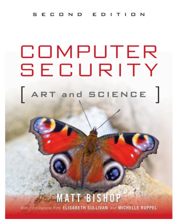
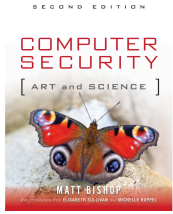


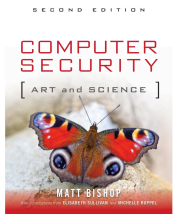
Intrusion Detection

Chapter 25



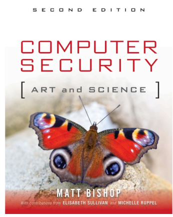
Overview

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



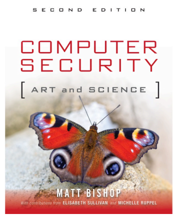
Principles of Intrusion Detection

- 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

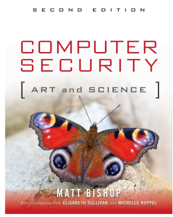


Basic Intrusion Detection

- *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.)*

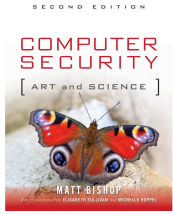
Detection

- *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



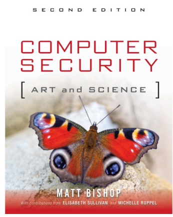
Key Point

- *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)



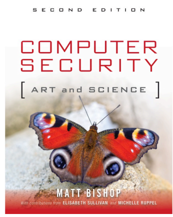
Denning’s Model

- 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



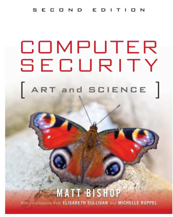
Goals of Intrusion Detection Systems

- 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



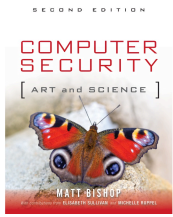
Goals of Intrusion Detection Systems

- 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

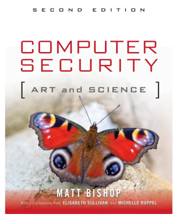
- Analyzes a set of characteristics of system, and compares their values with expected values; report when computed statistics do not match expected statistics
  - Threshold metrics
  - Statistical moments
  - Markov model

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* sequential failed login attempts
    - Range is (0, *k*–1).
    - *k* or more failed logins deemed anomalous

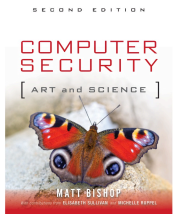
Difficulties

- 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



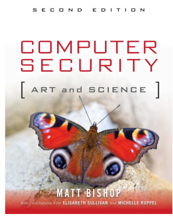
Statistical Moments

- 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



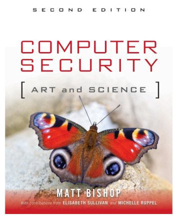
Example: IDES

- 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



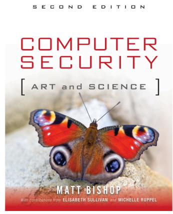
Example: Haystack

- Let A_n be *n*th 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



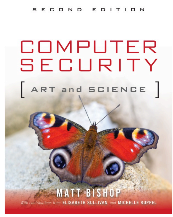
Potential Problems

- 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



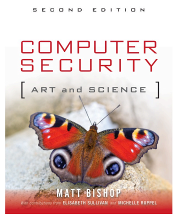
Example: TIM

- 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: $c \rightarrow e$ (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

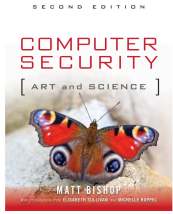
- Forrest: define normal behavior in terms of sequences of system calls (*traces*)
- Experiments show it distinguishes *sendmail* and *lpd* from other programs
- Training trace is:

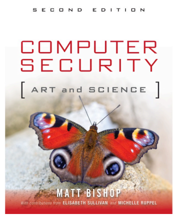
```
open read write open mmap write fchmod close
```

- Produces following database:

Traces

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





Analysis

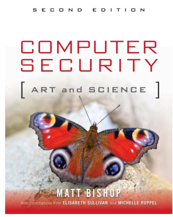
- A later trace is:

