

Security Aspects in Software Development

KU 2013/2014

“Web Application Security (WS)”

Daniel Hein Johannes Winter

Thomas Kastner

Institute for Applied Information Processing and Communications

Inffeldgasse 16a, 8020 Graz, Austria

securityaspects@iaik.tugraz.at

October 9, 2013

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Objectives	2
1.3	Plagiarism	3
2	Background story	3
3	Software requirements and setup	4
3.1	Configuring global Git settings	4
3.2	Cloning your repository for the first time	5
3.3	Setting up the database	6
3.4	Importing WS project into Eclipse workspace and testing the installation	6
4	Submission	7
4.1	Registration with the STicS system	7
4.2	Checklist	8
4.3	Deadline	8
5	Tasks	9
5.1	(7 points) Broken Authentication	9
5.2	(7 points) SQL Command Injection	9
5.3	(7 points) Path Traversal	10
5.4	(7 points) Lazy Cross-Site-Scripting	11
5.5	(5 points) Sensitive Data Exposure	11
5.6	(2 points) Compiler Based Information Disclosure	12

1 Introduction

Assignment title: Web Application Security (WS)
Group size: 2 students
Start date: October 09, 2013
Maximum score: 25 points (25% percent of final* mark)

HARD SUBMISSION DEADLINE: November 06, 2013 (23:59:59 CET)

See Section 4 for details on the submission process.

See Section 5 for the subtasks and questions of this assignment.

* Bar the oral exam.

1.1 Motivation

The web has become ubiquitous. Long since past are the days of simple static web pages serving information alone. Today, web applications have become prevalent. Web applications rely on technologies such as client side code execution, server side dynamic web page generation and client- and server-side data storage to enrich the user's experience. Erroneous use of these technologies can severely impact the security of the service provider and the user. The ease of access to web applications, in conjunction with the fact that some typical web application vulnerabilities are easy to exploit, while having a severe impact on security, makes teaching about web security a number one priority for us.

The OWASP¹ project has identified *Injection*, *Broken Authentication and Session Management*, and *Cross-Site-Scripting (XSS)* as the three most critical security risks for web applications in their "OWASP Top 10 - 2013" report [2]. With this in mind we have devised this exercise to teach you the skills necessary to avoid these common security risk, and a few other of the top ten risks.

1.2 Objectives

The goal of this exercise is to build up skills in secure web application development. In particular this exercise is intended to train the ability:

- to understand the security implications of web application authentication itself, and the applied security mechanisms. In addition, this exercise is designed to train your ability to detect, and repair problems in an existing authentication mechanism. In other words, you should know why user passwords are salted and then digested by a pseudo-random function, before being stored on the server, and you should understand why it is essential that passwords are not transmitted in plain over the Internet.
- to prevent, detect and repair
 - Command injection vulnerabilities, especially SQL injections.
 - Cross-Site-Scripting vulnerabilities.
 - Path traversal vulnerabilities.

Additionally, this exercise aims to train your ability to understand the security impact of command injections, cross-site-scripting and path traversal vulnerabilities. It should teach you understanding how attack data is introduced into a web application, and that user data should always be validated.

¹<https://www.owasp.org/>

- to protect sensitive information, like passwords, digested passwords, session cookies etc. Furthermore, this exercise is designed to teach you that executing any user supplied data can have unexpected side effects. Even if the user supplied data is just control information how a program should be compiled.

After finishing this assignment you should be able to better recognize potential vulnerabilities in existing source code, and to avoid the same issues when writing your own code. Moreover you should gain an understanding about the anatomy of different vulnerabilities allowing you to better assert risks and impacts of typical web-application security vulnerabilities.

