

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

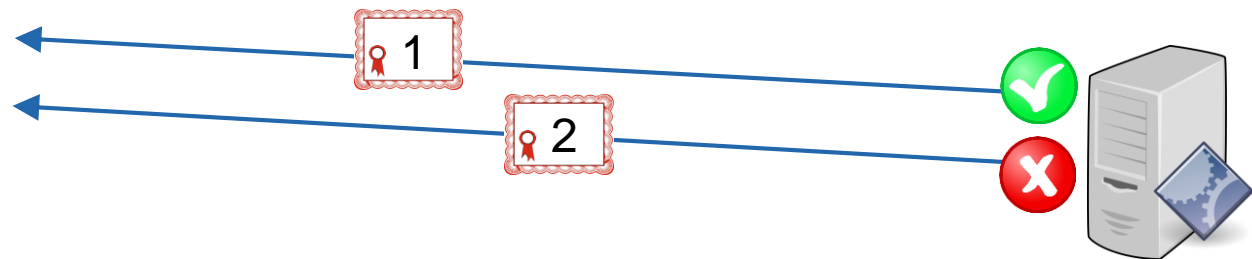


Client TLS Testing

- 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

```
[rema:tcal marc$ python3 tcal.py www.tcal.test -c x509  
Generating x509 test certificates...  
100.0% of all test certificates generated  
Waiting for test start
```

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

Passed means the
behavior is according
to the RFC / no
security problem

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

Failed means not
compliant with the
RFC / potential
security problem

- **+120 certificate tests** integrated, covering various aspects of certificates and certificate chains

Client TLS Testing – Web-based Service

Home About Contact Runs
REMA@ZHAW.CH

tcald Service Run Configuration

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam maximus leo a dolor luctus congue. Nulla a sem ligula. Sed vel porttitor sapien, quis iaculis eros. Phasellus tincidunt quis nulla non lobortis. Cras ultrices eget nulla vitae vestibulum. Donec mattis malesuada arcu quis auctor. Vestibulum a velit augue. Aliquam varius vitae dolor vel dignissim. Ut ac placerat nisl. Maecenas ac quam eu massa accumsan faucibus at vel neque. Nullam vel justo nisi. Sed tristique ut ligula eleifend tincidunt. Integer tempor consequat risus euismod aliquam. Ut blandit metus sit amet neque volutpat porta. Curabitur fermentum quam ut ante luctus luctus.

Name

Description

[CREATE INSTANCE](#)

Home About Contact Runs
REMA@ZHAW.CH

High	Missing Intermediate Basic Constraints Extensions	A chain of certificates where the first intermediate CA is missing a basic constraint extension.	Success
High	Multiple Cn Invalid First	Subject contains multiple CN-entries.	Failure
High	Multiple Cn Valid First	Subject contains multiple CN-entries.	Success
High	Not Authorized For Handshake	A certificate that has an unsuitable value in the key usage extension (cRLSign=true, not allowed for TLS-Handshake).	Success
High	Unknown Critical Extension	A certificate that has a non-standard critical extension entry.	Success
Low	Common Name Encoding Bmp String	Valid certificate signed by root-ca with common name BmpString-encoded	Failure
Low	Common Name Encoding Graphic String	Valid certificate signed by root-ca with common name GraphicString-encoded	Success

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

		AP	BPM	BDT	DF	DJ	LDSVM	LR	NN	SVM
Non Obfuscated	p	80.46%	92.44%	99.06%	98.50%	97.93%	93.53%	78.31%	95.64%	81.65%
	r	66.31%	78.03%	98.97%	98.14%	98.10%	88.40%	68.28%	90.02%	66.82%
	F1	72.70%	84.63%	99.01%	98.32%	98.02%	90.89%	72.95%	92.74%	73.50%
	s	7752	7752	7752	7752	7752	7752	7752	7752	7752
Obfuscated	p	89.21%	92.61%	99.65%	99.37%	99.36%	96.14%	89.68%	96.68%	89.39%
	r	94.54%	97.73%	99.68%	99.49%	99.30%	97.92%	93.58%	98.61%	94.90%
	F1	91.80%	95.10%	99.67%	99.43%	99.33%	97.02%	91.59%	97.63%	92.07%
	s	22842	22842	22842	22842	22842	22842	22842	22842	22842

Fig. 1. Performance of the classifiers to classify non-obfuscated and obfuscated scripts, using all features.

- **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

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
 - S. Aebersold, K. Kryszczuk, S. Paganoni, B. Tellenbach, T. Trowbridge. Detecting Obfuscated JavaScripts Using Machine Learning. ICIMP 2016
 - B. Tellenbach, S. Paganoni, M. Rennhard. Detecting Obfuscated JavaScripts from Known and Unknown Obfuscators using Machine Learning (under review)

Classification Result

Classification of 130137 bytes JavaScript.

Scored Label: NonObfuscated

Scored Probability: 2.60754968621768E-05