Fuzzing on the HTTP protocol implementation in mobile embedded web server.

Matthieu Barreaud, Guillaume Bouffard, Nassima Kamel, and Jean-Louis Lanet

Smart Secure Devices (SSD) Team – XLIM Labs, Université de Limoges
83 Rue d'Isle, 87000 Limoges, France
{matthieu.barreaud,guillaume.bouffard,nassima.kamel,jean-louis.lanet}@xlim.fr

Abstract. The fuzzing is a technique which allows to generate invalid, unexpected, or random data to supply them in the various inputs of the software or the protocol to be tested. That allows to find situations not expected by the programmers and sometimes to influence the functioning of the target. Our work aims to check implementations of the HTTP protocol in smart card embedded web servers. For that, we have used the fuzzing method to found vulnerabilities and compliance of this sort of web server. Moreover, working on black box forced us to use PyHAT to collect a maximum of information of the target features. Thus, we can reduce the amount of properties to analyze. Our fuzzing program is based on the Peach framework adapted to our needs. Then, with data model and state model of the target, Peach will generate the fuzzing data. We have also defined mutators to represent the mutations types used. Results generated by logs files are finally automatically analyzed to understand the behavior of the application and to detect if some fuzzed data succeed to take up vulnerabilities.

Keywords: smart card web server, fuzzing, embedded HTTP protocol

Acknowledgments

The authors thank Mamadou Balde, Amine Belhocine, Silvère Cainaud, Jérémie Clement, Romain Severin, Nicolas Tarriol and Lylia Tikobaini for their contribution to this work.

1 Introduction

The recent years, the evolution of embedded systems and their complexity have increased steadily. Nowadays, it is possible to embed a web server in a smart card. This technology uses the HTTP protocol and allows the holder to provide services and custom interfaces. Due to constrained resources systems in which this technology is running, it should necessary to test it in order to uncover bugs and other vulnerabilities.

This new technology was announced by Open Mobile Alliance (OMA) which describes the Smart Card Web Server (SCWS) specification based on the version 1.1 of the HTTP protocol for the smart cards dedicated to mobile phones. It is defined for smart cards based on Java Card 2.2 platform and for low-cost devices. In other hand Oracle (formerly Sun Microsystem) proposes the Java Card 3 platform which also offers an embedded web server and new features through the enhancement of the framework with new supported Java API and programming. However, it is dedicated to high-cost devices.

Our work aims are to test the compliance and the robustness of the HTTP protocol implementation of the embedded web servers. Thus, we have choose the fuzzing technique for its effectiveness in auditing different types of applications and systems. Based on a black box model, we have not knowledge of the target implementation. We are mainly interested to test the HTTP protocol implementation. In a previous work [12], we have presented the effectiveness of this technique in the verification of compliance of the BIP protocol (the transport layer) available on a smart card with the specification defined by ETSI (European Telecommunications Standards Institute) [9].

This paper is split in two parts. The first one is preamble, where we present a state of art of the smart card web server, the HTTP protocol and the fuzzing. In the second part, we describe our contribution and explain the main point of our tool: in particular how to generate testing data and a logging interpretation; and we finish with our future works and conclusion.

2 State of the art

2.1 Smart Card Web Server

The Smart Card Web Server (SCWS) is a HTTP/1.1 web server (RFC 2616 standard [10]) in the embedded smart cards. It is used to provide services, personalize mobile interface and to easiness the card administration. The SCWS is both a server, and a client application. In server mode, it is used by the subscriber with a WAP browser whereas in client mode, a Card Issuer may administrate the SCWS from a server [3,7].

In server mode, the card communicates *via* a Bearer Independent Protocol (BIP) channel, which is not the Card Issuer communication channel, allowing the SCWS to be run independently from the Card Issuer network. To communicate with the mobile, the card uses BIP commands [6,9]. These commands are composed of one or many TLV (Tag, Length and Value). They belong to the Subscriber Identity Module (SIM) Application Toolkit (SAT) technology defined by the ETSI [9]. The SAT consists of a set of commands programmed into the SIM which defines how the SIM should interact with the outside. Moreover, SAT initiates commands independently of the handset and the network (proactive commands). In addition to the SCWS application, the whole environment is composed of 1:

– An handset, in which are implemented:
  • A BIP Gateway to ensure the translation of the data format between the SCWS and the WAP browser,
  • The HTTP application or WAP browser which sends request to the SCWS via the BIP Gateway, at the subscriber requirement.
– The Over The Air (OTA) server which stores the Card Issuer data are needed to remotely administrate the SCWS.