1.3 Plagiarism

△ *Plagiarism will not be tolerated and any contributions containing plagiarized material will automatically receive a negative rating. It is explicitly allowed to use material and code snippets shown during the lecture or the exercises.*

2 Background story

In this assignment we ask you to assume the role of security consultants. As such, we ask *you* to secure the web front-end to the *SASE SUBMISSION SYSTEM*. The *SASE SUBMISSION SYSTEM* service allows students to test their submission for a C programming task. Students upload a submission, the system will compile it, and the compiler output is displayed to the student. Thus, the student knows that his, or her submission compiles in the submission environment.

The *SASE SUBMISSION SYSTEM* should only be accessible to students. Therefore, the system requires registration, before it can be used. Relevant areas are protected by requiring authentication.

The *SASE SUBMISSION SYSTEM* also comes with an administration interface. The administration interface allows system administrators to view registered users, delete users, and edit system wide configuration properties, such as the compiler command used to compile student submissions.

In detail the *SASE SUBMISSION SYSTEM* service is capable of:

- welcoming new students.
Address: `http://localhost:8080/sase.submission.system/`
- registering new students.
Address: `http://localhost:8080/sase.submission.system/signin.jsp`
- authenticating existing users, by accessing any protected URL such as `http://localhost:8080/sase.submission.system/submission/`, or `http://localhost:8080/sase.submission.system/admin/`.
- listing and deleting existing users.
Address: `http://localhost:8080/sase.submission.system/admin/users.jsp`
- editing global configuration properties.
Address: `http://localhost:8080/sase.submission.system/admin/config.jsp`
- logging-out an authenticated user.
Address: `http://localhost:8080/sase.submission.system/logout.do`

Sadly, during development, the *SASE SUBMISSION SYSTEM* was invaded by *Fuzzy the Bug*, destroyer of web sites, maker of vulnerabilities and devourer of secrets. *Fuzzy the Bug* has seriously damaged the integrity of the website and introduced a number of security vulnerabilities. Faced with *Fuzzy the Bug's* evil genius we were left with only choice: we needed to get competent help. Thus we ask you, as security

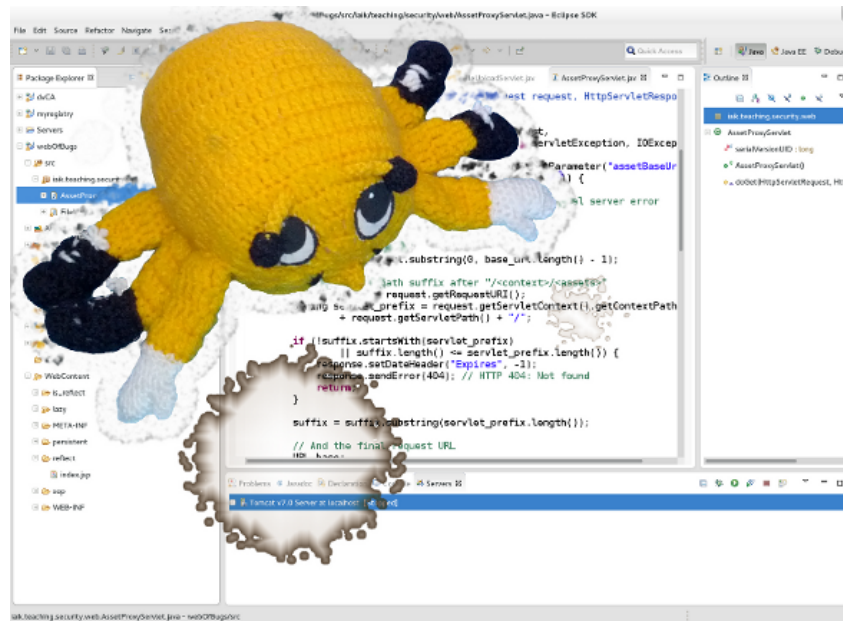


Figure 1: *Fuzzy the Bug* invading an integrated development environment

consultants, to help us out. Your assignment is to detect, exploit, and ultimately fix some of the many security problems. A detailed list of problems for you to look at is contained in Section 5.

