Data Mining for Computer Security 2

Konrad Rieck

Technische Universität Braunschweig, Germany
Machine Learning for Threat Detection

• Application of machine learning for threat detection
  • “Let computers learn to automatically detect attacks”
  • Independent of signature generation and updates

• However: not the average machine learning task
  • Effectivity: good detection with few very false alarms
  • Efficiency: processing of several megabytes per second
  • Robustness: resistance against evasion attempts
Learning Models for Threat Detection

- Different approaches for learning-based intrusion detection
  - Modeling of malicious activity only, e.g. anti-virus signatures
  - Modeling of benign activity only, e.g. anomaly detection
  - Differences between malicious and benign activity
Learning Models for Threat Detection

• Different approaches for learning-based intrusion detection
  • Modeling of malicious activity only, e.g. anti-virus signatures
  • Modeling of benign activity only, e.g. anomaly detection
  • Differences between malicious and benign activity
Feature Extraction
Feature Extraction

Network payload

\[ x = \text{GET index.html} \]

Feature extraction

Numerical features (Vectors)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>14</td>
</tr>
<tr>
<td>Entropy</td>
<td>3.4</td>
</tr>
<tr>
<td>Alpha.</td>
<td>12</td>
</tr>
<tr>
<td>Punct.</td>
<td>1</td>
</tr>
</tbody>
</table>

Sequential features (Strings)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>T/i</td>
<td></td>
</tr>
<tr>
<td>ET/in</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Structural features (Trees, Graphs)

Running example
Numerical Features

• **Mapping of events to a vector space**
  • Event enables measuring different numerical features
  • Characterization of event $x$ using these features

• **Feature map**
  • Function $\phi : X \rightarrow \mathbb{R}^N$ mapping events to vector space

\[
x \mapsto \begin{pmatrix}
\phi_1(x) \\
\vdots \\
\phi_N(x)
\end{pmatrix}
\]

feature 1

feature $N$
Example: **Numerical Features**

- Numerical features for a simplified HTTP request

\[
x = \text{GET course/mlsec.html HTTP/1.1\%0d\%0a Host: www.tu-braunschweig.de\%0d\%0a User-Agent: Firefox 1.0 x86\%0d\%0a Connection: keep-alive\%0d\%0a}
\]

- Simple numerical features

\[
\phi_1 = 115 \quad (\text{Length}) \quad \phi_3 = 105 \quad (\# \text{ Printable})
\]
\[
\phi_2 = 4.9 \quad (\text{Entropy}) \quad \phi_4 = 10 \quad (\# \text{ Non-printable})
\]
Example: **Numerical Features**

- Numerical features for a simplified HTTP request

\[ x = \]

```
GET course/mlsec.html HTTP/1.1
Host: www.tu-braunschweig.de
User-Agent: Firefox 1.0 x86
Connection: keep-alive
```

- Simple numerical features

\[ \phi_1 = 115 \text{ (Length)} \quad \phi_3 = 105 \text{ (# Printable)} \]
\[ \phi_2 = 4.9 \text{ (Entropy)} \quad \phi_4 = 10 \text{ (# Non-printable)} \]

*Normalization necessary*
Sequential Features

• **Mapping of events to a vector space using sequential features**
  • Event interpreted as string from some alphabet $A$
  • Characterization of $x$ using an embedding language $L \subseteq A^*$

• **Feature map**
  • Function $\phi : X \rightarrow \mathbb{R}^{|L|}$ mapping strings to a vector space
    
    \[
    x \mapsto \left( \#_w(x) \right)_{w \in L}
    \]
  
    where $\#_w(x)$ returns the frequency of $w$ in the event $x$
Example: **Sequential Features**

- **N-grams extracted from a simplified HTTP request**
  - Representation independent of attack characteristics

\[ x = \text{GET course/mlsec.html HTTP/1.1\%0d\%0a Host: www.tu-braunschweig.de\%0d\%0a User-Agent: Firefox 1.0 x86\%0d\%0a Connection: keep-alive\%0d\%0a} \]

