Security Lab – Security Testing Tools

VMware

- You will work with the **Ubuntu image**, which you should start in networking-mode **Nat**.

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:

- **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".
- **Black-Box Security Testing of the running program**: In this case, the testing tool communicates/interacts with the running program, using the "normal" communication interface, e.g. HTTP with web applications.

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

- **Findbugs**[1]: 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 security-relevant vulnerabilities
- **HP Fortify Source Code Analyzer (SCA)**[2]: Fortify SCA is a commercial and very powerful static code analysis tool, which supports several programming languages. We have an education 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 even external projects).
- **Arachni**[3]: Arachni is probably the most powerful open source web application vulnerability scanner. 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.

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

2 Basis for this Lab

Automated testing tools can produce large amounts of messages, especially when testing large projects. We therefore work with a small project that you know well from the lecture: the Marketplace application.

You will analyze the initial version (Marketplace V01, the version that was used in the lecture as the basis and that contains several vulnerabilities) and the final version (Marketplace V10, the final version produced in the lecture, which should be secure).

On the Ubuntu image, there exist two Eclipse projects *Testing_MarketplaceV01* und *Testing_MarketplaceV10*, which correspond to these two versions.

- To start Eclipse, open a terminal, change to */home/user/eclipse*, and enter *./eclipse*.

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

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

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

- The project directories are */home/user/workspace/Testing_MarketplaceV01* and *V10*.

3 Deploying the Application

To perform the tests, Findbugs requires the war files (they contain the Java bytecode) and Arachni requires the running application. Deploy both versions to the Tomcat 7 server, which is installed on the image (under */home/user/tomcat7*). This is done as follows (for both projects):

- Right-click the project in the *Project Explorer* (left window) in Eclipse, select *Export...*, and then select *Web* → *WAR file* in the list.
- Enter the following in the field *Destination* in the window that has just been opened: */home/user/tomcat7/webapps/testing_marketplaceV01.war*
- for version V01 and */home/user/tomcat7/webapps/testing_marketplaceV10.war* for version V10.

To start or stop Tomcat, enter the following in a terminal in directory */home/user/tomcat7/bin* (as *user*):

- `./startup.sh`
- `./shutdown.sh`

Using the browser, you reach the two versions of the application with:

- `http://localhost:8080/testing_marketplaceV01` and
- `http://localhost:8080/testing_marketplaceV10`

Test if this works as expected.

4 Task 1: Findbugs

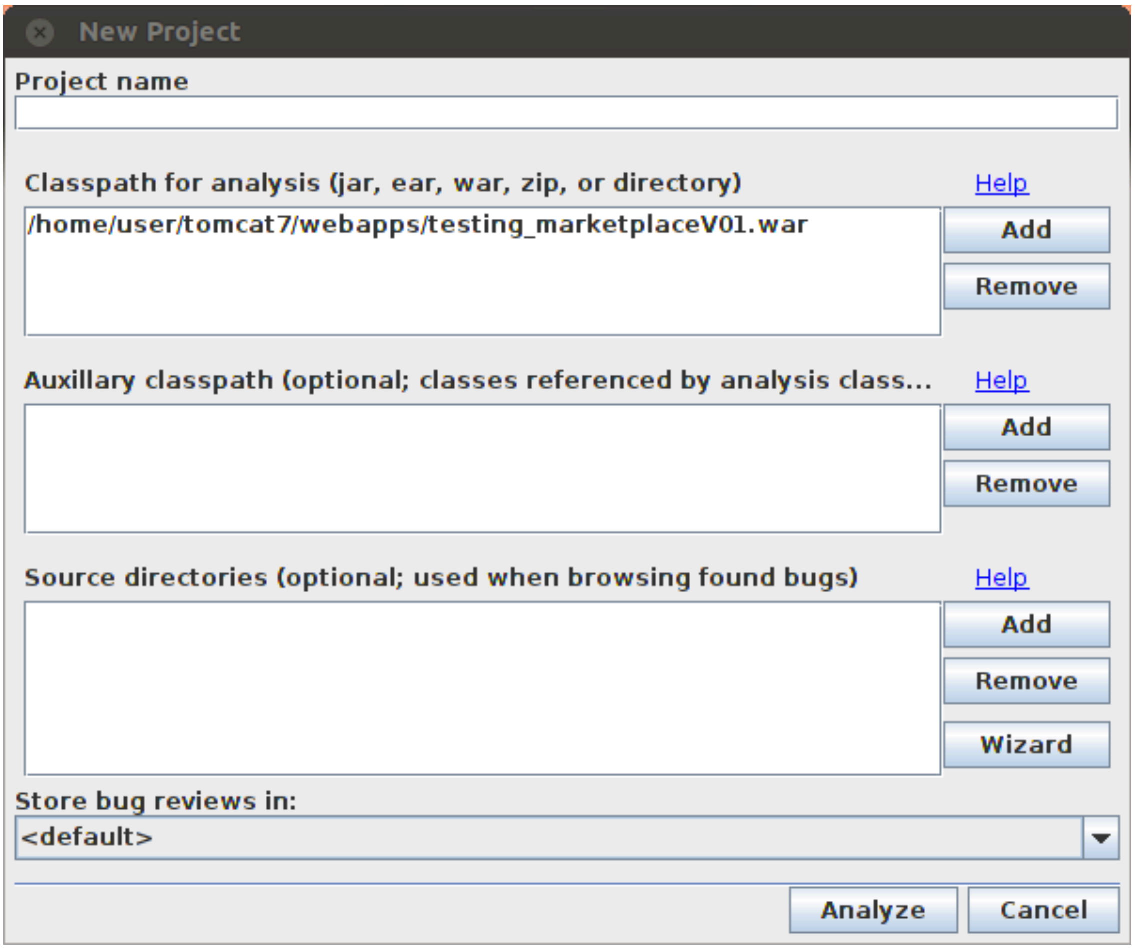
Your first task is to analyze the code with Findbugs