3 Software requirements and setup

The reference platform for this exercise is the VMware[1] image, which can be found in the download section ² of the course website. The reference image is based on Xubuntu(Precise Pangolin 12.04 LTS) Linux with Apache Tomcat 7.0.12, Oracle JAVA SDK 1.7.0_40, MySQL 5.5.32, lemon 3.7.9, gperf 3.0.3, clang 3.0.6 and CUnit 2.1.2(unstable). It is shipped with a pre-installed Eclipse(Kepler), and the Apache Tomcat v7.0 server is configured to be executed by Eclipse, thus allowing debugging of new and existing web application code in Eclipse.

△ If you prefer to use your own setup, bear in mind that we can and will only support the reference platform virtual appliance. We can give no guarantees about the compatibility of the software used in this assignment with non-reference platform setups.

In the remainder of this document we assume that you are using our VMware reference image. Unless explicitly noted otherwise, all command line examples shown here are to be run inside the virtual environment of the reference image and *not on your host computer*.

☞ The default user account on the reference image is “sase” with password “sase”.

3.1 Configuring global Git settings

All Git commits into your repository must be done with the proper author name and e-mail information. The proper author name is your real name, the proper email is your Graz University of Technology student email account. Configure those properties before your first commit. To configure the reference

²http://www.iaik.tugraz.at/content/teaching/master_courses/sicherheitsaspekte_in_der_softwareentwicklung/practicals/downloads/

environment log into your virtual machine, open a terminal window and set Git's `user.email` and `user.name` properties as shown (don't forget to update the placeholders!):

```
sase@vm:~$ git config --global user.name "Firstname Lastname"
sase@vm:~$ git config --global user.email "firstname.lastname@student.tugraz.at"
```

3.2 Cloning your repository for the first time

All registered two-person groups will receive Git login credentials for our teaching Git server. Issuing of Git credentials by email starts on the October 9th, 2013. Once you have received your credentials by e-mail you should proceed to set-up your virtual machine for use with your repository.

△ *Your SSH authentication key must be loaded before invoking any Git commands that communicate with our teaching Git server. You can use `ssh-add` to load a key. To list all keys loaded in the current session you can use `ssh-add -l` command. If your SSH authentication key is not loaded, the authentication will fail and you will be prompted for a password.*

Note that our teaching Git server NEVER asks for any password when doing a `git clone`, `git pull`, or `git push`. Triple-check your client configuration (repository name, `ssh-add -l`) and the SSH keys registered with the teaching Git self-management tool³ if you see a password prompt.

Your Git repositories are initially empty, and you have to download and apply the assignment patch from the course website. We recommend to use the following sequence of steps for the initial setup:

1. Create an SSH keypair (e.g. using `ssh-keygen`) for use in this course and register the *public* key with IAIK's student key management system at <https://keyman.teaching.student.iaik.tugraz.at>. Each group member should use his or her own key.

△ *We highly recommend to create a fresh SSH keypair exclusively for use in this course. Please do not reuse any existing keypairs, which you may have used in previous IAIK lectures, as the policies of IAIK's teaching GIT server may refuse repository access if such "key-reuse" is detected.*

2. Properly configure the `user.email` and `user.name` options of the GIT installation in your virtual machine.
3. Inside your virtual machine, load your SSH *private* key with `ssh-add`.

The reference virtual machine automatically starts an SSH authentication agent upon login. You can check the list of currently loaded keys by running `ssh-add -l`.

4. Clone your repository from the server, using the following command: (do not forget to replace `sase2013gXX` with your own group number)

```
sase@vm:~$ git clone git@teaching.student.iaik.tugraz.at:sase2013gXX.git ~/repo
```

This step will checkout a local working copy of your repository into the `/home/sase/repo` directory, and setup the required GIT remote repository configuration.

5. Download the web-security task patch from the practicals section⁴ of the course homepage, and save it in your virtual machine.

For the next steps let us assume that you saved the patch as `/home/sase/ws.patch`.

