Techniques for Finding Vulnerabilities in Web Applications

Mihai SĂNDULESCU
IT&C Security Master Program
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
mihai.sandulescu88@gmail.com

Abstract: The current trend is to move everything on the Internet. Because a lot of companies store sensitive user information, security has become mandatory. Usually, software developers don’t follow some basic practices in order to secure their applications. This paper will present in the second chapter, the white-box, black-box and gray-box methods which can be used in order to test applications for possible vulnerabilities. It focuses on fuzz testing, which is a black-box testing method, presented in the third chapter. The fourth chapter presents the stages of a fuzzing test and in the final chapter, we show a basic practical example on how to use the Burp Suite fuzzer to find a vulnerability.

Key-Words: vulnerabilities, fuzzing, black-box, white-box, web application, owasp, burp suite.

1. Introduction

The Internet is by far the biggest network on the planet. Nowadays, companies are using more and more the web applications as tools for delivering services. It has the advantage that the application is installed in one place, and there is no need to be installed at the client. Updates are made very easy on web applications compared to desktop applications where every client needs to apply some patches. But obviously there is a disadvantage: the application is public. The fact that is public is a big advantage but we say disadvantage because there are some peoples, bad intentioned, who try to steal sensitive information. Combined with the unaware of software developers regarding security, it can have a dramatic impact on information privacy of the users. Furthermore, imagine a denial-of-service attack on Amazon’s site. It can lead to unexpected loss of money.

There are some techniques, which we will present in this paper that can be used to test an application for possible vulnerabilities.

2. Testing techniques

If we try to find one technique that can be used to find all the vulnerabilities of an application, there is no ending point is this research. There are no testing methods that can show us all the vulnerabilities of the application that we are testing. Every testing method has its own advantages and disadvantages.

Taking into account the knowledge of the application we are testing, at a high level of categorization, we can divide testing in: white-box testing, black-box testing and gray-box testing.

2.1 White-box testing

White-box testing, also called clear-box testing, structural-testing, as the name suggests, implies the knowledge of internal structure of the application, having access to the code, the specifications of the application and sometimes the possibility to collaborate with the developer of the application.

White-box testing is a structural testing method, where code analysis is made in order to test the different reaction of the code in conjunction with different input values. We can divide the design techniques, used in a white-box testing plan, into: control flow testing, data flow testing, path testing, decision coverage, branch testing and statement coverage.

Advantages of using white-box testing:
- **Good coverage of the application code:** because we have access to the entire code of the application, every code path can be verified for possible vulnerabilities. It can lead to false positives because not all the code is accessible to the exterior of the application.

Disadvantages of using *white-box* testing:
- **Complexity:** it is quite difficult to manually verify the entire code of an application and the tools used to automate the tests can produce a lot of false positives.
- **Big costs:** white-box testing needs a tester who knows the programming language of the application. This can increase a lot the total costs of the test.

### 2.2 Black-box testing

Unlike white-box testing, black-box testing is based on modifying inputs and passing them to the application. There is no need of knowledge of application code or specifications.

![Black box testing](image)

*Figure 1. Black-box testing*

Usually, black-box testing is based on analyzing the reaction of an application to different modified inputs. It is a functional testing of an application which tries to show what the application does in certain circumstances. It is very used in web applications, where we can modify the different fields of the HTML protocol in order to crash the web application or, why not, the entire web server.

Advantages of black-box testing:
- **Reproducibility:** because black-box testing is not aware of the application code, the test can be easily reproduced over different implementations of an application;

**Availability:** a big problem in testing is when we don’t have the code of the application. Because black-box testing is independent of the code, it is always available;

- **Simplicity:** The tester doesn’t need to have knowledge about the programming language that is used is the application;

Disadvantages of black-box testing:
- **Code coverage:** it is very hard to know how much code we have covered in our black-box test. We don’t really know when to stop the test.
- **Difficulty:** although we don’t have to be programmers to make a black-box test, it can be very hard to make complex tests which require multiple variables changes. It’s also a matter of intelligence in choosing the right inputs to manipulate.

A very used technique in black-box testing is fuzz testing which we will describe is the next chapter.

### 2.3 Gray-box testing

Situated between the white-box testing and the black-box testing is the gray-box testing. It has the characteristics of the black-box testing where we can add additional information off the application though the reverse code engineering. Reverse code engineering takes a binary code file and tries to make it human readable. It is used in conjunction with binary code auditing, which is using specialized tools to decompile files to an intermediate human readable language.