4.1 Analysis of Marketplace V01

Start the graphical mode of Findbugs by entering the following in a terminal (as *user*):

```
java -jar /home/user/findbugs-2.0.3/lib/findbugs.jar
```

Choose menu *File* → *New Project....* , click the *Add* button next to the box *Classpath for analysis*, and select the first version: */home/user/tomcat7/webapps/testing_marketplaceV01.war*. The result should look as in the following screenshot:


New Project

Project name

Classpath for analysis (jar, ear, war, zip, or directory)

/home/user/tomcat7/webapps/testing_marketplaceV01.war

Auxillary classpath (optional; classes referenced by analysis class...

Source directories (optional; used when browsing found bugs)

Store bug reviews in:

<default>

Clicking *Analyze* starts the analysis. After a few seconds, you should get the results. If you get a message about missing classes, simply ignore it.

Focus on the reported *security* bugs and analyze the results. Briefly summarize the security issues reported by Findbugs:

Automated testing tool often produce false positives (false alerts). Therefore, the results should always be critically analyzed, which you'll do next.

Are the security bugs reported by Findbugs actual vulnerabilities or false positives? If you click the vulnerability in the top left window, you get additional information about it. Sometimes, looking into the source code of the affected classes (using Eclipse) is also a good idea.

Findbugs also provides you with recommendations about how to prevent a reported vulnerability. What are the recommendations by Findbugs in the current caes? And do you think these are good recommendations that truly help to effectively prevent the vulnerability?

4.2 Analysis of Marketplace V10

Perform another test, this time using V10, which hopefully contains fewer or no vulnerabilities. The WAR file to use is */home/user/tomcat7/webapps/testing_marketplaceV10.war*. Does Findbugs still report *b.* bugs? Explain the results. Again, looking into the affected Java classes will be helpful.

4.3 Findbugs – Conclusions

Maybe you are somewhat disappointed that Findbugs could not find more vulnerabilities in V01 because from the lecture you know that there exist further vulnerabilities in this version. Findbugs is indeed not very powerful in detecting security vulnerabilities but unfortunately, there exist no really good and freely available static code analysis tools. Still, Findbugs is suited to find at least a few vulnerabilities with little effort and if you don't have access to a commercial tool, Findbugs is still better than using nothing at all.

5 Task 2: HP Fortify SCA

Your second task is to test the code using the static code analyzer Fortify SCA of HP.

5.1 Analysis of Marketplace V01

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

```
cd /home/user/HP_Fortify_SCA_and_Apps_4.10/bin
./auditworkbench
```

When the window opens, click *Advanced can...*. In contrast to Findbugs, Fortify SCA analyzes the source code and correspondingly, you must select the base directory of the project: */home/user/workspace/Testing_MarketplaceV01*. In the following *Commandline Builder* window, click *AddDirectory...* and choose */home/user/tomcat/lib*. This provides Fortify SCA with the libraries used by the code. In the drop list next to *Java version*, select *JDK 1.7* and start the scan by clicking *Scan*.