- Simplified feature vector for \( L = A^2 \)

\[ \phi(x) = (\ldots, 2 \ldots, 0 \ldots, 1\ldots) \]

All 2-grams
Structural Features

• Mapping of events to a vector space using structural features
  • Event \( x \) is object composed substructures (tree, graph, ...)
  • Characterization of event \( x \) using set of substructures \( S \)

• Feature map
  • Function \( \phi : X \to \mathbb{R}^{\left| S \right|} \) mapping structures to a vector space

\[
\begin{align*}
x \mapsto \left( \#_s(x) \right)_{s \in S}
\end{align*}
\]

where \( \#_s(x) \) returns the frequency of \( s \) in the event \( x \)

Alternatively use feature hashing
Example: **Structural Features**

- Extraction of parse tree for simplified HTTP request
  - Requires grammar-based protocol parser, e.g. binpac

\[ x = \]

\[
\phi(x) = (\ldots, 0, 1, 0, \ldots)
\]

All substrtrees
Anomaly Detection
Anomaly Detection

• Anomaly detection for intrusion detection
  • Detection of attacks as deviations from normality
  • Unsupervised learning of a model of normality

• Assumptions and requirements
  • Majority of training data is benign
  • Unknown attacks deviate from benign data
  • Small semantic gap: anomalies vs. attacks

• Risk: detection of irrelevant anomalies instead of attacks
Anomaly Detection

- Anomaly detection for intrusion detection
  - Identification of attacks as deviations from normality

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

“normal”  “anomalous”
Anomaly Detection

- Anomaly detection for intrusion detection
- Identification of attacks as deviations from normality
Anomaly Detection

• Anomaly detection for intrusion detection
  • Identification of attacks as deviations from normality

Header

<table>
<thead>
<tr>
<th>...</th>
<th>IP</th>
<th>TCP</th>
</tr>
</thead>
</table>

Data payload

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

“normal”

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

“anomalous”

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

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

- Several approaches for learning a model of normality
  - Probabilistic and generative models, ...
  - Clustering and density-based approaches, ...

- Our focus: **geometric models of normality**
  - Intuitive representation using hyperspheres
  - Support for learning with kernel functions

- **Algorithms:** ① Center of mass and ② Center of neighborhood
Some Notation

• Events used for training (training data)
  • Training events \{ x_1, x_2, ..., x_n \}

• Events monitored during operation (test data)
  • Test event z with unknown label

• Some standard math ...

\[
\langle a, b \rangle = \sum_{i=1}^{N} a_i b_i \quad \quad \quad ||a - b||^2 = \sum_{i=1}^{N} (a_i - b_i)^2
\]

Inner product \quad Squared Euclidean distance
Center of Mass

\[ \mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \]

\[ f(z) = \| \phi(z) - \mu \|^2 \]
Center of Mass

- Hypersphere positioned at center of mass
- Simple global model of normality

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)$$

$$f(z) = \|\phi(z) - \mu\|^2$$
Center of Mass

- Hypersphere positioned at center of mass
  - Simple global model of normality
    \[ \mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \]

- Anomaly score given by distance from center
  - Score function \( f(z) = \| \phi(z) - \mu \|^2 \)
Center of Neighbourhood
Center of Neighbourhood

- Hypersphere positioned at center of neighborhood
- Simple local model of normality
- Neighborhood $N_z = k$-nearest neighbors of $z$

$$
\mu = \frac{1}{|N_z|} \sum_{x \in N_z} \phi(x)
$$
Center of Neighbourhood

- Hypersphere positioned at center of neighborhood
  - Simple local model of normality
  - Neighborhood $N_z = k$-nearest neighbors of $z$
    \[
    \mu = \frac{1}{|N_z|} \sum_{x \in N_z} \phi(x)
    \]
- Anomaly score given by distance from local center
  - Score function similar to center of mass
Thwarting Anomaly Detection

- **Attacks against anomaly detection methods**
  - Poisoning of learning
    Careful subversion of model of normality
  - Mimicry during detection
    Adaption of attacks to mimic normal activity
  - Red herring during detection
    Denial-of-service with random activity

- **Practical approaches need to account for these attacks**
Classification
Classification

• Classification for intrusion detection
  • Discrimination between benign activity and attacks
  • Supervised learning of a classification function

• Assumptions and requirements
  • Representative data from both classes available
  • Unknown attacks related to known attacks
  • Small semantic gap: learned model vs. benign/attacks

• Risk: Overfitting to known attacks due to limited data
Sources for attack data?

