CS5331 Web Security Assignment 3: Web Scanner

By Jeremy Heng

Last Updated: April 10, 2018

Contents

1 Introduction **1**
1.1 Vulnerability Categories 2

2 Administration **2**
2.1 Deadline 2
2.2 Grading Scheme 2
2.3 Contact 3

3 Public Benchmarks **3**

4 Final Assessment **3**

5 Scope and Formats **4**
5.1 SQL Injection 4
5.2 Server Side Code Injection 5
5.3 Directory Traversal 6
5.4 Open Redirect 6
5.5 Cross Site Request Forgery 7
5.6 Shell Command Injection 8

6 Exploit Scripts **8**

7 Scanner Structure **8**
7.1 Crawling and Identification of Injection Points 8
7.2 Payload Generation 9
7.3 Payload Injection 9
7.4 Generate Exploit Scripts 9

1 Introduction

In this assignment, you are expected to implement a **web vulnerability scanner** from scratch, within reason. The scanner can be implemented in any programming language you wish to use and you may use any open source libraries. However, the core scanning logic must be implemented by you. The assignment will be done in your teams.

Your objective will be to:

1. Crawl for available web endpoints and URLs.
2. Determine if the endpoint is vulnerable.
3. Generate an exploit if the endpoint is vulnerable.

The final assessment will be an **on-site test** in which your team is given three hours to find as many bugs as you can with your scanner. You may tweak your scanner as much as possible during this period.

1.1 Vulnerability Categories

Your scanner should discover the following vulnerability classes:

- SQL Injection
- Server Side Code Injection
- Directory Traversal
- Open Redirect
- Cross Site Request Forgery
- Shell Command Injection

2 Administration

2.1 Deadline

In the weeks leading up to the final asessement, you will be given time to develop and test your scanner against the public benchmarks. (See below for information about the benchmark)

The final assessment will be held before your exam week on 24 April during Reading Week at 6.30pm. The venue will be announced later.

2.2 Grading Scheme

This assignment comprises of 30% of your total grade. The final assessment will have a maximum of 21 points.

Each vulnerability category is assigned a maximum of 6 points. Each vulnerability found is granted 1 point for a successful detection and 1 point for successful generated exploit. You may score up to the maximum of 6 points for a category. You may pick any of the categories to target. Thus, one possible way to get a full score is to find and successfully exploit 3 bugs from three of the categories. You do not need to score fully for each of the categories and are allowed to focus on the categories you are most comfortable with. You can score a

maximum of 18 points for this section. However, we encourage you to attempt finding as many vulnerabilities as you can from all of the categories.

- SQL Injection: 3 x 2
- Server Side Code Injection: 3 x 2
- Directory Traversal: 3 x 2
- Open Redirect: 3 x 2
- Cross Site Request Forgery: 3 x 2
- Shell Command Injection: 3 x 2

A further 3 points will be given to a post-assessment writeup to describe your implementation.

Additionally, your scanner should not raise too many false positives. If the TA deems that the scanner is raising too many of such false positives, **up to 5 points may be deducted**.

2.3 Contact

If you have any queries, please contact jeremy@u.nus.edu.

3 Public Benchmarks

A public benchmark containing vulnerabilities representative of the in-scope vulnerability classes will be released for you to test your implementation against the base standard. The access details to these benchmarks will be released in a separate document along with download links to the virtual machine.

Please note that these benchmarks will probably not be sufficient for the final assessment. Part of your task will be to develop **your own benchmarks**.

4 Final Assessment

The assignment will be graded during a final **on-site three hour assessment**. The time is provided for you to engage your scanners against target web applications. No source code to these private benchmarks will be provided.

You will be provided with:

1. Hostnames of targets (no SSH access).
2. Hostname of scanning machine (granted SSH access).

The target will be accessible over the internet. Please restrict your scanning only to the provided hostname.