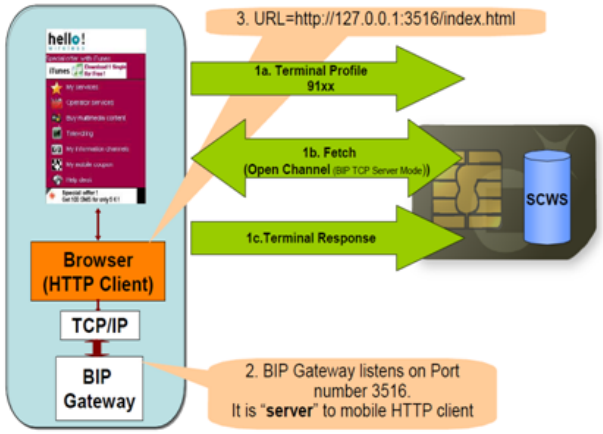

Fig. 1. Communication between the mobile and the SCWS

2.2 HTTP Protocol

The HTTP protocol is encapsulated into the BIP transport protocol. A HTTP message has the following components [10]:

– An URI used to identify the resource requested. The URI must be written as defined in the RFC 2616 format:

```
http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]
```

– A header which contains information as:

```
Message header = field-name ":" [field-value]
```

– A body which contains the HTTP methods:
  • The `GET` method is used to retrieve the resource (in the form of an entity) identified by the `Request-URI`.
  • The `HEAD` method is identical to GET except that the server does not return a body message in the response.

- The `POST` method is used to request the origin server accept the resource enclosed in the request as a new subordinate of the resource identified by the `Request-URI` in the `Request-Line`.
- The `PUT` method requests enclosed static resource be stored under the supplied `Request-URI`. This method is used only static resources.
- The `DELETE` method allows to delete the resource identified by the `Request-URI` field.
- The `OPTIONS` method represents a request to obtain information about the communication options available on the request/response chain identified by the `Request-URI`.

The `TRACE` and `CONNECT` methods are supported by the server applets only:

- The `TRACE` method is used to invoke a remote, application-layer loopback of the request message.
- The `CONNECT` method is used with a proxy that can dynamically switch to being a tunnel.

The first line of the response is always the `Status-Line`, which consists in the protocol version followed by a numeric status code and its associated textual meaning:

- The status code is a 3-digit number, indicating if the request is successfully executed or not:
  - Informational `1xx` indicates a provisional response;
  - Successful `2xx` indicates that the client's request is successfully received, understood and accepted;
  - Redirection `3xx` is not supported;
  - Client error `4xx` is returned for cases in which the client seems to be erred;
  - Server error `5xx` is returned when the SCWS server has erred or is incapable of performing the request.
- The reason gives a short textual description of the status code
- A text message indicating the error type (same as standard reason-phrase)
- An error style sheet (XML output)

2.3 The Fuzzing

Fuzzing is a technique which aims to find errors in software implementations by injecting invalid data [5], [15]. The main goal of fuzzers is to crash the machine, application, protocol, etc. It is the inherent limit of this technique. The main advantage is to search vulnerabilities with a low-cost material. Fuzzing data can be generated in three different ways [13]:

- Random data generation which has the inconvenient to be blind and, in most cases, these data are filtered and rejected by the target.
- The fuzzer generates invalid data from a data model created by the user. Then it sends them to the application or protocol to test. This method is time consuming because it needs to know the protocol integrally but it is the best effective method.

– The mutation fuzzer takes a well known and valid session like a file or a network capture to mute it and send it to the application or protocol to fuzz. This method does not need target knowledge but is limited because it treats only the test file cases.

Most existent fuzzers are designed to test network protocols such as Xiao *et al* [16] which provides a tool to test at TCP/IP layer, but their use is not limited to this unique area. There are many fuzzing frameworks based on APIs that can be used to implement tools for auditing at different levels (files, API, arguments for a command line, standard input, etc.) as Spike [2], Peach [8], Sulley [14] and Fusil [11].

In our research we have chose Peach thanks to its flexibility and because it uses both data and state models, and mutation to generate fuzzing data. This choice allows us to parallelize the fuzzing which reduces time of test. It is very important, in particular while using smart cards.

3 Our fuzzing tool

3.1 Aims

Our fuzzing tool works in the black box model – *ie* we do not have any knowledge on the protocol implementation in the card. We must verify the correctness of the HTTP protocol implementation and analyze the behavior of the smart card. We want to determine when the smart card crashes or has an unexpected behavior. For that, we must analyze the status word and response returned. The status word corresponds to the card state after having executed the request.

