CS259D: Data Mining for CyberSecurity
Administrativia

- HW due tonight
- Time for guest lecture on Friday
- Projects
Web security

- Web servers accessible by outside world
- Web apps developed with security as an afterthought
- Example: Target breach
## Popularity of web-related attacks

<table>
<thead>
<tr>
<th>Year</th>
<th>Total</th>
<th>Web-related</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>809</td>
<td>109</td>
<td>13.5%</td>
</tr>
<tr>
<td>2000</td>
<td>800</td>
<td>186</td>
<td>23.3%</td>
</tr>
<tr>
<td>2001</td>
<td>588</td>
<td>120</td>
<td>20.4%</td>
</tr>
<tr>
<td>2002</td>
<td>376</td>
<td>100</td>
<td>26.6%</td>
</tr>
<tr>
<td>Total</td>
<td>2573</td>
<td>515</td>
<td>20.0%</td>
</tr>
</tbody>
</table>
Web-related attack detection

- **Misuse-based**
  - Example: Snort
    - 1037 out of 2464 signatures
  - Hard to keep up-to-date
    - Time-intensive, error-prone, requires significant security expertise
  - Challenge with apps developed in-house

- **Anomaly-based**
  - Applicable to custom-developed web apps
  - Support detection of new attacks
Anomaly detection method

- **Input:** web server log files
  - Common Log Format (CLF)
- **Analysis:** build profiles for apps & active docs
  - Lower error rates than generic profiles
  - Use multiple models
    - Reduce vulnerability to mimicry attacks
- **Output:** anomaly score for each web request
Data: Model

- An ordered set $U = \{u_1, u_2, \ldots, u_m\}$ of URIs
  - Extract from successful GET requests
    - $200 \leq \text{return-code} < 300$
- Components of $u_i$
  - Path to desired resource: $\text{path}_i$
  - Optional path information: $\text{pinfo}_i$
  - Optional query string: $q$
    - Following a $\text{?}$ Character
    - Passing parameters to referenced resource
    - Attributes and values: $q = (a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n)$
    - $S_q = \{a_1, a_2, \ldots, a_n\}$
- URIs without query strings not included in $U$
- $U_r$: subset of $U$ with resource path $r$
  - Partition $U$
  - Anomaly detection run independently on each $U_r$
Data: Example record

- Path: /scripts/access.pl
- q: user=johndoe&cred=admin
- a₁ = user, v₁ = johndoe
- a₂ = cred, v₂ = admin
- S₉ = {user, cred}
Anomaly score

• Each model
  ◦ returns probability $p$ of normalcy
  ◦ Has an associated weight $w$
    • default value = 1

• Anomaly score =

$$\sum_m w_m \times (1 - p_m)$$
Attribute length

- Fixed size tokens
  - Session identifiers
- Short input strings
  - Fields in an HTML form
- Example:
  - Buffer overflow: shell code & padding
    - Several hundred bytes
  - XSS
Attribute length

- **Learning:** Estimate mean $\mu$ and variance $\sigma^2$ of lengths in training data

- **Chebyshev inequality:**
  $$p(|x - \mu| > t) < \frac{\sigma^2}{t^2}$$

- **Detection:**
  - strings with length larger than mean
    - If length < mean, $p = 1$
    - Padding not effective

  $$p = p(|x - \mu| > |l - \mu|) < \frac{\sigma^2}{|l - \mu|^2}$$
Observations about attributes:
- Regular structure
- Mostly human readable
- Almost always contain only printable characters

Character distribution: sorted relative frequencies
- Example: passwd => 0.33, 0.17, 0.17, 0.17, 0.17, 0,…, 0
- Fall smoothly for human-readable tokens
- Fall quickly for malicious input

Example:
- Buffer overflow: needs to send binary data & padding
- Directory traversal exploit: many dots in attribute value
Attribute character distribution

- **Learning:**
  - character distribution of each observed attribute is stored
  - Average of all character distributions computed

- **Detection:**
  - Variant of the Pearson $\chi^2$-test
  - Bins: $\{[0], [1, 3], [4, 6], [7, 11], [12, 15], [16, 255]\}$
  - For each query attribute:
    - Compute character distribution
    - Observed values $O_i$: Aggregate over bins
    - Expected values $E_i$: Learned character distribution attribute length
    - Compute: $\chi^2 = \sum_{i=0}^{5} \frac{(O_i - E_i)^2}{E_i}$
    - Read corresponding probability
Structural inference

- Simple manifestations of an exploit
  - Unusually long parameters
  - Parameters containing repetitions of non-printable characters
- Evasion
  - Replace non-printable characters by groups of printable characters
- Parameter structure: regular grammar describing all of its legitimate values
- Detect exploits requiring different parameter structure
  - Examples: Buffer overflow, directory traversal, XSS
Structural inference

- Learning: Markov model/Non-deterministic finite automaton (NFA)
  - \( P_S(o) \): probability of emitting symbol \( o \) at state \( S \)
  - \( P(t) \): probability of transition \( t \)
  - Output: paths from Start state to Terminal state

- For a word \( w = (o_1, o_2, \ldots, o_k) \)

\[
p(w) = p(o_1, o_2, \ldots, o_k) = \sum_{p: \text{paths } S_i \in p} \prod_{i} p_{S_i}(o_i) \times p(t_i)
\]
Structural inference