3. **Fuzzing**

Fuzzing is a black-box testing method which consists basically in sending malformed input to an application in an automated or semi-automated manner. Sometimes, it is identified with *fault-injection*.

Fuzzing was introduced for the first time in 1988 by Professor Barton Miller at University of Wisconsin Madison. The project “The fuzz generator” was used to...
test the robustness of various UNIX applications sending them some malformed strings. There are two main categories of fuzzing: *mutation-based* and *generation-based* fuzzing.

- **Mutation-based** fuzzing is also called dumb fuzzing because we take the inputs of an application and blindly mutate them, without any knowledge. A well-known method of blind mutation is bit flipping, where the bits are flipped randomly;

- **Generation-based** fuzzing is also called intelligent fuzzing because it consists of generating the input based on some knowledge (the format of a protocol, etc.).

### 3.1 Phases in a fuzzing test

Although there is no well-defined list of a fuzzing test, we can have the following primary phases:

- **Identifying the target**: when we are testing our own application, there is no need of finding the target to fuzz. But normally, the fuzz testing is made by another person, who doesn’t know the application. In this case, it’s very important to identify additional possible libraries that are used in the application, not only the application itself.

- **Identifying the inputs**: is one of the most important phases in the process of fuzzing. Imagine we have a compiled desktop application and we don’t have any information of it. We may need the help of other tools to be capable of identifying the inputs of the application (for example *filemon*). In web application it may be easier to find the inputs. The overall success of the fuzzing test is relies on this phase.

- **Generate data for fuzzing**: after the selection of input fields is made, we must decide which type of data we are generating for fuzzing. It’s also a very important phase in fuzzing like identifying the inputs because the data that is generated can find the vulnerability. Some knowledge of existing vulnerabilities is required in order to succeed a fuzzing test.

- **Execute the fuzzing test**: is where the process of fuzzing takes place. It can consist of sending a modified HTTP request, opening a file with modified parameters, calling a web service, etc. Because we are modifying the parameters and we are generating lot of data, automation of the execution is a must. There are few situations where we can manually fuzz an application: in general, when we are sure that a fuzzing data can discover some vulnerabilities.

- **Watch for any exceptions**: when we are generating big amounts of data, it is crucial to know what modified data caused the application to crash or generated a vulnerability. In general, when a request is made to an application, we are waiting the response in order to make the correlation between the request and response.

- **Determine exploitability**: not all the vulnerabilities of an application are exploitable. When all the data has been sent to the application and all the vulnerabilities were discovered, it is necessary to decide whether these vulnerabilities are exploitable or not. In general, this process requires some security knowledge.

### 3.2 Fuzzer types

Fuzzers are applications that automate the process of fuzzing. They are very useful in any kind of fuzzing. Based on the location of the application, the main types of fuzzers are:

1. **Local fuzzers**: local fuzzers target local applications. The main types of local fuzzers are:
Command-line fuzzers: used to fuzz applications that take parameters passed to the application by the user in a command line environment. Command line fuzzers: iFuzz.

Environment variable fuzzers: are used in applications that use environment variables (for example they use the `getenv` function in UNIX). Sharefuzz and iFuzz are fuzzers of this type;

File fuzzers: are the most used local fuzzers. Nowadays, almost any application works with files so a file fuzzer is a good candidate in finding vulnerabilities of such applications. The basic approach is to modify the input files in different ways in order to generate some exceptions. The files can be mutated or parts of it can be intentionally regenerated. Among file fuzzers we find: FileFuzz, SPIKEfile and notSPIKEfile, PAIMEIfuzz.

2. Remote fuzzers: can be grouped into:
   - Network fuzzers: are used to fuzz simple network protocols like FTP were all the components are human readable strings, and complex protocols, where the fields are ASCII codes not so user friendly. Some network fuzzers are: SPIKE and PEACH.

   - Web fuzzers: are tools that can communicate over HTTP protocol in order to send malformed input to a web application. They can be used to automate the detection of various web vulnerabilities like SQL Injection, Denial of Service, XSS and so on. Among the most popular web fuzzers we have: WebScarab, Burp Suite[9], SPI Fuzzer by SPI Dynamics which was acquired by HP, Web application test tools by Codenomicon.