6. Change into the directory of your local working copy and apply the task patch with the `git am` command:

```
sase@vm:~$ cd ~/repo/
sase@vm:~/repo$ git am ~/ws.patch
```

³<https://teaching.student.iaik.tugraz.at>

⁴http://www.iaik.tugraz.at/content/teaching/master_courses/sicherheitsaspekte_in_der_softwareentwicklung/practicals/downloads/

Applying the patch may produce some warnings about squelched whitespace errors, which can be safely ignored. After the patch has been applied to your local working copy, there will be a `tasks/ws/src/sase.submission.system` directory containing the source code for the WS assignment. This directory is called *SASE SUBMISSION SYSTEM* directory in the remainder of this document.


7. Push your changes back to the GIT server, to allow your group colleague to merge them into his or her local working copy.

To push your changes to the server, use the `git push` command:

```
sase@vm:~$ cd ~/repo/  
sase@vm:~/repo$ git push
```

To pull your colleague's changes from the server, use the `git pull` command:

```
sase@vm:~$ cd ~/repo/  
sase@vm:~/repo$ git pull
```

 *Subversion users: Please be aware, that the `git commit` command only operates on your local working copy. To publish your commits to the GIT server, you need to explicitly do a `git push`.*

3.3 Setting up the database

The reference virtual machine already comes with a preinstalled MySQL database server. It is however necessary to initialize the database required by the *SASE SUBMISSION SYSTEM*, before the web application can be used.

The SQL script to initialize the WS database can be found in

```
tasks/ws/src/sase.submission.system/db-init.sql
```


in your local working copy of the GIT repository. The script contains all necessary SQL statements to recreate the database tables and to populate them with the initial “admin” credentials of the *SASE SUBMISSION SYSTEM*.

To re-initialize the database, first start the `mysql` command line tool and log in as MySQL root user with password `sase`:

```
sase@vm:~$ mysql --user=root --password=sase
```

In the MySQL command interpreter issue the following commands:

```
source ~/repo/tasks/ws/src/sase.submission.system/db-init.sql  
quit
```

 *You can use the same commands, to reinitialize the database, for example if you want to start from scratch.*

3.4 Importing WS project into Eclipse workspace and testing the installation


Start Eclipse. After starting Eclipse,

1. open *File* and select *Import*
2. go to *General* and select *Existing Projects into Workspace*
3. browse to your *SASE SUBMISSION SYSTEM* directory
(default: `~/repo/tasks/ws/src/sase.submission.system/`)

To run the project right click on the project, select *Run As > Run On Server* and choose the pre-configured *Tomcat v7.0 Server*. This operation tells Eclipse to publish the `sase.submission.system` web application to the *Tomcat* server. You must do this at least once, afterwards you can also start the *Tomcat* server in Eclipse using the server tab in the area below the editor. The web application should then be available in any browser in the VM.

After cloning your repository (see above) and importing it into Eclipse, proceed to test your setup by starting the web application as described above, then start a web-browser *in* the VM and navigate to the following URL:

- `http://localhost:8080/sase.submission.system`

 You will find Chromium as well as Firefox application launchers on the desktop. The launchers are configured to use the `http://localhost:8080/sase.submission.system` as start page.

4 Submission

The first part of the submission for the Web Application Security assignment consists of informing us via STicS⁵, the Student Tick System, about which tasks you have prepared for the oral exam. The second part of the submission consists of discussing your answers at the oral exam on the 7th and 8th of November, 2013.

△ *For this task you do not have to hand in any code. During the oral exam we will discuss your answers to the questions in Section 5 and ask you to practically demonstrate some simple attacks. Your overall score for this assignment solely depends on your performance at the oral exam. Failing to correctly answer prepared question will degrade your score. (Worst case: 0 points)*

You can achieve a maximum of 25 points for the Web Application Security assignment. In total there are 32 tasks in Section 5, which are worth 35 points. For you as a student that means that you can mix and match your tasks according to your own preferences. You do *not* have to prepare answers for all 32 tasks! If you want to get the maximum achievable points, select enough tasks to achieve 25 points, and tell us which tasks you have prepared. These are the tasks that you will get asked about at the oral exam.

