Security Lab – Security Testing Tools

VMware

- This lab can be solved with the **Ubuntu image**, which you should start in networking-mode **Nat**. The remainder of this document assumes you are working with the Ubuntu image.
- Solving parts of this lab can also be done without the image, but this requires you to install several software components on your own system. It's therefore strongly recommended to solve the entire lab using the image.

1 Introduction

In this lab, you'll learn about the possibilities of automated testing of software with respect to security. If possible, you should use this in your projects or when analyzing software of others in the context of penetration tests because they offer a good and efficient way to find at least some vulnerabilities with little effort.

There are different possibilities to test software in an automated way. Here, we look at the following two approaches that dominate in practice:

- **Black-Box Security Testing of the running program**: In this case, the testing tool communicates / interacts with the running program, using the standard communication interface, e.g. HTTP with web applications.
- **Static Code Analysis**: Here, source code or executable (compiled) code is analyzed. The software to test is not run during the analysis – therefore the name «static analysis».

In this lab, you'll use the following two tools:

- **Arachni**[1]: Arachni is one of the most powerful open source web application vulnerability scanners. Arachni belongs to the category of black-box security testing tools that analyze a web application with respect to security by interacting with it via HTTP.
- **HP Fortify Source Code Analyzer (SCA)**[2]: Fortify SCA is a commercial static code analysis tool, which supports several programming languages. We have a license to use the full functionality of the tool for educational purposes, but you are not allowed to use the tool for other purposes (e.g. for your student projects or external projects).
    - Sidenote: The reason we are using a commercial tool is that the free / open source market does not really offer a good static code analysis tool to find security bugs. If you want to try one for Java code, we recommend Findbugs[3]. Findbugs is an open source tool to find vulnerabilities in Java code. It's one of the few freely available static code analysis tools that can find some security-relevant vulnerabilities, but Fortify SCA is a more powerful tool.

The goal of this lab is that you get familiar with these two tools, that you can use them, that you can interpret their results, and that you understand the benefits but also the limitations of such tools.

2 Task 1: Analyzing Marketplace_v01 with Arachni

In the first task, you are performing an analysis of Marketplace_v01 using Arachni. Marketplace_v01 corresponds to the first (basic) version of the Marketplace application that was introduced in the lecture.

[1] http://www.arachni-scanner.com

[2] https://www.hp.com

[3] http://findbugs.sourceforge.net

2.1 Basis for this task

To run Marketplace_v01 on the image, do the following:

- Download *Marketplace_v01.zip* and *Marketplace.sql* from OLAT.
- Move the files to an appropriate location (e.g. in a directory *securitylabs* in the home directory */home/user*).
- To create the database scheme and the technical user used by the Marketplace application to access the database, do the following (this can be repeated at any time to reset the database):
  - Open *MySQL Workbench*.
  - Click on the left on *Local Instance 3306* and enter *root* as password.
  - Choose *Open SQL Script...* in the menu *File* and select the downloaded file *Marketplace.sql*.
  - Click the *Execute* icon.
- Unzip *Marketplace_v01.zip*. The resulting directory *Marketplace_v01* contains a Java EE project based on Maven. This should be importable in any modern IDE. The remainder assumes you are using NetBeans, which is installed on the Ubuntu image.
- Start *NetBeans* and open the project.
- To build the project, right-click *Marketplace_v01* in the *Projects* tab and select *Clean and Build*.
- To run *Marketplace_v01*, right-click *Marketplace_v01* in the *Projects* tab and select *Run*. If the application server (Payara) is not yet running, it will be started, which takes some time.
- Under the *Service* tab and expanding *Servers*, the Payara server is listed (it is listed as *GlassFish Server*). Sometimes, it may be necessary to restart Payara, do this with a right-click and *Restart*. Under *Applications*, you can also undeploy applications if necessary.
- The application is reached with *http://localhost8080/Marketplace_v01*.