The analysis will take longer than with Findbugs, approximately 2 minutes. As soon as the analysis has been completed, the results are shown in the *Audit Workbench* window.

In the top left you can see the identified vulnerabilities, separated in priorities *Critical – High – Medium – Low*. For the reminder of working with Fortify SCA, consider all findings with priorities *Critical – High – Medium*. In the *Low* category, only consider findings with respect to Access Control and CSRF (usually the top two findings in the list). Of course you can study the other *Low*-findings if you want, but we will not focus on them here.

Are there vulnerabilities that were also reported by Findbugs? If yes, which ones?

Did Findbugs find vulnerabilities that could not be found by Fortify SCA? If yes, which ones?

List the vulnerabilities that were reported by Fortify SCA, but that where not reported by Findbugs:

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 right 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* is also interesting as it shows the data flow[4] of the involved classes, method, and variables – in the case of the SQL injection vulnerabilities, one can for instance clearly see how the data received from the user find their way into the SQL queries.

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

Use this provided information and your knowledge of the Marketplace application from the lecture to answer the following question: Are the additional (compared to Findbugs) vulnerabilities reported by Fortify SCA actual vulnerabilities or false positives?

5.2 Analysis of Marketplace V10

Select *Close Project* in the menu *File* and perform another test as described above, this time using V10. The base directory of the project is */home/user/workspace/Testing_MarketplaceV10*.

Which of the vulnerabilities of V01 are no longer reported and why?

Some of the vulnerabilities reported in V01 are still present, but their priority has been reduced from *Critical* to *Medium*. Which are the corresponding vulnerabilities and what's the reason for this reduction of the priority? What could be done in addition to completely get rid of the vulnerability? Looking at the *Recommendations* should help you to answer this question.

What do you say about the other reported vulnerabilities?

5.3 Fortify SCA – Conclusions

You have seen that the tool is much more powerful than Findbugs, both in detecting vulnerabilities and delivering additional information that help to fix the vulnerability or to identify it as a false positive. Of course this has its price, the license costs are „several 1'000 USD" per user and year.

When looking at the various issues with priority *Low* you see that Fortify SCA can indeed detect a lot. Many of these issues of priority *Low* are not very security-relevant but they should nevertheless not simply be ignored, as some of the reported issues may be important. In the case of the Marketplace application, this concerns, e.g., the missing standard error handling in web.xml, which we discussed and added and in the lecture but which we then deactivated again as this is typical during application development.

6 Task 3: Arachni

In the last part, you'll experiment with Arachni. In contrast to the two static code analysis tools used above, Arachni interacts with the running web application. If you have deployed the both applications and started Tomcat as described in section 3, then both versions should be ready to be tested.

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

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

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[5] 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.

[5] Arachni can also – by configuring the credentials accordingly – access protected areas (which exist in V10 of the Marketplace application). For simplicity, this is left out in this lab.

6.1 Analysis of Marketplace V01

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

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

Logon with the default user credentials: *user@user.user* with password *regular_user*.

At the top, select *Scans* → *New*. As *Target URL*, enter

```
http://localhost:8080/testing_marketplaceV01/index.jsp
```

and use the *Default (Global)* configuration profile. Hit *Go!* to start the scan. The scan will take several minutes to complete, but first findings will show up soon and can be analyzed while the scan is still running. The blue button with the question mark next to the identified vulnerable URLs can be clicked to get additional details.

Which problems were identified by Arachni? Do only consider "red" or "orange" results.

Does Arachni report any vulnerabilities that were not reported by Fortify SCA above?

First, only consider the XSS vulnerability/-ies. Are they actual vulnerabilities or false positives? How did Arachni discover the vulnerability?

Analyze the recorded requests and responses that were used by Arachni to detect XSS vulnerabilities. This should provide you with hints about the strategy used by Arachni to discover XSS vulnerabilities. Based on your analysis, write down how you think Arachni performs these tests.

Now consider the blind SQL injection (timing attack) vulnerabilities. Read the description provided by Arachni and explain in your own words what Arachni did and why this is a strong indication that an SQL injection attack is possible. Also, try to reproduce the timing attack in the browser by interacting with the Marketplace application (the details given by Arachni should provide you with a suitable attack string) – were you successful?

Finally, there's another blind SQL injection vulnerability, this time with description "differential analysis". First, read the description and try to understand what Arachni did. Basically, Arachni submits many strings to the application that should all provide the same output (e.g. use various strings in a search that unlikely produce any result). But if one of them actually produces a significantly different result, it's an indication that an SQL injection vulnerability was found.