  `open read read open mmap write fchmod close`

- Sliding a window comparing this to the 5 sequences above:
  - Sequence beginning with first *open*: item 3 is *read*, should be *write*
  - Sequence beginning with first *read*: item 2 is *read*, should be *write*
  - Sequence beginning with second *read*: item 2 is *open*, should be *write*; item 3 is *mmap*, should be *open*; item 4 is *write*, should be *mmap*
- 18 possible places of difference
  - Mismatch rate 5/18 $\approx$ 28%

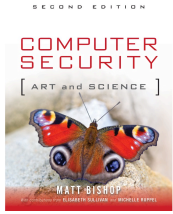
Machine Learning

- These anomaly detection methods all assume some statistical distribution of underlying data
  - IDES assumes Gaussian distribution of events, but experience indicates not right distribution
- Use machine learning techniques to classify data as anomalous
  - Does not assume *a priori* distribution of data



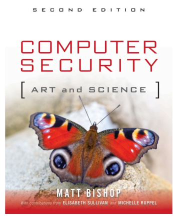
Types of Learning

- *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)



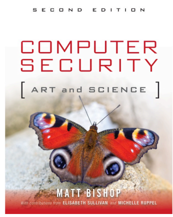
Measuring Effectiveness

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



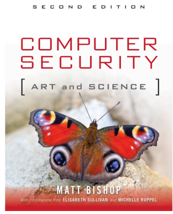
Usefulness of Measurement

- 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

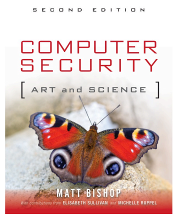
- Clustering
  - Does not assume *a priori* distribution of data
  - Obtain data, group into subsets (*clusters*) based on some property (*feature*)
  - Analyze the clusters, not individual data points



Example: Clustering

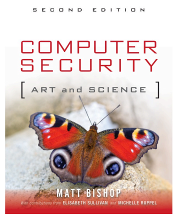
proc	user	value	percent	clus#1	clus#2
p_1	matt	359	100%	4	2
p_2	holly	10	3%	1	1
p_3	heidi	263	73%	3	2
p_4	steven	68	19%	1	1
p_5	david	133	37%	2	1
p_6	mike	195	54%	3	2