- \( w = ab \)
- \( P(w) = 0.3 \times 0.5 \times 0.2 \times 0.5 \times 0.4 + 0.7 \times 1.0 \times 1.0 \times 1.0 \times 1.0 = 0.706 \)
Structural inference

- Goal: Find a model with highest likelihood given training examples
- Bayesian model induction:
  \[ P(\text{model} | \text{training data}) = p(\text{training data} | \text{model}) \times \frac{p(\text{model})}{p(\text{training data})} \]
- \( P(\text{training data}) \) a scaling factor; ignored
- \( P(\text{training data} | \text{model}) \) computed as last slide
- \( P(\text{model}) \): preference towards smaller models
  - Total number of states: \( N \)
  - Total number of transitions at each state \( S \): \( T(S) \)
  - Total number of emissions at each state \( S \): \( E(S) \)

\[
P(\text{Model}) \propto \frac{1}{\prod_s (N + 1)^{T(S)} \times (N + 1)^{E(S)}}
\]
Structural inference: Learning

- Start with a model exactly reflecting input data
- Gradually merge states
- Until posterior probability does not increase
- Cost: $O((n^*L)^3)$ with $n$ training input strings, and $L$ maximum length of each string
  - Up to $n^*L$ states
  - $(n^*L)(n^*L-1)/2$ comparisons for each merging
  - Up to $n^*L-1$ merges

- Optimizations
  - Viterbi path approximations
  - Path prefix compression
  - Cost: $O(n^*L^2)$
First option: Compute probability of query attribute
  ◦ Issue: probabilities of all input words sum up to 1; all words have small probabilities

Output:
  ◦ $p = 1$ if word is a valid output of Markov model
  ◦ $p = 0$ otherwise
Token finder

- Goal: determine whether values of an attribute are drawn from an enumerated set of tokens
- Example: flags, indices
- Learning:
  - Growth in # of different argument instances compared to total # of argument instances
  - Compute correlation between these numbers:
    - $F(x) = x$
    - $G(x) = G(x-1) + 1$ if $x$-th value is new
    - $G(x) = G(x-1) - 1$ if $x$-th value was seen before
    - $\text{Corr} = \frac{\text{Covar}(F, G)}{\sqrt{\text{Var}(F) \times \text{Var}(G)}}$
    - If $\text{Corr} < 0$, then enumeration
    - If enumeration, then store all values for use in detection phase
Token finder: Detection

- If enumeration: value expected to be among stored values
  - Output $p = 1$ or $p = 0$ correspondingly
- If random: $p = 1$
Attribute presence/absence

- Observation: URIs typically produced not directly by user, but by scripts, forms, client-side programs
  - Result: regularity in number, name, order of parameters
  - Hand-crafted attacks typically break this regularity
    - Incomplete or malformed requests to probe/exploit web app
      - Missing argument
      - Mutually exclusive arguments appearing together
Attribute presence/absence

- Learning: Record set $S_q$ for each query $q$ during training in a hash table
- Detection: Lookup the attribute set in hash table
  - Return $p = 1$ or $p = 0$ correspondingly
Attribute order

- Legitimate invocations often contain same attributes in same orders
  - Sequential program logic preserves order even when some attributes left out

- Learning:
  - Attribute $a_s$ precedes $a_t$ if as and at appear together in parameter list of at least one query and $a_s$ comes before $a_t$ when they appear together
Attribute order

- Directed graph
- # vertices = # attributes
- For each training query, add edges between nodes of ordered attribute pairs
- Find all strongly connected components (SCC) of the graph
- Remove edges between nodes in same SCC
- For each node, find all reachable nodes
- Add corresponding pairs to set of precedence orders
Attribute order

• Find all order violations
  ◦ Return $p = 0$ or $p = 1$ correspondingly
Access frequency

- Frequency patterns of different server-side web applications
- Two types of frequencies:
  - Frequency of application being accessed from a certain client (IP address)
  - Total frequency of all accesses
- Attacks
  - Probing
  - Guess parameter values
  - Evasion: slow down
Access frequency

- Learning:
  - divide training time to intervals of fixed time (e.g., 10 sec)
  - Count accesses in each interval
  - Find total and client-specific distributions

- Detection:
  - Chebyshev probability for total, and for client
  - Return average of the two probabilities
Inter-request time delay

- Regular delay between each successive request
  - Surveillance
  - Scripted probes
- Learning: Find distribution of normal delays
  - Similar to character distribution model
- Detection: Pearson $\chi^2$-test
Invocation order

- Order of invocation of web-based applications for each client
  - Infer session structure regularity
  - Similar to structural inference model
- Learning: group queries based on source IP
  - Session: Queries within an interval of time
  - Build NFA for sessions
- Detection: $p = 1$ or $p = 0$ depending on session being an output of NFA
# Evaluation

<table>
<thead>
<tr>
<th>Data set</th>
<th>Number of alerts</th>
<th>Number of queries</th>
<th>False positive rate</th>
<th>Alarms per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>206</td>
<td>490,704</td>
<td>0.000419</td>
<td>4944</td>
</tr>
<tr>
<td>UCSB</td>
<td>3</td>
<td>4617</td>
<td>0.000650</td>
<td>0.01</td>
</tr>
<tr>
<td>TU Vienna</td>
<td>137</td>
<td>713,500</td>
<td>0.000192</td>
<td>1.71</td>
</tr>
</tbody>
</table>
Reference

• “A multi-model approach to the detection of web-based attacks”, 2005