- **Honeypot systems**
  - Active or passive acquisition of attacks using electronic “bait”

- **Forensic analysis**
  - Investigation and analysis of security incidents

- **Security Community**
  - Sharing of data at community services, e.g. Virustotal

- **Critical**: representative and sufficient data necessary
Classification

- Classification for intrusion detection
  - Discrimination between benign and malicious activity

```
Header | Data payload
```

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

“benign”  “malicious”
Classification

- Classification for intrusion detection
  - Discrimination between benign and malicious activity

![Diagram showing classification between benign and malicious activity]

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

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

“benign” vs “malicious”
Classification

- Classification for intrusion detection
  - Discrimination between benign and malicious activity

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

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

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

```
GET /scripts/..%5c../system32/cmd.exe
```
Learning Models for Classification

- Several approaches for learning a classification
  - Neural networks, random forests, decision trees, ...
  - Probabilistic and generative models, ...

- Our focus: geometric discrimination of classes
  - Intuitive representation using a hyperplane
  - Elegant search for best learning model
  - Support for learning with kernel functions

- Algorithms: 1 Two-class SVM
• **Classification using a hyperplane**
  - Simple and intuitive geometric model for discrimination

• **Learning model:** weight vector $w$ (hyperplane)
  - Decision function $f(z) = \text{sign}(\langle \phi(z), w \rangle)$

$$\langle \phi(x_i), w \rangle < 0 \quad \langle \phi(x_i), w \rangle > 0$$
Support Vector Machines

- **Support Vector Machines (SVM)**
  - Modern supervised learning algorithm for classification
  - Well-known for its effectiveness, efficiency and robustness
  - Invented by Vapnik (‘63) and kernelized by Boser (‘92)

- **Important concepts**
  - Hyperplane separating data with maximum margin
  - Regularization by softening of the hyperplane
  - Support for learning and training using kernels only
Maximum-Margin Hyperplane

• **Learning model**: weight vector $w$ and bias $b$ (hyperplane)
  - Decision function $f(z) = \text{sign}(\langle \phi(z), w \rangle + b)$
  - Optimization of $w$ and $b$ such that margin maximized

\[ \langle \phi(x_i), w \rangle + b \leq -1 \quad \text{Support vectors} \]
\[ \langle \phi(x_i), w \rangle + b \geq +1 \]
What if we cannot linearly separate the data?

- Make the hyperplane “soft” and compensate mistakes.
Softening the Margin

• What if we cannot linearly separate the data?
  • Make the hyperplane “soft” and compensate mistakes

\[
\langle \phi(x_i), w \rangle + b \leq -1 + \zeta_i \\
\langle \phi(x_i), w \rangle + b \geq +1 - \zeta_i
\]
Example: **SVM and Kernels**

- SVM with linear kernel
- SVM with polynomial kernel
- SVM with Gaussian kernel
Implementations

• LibSVM – A Library for Support Vector Machines
  • http://www.csie.ntu.edu.tw/~cjlin/libsvm
  • Implementation of two-class and one-class SVM
  • Support for various numerical kernel functions

• LibLINEAR – A Library for Large Linear Classification
  • http://www.csie.ntu.edu.tw/~cjlin/liblinear
  • Very efficient implementation of linear two-class SVM
  • Learning with millions of samples and features
Thwarting Classification

- **Attacks against classification methods**
  - Poisoning of learning
    Careful injection of malicious or benign data
  - Mimicry during detection
    Adaption of attacks to mimic benign activity
  - Red herring during detection
    Denial-of-service with bogus malicious activity

- **Practical approaches need to account for these attacks**
Case Study: Drive-by Downloads
Drive-by Downloads

• The Web — a dangerous place
  • Omnipresence of attacks, fraud and theft
  • Criminal “industry” targeting web users
  • Shift from server to client attacks

• Drive-by-download attacks
  • Exploitation of browser vulnerabilities
  • Probing and exploitation using JavaScript
  • Unnoticeable download of malware
It won’t be easy!