- 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

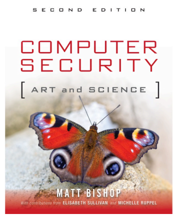


Example

- 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

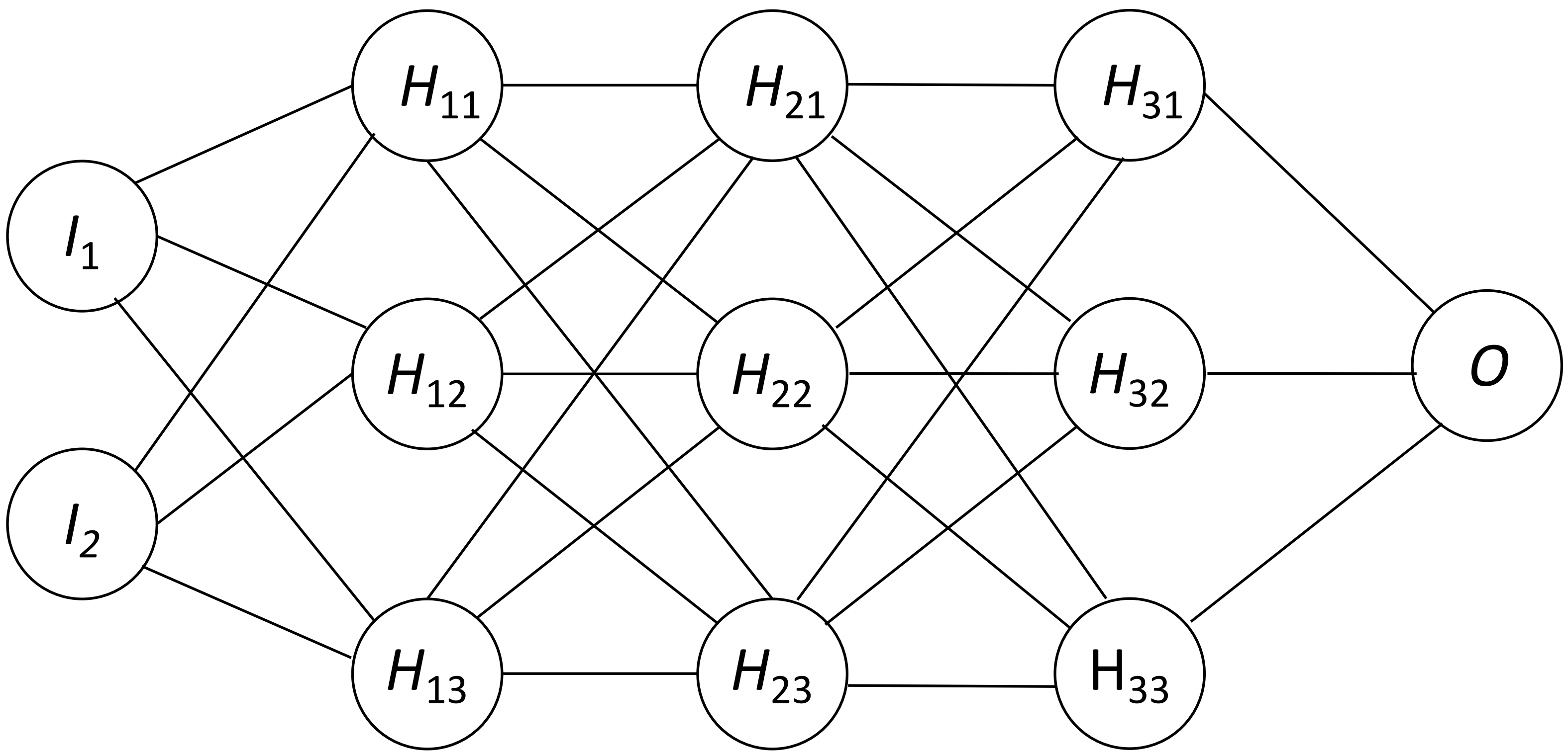
- If following features used:
  - Length of time of connection
  - Number of packets from destination
  - Number of data bytes from source