2.2 Scanning the Application

Start Arachni by entering the following in a terminal as *user*:

```
/opt/arachni/bin/arachni_web
```

Arachni provides a web interface. Use the browser to access Arachni:

```
http://localhost:9292
```

Log in with the default user credentials: *user@user.user* with password *regular_user*.

Starting from a base URL, Arachni first tries to discover resources of the web application and then starts looking for vulnerabilities. To get the most out of Arachni you have to invest quite some time to understand all its configuration options and configure it for your needs. This significantly goes beyond the scope of this lab. But for a first scan, the default settings work quite well.

At the top, select *Scans* → *New*. As *Target URL*, enter the URL below. You must use the external IP address of your virtual host as Arachni does not accept *localhost*. Determine this address with *ifconfig* and use it instead of *192.168.57.130* in the URL.

```
http://192.168.57.130:8080/Marketplace_v01
```

Hit *Go!* to start the scan. The scan will take a few minutes to complete. As a result, several detected vulnerabilities are listed, ordered according to their criticality.

2.3 Analysis of the Results

With automated security testing tools, it's always important to analyze the results to verify which of them are actual vulnerabilities (*true positives*) and which are false alerts (*false positives*). In the fol-

lowing, you will do this. Focus on the issues with a criticality of at least *low*, i.e. ignore the *Informational* issues for now.

Consider the following when analyzing the issues:

- The blue button with the question mark next to the identified vulnerable URLs can be clicked to get additional details. It's absolutely necessary that you analyze these details to truly understand what's going on. What's often very helpful is the information about *Injected seed* and *Proof*.
- Sometimes, there are multiple entries that identify the same vulnerability. For instance, with Marketplace_v01, Arachni reports a Cross-Site Scripting issue with *search.xhtml* twice because it has been found by Arachni with two different approaches.
- Arachni often does not present an immediate exploit but merely a proof of concept that a specific vulnerability exists. For instance, in the case of a Cross-Site Scripting vulnerability, this may mean that the injected data by Arachni is not sufficient to trigger the vulnerability, but provides «strong hints» for Arachni that the vulnerability is present.
- During the analysis of the results, you should try to verify the reported vulnerabilities by directly interacting with the tested application. This is especially important in the case of injection-based vulnerabilities (SQL Injection, Cross-Site Scripting etc.). In these cases, you should therefore always send the proof of concept injection data as reported by Arachni to the application to verify if the behavior / response of the web application corresponds to what is reported by Arachni. If this does not convince you that the reported vulnerability is a true / false positive, you should do a more detailed manual analysis. For instance, in the case of a reported Cross-Site Scripting vulnerability, it is often reasonable to verify that Javascript code can truly be injected into the browser (e.g. code using the *alert* function to show a pop-up window). Another possibility to verify a vulnerability is by analyzing the source code. In general, you should do «as much manual analysis as required so you are convinced that the issue is a true positive or false positive».
- If Arachni reports issues that you don't know yet (e.g. missing HTTP headers), browse the Web to get additional information.
- If you have to analyze the traffic in details, *Burp Suite* is installed on the image:
  - Start Burp Suite it in a terminal (as *user*):
    `java -jar /opt/Burpsuite/burpsuite_free_v1.7.03.jar`.
    After startup, select *Temporary Project* on the first screen and *Load from Configuration File /opt/Burpsuite/securitylab.json* on the second screen. This makes sure the proxy listens on port 8008.
  - Burp Suite must be used by the browser as a proxy and with the installed Firefox add-on *FoxyProxy*, activating a proxy is easy: Simply select in Firefox *Tools* → *FoxyProxy Standard* the entry *Use Proxy "localhost:8008" for all URLs*. To stop using the proxy, select *Completely disable FoxyProxy*.

With these hints, you should be ready to analyze the issues reported by Arachni. In the following table, list all issues detected by Arachni (with criticality low or higher), mark whether they are true positives (TP) or false positives (FP), and explain (in one or two sentences) why the issue is a TP or a FP. The first entry serves as an example.