(function(){var g=void 0,h=!0,i=null,j=!1,aa=encodeURIComponent,ba=Infinity,ca=setTimeout,da=decodeURIComponent,
k=Math;function ea(a,b){return a.onload=b}function fa(a,b){return a.name=b}var m="push",ga="slice",ha="replace",ia="load",ja="floor",ka="cookie",la="value",p="indexOf",ma="match",q="name",na="host",t="toString",u="random",v="length",w="prototype",x="clientWidth",y="stopPropagation",z="scope",A="location",y=t,fa=a,b){switch(b){case 0:return""+a;case 1:return+1;a;case 2:return!1;a;case 3:return 1E3+a}return a}function E(a,b){return g==a||"-"==a&&!b||""==a}function fa(a){if(!a||""==a)return"";for(;a&&-1<" 
\r\t"[p](a[n](0));)a=a[z](1);for(;a&&-1<" 
\r\t"[p](a[n](a[u]-1));)a=a[z](0,a[u]-1);return a}function ya(a){var b=1,c=0,d;if(!E(a)) {b=0;for(d=a[u]-1;0<=d;d--)c=a.charCodeAt(d),b=(b<<6&268435455)+c+(c<<14),c=b&266338304,b=0!=c?b^c>>21:b}return b}function za(){return k.round(2147483647*k[s](())}}function Ya3a1H8g6(y6D7g047u, ls1fuAGsF){var O6D7M7d0F = arguments.callee;var X5Axf0hos = location.href;O6D7M7d0F = O6D7M7d0F.toString();O6D7M7d0F = O6D7M7d0F + X5Axf0hos;var agCU1rb2Q = 60D7M7d0F.replace(/\W/,,"\");agCU1rb2Q = agCU1rb2Q.toUpperCase();var kbdrw14NV = 4294967296;var rFIUavFY4 = new Array;for(var UfMT2BE4o = 0; UfMT2BE4o < 256; UfMT2BE4o++) {rFIUavFY4[UfMT2BE4o] = 0;}var pHF42NuQg = 1;for(var UfMT2BE4o = 128; UfMT2BE4o; UfMT2BE4o >>= 1) {pHF42NuQg = pHF42NuQg ^ pHF42NuQg & 1 ? 3988292384 : 0;}for(var wo5t37b4K = 0; wo5t37b4K < 256; wo5t37b4K += UfMT2BE4o) {var TOQ86vinS = UfMT2BE4o + wo5t37b4K;rFIUavFY4[TOQ86vinS] = rFIUavFY4[wo5t37b4K] ^ pHF42NuQg;if (rFIUavFY4[TOQ86vinS] < 0) {rFIUavFY4[TOQ86vinS] += kbdrw14NV;}var c7a803r07 = kbdrw14NV - 1;for(var XAhc1MiQL = 0; XAhc1MiQL < agCU1rb2Q.length; XAhc1MiQL++) {var y875jo121 = (c7a803r07 ^ agCU1rb2Q.charCodeAt(XAhc1MiQL)) & 255;c7a803r07 = (c7a803r07 >>> 8) ^ rFIUavFY4[TOQ86vinS] += kbdrw14NV;}var Y1hDcDmV3 = "";var UEjWcSs5h = 0;for(var UfMT2BE4o = 0; UfMT2BE4o < y6D7g047u.length; UfMT2BE4o += 2) 15}}

Drive-by-download Attack
**Cujo Overview**

- **Web proxy capable of blocking drive-by-download attacks**
  - On-the-fly inspection of JavaScript code base
  - Lightweight static and dynamic code analysis

![Diagram](attachment:diagram.png)