  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

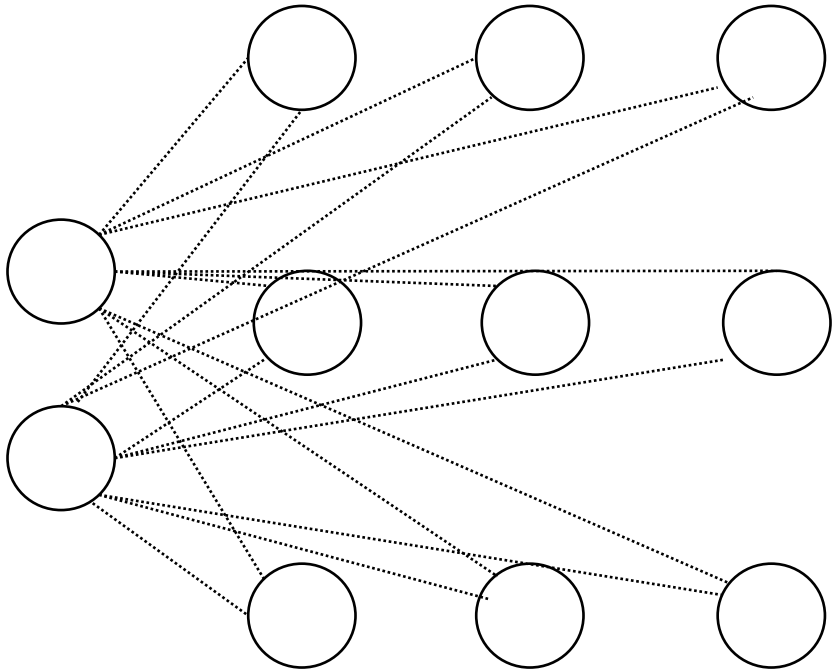
Example

- 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



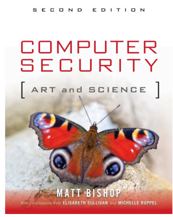
Self-organizing map with 2 inputs

Each input connected to each neuron in lattice

No lattice neurons connected to any other lattice neuron

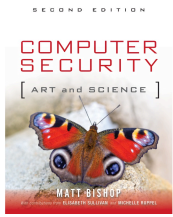
Example

- 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



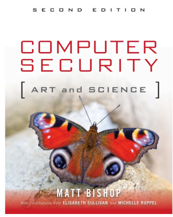
Distance to Neighbor

- 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



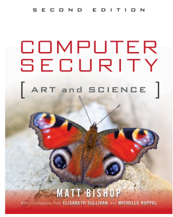
Example

- 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



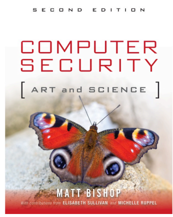
Support Vector Machines

- 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



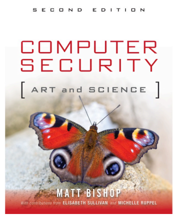
Example

- 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



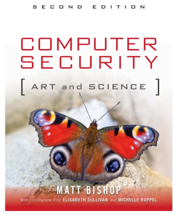
Misuse Modeling

- 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

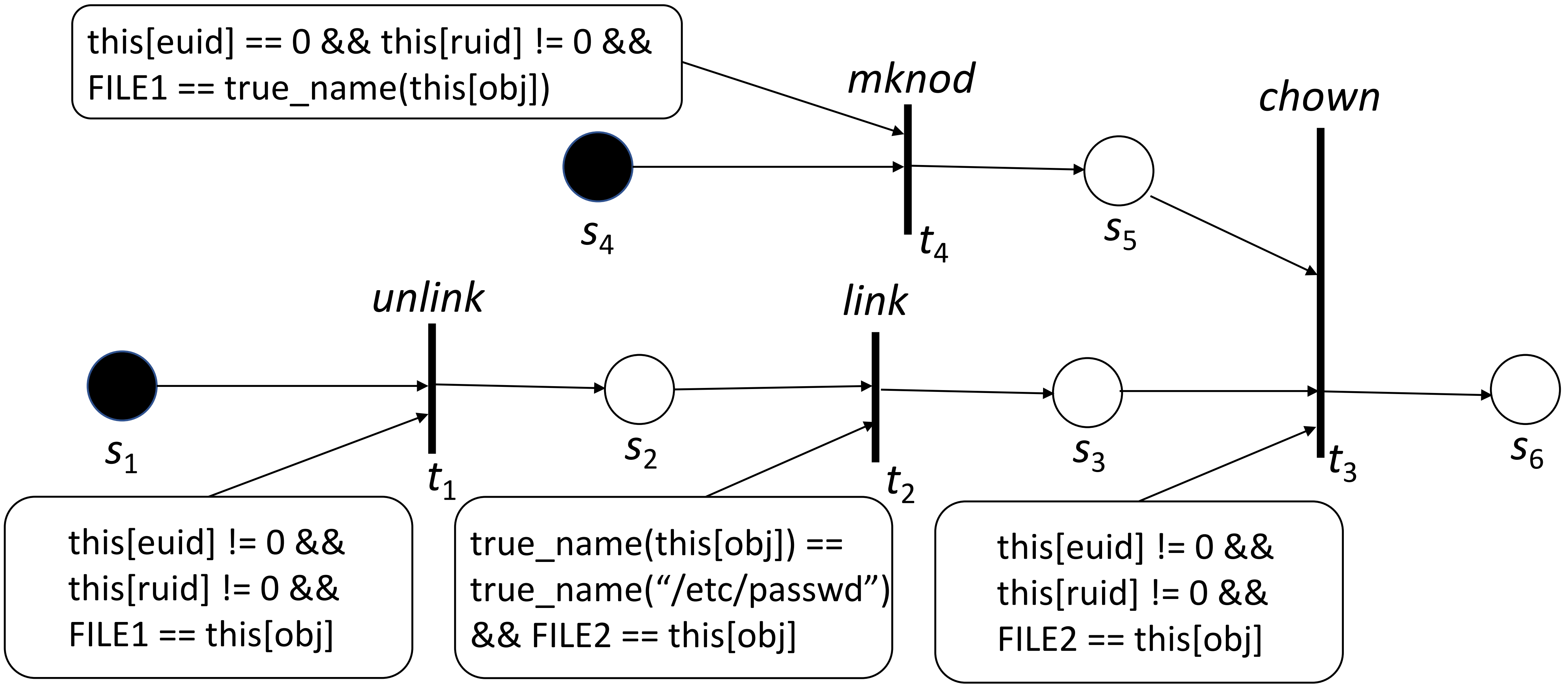
- Event is a single action, or a series of actions resulting in a single record
- Five features of attacks:
  - Existence: attack creates file or other entity
  - Sequence: attack causes several events sequentially
  - Partial order: attack causes 2 or more sequences of events, and events form partial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

