetch bit set, access key matches storage key: process can write to page
  - Fetch bit set, access key non-zero and does not match storage key: no access allowed

Example: Cisco Router

- Dynamic access control lists

```
access-list 100 permit tcp any host 10.1.1.1 eq telnet
access-list 100 dynamic test timeout 180 permit ip any host \
   10.1.2.3 time-range my-time
time-range my-time
   periodic weekdays 9:00 to 17:00
line vty 0 2
   login local
   autocommand access-enable host timeout 10
```

- Limits external access to 10.1.2.3 to 9AM–5PM
  - Adds temporary entry for connecting host once user supplies name, password to router
  - Connections good for 180 minutes
    - Drops access control entry after that

Type Checking

- Lock is type, key is operation
  - Example: UNIX system call *write* can't work on directory object but does work on file
  - Example: split I&D space of PDP-11
  - Example: countering buffer overflow attacks on the stack by putting stack on non-executable pages/segments
    - Then code uploaded to buffer won't execute
    - Does not stop other forms of this attack, though …

More Examples

- LOCK system:
  - Compiler produces “data”
  - Trusted process must change this type to “executable” becore program can be executed
- Sidewinder firewall
  - Subjects assigned domain, objects assigned type
    - Example: ingress packets get one type, egress packets another
  - All actions controlled by type, so ingress packets cannot masquerade as egress packets (and vice versa)

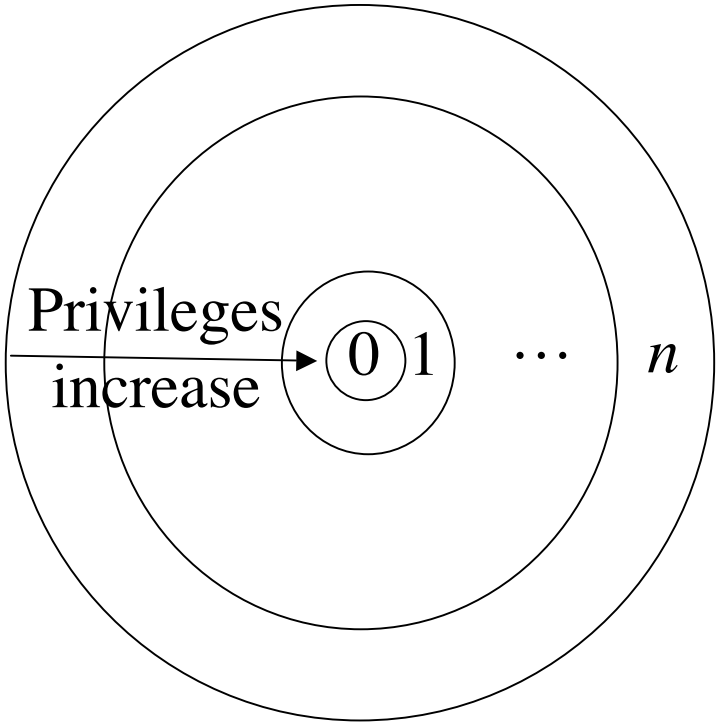
Sharing Secrets

- Implements separation of privilege
- Use (*t*, *n*)-*threshold scheme*
  - Data divided into *n* parts
  - Any *t* parts sufficient to derive original data
- Or-access and and-access can do this
  - Increases the number of representations of data rapidly as *n*, *t* grow
  - Cryptographic approaches more common

Shamir's Scheme

- Goal: use (t, n)-threshold scheme to share cryptographic key encoding data
  - Based on Lagrange polynomials
  - Idea: take polynomial $p(x)$ of degree t–1, set constant term ($p(0)$) to key
  - Compute value of p at n points, *excluding* x = 0
  - By algebra, need values of p at any t distinct points to derive polynomial, and hence constant term (key)

Ring-Based Access Control



- Process (segment) accesses another segment
  - Read
  - Execute
- *Gate* is an entry point for calling segment
- Rights:
  - *r* read
  - *w* write
  - *a* append
  - *e* execute

Reading/Writing/Appending

- Procedure executing in ring r
- Data segment with *access bracket* (a_1, a_2)
- Mandatory access rule
  - $r \leq a_1$ allow access
  - $a_1 < r \leq a_2$ allow r access; not w, a access
  - $a_2 < r$ deny all access

Executing

- Procedure executing in ring r
- Call procedure in segment with *access bracket* (a_1, a_2) and *call bracket* (a_2, a_3)
  - Often written (a_1, a_2, a_3)
- Mandatory access rule
  - $r < a_1$ allow access; ring-crossing fault
  - $a_1 \leq r \leq a_2$ allow access; no ring-crossing fault
  - $a_2 < r \leq a_3$ allow access if through valid gate
  - $a_3 < r$ deny all access

Versions

- Multics
  - 8 rings (from 0 to 7)
- Digital Equipment's VAX
  - 4 levels of privilege: user, monitor, executive, kernel
- Older systems
  - 2 levels of privilege: user, supervisor