In case a finding is «technically» a true positive but exploiting it does not really provide a benefit for the attacker (e.g. a CSRF «vulnerability» that is not associated with a sensitive action) or is very unlikely to be successful in practice, then identify the finding as a true positive and explain in the explanation field why exploiting it is not beneficial or very difficult for the attacker.

Note that you also have to hand in a screenshot of the Arachni analysis results to get the lab points, so don't forget to take that screenshot.

Issue	TP/FP	Explanation
XSS in lastName field of listUsers.xhtml	FP	Verification is not possible. Injecting of a proof of concept script such as <script>alert(“XSS”);</script> in the last_name field is not possible as control characters are sanitized by the application.
XSS in j_idt10:j_idt11 field of /faces/view/public/search.xhtml	TP	verified by entering <script>alert("XSS");</script> in the search field and sending POST request. The control characters in the request message are not sanitised on the server side and the code is sent back to the browser and executed.
Missing 'X-Frame-Options' header in Server /faces/view/public/search.xhtml	TP	action="/Marketplace_v01/faces/view/public/search.xhtml" method="post" input name="..." Kann ignoriert werden, CSRF-Attacke ist zwar möglich aber es können keine kritischen Daten (z.B. user credientials) geändert werden. No sensitive function is performed via search field, No anti-CSFR token required.
Private IP address disclosure in Header	FP	Kann ignoriert werden. Ist normal, wir testen lokal auf der Maschine.

3 Task 2: Analyzing Webshop with Arachni

In the second task, you are analyzing an Arachni report of the Webshop application, which you already used in a previous lab to manually find and exploit vulnerabilities. As the scan takes quite a long time, you won’t perform the scan yourself. Instead, you are using an Arachni report that was created by the instructors beforehand.

3.1 Basis for this task

To verify some of the vulnerabilities, you’ll have to run the Webshop application on the image. This works basically exactly the same as described in section 2.1:

- Download *Webshop.zip* and *Webshop.sql* from OLAT (from the previous lab).
- Create the database scheme and technical user using *Webshop.sql*.
- Build and run the application by unzipping *Webshop.zip*, opening the project *Webshop* in NetBeans, and building and running it.
- The application is reached with *http://localhost8080/Webshop*.

As a side effect of this, you have imported the project into NetBeans, which provides you with access to the code. This may also be beneficial during analysis of the findings.

The Arachni report you have to analyze is available on OLAT. Download *Webshop-Arachni.zip*, unzip it, and open *index.html* with the browser, which shows an overview of the issues. Then, select the *Issues* tab at the top to analyze the details.

3.2 Analysis of the Results

Just like in the previous task, analyze the issues with criticality low or higher and ignore the informational issues. The issues are listed in the table below. Take into account the following hints:

- Blind SQL Injection (timing attacks): The details results should already tell you more or less what happened, but here's some additional information about the strategy of Arachni: Arachni typically does not know anything about the database structure. To verify that user input is interpreted as SQL commands (which is the fundamental problem with SQL injection), Arachni therefore injects the *sleep* database function. If the response time of the application is much higher than without the injected *sleep* function, there's a strong indication that the *sleep* function was actually interpreted as a database command, which serves as a proof of concept of an SQL injection vulnerability.
  - Arachni reports three Blind SQL Injection findings. You have to analyze only the one in *productSearch.action* and can ignore the other two (*editClient.action* and *login.action*, they work very similar).
  - When analyzing that issue, looking at the code in method *searchProducts* in class *DBAccess.java* may be helpful. However, you should also try to verify the vulnerability directly with the browser. Arachni often uses *16 seconds* as parameter to the *sleep* function. To speed up verification, you can use a lower value.
- Just like in the first task, «convince yourself» whether a vulnerability is a true or false positive. This can again either be done by interacting with the application or by inspecting the source code of the Webshop application. Of course, it may be that Arachni reports some vulnerabilities you have also found during the manual analysis you did during a previous lab – in these cases, verification should be straightforward.
- The Arachni scan was performed with the rights of the user *admin*. It may therefore be necessary to log in as admin to verify some of the vulnerabilities. The password of user *admin* is *admin*.