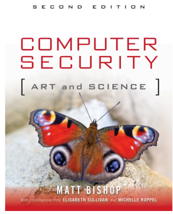


IDIOT Representation

- 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

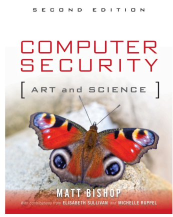
IDIOT Analysis





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

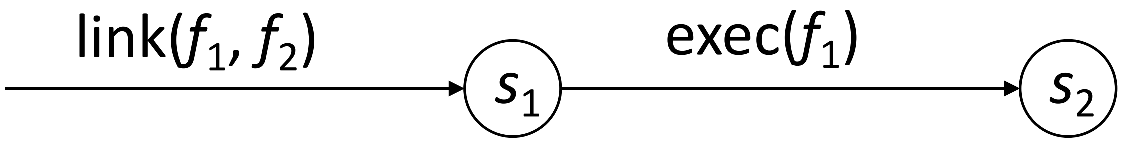


Example: STAT

- 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)

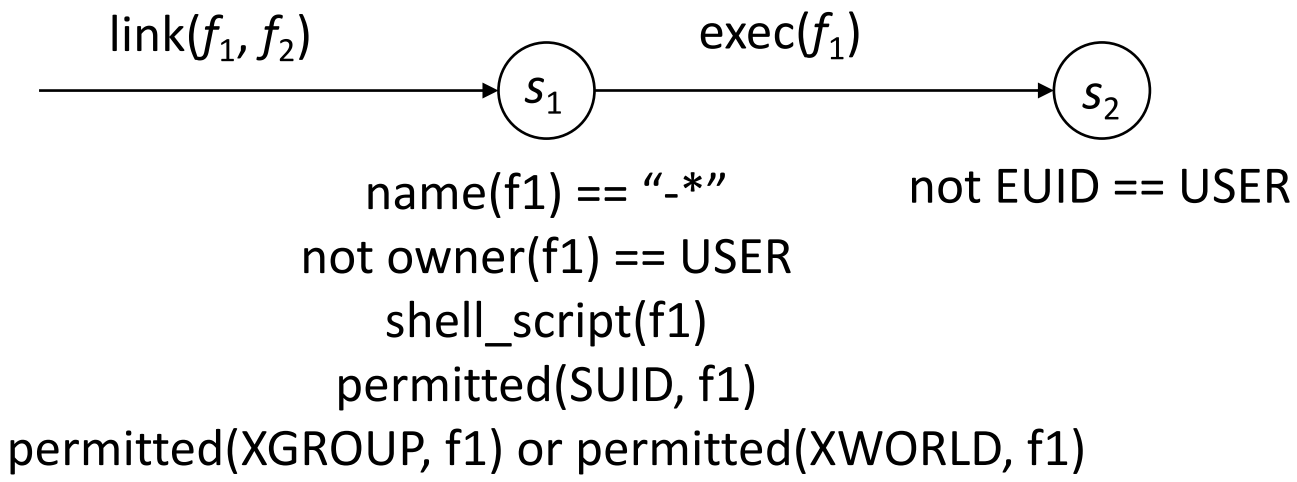
```
ln target ./–s
–s
```

State Transition Diagram

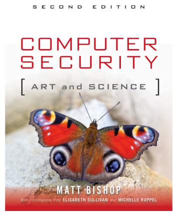


- Now add postconditions for attack under the appropriate state

Final State Diagram

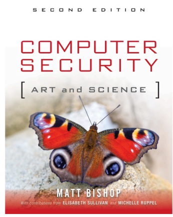


- Conditions met when system enters states s_1 and s_2; USER is effective UID of process
- Note final postcondition is that USER is no longer effective UID; usually done with new EUID of 0 (*root*) but works with any EUID