To reiterate, you select which tasks you want to prepare for the oral exam. Each task you choose to prepare increases your achievable points, but also increases your personal pool of questions that we use to select your oral exam questions.

4.1 Registration with the STicS system

To register with the STicS system you have to be registered for the Security Aspects in Software Development practical exercises in TUGRAZonline (Course number: 705.025).

If you are enrolled for the practicals in TUGRAZonline, you can obtain a STicS account by simply requesting a STicS password reset at `https://stics.iaik.tugraz.at/` using your student account email address (e.g. `my.name@student.tugraz.at`) as username.

Once you log into STicS you can select the tasks you have prepared for the WS assignment.

⁵`https://stics.iaik.tugraz.at/`

4.2 Checklist

Here is a checklist with all steps required to successfully complete this task:

- ☐ **Enroll for the Security Aspects in Software Development practicals** in TUGRAZonline (Course Number: 705.025).
- ☐ **Find a group partner and register your group** via an e-mail to securityaspects@iaik.tugraz.at. You will get login credentials for your GIT repository once the group registration has been processed.
- ☐ **Request a password for IAIK's Student Tick System (STicS)** at <https://stics.iaik.tugraz.at/>. Registration will be possible using your TUG student email address, after you have enrolled for the course in TUGRAZonline.
- ☐ **Grab the source code patch for this assignment** from the practicals section of the course website and import it into your GIT repository.
- ☐ **Prepare answers to the questions from Section 5** for your oral exam for this assignment on November 7th, or 8th. You can use your notes during the oral exam. We may ask you to practically demonstrate some short attacks in the virtual machine reference environment.
- ☐ **Tick the tasks and questions which you solved in the STicS system.** *Each group member* must tick the questions which he or she wants to answer during the oral exam. Both assignment partners may choose the same set of questions, if they want, but you may also chose different questions.
- ☐ **Select a timeslot for the oral exams at November 8th and 9th 2013.** Timeslots will be announced timely in the newsgroup.
- ☐ **Show up to the oral exam and discuss your answers with us.** You can use your notes during the oral exams. Points will only be awarded for questions which haven been ticked in the STicS system before the deadline.

⚠ Your overall score for this assignment depends on your performance at the oral exam. Failing to correctly answer prepared question will degrade your score. (Worst case: 0 points)

4.3 Deadline

HARD SUBMISSION DEADLINE: November 06, 2013 (23:59:59 (CET))

⚠ We strongly recommend that you select the tasks you have prepared in STicS at least one or two days before the ultimate submission deadline. Last minute submissions are always risky with respect to unforeseen technical difficulties.

5 Tasks

This section discusses the theoretical and practical tasks to be done as part of the Web Application Security assignment. The tasks described in the following have an accumulated points value of 35 points. You can only achieve a maximum of 25 *points* for the Web Application Security assignment. This difference is intentional. The idea is that you can select which tasks you want to solve based on your personal preferences. For details see Section 4.

5.1 (7 points) Broken Authentication

The *SASE SUBMISSION SYSTEM* authentication mechanisms uses client side password digestion to protect the confidentiality of the student's passwords. Password digestion is based on concatenating a salt to the password and then hashing it repeatedly using the MD5 cryptographic hash function. With many web applications password digestion is done on the server. The *SASE SUBMISSION SYSTEM* digests passwords on the client side, directly in the browser using JavaScript, thus not even the server sees the password in plaintext. Unless of course, ...

