Swiss Cyber Storm 2016 – Turbo Talk

Client TLS Testing
Detecting Obfuscated JavaScripts

Prof. Dr. Marc Rennhard
Head of Information Security Research Group
Institute of Applied Information Technology (InIT)
ZHAW School of Engineering
rema@zhaw.ch
About the Information Security Research Group

- Part of InIT at ZHAW
  - InIT: Institute of Applied Information Technology
  - ZHAW: Zurich University of Applied Sciences

- 3 professors/lecturers, 8-10 researchers

- Main activity: Applied research projects with industrial / academic partners
  - ≈20 large R&D projects during the past 10 years (mostly CTI and EU)

- One key research area: Automated security analysis and security testing
Client TLS Testing

• Motivation
  • TLS is the *most widely used* secure communication protocol
  • Several services and tools to test the security of TLS servers
  • Only few tools to test the security of *client-side* TLS implementations and configurations

• Goal: Develop a powerful tool for client TLS testing
  • Current focus on testing the processing of server certificates
  • Very security-critical component as wrong handling of certificates may allow e.g. MITM attacks

Use any TLS client application to be tested

Testing tool runs as TLS server application
Client TLS Testing – Tool Usage in Practice

Now the client application to test can initiate repeatedly TLS sessions with the testing tool and during each TLS session, a certificate test case is carried out:

Running Test: WeakRsa512Key
Description: Valid certificate with a weak 512-bit rsa-key
Server ready, listening on port 1025 for TLS connection...
Connection established to remote client 127.0.0.1:64037
Test passed

Running Test: MultipleCnInvalidFirst
Description: Subject contains multiple CN-entries.
Server ready, listening on port 1025 for TLS connection...
Connection established to remote client 127.0.0.1:53129
Test failed

- +120 certificate tests integrated, covering various aspects of certificates and certificate chains
### Client TLS Testing – Web-based Service

#### tical Service Run Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Browser XY - 2016-10-19</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>Issue Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Missing Intermediate Basic Constraints Extensions</td>
<td>Success</td>
</tr>
<tr>
<td>High</td>
<td>Multiple Cn Invalid First</td>
<td>Failure</td>
</tr>
<tr>
<td>High</td>
<td>Multiple Cn Valid First</td>
<td>Success</td>
</tr>
<tr>
<td>High</td>
<td>Not Authorized For Handshake</td>
<td>Success</td>
</tr>
<tr>
<td>High</td>
<td>Unknown Critical Extension</td>
<td>Success</td>
</tr>
<tr>
<td>Low</td>
<td>Common Name Encoding Bmp String</td>
<td>Failure</td>
</tr>
<tr>
<td>Low</td>
<td>Common Name Encoding Graphic String</td>
<td>Success</td>
</tr>
</tbody>
</table>

---

Client TLS Testing – Status and Outlook

• Current status and next steps
  • Tool works well to efficiently test client-side TLS implementations
  • Systematically test all widespread browsers and TLS libraries with the goal to find security problems

• Future plans
  • Public release of the testing tool
    ▪ Provide Web-based service / release tool as open source software
  • Extend with further tests, e.g. TLS protocol fuzzing

• Thanks to all people involved!
  • Stefan Berhardsgrütter, Lucas Graf, Damiano Esposito (InIT)
  • Tobias Ospelt (modzero)
Detecting Obfuscated JavaScripts

• Motivation
  • JavaScript is often used as an attack vector to deliver malware
    ▪ XSS vulnerabilities, Web-based malware distribution (drive-by),...
  • Detection using signature-based approaches are not ideal, easy to circumvent e.g. by obfuscating JavaScripts
  • Most malicious JavaScripts are obfuscated
  • Benign JavaScripts are usually not obfuscated

• If obfuscated JavaScripts can be reliably detected, this serves as a good first indicator whether a script is malicious / benign

• Goal of the project: Find a method to classify JavaScripts as obfuscated / non-obfuscated with high accuracy
  • Based on a machine learning approach
Detecting Obfuscated JavaScripts – Data Set

• Having a large, representative data set with correctly labelled samples is key to machine learning.

• Our data set consists of +100,000 JavaScripts:
  • From different sources: top global websites, JavaScript libraries, MELANI (malicious samples).
  • Includes samples from more than 10 different obfuscators.

• The data set was used to train different binary classifier:
  • The trained classifier takes any JavaScript as input and classifies it as obfuscated / non-obfuscated.
Detecting Obfuscated JavaScripts – Classification Performance

- **Boosted Decision Tree** performed best to solve the problem
- Two important values to assess the performance of trained machine learning models are **precision and recall**
  - With BDT, we could achieve values of **99% or better**
  - Less than 1 out of 100 JavaScripts is classified incorrectly

<table>
<thead>
<tr>
<th></th>
<th>AP</th>
<th>BPM</th>
<th>BDT</th>
<th>DF</th>
<th>DJ</th>
<th>LDSVM</th>
<th>LR</th>
<th>NN</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non</td>
<td>80.46%</td>
<td>92.44%</td>
<td>99.06%</td>
<td>98.50%</td>
<td>97.93%</td>
<td>93.53%</td>
<td>78.31%</td>
<td>95.64%</td>
<td>81.65%</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>66.31%</td>
<td>78.03%</td>
<td>98.97%</td>
<td>98.14%</td>
<td>98.10%</td>
<td>88.40%</td>
<td>68.28%</td>
<td>90.02%</td>
<td>66.82%</td>
</tr>
<tr>
<td></td>
<td>72.70%</td>
<td>84.63%</td>
<td>99.01%</td>
<td>98.32%</td>
<td>98.02%</td>
<td>90.89%</td>
<td>72.95%</td>
<td>92.74%</td>
<td>73.50%</td>
</tr>
<tr>
<td></td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
<td>7752</td>
</tr>
<tr>
<td>Obfuscated</td>
<td>89.21%</td>
<td>92.61%</td>
<td>99.65%</td>
<td>99.37%</td>
<td>99.36%</td>
<td>96.14%</td>
<td>89.68%</td>
<td>96.68%</td>
<td>89.39%</td>
</tr>
<tr>
<td></td>
<td>94.54%</td>
<td>97.73%</td>
<td>99.68%</td>
<td>99.49%</td>
<td>99.30%</td>
<td>97.92%</td>
<td>93.58%</td>
<td>98.61%</td>
<td>94.90%</td>
</tr>
<tr>
<td></td>
<td>91.80%</td>
<td>95.10%</td>
<td>99.67%</td>
<td>99.43%</td>
<td>99.33%</td>
<td>97.02%</td>
<td>91.59%</td>
<td>97.63%</td>
<td>92.07%</td>
</tr>
<tr>
<td>s</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
<td>22842</td>
</tr>
</tbody>
</table>

Fig. 1. Performance of the classifiers to classify non-obfuscated and obfuscated scripts, using all features.
Detecting Obfuscated JavaScripts – Conclusions and Outlook

• Key Findings
  • Machine learning works well to classify obfuscated / non-obfuscated JavaScripts
  • Machine Learning is no magic solution: correctly classifying a script that uses an obfuscator not present in the training set is much more difficult
  • Try it out: http://jsclassify.azurewebsites.net/

• Future work
  • Our classifier serves as a good indicator for malicious / benign JavaScripts, but the ultimate goal is to have a classifier that outputs malicious / benign
  • Main obstacle: Number of malicious samples in the data set is currently not representative enough (only about 2’700 samples)

• Publications
  • B. Tellenbach, S. Paganoni, M. Rennhard. Detecting Obfuscated JavaScripts from Known and Unknown Obfuscators using Machine Learning (under review)