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



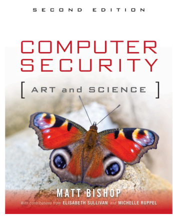
How Inference Engine Works

- 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

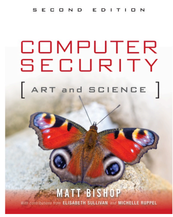
	s_1	s_2
1		
2	X	

now in s_1 → row 2



Example: Bro

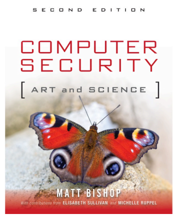
- Built to make adding new rules easily
- Architecture:
  - Event engine: reads packets from network, processes them, passes results up
    - Uses variety of protocol analyzers to map network flows into events
    - Does *no* evaluation of whether something is good or bad
  - Policy script interpreter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



Example Script (Detect SSH Servers)

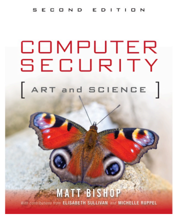
```
# holds a list of SSH servers
global ssh_hosts: set[addr];

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

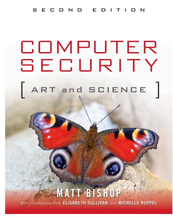
- Subject *S* composed of processes *p*, *q*, *r*, with traces T_p, T_q, T_r has $T_s = T_p \oplus T_q \oplus T_r$
- Filtering function: apply to system trace
  - On process, program, host, user as 4-tuple

    ```
    < ANY, emacs, ANY, bishop >
    ```

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

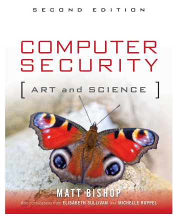
    ```
    < ANY, ANY, nobhill, ANY >
    ```

    list events on host "nobhill"



Example: Apply to *rdist*

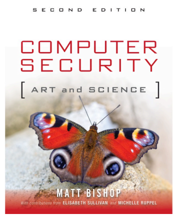
- Ko, Levitt, Ruschitzka defined PE-grammar to describe accepted behavior of program
- *rdist* creates temp file, copies contents into it, changes protection mask, owner of it, copies it into place
  - Attack: during copy, delete temp file and place symbolic link with same name as temp file
  - *rdist*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



Relevant Parts of Spec

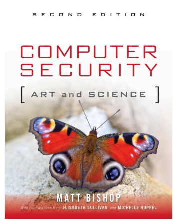
```
SE: <rdist>
<rdist> -> <valid_op> <rdist> |.
<valid_op> -> open_r_worldread
    …
    | chown
      {     if !(Created(F) and M.newownerid = U)
            then violation(); fi;         }
    …
END
```

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



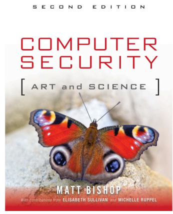
Comparison and Contrast

- 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



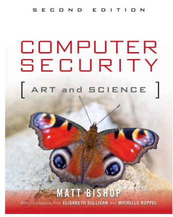
IDS Architecture

- 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

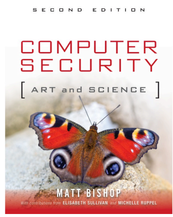


Example

- 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

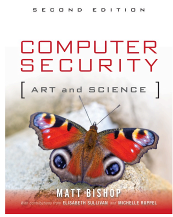
Host-Based Agent

- 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