PACLs

- Propagated Access Control List
  - Implements ORGON
- Creator kept with PACL, copies
  - Only owner can change PACL
  - Subject reads object: object's PACL associated with subject
  - Subject writes object: subject's PACL associated with object
- Notation: $PACL_s$ means *s* created object; PACL(*e*) is PACL associated with entity *e*

Multiple Creators

- Betty reads Ann's file *dates*

  $PACL(Betty) = PACL_{Betty} \cap PACL(dates)$

  $= PACL_{Betty} \cap PACL_{Ann}$

- Betty creates file *dc*

  $PACL(dc) = PACL_{Betty} \cap PACL_{Ann}$

- $PACL_{Betty}$ allows Char to access objects, but $PACL_{Ann}$ does not; both allow June to access objects
  - June can read *dc*
  - Char cannot read *dc*

Key Points

- Access control mechanisms provide controls for users accessing files
- Many different forms
  - ACLs, capabilities, locks and keys
    - Type checking too
  - Ring-based mechanisms (Mandatory)
  - PACLs (ORCON)

Chapter 25: Intrusion Detection

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

Principles of Intrusion Detection

- Characteristics of systems not under attack
  - User, process actions conform to statistically predictable pattern
  - User, process actions do not include sequences of actions that subvert the security policy
  - Process actions correspond to a set of specifications describing what the processes are allowed to do
- Systems under attack do not meet at least one of these

Example

- Goal: insert a back door into a system
  - Intruder will modify system configuration file or program
  - Requires privilege; attacker enters system as an unprivileged user and must acquire privilege
    - Nonprivileged user may not normally acquire privilege (violates #1)
    - Attacker may break in using sequence of commands that violate security policy (violates #2)
    - Attacker may cause program to act in ways that violate program's specification

Basic Intrusion Detection

- *Attack tool* is automated script designed to violate a security policy
- Example: *rootkit*
  - Includes password sniffer
  - Designed to hide itself using Trojaned versions of various programs (*ps*, *ls*, *find*, *netstat*, etc.)
  - Adds back doors (*login*, *telnetd*, etc.)
  - Has tools to clean up log entries (*zapper, etc.)*

Detection

- *Rootkit* configuration files cause *ls*, *du*, etc. to hide information
  - *ls* lists all files in a directory
    - Except those hidden by configuration file
  - *dirdump* (local program to list directory entries) lists them too
    - Run both and compare counts
    - If they differ, *ls* is doctored
- Other approaches possible

Key Point

- *Rootkit* does *not* alter kernel or file structures to conceal files, processes, and network connections
  - It alters the programs or system calls that *interpret* those structures
  - Find some entry point for interpretation that *rootkit* did not alter
  - The inconsistency is an anomaly (violates #1)

Denning's Model

- 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

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
  - 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

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
    - $A_{k,j}$ 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 $T_L \le A_{n+1} \le T_U$
  - 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: $c \rightarrow e$ (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 *open* (*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