Issue	TP/FP	Explanation
Cross-Site Request Forgery in editSeller.action (this function is reachable as user *admin*)	TP	Technically feasible but not interesting for an attacker.
Blind SQL Injection (timing attack) in productSearch.action (ignore the two other SQL Injection findings)	TP	Critical Attacker can harverst sensitive data stored in the database.

Cross-Site Scripting (XSS) in editClient.action (this function is reachable as user *admin*)	TP	Technically feasible to store a script, can be interesting for an attacker if the script is executed with admin rights.
Unencrypted password form	TP	Credentials should always be sent over HTTPS.
Missing 'Strict-Transport-Security' header	FP	The WebShop application uses both HTTP and HTTPS protocols. Enforcing HTTPS for all resources is an additional security feature but not necessary in this case.
Password field with auto-complete (4 findings, all are basically the same)	TP	Clear text password is visible with HTML inspector of any modern desktop browser.
Missing 'X-Frame-Options' header	TP	It is a good idea to include this header to avoid clickjacking.

4 Task 3: Analyzing Marketplace_v01 with Fortify SCA

Your third task is to perform a static code analysis of Marketplace_v01 Using Fortify SCA.

4.1 Performing the Scan

Start Fortify SCA in a terminal by entering the following as *user*:

```
cd /opt/HP_Fortify_SCA_and_Apps_16.10/bin
./auditworkbench
```

When the window opens, click *Scan Java Project...*. Select the source code directory of Marketplace_v01, which is *Marketplace_v01/src* (make sure to include the *src* directory in the path selection by double-clicking it the path selection dialogue). In the following *Java Version...* window, select *1.8* and then *OK*. Next the *Audit Guide Wizard* window is opened, leave everything as it is and click *Scan*.

The analysis will take a few minutes. As soon as the analysis has been completed, the results are shown in the *Audit Workbench* window. To make sure all findings are listed, select *Security Auditor View* at the *Filter Set* setting at the top left of the window.

The findings are separated in severity levels *Critical – High – Medium – Low*. However, the chosen setting is not always reasonable as, for instance, two SQL Injection findings are listed under Low,

which does not make any sense. It's therefore best to select the *green* tab, which shows all findings, and then *OWASP Top 10 2013* in the *Group By* drop list, which arranges the findings according to OWASP Top 10.

A valuable feature of Fortify SCA is the additional information that is delivered with each identified vulnerability. To access it, click a vulnerability in the top left window (e.g. an SQL injection vulnerability). You then get the corresponding source code in the middle window and the critical code section is marked. In the lower part, you find lots of additional information. The tabs *Summary*, *Details* and *Recommendations* deliver information about the vulnerability and how it can be fixed. The recommendations are often described very detailed. The tab *Diagram* may also be interesting as it sometimes shows the data flow[4] of the involved classes, method, and variables.

Note that you also have to hand in a screenshot of the Fortify SCA analysis results to get the lab points, so don't forget to take that screenshot.

4.2 Analysis of the Results

Again, your task is to analyze the findings. Focus on the OWASP Top 10 findings, i.e. you can ignore the ones assigned to category *none* in the top left window (you are of course invited to check them, but they don't have to be analyzed to get the lab points). It should be possible to verify the OWASP Top 10 findings by inspecting the source code and your general knowledge (from the lecture) about the application. If a finding has already been verified in task 1, you can refer to the analysis done there.