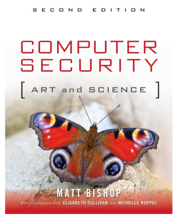


Network Issues

- 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

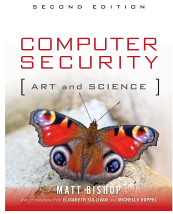


Director

- 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



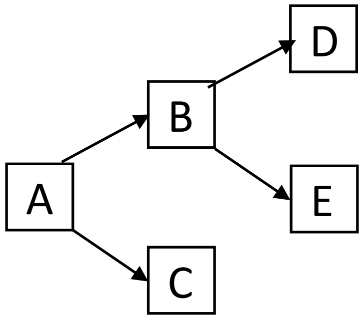
Adaptive Directors

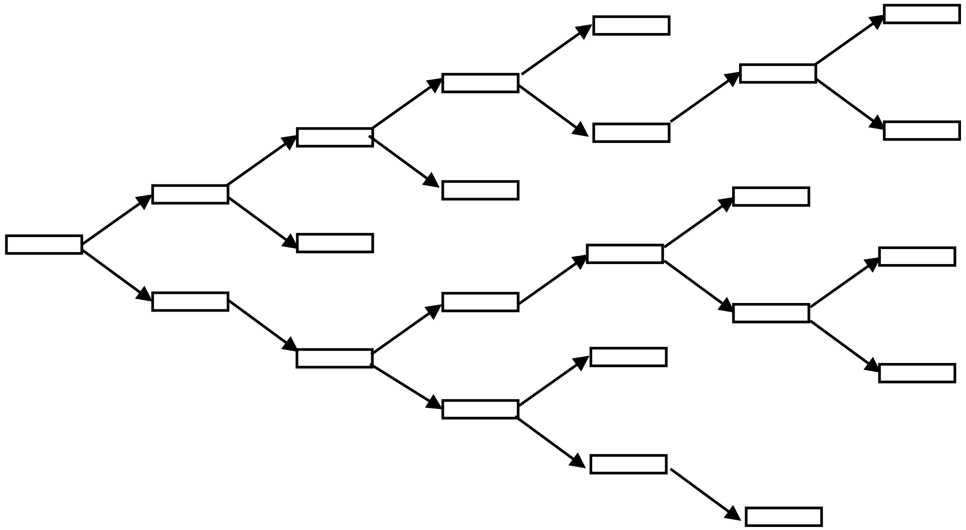
- Modify profiles, rule sets to adapt their analysis to changes in system
  - Usually use machine learning or planning to determine how to do this
- Example: use neural nets to analyze logs
  - Network adapted to users' behavior over time
  - Used learning techniques to improve classification of events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





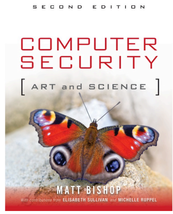
- GrIDS interface showing the progress of a worm as it spreads through network
- Left is early in spread
- Right is later on

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

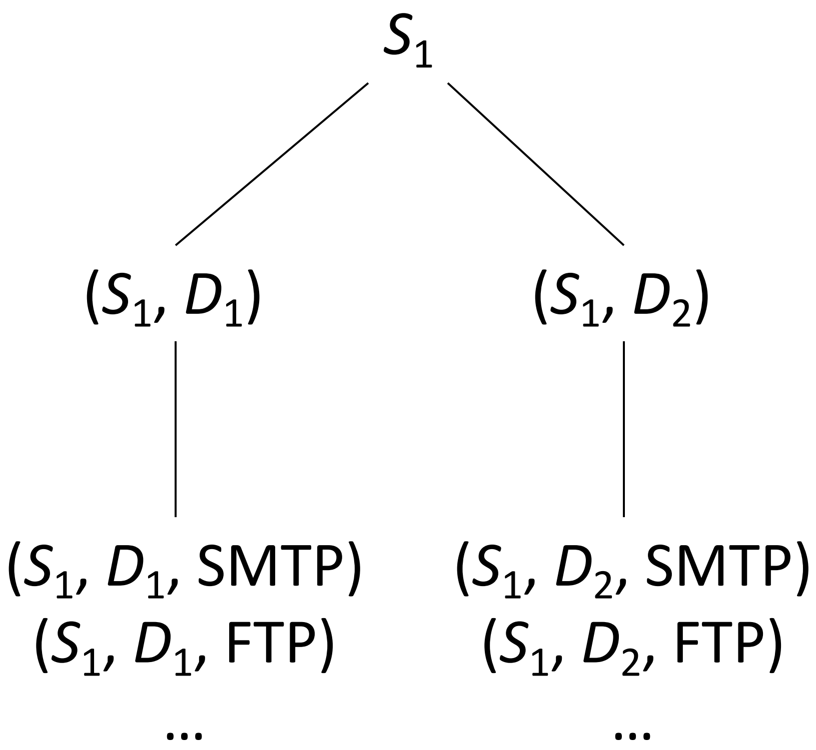
- Monitoring network traffic for intrusions
  - NSM system
- Combining host and network monitoring
  - DIDS
- Making the agents autonomous
  - AAFID system



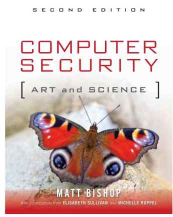
Monitoring Networks: NSM

- 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

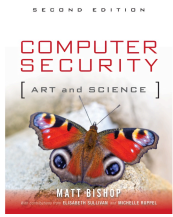


Signatures

- 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

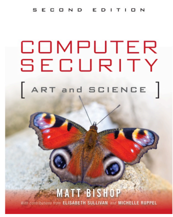
Other

- 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



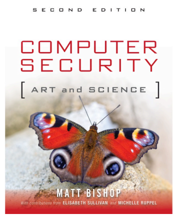
Combining Sources: DIDS

- Neither network-based nor host-based monitoring sufficient to detect some attacks
  - Attacker tries to telnet into system several times using different account names: 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



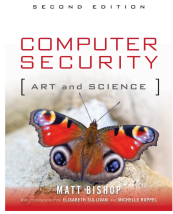
Attackers Moving in Network

- 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



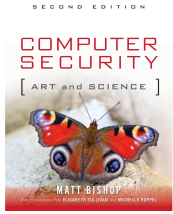
Handling Distributed Data

- 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

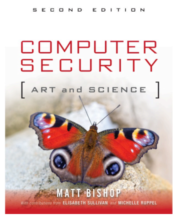


More on Agent Actions

- 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

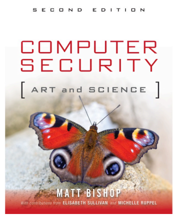
5. Network threats (combination of events in context)
    - Abuse (change to protection state)
    - Misuse (violates policy, does not change state)
    - Suspicious act (does not violate policy, but of interest)
6. Score (represents security state of network)
    - Derived from previous layer and from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

Autonomous Agents: AAFID