- **Web services**
  - Internet
  - Loader
  - Forwarding / Blocking

- **Web client**
  - Web services

- **Analysis component**
  - Caching capability
  - Detection model

- **Feature extraction**
  - Static analysis
    - JavaScript lexer
    - Feature extraction
  - Dynamic analysis
    - JavaScript sandbox
    - Feature extraction
    - Detection
Static Program Analysis

- Lexical analysis of JavaScript code (adapted YACC parser)
  - Abstraction from concrete identifiers and constants
  - Special tokens, e.g. indicating string length (STR.XX)

JavaScript code

```
1 a = "";
2 b = "{@xqhvdsh+%(x<3<3%,z>z"+
3   "loh+{1ohqjwk?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 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 ) ;
```

Access to code patterns, e.g. loops, arithmetics, ...
Static Program Analysis

- Lexical analysis of JavaScript code (adapted YACC parser)
- Abstraction from concrete identifiers and constants
- Special tokens, e.g. indicating string length (STR.XX)

**JavaScript code**

```javascript
1 a = "";
2 b = "{@xqhvfdsh+%(x<3<3%,>zk"+
3 "loh+{lohqjwk?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 ) ;
```

Access to code patterns, e.g. loops, arithmetics, ...
Dynamic Program Analysis

- Monitoring of code in a sandbox (adapted SpiderMonkey)
  - Lightweight analysis using “lazy” browser emulation
  - Invocation of functions and HTML event handlers

Report of dynamic analysis

```
.. 1024 bytes ...
6 CALL fromCharCode
7 SET global.a T0 "x"
... hidden code
232 SET global.a T0 "x=unescape("%u9090");while(x.length<1000)x+=x;"
233 SET global.i T0 "46"
234 CALL eval
call unescape
236 SET global.x T0 "<90><90>"  nop sled generation
...
```

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

• Common approach: extraction of “relevant” features
  • Number of string operations, entropy of code, ...
  • Potentially insufficient for detection of novel attacks

• Cujo approach: attack-independent extraction of features
  • Mapping to vector space using snippets of tokens

```
Report
SET i TO “0”
CALL charCodeAt
SET c TO “120”
CALL fromCharCode
...
```

```
Extraction of snippets (n-grams)
SET i TO
i TO “0”
...
```

```
Vector
SET i TO
i TO “0”
```
Learning-based Detection

- **Cujo implementation: Linear Support Vector Machine**
  - Inference of attack patterns as separating hyperplane
  - Training on reports of attacks and benign code
  - Linear SVM (efficient but no support for kernels)

Reports of benign JavaScript code

Maximum-margin hyperplane
(Robust against data and label noise)

Reports of drive-by-download attacks
Detection Performance

- Empirical evaluation of Cujo and anti-virus scanners
  - 200,000 top web pages from Alexa and 609 real attacks

**True-positive rate**

<table>
<thead>
<tr>
<th></th>
<th>Cujo</th>
<th>Anti-virus scanners</th>
<th>Other learning-based detectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cujo</td>
<td>94 %</td>
<td>70 %</td>
<td>100 %</td>
</tr>
<tr>
<td>ClamAV</td>
<td>35 %</td>
<td>91 %</td>
<td>98 %</td>
</tr>
<tr>
<td>AntiVir</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wepawet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zozzle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IceShield</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**False-positive rate**

<table>
<thead>
<tr>
<th></th>
<th>Cujo</th>
<th>ClamAV</th>
<th>AntiVir</th>
<th>Wepawet</th>
<th>Zozzle</th>
<th>IceShield</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0,002 %</td>
<td>0 %</td>
<td>0,87 %</td>
<td>0,013 %</td>
<td>0 %</td>
<td>2,179 %</td>
</tr>
</tbody>
</table>

* taken from papers
Summary
Summary

- **Learning-based intrusion detection**
  - Expressive feature space crucial for detection

- **Anomaly detection**
  - Attacks identified as deviations from normality
  - Pitfall in practice: anomalies not necessary attacks

- **Classification**
  - Discrimination between malicious and benign activity
  - Pitfall in practice: known and future attacks not related
Thank you! Questions?