

Web Application Penetration Testing: An Introduction

Andrea Hauser

Offense Department, scip AG

anha@scip.ch

<https://www.scip.ch>

Marc Ruef (Editor)

Research Department, scip AG

maru@scip.ch

<https://www.scip.ch>

Abstract: Web application penetration testing always requires good preparation. Defining the customer's scope and expectations is essential for a successful test session. In addition to manual testing, automated tools are always used to help find the "low-hanging fruit". What is most important, however, is the tester's intuition.

Keywords: API, BlackHoodie, Browser, Burp, Exploit, Firefox, Linux, Nmap, OWASP, Penetration Test

1. Preface

This paper was written in 2019 as part of a research project at scip AG, Switzerland. It was initially published online at <https://www.scip.ch/en/?labs.20191024> and is available in English and German. Providing our clients with innovative research for the information technology of the future is an essential part of our company culture.

2. Introduction

This is a supporting article related to my talk held on October 21, 2019, on *Blackhoodie @HackLu* [1]. The talk provides an introduction to web application penetration testing from my personal perspective. I begin by introducing the necessary preparations for web application penetration testing and then discuss how to carry out a web application penetration test.

3. Preparation phase

Before getting started with a web application penetration test, you first need to set up a testing environment and define the scope. At the very beginning, I also start thinking about the reporting method and prepare a checklist of everything I am going to test. This is to make sure I don't forget anything during the testing process and also to remind myself again of what needs to be tested as well as to address any potential flaws before starting the test.

It's equally important to prepare a test procedure with a checklist and to ensure that the tools are up-to-date and ready to use. When performing a web application penetration test, I use the following tools at a bare minimum:

- *Kali Linux* [2]
- Browsers, usually Firefox and Chrome
- *BurpSuite Professional* [3] or the free alternative *OWASP ZAP* [4]
- *nmap* [5]

Scoping is also necessary at the very beginning of the project, so that you are on the same page as the customer in

terms of how the web application penetration test will be performed and what it should achieve. Scoping is completed together with the customer during a kick-off meeting. The following questions must be addressed by the kick-off meeting, at the very latest:

- What's going to be tested (target URL, user roles, etc.)?
- Is the web application accessible externally?
- If the web application cannot be accessed externally, where exactly should the test be performed?
- How much information will be given to the tester?
- Will it be a black-box, gray-box or white-box test?
- Will the tester be given access to the documentation or source code?
- If access to the documentation or source code is allowed, when will the tester receive the documents?
- How many different user roles are there?
- Will the tester be given access to test accounts, or can they create these accounts themselves?
- When will any test accounts be created and the tester given access to them?
- When will the test take place?
- How long will the test take?
- When can the first results or first draft of the report be expected?
- Why is the web application penetration test being performed?
- What does the customer expect to achieve with the web application penetration test?
- What is to be excluded from the test?
- Are there any areas not covered in the scope?
- Who is the designated contact person during the test?
- Who is to be notified if any serious vulnerabilities are found?

All of the main items should be clarified before starting the web application penetration test.

4. Procedure for the web application penetration test

There are plenty of good resources available for web application penetration testing. I mainly use the resources provided by OWASP [6], including the *OWASP Testing Guide* [7] and the *OWASP application security verification standard* [8].

Before I start testing for specific vulnerabilities, I first take an overall look at each web application and try to interact with as many elements of the web application as possible. This includes evaluating differences between various user roles. While browsing, my proxy is running the whole time in the background. This allows me to gain the most comprehensive overall picture possible from the history of all of my requests and responses to them. The first time I browse through the application, I take mental notes of potentially interesting possibilities for interaction, such as login screens and search fields. Then I browse through the application a second time and more closely examine the requests that are sent as well as their responses. This also allows me to discover interesting targets for manual testing.