   - Browser fuzzers: although browser fuzzers represent a special category, they are related to web fuzzers. Browser fuzzers can be used to change Component Object Model (COM) objects, CSS files, HTML parsing, server response headers, graphics, etc. Useful browser fuzzers include: mangleme, CSSDIE, COM Raider, Hamachi.

4. Fuzzing in web applications

The fuzzing of web applications implies the modification of the HTTP protocol. It is related to network fuzzing and the targets are web application and the web servers at the same time. When fuzzing a web application we must send over the network modified HTTP request, wait the web server to process the request and the capture the response from the server. Because we need to send a lot of requests in a fuzzing test, it will be good to have a fuzzer that can make multiple requests at the same time using multiple threads. This is one of the criteria of choosing Burp Suite[9] in the hands-on experiment of this paper.

In general, the HTTP protocol is made of name-value pairs which are quite simple to include in a fuzz test. In reality, you must have some security knowledge in order to know what values to choose to discover vulnerabilities.

The main variables that can be fuzzed in a web application are:

1. **The used METHOD**: defines the action to be executed by the web server. The most used methods are GET and POST, which are user to retrieve resources from the server. Other methods used are: PUT, DELETE, HEAD, TRACE, OPTIONS. The OPTIONS method is used to ask the server which methods it can support. Because there are already known vulnerabilities of some methods, having the OPTION method enabled...
on the server can be considered vulnerability.

2. **The basic URL**: represents the location of the asked resource on the web server. There are two types of URLs: absolute URLs where the domain is specified (http://www.oneexample.com/mypage.html) and relative URLs where the domain is not specified (/mypage.html). In general, the relative part of an URL can be divided into: 

```
[path]/[page]
```

extension?name=[value]&name=[value]. This representation makes it very easy to find the inputs in a fuzzing test where the inputs are the words shown in bolded font. Also we can fuzz the separators used is this structure like “?”, “=” , “&”. One advantage in absolute URLs is that we can fuzz the subdomains of an application in order to find additional information.

3. **The used PROTOCOL**: actually is the version of the HTTP protocol that is used is the request. It is known that old versions of the HTTP protocol have some vulnerabilities, so fuzzing the version of it can be useful in a fuzzing test.

4. **The HEADERS**: HTTP headers are used to customize the requests made to the server (we can ask for a file, put it in the browser cache, etc.). The headers have the following structure:

```
[Header name]: [Header value]
```

All the header names, values and separators (”,“) can and should be fuzzed.

5. **The COOKIES**: the HTTP protocol is a stateless protocol where each request is not aware of another request that have been made. For this reason, there were developed some mechanisms in order, for example, to retain the login of a user, so that user doesn’t have to login on each request. A very used method is sending to the user a cookie token that uniquely identifies that user. It can be stored in clear or encrypted in the cookies of the browser. Because the cookies can contain sensitive information, they are real targets in the fuzzing process of testing the application.

The main structure of a cookie is:

```
Cookie: [name1]=[value1];[name2]=[value2];
```

As with the headers, cookies can easily be fuzzed.

6. **The POSTed DATA**: as you probably already know, there are two main ways to pass parameters to the web application: through the query string part of the URL using the GET method, or by the posted data in the headers through the POST method. The values are sent in the following format:

```
[name1]=[value1]&[name2]=[value2]
```

The parameters sent to the application through the POST or GET methods are the most important inputs to fuzz in order to find vulnerabilities in a web application because they have direct access to the code of that application.

5. **Hands-on experiment**

In this example we will use Burp Suite[f] fuzzer from Portswigger Security in order to show how to use a fuzzer to find vulnerabilities in web applications. Because configuring all the HTTP headers of a request can be very difficult and needs very good knowledge of the protocol, usually the web fuzzers implement a proxy, used as a “man in the middle” tool between the browser and the web server, to capture web traffic.
With this architecture, we are able to modify requests and responses in real time, if we want to. Like almost all the web fuzzers, Burp Suite follows this rule. It consists of multiple plugins used to find vulnerabilities in web application but the actual fuzzer is called Intruder.

But let’s have a look on how to use Burp Suite to find some vulnerabilities.

The first step in the fuzzing process is to configure the web browser to use the Burp Suite’s proxy. In order to find the proxy’s configuration, we must go under the “Proxy” tab - “Options” sub-tab in Burp Suite.

