HTTP fuzzing in Smart Card Web Server

Matthieu BARREAUD, Guillaume BOUFFARD & Jean-Louis LANET
{matthieu.barreaud, guillaume.bouffard, jean-louis.lanet}@xlim.fr
SSD team – XLIM – University of Limoges

This work is based on the M1 students project. Thanks to Mamadou BALDE, Amine BELHOCINE, Silvère CAINAUD, Jérémie CLEMENT, Romain SEVERIN, Nicolas TARRIOL and Lylia TIKOBAINI.
Outline

- Introduction
- State of the art
  - SCWS / BIP
  - The HTTP protocol
  - The fuzzing
- Fuzzing on a SCWS
  - Aims
  - HTTP smart card features
  - Tests generation
  - Logging
  - Parallelization
- Experimental results
- Conclusions
Introduction

- Smart cards are essentials:
  - Payment: credit card, moneo, etc.
  - Transport: ticketing in public transport, etc.
  - Insurance: Health Care card, etc.
  - Telephony: SIM card (mobile), etc.

- Hyper Text Transfer Protocol (HTTP) is a new implemented technology in smart card world.

- Problem:
  - How to ensure the HTTP implementation robustness?
State of the art > SCWS / BIP

- mNFC JavaCard 2.2 Smart Card Web Server (SCWS)
- http://127.0.0.1:3516
- Short-range wireless technologies (~ 10 cm)
- SCWS is both a server and a client application:
  - In server mode, SCWS is used by the subscriber using a WAP browser implemented in his handset.
  - In client mode, SCWS is used by the Card Issuer in order to administrate the SCWS from a server.
- In our case, we use the SCWS in server mode.
- SCWS communicates with BIP commands
State of the art > SCWS / BIP

1a. Terminal Profile
91xx

1b. Fetch
(Open Channel (BIP TCP Server Mode))

1c. Terminal Response

2. BIP Gateway listens on Port number 3516. It is “server” to mobile HTTP client

3. URL=http://127.0.0.1:3516/index.html
We can access the SCWS in:

- Remote to administrate the SCWS
  - Administrative commands: add a user, change a password, etc.
  - The Access Control Policy (ACP) is stocked in the card

- Local to communicate with the SCWS

<table>
<thead>
<tr>
<th>Method</th>
<th>Supported</th>
<th>Additional comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>Mandatory</td>
<td>Mandated for HTTP 1.1 server implementations</td>
</tr>
<tr>
<td>HEAD</td>
<td>Mandatory</td>
<td>Mandated for HTTP 1.1 server implementations</td>
</tr>
<tr>
<td>POST</td>
<td>Mandatory</td>
<td>Support for forms in user interface</td>
</tr>
<tr>
<td>PUT</td>
<td>Mandatory</td>
<td>Support for remote administration</td>
</tr>
<tr>
<td>DELETE</td>
<td>Mandatory</td>
<td>Support for remote administration</td>
</tr>
<tr>
<td>TRACE</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>CONNECT</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>
HTTP request representation:

```
GET /index.html HTTP/1.1 \r\nHost: 127.0.0.1:3516 \r\n...\r\n\r\n```

Creation of an interactive HTTP Backus Normal Form (BNF)
It is a software testing technique

It provides invalid, unexpected or random data to the inputs of a computer program, protocol implementation, ... in order to crash it.

Fuzzing is used to find security flaws in software or computer systems.

There are two types of fuzzer:

- *Mutation based fuzzer*: mutates existing data samples to create test data
- *Generation based fuzzer*: defines new test data based on data models and state models
Advantages / Drawbacks of fuzzer types:

- **Mutation**
  - Limited
  - Fast to implement
  - Slow execution because of cases number

- **Generation**
  - Full
  - Implementation is time consuming
  - Slow execution because of cases number
The fuzzing > Aims

- We verify the correctness implementation of the smart card HTTP protocol
- Work in black box without knowledge of the system
- Have a generic data model, usable anywhere (not only on smart card)

- Accurate results analysis:
  - Determine if the smart card has a not expected behavior
  - We analyze the card return value.

- Choose the type of mutation:
  - Mutate all possible values or only part of them?
The fuzzing > HTTP features

- How to know the SCWS implemented features?

- We developed PyHAT!

- Which reduces the amount of methods to fuzz
The fuzzing > HTTP features

* 0000000000 0000 0000 0 0000000000*
* 888 888 0000 0000 888 888 888 888 888 888*
* 888000888 888 888 888000888 8 88 888 888*
* 888 888 888 888 888 8000888 888 888 888*
* 888 88880888 8880880 8880 8880 88880*
* * 888 88880888 8880880 8880 8880 88880*
* 1. SmartCard Web Server
* 2. Other Web Server
* 3. Quit
* Your choice : 1

* Transferring data to 127.0.0.1 on port 3516

* List of implemented HTTP methods
  * HEAD
  * TRACE
  * GET
...

* List of supported HTTP versions
  * HTTP/0.9
  * HTTP/1.0
  * HTTP/1.1

* List of supported encoding
  * gzip
  * compress
  * deflate
  * identity

* List of parsed headers
  * Cache-Control
  * Connection
  * Date
...

* List of web pages on secinfo.msi.unilim.fr
  * /xmlrpc.php
  * /wp

* 0000000000 0000 0000 0 0000000000*
* 888 888 0000 0000 888 888 888 888 888 888*
* 888000888 888 888 888000888 8 88 888 888*
* 888 888 888 888 888 8000888 888 888 888*
* 888 88880888 8880880 8880 8880 88880*
* * 888 88880888 8880880 8880 8880 88880*
* Your choice : 2
* Host : secinfo.msi.unilim.fr
* Port [80] :

* Transferring data to secinfo.msi.unilim.fr on port 80

* List of implemented HTTP methods
  * HEAD
  * TRACE
  * GET
...

* List of supported HTTP versions
  * HTTP/0.9
  * HTTP/1.0
  * HTTP/1.1

* List of supported encoding
  * gzip
  * compress
  * deflate
  * identity

* List of parsed headers
  * Cache-Control
  * Connection
  * Date
...

* List of web pages on secinfo.msi.unilim.fr
  * /xmlrpc.php
  * /wp

*
The fuzzing > Test generation

PyHAT

PEACH

Search HTTP implemented features

Send invalid data

Return card state

Interface

APDU Response

Send invalid data

Smart Card
The fuzzing > Logging

- During the fuzzing step, we need to:
  - Log each test
  - To find some implementation errors

- A log file is created for each fuzzed method

- The request sent is saved with the full APDU command
The fuzzing > Logging

----- RECEIVE DATA TERMINAL RESPONSE -----
TRANSMIT (total): 80 14 00 00 44 01 03 01 42 01 02 02 82 81 03 01 00 36 81 32 47 45 54 20 2F 74 65 73 74 31 2E 68 74 6D 6C 20 48 54 50 2F 31 2E 30 2D 0A 48 6F 73 74 3A 20 31 32 37 2E 30 2D 0A 37 01 00

TRANSMIT (request only):
GET /test1.html HTTP/1.1
Host: 127.0.0.1:3516

RESPONSE (total):
STATUS WORD : 91 FE
The fuzzing > Logging

----- SEND DATA -----

TRANSMIT (total): 80 12 00 00 FE

RESPONSE (total): 48 54 54 50 2F 31 2E 31 20 32 30

30 20 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74

68 3A 31 33 37 0D 0A 45 54 61 67 33 22 0D ...

RESPONSE (request only):

HTTP/1.1 200

Content-Length: 137

ETag: “0003”

Content-Encoding: gzip

Content-Type: text/html

STATUS WORD : 90 00
The fuzzing > Parallelization