The scanning machine will be running Ubuntu 16.04 and will be guaranteed to be geographically close to the targets. You may use the scanning machine to launch your attacks but this is not required.

Some of the target web applications will require authentication in certain places. You are allowed a measure of manual intervention during the assessment. For instance, you may supply your scanner with either your credentials or a cookie.

Access to the two machines will only be granted during the final assessment.

Once the assessment begins, you will **configure and begin scanning** of the target machine. In the event you require a reset or restore of the target machine, please contact the available TAs. Please avoid getting into this situation where possible as restoring the virtual machines will eat into your time.

Before the assessment ends, please ensure you have the following things prepared:

1. All logs produced by your scanner including details about endpoints discovered and crawled.
2. The required result files described in the 'Scope and Formats' section.
3. A list of vulnerable endpoints.
4. The automatically generated exploits.
5. The scanner scripts/source code.

Once the time is up, please zip up (or tar.gz) your material with the group name (G01.zip) and upload it immediately to the IVLE once the assessment is complete.

The grading will then begin with the assessor going to each group to validate the findings. You are required to **explain briefly how your scanner works and to demonstrate that your auto-generated exploits work**. You will be required to show proof that the vulnerabilities were detected through the scanner and that the exploits were generated automatically so please keep your logs on hand.

After the assessment, you are required to do a writeup on the following things:

1. A brief walkthrough of your code and the design decisions.
2. A description of any tweaks you performed over the course of the on-site assessment.

This writeup is to be submitted to the IVLE submission folder (Student Submission/Assignment 3 Submission) by the next day in the PDF format and titled with your group name (G01.pdf).

5 Scope and Formats

This section details the scope of the vulnerability classes and the required result files your scanner is to produce. You may add more fields or data to the result files as you desire.

The objective of these result files are to capture the essential data about the vulnerable endpoint and to express the information required to develop an exploit. These files are **not the exploits**.

5.1 SQL Injection

Scope The following SQL engines may be used: MySQL and SQLite. The objective is to inject an SQL query to the backend database to perform arbitrary lookups or writes.

Verification The success condition is to demonstrate that sensitive information may be leaked from or that information may be written to the database. Other more impactful conditions such as obtaining remote code execution is also acceptable.

Format The result output by your scanner should be in the JSON form:

```
{
   "class":"SQL Injection",
   "results":{
      "https://target.com":[
         {
            "endpoint":"/search",
            "params":{
               "key1":"value1"
            },
            "method":"POST"
         },
         ...
      ]
   },
   ...
}
```

5.2 Server Side Code Injection

Scope This class covers local file inclusion, remote file inclusion, and PHP code injection. In essence, the class involves allowing an attacker to arbitrarily control what a web server renders or executes.

Verification The success condition is to demonstrate that arbitrary or controlled code can be executed by controlling the path of a file or injecting server side code. The objective is to run `uname -a` on the target system or to obtain a reverse shell. If this is not possible, you may attempt to retrieve the source code of the `index.php` file, instead.

Format The result output by your scanner should be in the JSON form:

```
{
   "class":"Server Side Code Injection",
   "results":{
      "https://target.com":[
         {
            "endpoint":"/main.php",
            "params":{
               "key1":"../../uploads/shelled"
            },
            "method":"GET"
         },
         ...
      ]
   },
   ...
}
```

5.3 Directory Traversal

Scope The objective is to obtain the contents of an arbitrary file on the target system.

Verification The success condition is to demonstrate that you can display the contents of `/etc/passwd`.

Format The result output by your scanner should be in the JSON form:

```
{
   "class":"Directory Traversal",
   "results":{
      "https://target.com":[
         {
            "endpoint":"/image.php",
            "params":{
               "key1":"../etc/passwd"
            },
            "method":"GET"
         },
         ...
      ]
   },
   ...
}
```

5.4 Open Redirect

Scope The objective is to coerce a browser to navigate to a controlled URL.