In this case, we have our proxy running on localhost (127.0.0.1) on PORT 8080. As you can see the Running checkbox, our proxy is running without any problem. If you have any problem with it, restart Burp Suite or watch for any other tools/web applications running on port 8080 (in this case). Next, we must configure our browser to user this proxy, such that all the web requests we make pass through this proxy.

For Internet Explorer we must go under “Tools -> Internet Options -> Connections -> Lan Settings” and configure the proxy with the settings found in Burp Suite. Now we can use Firefox or Google Chrome to navigate some pages in order to capture the requests. At the moment of this paper, there is a bug with using the Internet Explorer proxy’s. When we configure IE’s proxy, the others browsers (which are configured to take global settings of the proxy) work very well. Also be aware about the live interception in Burp Suite under Proxy -> Interception. If intercept is on, you are asked to change the request or you can just forward it.

As you can see in Figure 4, we are making a request on a local web application, on port 8000, requesting the Messages.aspx page. For this paper we have created a vulnerable application in term of a demo forum which we will use to demonstrate how to fuzz a web application.

For defining the target, we go under the Target tab -> SiteMap. Here we can see all the requests that the browser did, whether is a local request or a request to an external resource.

Now, we must locate the request that we want to fuzz. Once located, right click on the request and choose “Send to Intruder”, as shown in the next picture.
Once we have selected our target, it’s the right moment to find any inputs that we think they can generate some exceptions. To watch the HTTP protocol that was generated by the application we must go under the Intruder’s tab. It’s actually the most important tab in the fuzzing test. The first sub-tab is Target which is already configured with the Host: localhost and the Port: 8000. Under the Positions sub-tab we find the HTTP protocol used in the request. The input variables used in the fuzzing test are delimited by two “§” symbols. As we can see, Burp Suite[9] has already selected for us two inputs to be fuzzed: the value of “TopicId” posted parameter and the value of the “ASP.NET_SessionId” which is the session token.

For the purpose of this paper, we don’t need to fuzz the session token or analyze it to check his randomness. We will select the “§kju10zwte5qvofm4ta1gb302§” value (in this case) and press the “Clear §” button. The attack type is not important for a single input to fuzz. I won’t describe the different types of attacks that we can make because it’s not the purpose of this paper.

As we can see, we have the parameter TopicId which has the value of 1. It’s not very hard to find out that changing the value of 1 with a string value or a big number can cause the application to crash. There are a lot of developers who don’t validate the parameters before using them. Let’s change this value with some string values. In order to do that we must go under the Payloads tab. Burp Suite[9] offers many possibilities in order to manipulate the input values. We can use text files, string generators, number generators and so on. For our test, a “Simple list” type is sufficient to test the TopicId with some string values. To add the values, we use the Payload Options bloc, in the Add text box we enter our values and press Add button. For the purpose of this test we have added simple string values.

We can see in the “Request count” field that the total number of requests is 5. Now that we have defined our attack surface, we can begin fuzzing. To start the automated fuzzing, we choose Intruder -> Start attack from the upper menu. A new window will appear that shows all the requests that are made. After the fuzzing process is finished, we can look at the requests made by the fuzzer and the status response of the server. In our test, as we expected, the web server respond with a “500 Internal server error” for the requests where TopicId has a string value. Furthermore, for each request we can see the request itself and its associated response from the server. In the response from the server, we found out that the application sent detailed information about the error that has occurred: “Input string was not in a correct format”. This is again another vulnerability because sensitive information is sent to a possible attacker.

6. Conclusions

As we have seen so far, it is very easy to make mistakes in developing an application. There are a lot of applications that we can use to test our code, or
other’s code, through a simple fuzzing test. The example that we have shown in this paper is a basic example, but we can find a lot of applications that we can use this kind of simple fuzzing to find possible vulnerabilities. There are also more complex tests that can be made with Burp Suite\cite{9} or with other web fuzzers. For example, Burp Suite\cite{9} Pro has a “Sequencer” plugin that can be used to test the randomness of different selected inputs. It can be very useful to test the randomness of a session token and calculate the strength of the algorithm used.

Acknowledgement

Parts of this research have been published in the Proceedings of the 7th International Conference on Security for Information Technology and Communications, SECITC 2013.

References

[2] Paco Hope, Ben Walther, Web security testing cookbook – Systematic techniques to find problems fast, O’Reilly, 2008;