Issue	TP/FP	Explanation
OWASP A1 - Injection: SQL injection in ProductDatabase.java:18 SQL injection in ProductDatabase.java:20	TP	Critical, because attacker execute arbitrary SQL commands and collect sensitive data
OWASP A3 - XSS: reflected XSS in search.xhtml	TP	Client input is not sanitised on server side. Unchecked input data is sent back to the browser.
OWASP A5 - Security Misconfiguration: J2EE Misconfiguration - Missing Error Handling in web.xml	TP	No standard error handling - no default error page is displayed. Nothing stopps an attacker from exploring the application and mining information about it.
OWASP A6 - Sensitive Data Exposure: Sensitive Data Exposure (Privacy Violation) in PurchaseDatabase.java:24	TP	Mishandling of confidential information. In case of a runtime exception the INSERT query containing the PURCHASE table structure and credit card number is printed to the client. If connection is not secured via TLS then confidential data (cc number) is sent in the clear over the wire.

[4] Many analytical methods of static code analysis tools are based on data flow analysis.

OWASP A8 - CSRF: checkout.xhtml:26	TP	Missing CSRF token to prevent attacker from making unauthorised requests. If a shop customer is logged in to the application and an attacker tricks the customer in navigating to a malicious site that posts the form data on load in a hidden iframe, then the attacker can go shopping with the customer's credit card.
OWASP A8 - CSRF: search.xhtml:21	TP	Technically feasible, but no worries, no sensitive function is performed via search field
OWASP A8 - CSRF: search.xhtml:52	TP	Attacker could mess with a customer's shopping cart.

5 Task 4: Conclusion

Although we only scratched the surface of automated security testing tools, you could certainly get a first impression about their value and their capabilities. This final task serves to conclude your first experiences with security testing tools and to summarize their possibilities and limitations by providing answers to the following three questions. This should also show you that black-box security testing tools and static code analysis tools complement each other as they have different strengths and limitations.

Question 1: How do you rate the capabilities of the two fundamental approaches *black-box security testing* (e.g. Arachni) and *static code analysis* (e.g. Fortify SCA) to find SQL Injection and Cross-Site Scripting vulnerabilities? Provide an answer that rates the advantages of both approaches.

black box testing can give an indication for SQL injection vulnerabilites.
Whereas with static code analysis it is very easy to find SQL queries that can be abused.
I would rate both approaches equal regarding SQL injection vulnerabilites.

Possible XSS vulnerabilites are easily detected by both methodes.
I cannot think of an advantage of one approach over the other.

Question 2: Assume we have an application that contains access control vulnerabilities, i.e. users can access areas of the application that should not be accessible by them. Your goal is to find such vulnerabilities using either of the two approaches discussed in this lab. Which one is better suited? Is it actually possible to find such vulnerabilities using the two types of security testing tools discussed here?

Arachni (black box testing) would have to guess resources and try to access them. This is very unlikely.
Using a black box approach is not recommended.

This kind of vulnerability is best detected with a static code analysing tool.

Question 3: Finally, how to you rate the capabilities to detect configuration problems of the application itself (e.g. problems in web.xml) and the underlying application server? Again, provide an answer that rates the advantages of both approaches.

SCA helps avoiding pitfalls in setting up an application before deploying it.

But if the application is deployed and running, a security-unaware administrator could alter the web server configuration. In this situation, periodically running automated black-box tests are clearly the better approach.

Lab Points

For **2 Lab Points** you must submit your solution of the 4 tasks (for each task, you can get ½ point) by e-mail to the instructor. *Use SecLab – Security Testing Tools - group X - name1 name2* as the subject, corresponding to your group number and the names of the group members. You can submit either pdf (e.g. by annotating directly this document) or doc(x) formats, and of course it's also OK if you scan and submit a handwritten solution. However, your solution to tasks 1 – 3 should be formatted in the same way as the tables above.

In addition, you must include screenshots of the Arachni and Fortify SCA scans you did during tasks 1 and 3.

Solutions that were obviously copied from others won't give any points.