Verification The success condition is to demonstrate that you can redirect the victim to `https://status.github.com/messages`.

Format The result output by your scanner should be in the JSON form:

```
{
   "class":"Open Redirect",
   "results":{
      "https://target.com":[
         {
            "endpoint":"/redirect.php",
            "params":{
               "url":"https://attacker.com"
            },
            "method":"GET"
         },
      ...
    ]
  },
  ...
}
```

5.5 Cross Site Request Forgery

Scope Carry out a sensitive operation on behalf of other users by exploiting the lack of a proper CSRF token or other CSRF protection mechanisms. This coercion is done from the context of a domain the attacker controls. What constitutes a sensitive operation or data is subjective and is left up to the team to distinguish. Note that forms that submit non-sensitive data may be allowed to be left unprotected by CSRF protections.

Verification Validation of this attack will be left up to the teams discretion. One possible way to demonstrate that the exploit works is to show that a user's data changes in the web application when submitting a form as the user on a domain the team controls.

Format The result output by your scanner should be in the JSON form:

```
{
  "class":"CSRF",
  "results":{
    "https://target.com":[
      {
        "form_page":{
          "endpoint":"/update_profile.php",
          "form_id":"form1",
          "params":{
            "user":"admin"
          },
          "method":"GET"
        },
        "action":{
          "endpoint":"/update.php",
          "params":{
            "name":"New Name",
            "age":15
          },
          "method":"POST"
        }
      }
    ],
    ...
  },
  ...
}
```

5.6 Shell Command Injection

Scope Execute shell commands by injecting them into vulnerable endpoints.

Verification The objective is to execute `uname -a` or obtain a reverse shell.

Format The result output by your scanner should be in the JSON form:

```
{
   "class":"Command Injection",
   "results":{
      "https://target.com":[
         {
            "endpoint":"/ping",
            "params":{
               "url":"https://attacker.com; uname -a"
            },
            "method":"POST"
         },
         ...
      ]
   },
   ...
}
```

6 Exploit Scripts

In addition to the result files above, you are also required to automatically generate standalone script files to launch the attacks. These script files will be used during the final assessment to validate your findings.

7 Scanner Structure

The following is a suggested architecture for you to follow when building your scanner. It is not compulsary to implement it in this fashion. The scanner is split into four components.

7.1 Crawling and Identification of Injection Points

Crawling Web Pages The aim of this phase is to find out as many pages in the web application as possible. The crawling function takes in the start URL, the depth to crawl (optional), maximum number of URLs (optional). You can use open source crawlers like Scrapy or implement your own crawler. We encourage you to use Scrapy, so that you can later on discuss on the forum about how to do things with Scrapy. Meanwhile, there are tons of helpful tutorials about how to use Scrapy on the Internet. An important thing to note is that some pages of an application will only be available in certain contexts such as when you are logged in.

Identification of Injection Points After crawling a web page, you will need to identify injection points in each of the crawled pages. An attacker can inject his attack vector at numerous points. Some of the injection points are (but are not limited to):

- GET requests parameters
- POST requests parameters
- Request Headers (including cookies)

7.2 Payload Generation

This phase is specific to the vulnerability category. These payloads will be injected in the injection points that you have identified in the previous phase. For example, when testing for SQL Injection vulnerabilities, the scanner could take in information of all possible database types, versions and the current server information to return the payloads.

7.3 Payload Injection

The next step would be to inject every payload generated in phase 2 into your injection points discovered in phase 1. After each injection, you need to find out which of these (injection point, payload) pairs are exploitable. The output of this phase would be the list of confirmed exploits in the website.

7.4 Generate Exploit Scripts

For each of the exploitable (injection point, payload) pairs you found in phase 3, you need to generate an automated script which proves that the exploit is indeed vulnerable. Typically, you will have to demonstrate that you can achieve the scope of vulnerability category. Failing to automate any component in this script will not get you the full grade.