Questions:

- (1.a) ☐ (1 point) From a security point of view, what is the reason for concatenating the password with a salt?
- (1.b) ☐ (1 point) What is the security reason for digesting the password by applying a pseudo random function for a specific number of times?
- (1.c) ☐ (1 point) Have a look at the *sign in* process. Where is the salt and the number of iterations needed to digest the password created? Is the information created server-side, or client-side, and where exactly is it created in the web application code? How does the client obtain this information for digesting the password during *log in*?
- (1.d) ☐ (1 point) Having analyzed where and how the salt and the number of iterations is created, specify an attack, where an attacker can get another user to send her or his plaintext password to the server during sign in. What is the name of this kind of attack?
- (1.e) ☐ (1 point) Fix the vulnerability that allows an attacker to get another user to send his or her plaintext password during sign in. Describe how your fix works and document the code of your fix.
- (1.f) ☐ (1 point) Create, that is sign in, several users. Have a look at the users in the administration area. Do you notice anything peculiar with the salt values? Is it possible to exploit this behaviour? If so, which attack can you mount?
- (1.g) ☐ (1 point) Pinpoint the location of the salt vulnerability and fix it. Explain, why the problem occurred and how your fix repairs the problem.

5.2 (7 points) SQL Command Injection

The *SASE SUBMISSION SYSTEM* uses a database to store registered users, their roles, properties associated with their roles, as well as global configuration properties. The *SASE SUBMISSION SYSTEM* uses JDBC to send SQL commands to the database to insert, update or query information in the database. To protect against SQL command injection attacks, the *SASE SUBMISSION SYSTEM* uses both client side and server side input validation, as well as prepared statements. So an SQL command injection is impossible, or is it?

Questions:


- (2.a) ☐ (1 point) What is an SQL command injection? Give an arbitrary example of an SQL command injection vulnerability. Describe one method to prevent SQL command injections. Discuss the advantages, and disadvantages of your method.
- (2.b) ☐ (1 point) The *SASE SUBMISSION SYSTEM* is vulnerable to at least one SQL command injection. Where in the server code is/are the vulnerable statement(s)? How did you find them?
- (2.c) ☐ (1 point) Select an SQL command injection vulnerability and inject valid SQL into it. What vulnerability did you use, that is, where in the server side code is the vulnerability located? What is your attack SQL statement, and where did you enter it in the client (web browser)? What is the full SQL statement, that is the original SQL statement plus your attack statement that gets executed on the server?
- (2.d) ☐ (1 point) Given the above vulnerability, what can an attacker do with this vulnerability? Can you read data? Can you modify the database? Explain your answer!
- (2.e) ☐ (1 point) Use one SQL command injection vulnerability to read out the password field of the administrator. Which vulnerability did you use? Specifically, where is it located in the server code and where did you enter it on the client side? What does the attack statement look like? Again, specify the full statement that gets executed by the server.
- (2.f) ☐ (1 point) What can you do if you combine the above attack with the attack developed in (1.d)?
- (2.g) ☐ (1 point) Fix one SQL command injection vulnerability! Document your code changes.

5.3 (7 points) Path Traversal

The *SASE SUBMISSION SYSTEM* allows students to upload a program which is then compiled. If the uploaded file is a ZIP file, the *SASE SUBMISSION SYSTEM* will unzip the content into a temporary directory and compile the submission. The content of the ZIP file should only be extracted to the file system below the system's temporary directory (/tmp). At least in theory ...

Questions:

- (3.a) ☐ (1 point) The *SASE SUBMISSION SYSTEM* is vulnerable to a path traversal attack that allows uploading files and writing them to almost arbitrary locations. Where is this vulnerability located in the server side code?
- (3.b) ☐ (1 point) Can an attacker write to an arbitrary location? What limits the attacker, in where he or she can write files on the server?
- (3.c) ☐ (1 point) How can this vulnerability be exploited? Create an exploit that writes a file into the Tomcat web-root directory. The web-root directory is the directory, where Tomcat looks for files to serve in response to HTTP requests. Show that your attack was successful by retrieving the uploaded file from the server using your browser.