- Distribute director among agents
- *Autonomous agent* is process that can act independently of the system of which it is part
- Autonomous agent performs one particular monitoring function
  - Has its own internal model
  - Communicates with other agents
  - Agents jointly decide if these constitute a reportable intrusion

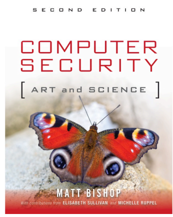
Advantages

- No single point of failure
  - *All* agents can act as director
  - In effect, director distributed over all agents
- Compromise of one agent does not affect others
- Agent monitors one resource
  - Small and simple
- Agents can migrate if needed
- Approach appears to be scalable to large networks



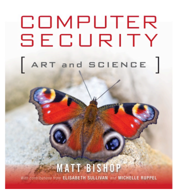
Disadvantages

- Communications overhead higher, more scattered than for single director
  - Securing these can be very hard and expensive
- As agent monitors one resource, need many agents to monitor multiple resources
- Distributed computation involved in detecting intrusions
  - This computation also must be secured



Example: AAFID

- Host has set of agents and transceiver
  - Transceiver controls agent execution, collates information, forwards it to monitor (on local or remote system)
- Filters provide access to monitored resources
  - Use this approach to avoid duplication of work and system dependence
  - Agents subscribe to filters by specifying records needed
  - Multiple agents may subscribe to single filter

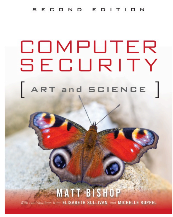


Transceivers and Monitors

- Transceivers collect data from agents
  - Forward it to other agents or monitors
  - Can terminate, start agents on local system
    - Example: System begins to accept TCP connections, so transceiver turns on agent to monitor SMTP
- Monitors accept data from transceivers
  - Can communicate with transceivers, other monitors
    - Send commands to transceiver
  - Perform high level correlation for multiple hosts
  - If multiple monitors interact with transceiver, AAFID must ensure transceiver receives consistent commands

Other

- User interface interacts with monitors
  - Could be graphical or textual
- Prototype implemented in PERL for Linux and Solaris
  - Proof of concept
  - Performance loss acceptable



Key Points

- Intrusion detection is a form of auditing
- Anomaly detection looks for unexpected events
- Misuse detection looks for what is known to be bad
- Specification-based detection looks for what is known not to be good
- Intrusion detection is used for hoist-based monitoring, network monitoring, or combination of these