Analyze the string that was used by Arachni to uncover this vulnerability and reproduce the attack by interacting with the Marketplace application. Explain why the used attack string produces a different result than others that were used by Arachni (although these "other" search strings are not listed). You may have to study the source code where the SQL query is built to answer this question.

Blind SQL injection attacks can not only be used to show that an SQL injection vulnerability exists, they are attacks on their own, as is described in the following. We start with the description of blind SQL injection by OWASP:

> *When an attacker executes SQL Injection attacks, sometimes the server responds with error messages from the database server complaining that the SQL Query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather then getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through SQL statements.*

To illustrate this. we consider another (not detected by Arachni) blind SQL injection vulnerability in the Marketplace application by taking the following two requests as an example:

- http://localhost:8080/testing_marketplaceV01/cart?productCode=0004'+OR+'50'='50
- http://localhost:8080/testing_marketplaceV01/cart?productCode=0004'+AND+'50'='51

In the first case, the response is a „normal" HTTP 200 response but in the second case, it's a HTTP 500 „Internal Server Error" response. The SQL query apparently resulted in an error in the second case, which is indeed true: a *NullPointerException* occurred because after reading a product from the database, the code does not check whether the product could actually be found. If you think about this, you'll realize that in the first request, OR '50'='50' was appended to the WHERE clause (which is always true and as a result, a product is returned) and in the second request, AND+'50'='51' was appended, which is always false and therefore nothing is returned – which then results in the exception

This already is a bug but can it be exploited? To do this, one has to be a bit creative. Based on the vulnerability identified above and by performing some additional manual tests, you'll realize that when appending something after the AND that is false, then the application behaves differently (the error occurs) than when appending something that is true (no error occurs). By appending suitable queries after the AND, this can be used to read all data from the database step-by-step.

We want to use the vulnerability to get existing user names. We assume the attacker knows the database structure. We now append the following statement with different user names after the AND:

```
(SELECT COUNT(*) FROM UserPass WHERE Username = 'username') = 1
AND '1' = '1
```

The additional AND '1' = '1 is required to get a syntactically correct Query. If a non-existing user name is used for *username* (e.g. *rumpelstilzchen*), the inserted SELECT query returns 0, which is ≠ 1, which means the logical expression is FALSE and the server returns the HTTP 500 error. But if an existing user name is used (e.g. *john*), the SELECT statement returns 1, the logical expression after the (first) AND is TRUE and the application does nor produce the error. In the same way, any information can be read from the database (e.g. passwords) without ever receiving the actual data directly from the database – therefore the name *Blind* SQL Injection. Of course, this requires many requests, but it can be easily automated. And there are tricks to accelerate this (for instance by reading not entire fields but individual characters) – but this definitely goes beyond the scope of this lab.

6.2 Analysis of Marketplace V10

Perform another scan with Arachni, this time using V10:

```
http://localhost:8080/testing_marketplaceV10/index.jsp
```

Which of the vulnerabilities found in V01 are no longer reported and why?

Which vulnerabilities found in V01 are still there and why? Are there new vulnerabilities?

6.3 Arachni – Conclusions

As you have seen, Arachni is an interesting tool that could detect several critical vulnerabilities.

7 Final Conclusions

Although we could only scratch the surface of automated security testing tools, you could certainly see that they are valuable. You have also seen that different tools could find different vulnerabilities, which means combining them is reasonable. Fortify SCA is certainly a very powerful (and expensive) tool, but combining the open source products Findbugs and Arachni could also discover the most critical vulnerabilities without producing false positives. In general, using the tools requires lots of security know how because otherwise, it's not possible to correctly interpret the reported vulnerabilities and propose and implement the right corrective measures. The quote “A fool with a tool is still a fool“ therefore applies here as well.

Of course, the tools mainly find the so-called „low-hanging fruit“, i.e. the vulnerabilities that can be also uncovered quite easily manually, especially if the tester has the necessary skills. But since attackers typically also use such tools in a first step, it’s very important to identify and remove these low-hanging fruit vulnerabilities and the tools are certainly helpful to do this. However, in particular in critical applications where there’s real (financial) gain for the attacker, automated testing tools should always be complemented with manual tests (penetration tests, manual code reviews of security-critical components) because the attackers are willing to invest more time and energy to uncover vulnerabilities.

Lab Points

For **2 Lab Points**, you must show the filled-in sheet (one per group) to the instructor. In addition, you must show the scan results of Fortify SCA und Arachni (using V01 of the Marketplace application) – to do so, perform these two scans at the end of the lab again and show the outputs of the tools.