After I have my first overall impression, I start testing the items previously identified in order to find vulnerabilities. At the same time, I run automated tools on a range of selected endpoints. The tester should not shy away from using automated tools when performing web application penetration tests, as these can help to find the “low-hanging fruit” at an early stage, allowing the tester to focus on more complex tasks. When using automated tools, however, I always make sure to log all executed requests and responses, so that I can briefly review them after running the tool. Sometimes there are unexpected responses that the automated tool does not recognize. This is where I come in as the tester and examine everything manually.

My procedure for web application penetration testing can be demonstrated with the category *A1 – Injection* [9] from the *OWASP Top 10 – 2017* [10] list of security risks.

4.1. A1:2017 – Injection

This category encompasses a very broad range of tests and includes all vulnerabilities that an attacker can exploit by using manipulated data to execute a command. These include manipulated SQL statements, operating system commands and many other vulnerabilities. Vulnerabilities of this sort should be identified using an SQL injection procedure.

In this sample scenario, I will test a web shop. This web shop allows products to be filtered by category. While first browsing through the web shop, I discovered that a normal filter-by-category request looks like this:

```
https://example.com/filter?category=Lamps
```

As the tester, I noticed here that the category filter could land in the WHERE clause of an SQL statement. This is a spot where I will definitely attempt an SQL injection. First, I try using the classic SQL injection ' OR 1=1--. A request would then look like this:

```
https://example.com/filter?category='%20OR%201=1--
```

Next, I compare the response from the injection with the response from an unfiltered request. If, for example, the number of products displayed in the response has changed, I can assume that the SQL injection worked.

From the back-end perspective, the whole sample scenario would look like this:

```
String category =
request.getParameter("category");
String query = "SELECT * FROM products WHERE
category = ' " + category + " ' AND released =
1";
```

In the case of an attack, the executed SQL statement would ultimately look like this:

```
SELECT * FROM products WHERE category='' OR
1=1--
```

to bypass the limiting AND released = 1.

More examples are covered in our article *SQL injection* [11]. A good tutorial, including several labs that let you test out vulnerabilities yourself, is available in the *Portswigger Web Security Academy* [12].

To close this case fully, I will also show how the vulnerability could have been prevented. SQL injections can be prevented, for example, with prepared statements. In the above case, that would specifically look like this:

```
String category =
request.getParameter("category");
String query = "SELECT * FROM products WHERE
category = ? AND released = 1";
PreparedStatement pstmt =
connection.prepareStatement(query);
pstmt.setString(1, category);
ResultSet results = pstmt.executeQuery();
```

Injects can generally be prevented by using secure API calls, whitelists and specific language-dependent escape characters. For more detailed information, refer to the *Injection prevention cheat sheet* [13].

5. Closing remarks

This article has hopefully provided some basic insights for anyone interested in the procedures for web application penetration testing. Careful preparation for the actual technical testing is important and first requires defining of the scopes and objectives of the web application penetration test. When performing the test, the tester should not only possess technical expertise, but also the intuition to identify areas of interest. Even a person who is not an IT security professional can acquire this intuition by monitoring the network requests sent every time they visit a website. By visiting several websites, they will gain a sense of how a “normal” website behaves.

6. External Links

- [1] https://www.blackhoodie.re/HackLu_schedule/
- [2] <https://www.kali.org/downloads/>
- [3] <https://portswigger.net/burp>
- [4] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- [5] <https://nmap.org/download.html>

[6] https://www.owasp.org/index.php/Main_Page
[7] https://www.owasp.org/index.php/OWASP_Testing_Project
[8] https://www.owasp.org/index.php/Category%3AOWASP_Application_Security_Verification_Standard_Project
[9] https://www.owasp.org/index.php/Top_10-2017_A1-Injection

[10] https://www.owasp.org/index.php/Category%3AOWASP_Top_Ten_Project
[11] <https://www.scip.ch/en/?labs.20190912>
[12] <https://portswigger.net/web-security/sql-injection>
[13] https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html