Moreover, we want to have a generic data model which represents the HTTP protocol usable on smart card or not. Indeed, our fuzzing tool can fuzz a web server defined by an IP address and a port or an embedded web server in a smart card. Finally, we choose the mutation type by defining an array of string values.

3.2 Representation

Before the fuzzing step we use our application PyHat which searches the implemented features on the card. That permits to reduce the number of cases to treat. This tool creates a XML file used by the fuzzer Peach [8]. Then, we have an interface which makes the link between:

– PyHat, the fuzzer Peach and the targets
– the fuzzer Peach, the targets and the logs

Finally, we have an analyzer tool which permits to identify the vulnerabilities found in the logs.

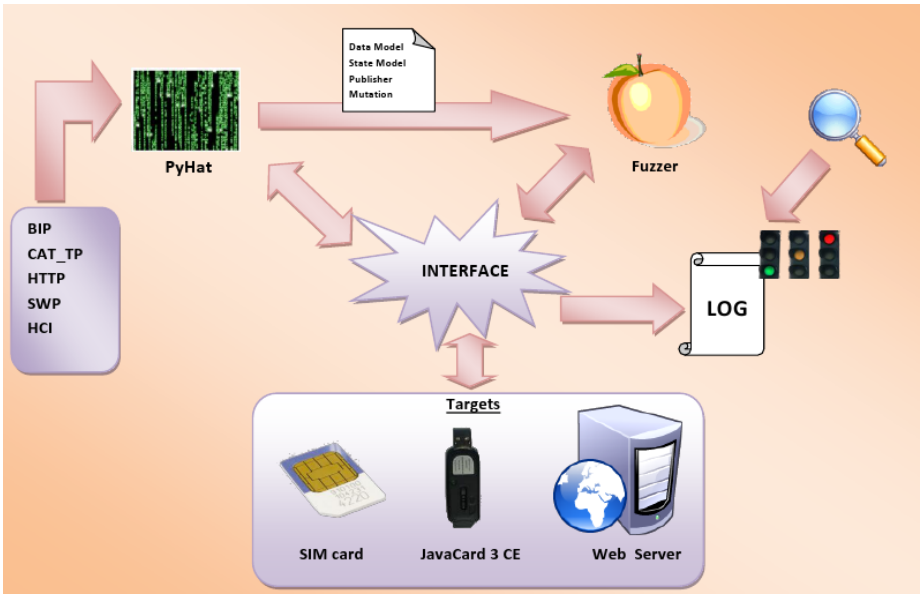

Fig. 2. Schema

3.3 HTTP smart card features

If we fuzz each method and their associated header, we have a huge amount of tests. These tests may spend a huge amount of time with a low bandwidth like with a smart card. In order to decrease this amount of tests, we have developed a tool which search the implemented HTTP features in the server. This tool, PyHAT, for Python HTTP Assessment Tool, may:

- Detect implemented HTTP methods: the HTTP protocol is consists in methods which provide an interaction between the client and the server. The main feature of PyHAT is to find them.
- Detect case sensitive requests: the HTTP protocol specification describes that the sent requests to the server must be case sensitive. Our analysis tool checks if server is compliant them.
- Detect the supported HTTP versions.
- Detect supported encoding data methods: to save bandwidth, the HTTP protocol allows to encode the resource within an encoding method chosen by the client. The RFC defines `gzip`, `compress` and `deflate` algorithms. Another way to get a resource is without encoding. It is the `identity` method.
- Detect server analysis request fields: previously, we explained how a HTTP request is formed. In addition of the first HTTP request line, other fields may be used but they may not be parsed by the HTTP server.
- Find the resources contained in the web-server.
- Output the result in a XML file which may be parsed by our fuzzer.

In order to find the implemented features by the HTTP server, we defined an analysis strategy based on the HTTP protocol specification. In this RFC, the card should, or must, implement deterministic responses for each client request. Our analysis tool checks these responses. Moreover, to list the resources contained by the HTTP server, we recursively check, starting by the index webpages, the URI which references the target.

3.4 Test generation

Peach Peach is a fuzzer written in Python-language which intends to generate invalid data on some protocols. For that, Peach uses XML file containing:

- the data models representing the structure of the protocol,
- the state models recreating the basic state logic needed to test the protocol,
- the publisher describing where the data are sent,
- the mutator representing the mutation types used.