```
peach.exe -pX,Y ./fuzz_http.xml

-p3,0
-p3,1
-p3,2

XML 1
XML 2
XML 3

Interface
```
The fuzzing > Experimental results

- No HTTP card response when "\r\n\r\n" is not in the request but there is a correct Status Word (90 00).

- SCWS does not compress in GZIP

- PUT method permits to overwrite existing webpages
The fuzzing > Experimental results

----- RECEIVE DATA TERMINAL RESPONSE -----
TRANSMIT (total): 80 14 00 00 3F 01 03 01 42 01 02 02
82 81 03 01 00 36 81 2D 47 45 54 ... 0A 0D 0A 37 01 00
TRANSMIT (request only): GET /index.html HTTP/1.1
Host: 127.0.0.1

RESPONSE (total):
STATUS WORD: 91 B1

----- FETCH (SEND DATA) -----
TRANSMIT: 80 12 00 00 B1
RESPONSE (request only):
HTTP/1.1 200
Content-Length: 85
ETag: “059B”
Content-Type: text/html

<html><head><title>Hello World</title></head>
<body><h1>Hello World</h1></body></html>

STATUS WORD: 90 00
The fuzzing > Experimental results

----- RECEIVE DATA TERMINAL RESPONSE -----
TRANSMIT (total): 80 14 00 00 C0 01 03 01 42 01 02 02
82 81 03 01 00 36 81 AE 50 55 54 20 ... 6C 3E 37 01 00
TRANSMIT (request only): PUT /index.html HTTP/1.1
Host: 127.0.0.1
Content-Length: 85
Content-Type: text/html

<html><head><title>AAAAAAAAAAAA</title></head>
<body><h1>BBBBBBBBBBB</h1></body></html>

RESPONSE (total):
STATUS WORD : 91 20

----- FETCH (SEND DATA) -----
TRANSMIT : 80 12 00 00 20
RESPONSE (request only):
HTTP/1.1 204
STATUS WORD : 90 00
The fuzzing > Experimental results

----- RECEIVE DATA TERMINAL RESPONSE ----- 
TRANSMIT (total): 80 14 00 00 3F 01 03 01 42 01 02 02 82 81 03 01 00 36 81 2D 47 45 54 ... 0A 0D 0A 37 01 00
TRANSMIT (request only): GET /index.html HTTP/1.1 Host: 127.0.0.1

RESPONSE (total):
STATUS WORD : 91 B1

----- FETCH (SEND DATA) ----- 
TRANSMIT : 80 12 00 00 B1
RESPONSE (request only):
HTTP/1.1 200 
Content-Length: 85 
ETag: “059C” 
Content-Type: text/html

<html><head><title>AAAAAAAAAAAAA</title></head>
<body><h1>BBBBBBBBBBB</h1></body></html>

STATUS WORD : 90 00
Conclusions

- First results of fuzzing have shown non-conformance of the HTTP specification
- These results can only be discovered through fuzzing
- Fuzzing is possible on smart cards
Any Questions?