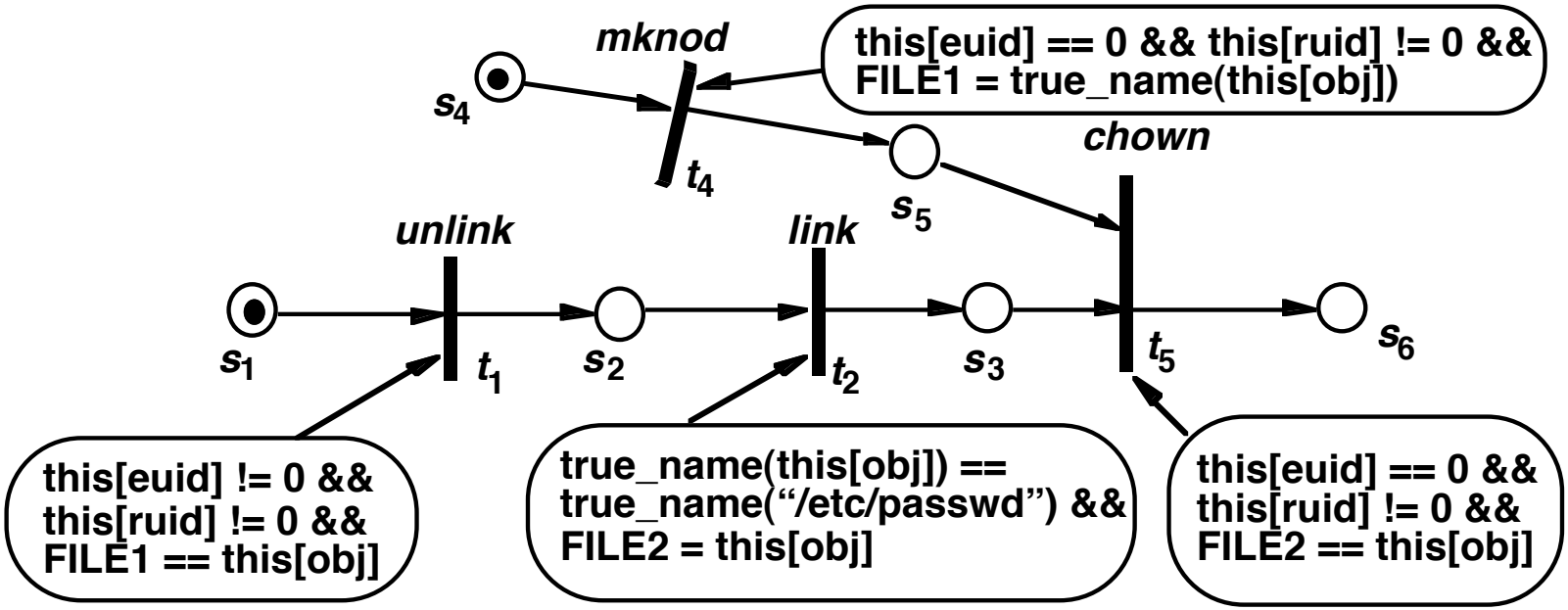
Example: IDIOT

- 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

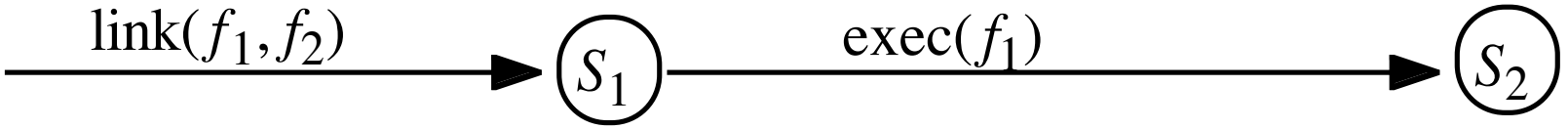
- New signatures can be added dynamically
  - Partially matched signatures need not be cleared and rematched
- Ordering the CPAs allows you to order the checking for attack signatures
  - Useful when you want a priority ordering
  - Can order initial branches of CPA to find sequences known to occur often

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

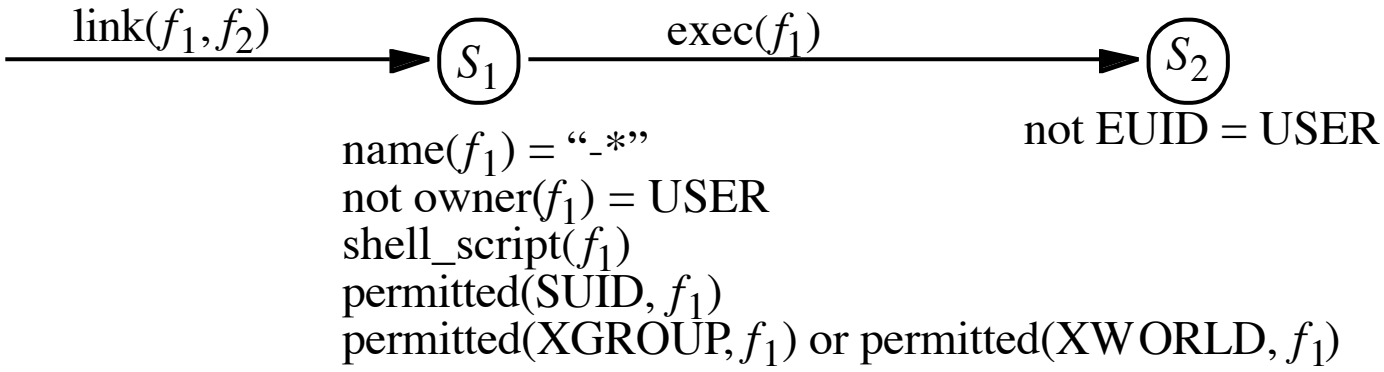
```
ln target ./–s
–s
```

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

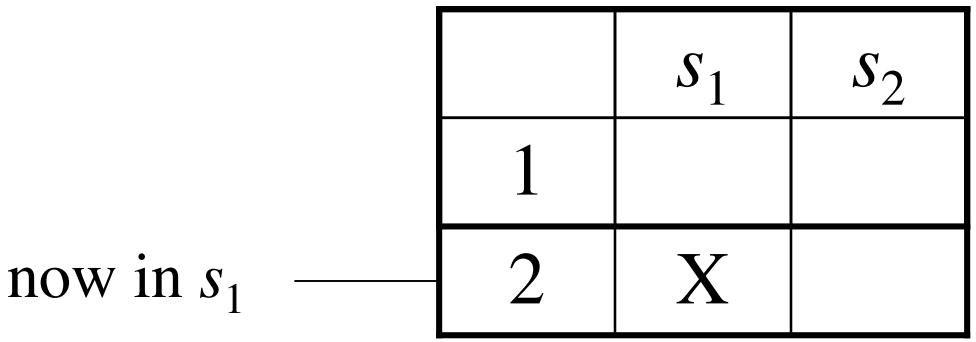
USTAT

- 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 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



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")
```