We have modified Peach to realize our needs. For example, we have created a publisher by using the library Pyscard [1] because Peach does not permit to communicate with smart cards. This publisher permits to encapsulate the HTTP request in the BIP protocol automatically. It logs all the APDUs sent and received by the smart card. Moreover, we have implemented basic functions in order to fuzz the HTTP protocol like `choice`, `minOccurs`, `maxOccurs`.

Peach permits to parallelize the fuzzing using the `-p` argument. This argument specifies the total number of card and the card to use. We need to have for each card a XML files where we indicate in the publisher the card to use. The parallelization is the solution of the time consuming problem of fuzzing.

Fuzzing strategy We automate the generation of data and state models corresponding to methods and headers of the XML file created by PyHAT. This tool creates several files, one for each method, for each header and for each card to fuzz. Then, we use the Peach fuzzer with our own modifications to generate random data corresponding to the data models generated by PyHAT. To create invalid data we use string mutations. We have a static array of string containing several special characters. Moreover, we mutate the default string to create an unexpected result.

Finally, in order to find some not expected behavior, we store each sent and response requests in ordered log files. We have a log for each fuzzed method and for each header. That allows us to find the way to an unexpected behavior. Moreover, for each request, we store:

- the HTTP sent and response requests (of course!)
- the BIP commands (if we have an overflow inter-layer)

However, during the fuzzing step, Peach inserts some tags to flag a detected unexpected behavior. Unfortunately, it exists some behavior detected as expected but actually is not compliance with the HTTP specification. So, these false positives are found by our analysis tool.

4 Conclusion

The fuzzing method allows us to discover vulnerabilities or implementation errors in the HTTP protocol. Moreover, with a low bandwidth, smart cards may be fuzz with a preanalysis and, we it possible, a parallelize that. After we verify the HTTP protocol implementation and we have already found some bugs and vulnerabilities. These experimental results will be treated in the extended paper version.

we are also interested on application audit dedicated to smart card embedded web server, in particular the web applications based on servlet. Our goal is to found implementation flaws and web vulnerabilities such as XSS (Cross site Scripting). Our approach is to design a fuzzing tool in white box, using data model and state model generated from a static analysis of the targeted application. There are many static analysis tools such as the `FindBugs` framework [4]. Currently, we are investigating with this framework to verify its compatibility with Java Card applications. We are also interested in the control and data flow management in our analysis. This part will be detailed in the final version of this paper.

References

1. Pyscard - python for smart cards, `http://pyscard.sourceforge.net/`
2. Aitel, D.: An introduction to spike, the fuzzer creation kit. immunity inc. white paper (2004)
3. Alliance, O.M.: Smartcard-web-server. Tech. rep., OMA (2008)
4. Almaliotis, V., Loizidis, A., Katsaros, P., Louridas, P., Spinellis, D.: Static program analysis for java card applets. Smart Card Research and Advanced Applications pp. 17–31 (2008)
5. Campana, G.: Fuzzing : injectez des données et trouvez les failles cachées. MISC pp. 28–37 (SEPT/OCT 2008)
6. Devrient, G..: Bearer independent protocol (bip). Tech. rep., G&D (2006)
7. Devrient, G..: Smart card web server - merging the sim and the world wide web. Tech. rep., G&D (2007)
8. Eddington, M.: Peach fuzzing platform, `http://peachfuzzer.com/FrontPage`
9. ETSI: ETSI TS 102 223 - Smart Cards : Card Application Toolkit (CAT) (2010)
10. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Rfc 2616: Hypertext transfer protocol–http/1.1, june 1999. Status: Standards Track (1999)
11. I.Maceno, M.: The Fusil Project, `http://www.labri.fr/perso/fleury/courses/SS08/download/memoirs/adjej-gunes-maceno-memoire.pdf`
12. M.Barreaud, J-L.Lanet, J.I.C.: Analyse de vulnérabilités sur cartes à puce à serveur web embarqué. SAR-SSI (2011)
13. Miller, C., Peterson, Z.: Analysis of Mutation and Generation-Based Fuzzing (2007)
14. P.Amini, A.: Sulley: Fuzzing framework (2007), `http://www.fuzzing.org/wp-content/SulleyManual.pdf`
15. Takanen, A., DeMott, J., Miller, C.: Fuzzing for software security testing and quality assurance. Artech House Publishers (2008)
16. Xiao, S., Deng, L., Li, S., Wang, X.: Integrated tcp/ip protocol software testing for vulnerability detection (2003)