 With an Eclipse controlled Tomcat the web-root is actually in the .metadata directory of the eclipse workspace, for example:

`~/workspace/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/`
- (3.d) ☐ (1 point) What happens if you upload a JSP file? Specifically, what will the server do with the JSP file if a user opens the file on the server by surfing to its address in a browser?
- (3.e) ☐ (2 points) Implement an attack, which exploits the above vulnerability to gain access to a user's user name and password. The attack should send that information to an attacker's server. Document the code of your attack, what you upload, and where you put it.
- (3.f) ☐ (1 point) Repair the path traversal vulnerability. Describe how your fix prevents the above path traversal attack.

5.4 (7 points) Lazy Cross-Site-Scripting

The *SASE SUBMISSION SYSTEM* relies heavily on JavaScript to implement its functionality, so JavaScript is an integral part of the web application. To prevent attackers from injecting JavaScript code into the application, the *SASE SUBMISSION SYSTEM* uses both client side, and server side input validation. If everything is implemented correctly, there should not be an attack vector for a cross-site-scripting attack. If only ...



Questions:

- (4.a) ☐ (1 point) What is a cross-site-scripting vulnerability? Where is the injected code executed? Describe the difference between reflective, persistent and lazy cross-site-scripting.
- (4.b) ☐ (1 point) The *SASE SUBMISSION SYSTEM* contains at least one lazy cross-site-scripting vulnerability that allows a normal user to inject JavaScript into the administration context. Where is this vulnerability located? Where can a normal user inject JavaScript, and where will it show up in the administration context?
- (4.c) ☐ (2 points) Write an attack that exploits the above lazy cross-site-scripting vulnerability to change the compiler command in the global configuration of the *SASE SUBMISSION SYSTEM*. Document your attack and describe your attack code.
- (4.d) ☐ (1 point) Why is changing the compiler command a problem? Who controls when the compiler command is executed? What command(s) could an attacker choose?
- (4.e) ☐ (1 point) Repair the lazy cross-site-scripting vulnerability. Describe how your fix prevents the lazy cross-site-scripting attack.
- (4.f) ☐ (1 point) Can you find a reflective cross-site-scripting attack in the submission page? How can it be exploited?

5.5 (5 points) Sensitive Data Exposure

The *SASE SUBMISSION SYSTEM* is committed to protecting the sensitive information of its users. It uses client-side password digestion to prevent even the application server from ever seeing the password in plaintext. Furthermore, it uses an authorization system to prevent users from accessing sensitive information of other users.

Questions:

- (5.a) ☐ (1 point) Start Wireshark and monitor the network traffic between the browser and the *SASE SUBMISSION SYSTEM*, while logging into the *SASE SUBMISSION SYSTEM* as a user. Use the information you gather to login as the user, without using the original password.
 *By default non-root users are not allowed to capture network packets with Wireshark on a Linux system. On the reference virtual machine, you can either use `sudo wireshark` to run Wireshark as root user, or alternatively follow the instructions shown in the warning dialog (telling you that Wireshark should not be run as root) to setup proper privileges for your `sase` user.*
 *Be sure to use the `lo` or any network interfaces when capturing packets being sent to `localhost` (`127.0.0.0/8` subnet).*
- (5.b) ☐ (2 points) How can the above attack be prevented? Configure a suitable protection mechanism in Tomcat. Also, configure the web application to enforce this protection mechanism. Describe the necessary steps.
- (5.c) ☐ (1 point) Try to read out the session cookie `JSESSIONID` using JavaScript. Can you read out the cookie? Discuss the security implications.

- (5.d) ☐ (1 point) Only the administrator should have access to the global configuration values and the list of registered users. Which server component is responsible for delivering the information to an administrator's browser? Can you find a way to get this information from the web interface without being an administrator? Fix this vulnerability.

5.6 (2 points) Compiler Based Information Disclosure

The *SASE SUBMISSION SYSTEM* allows students to upload a program which is then compiled, and the compiler output is shown to the student uploading the program. For the program to be compiled correctly, the submission must consist of exactly two source files: `main.c` and `submission.c`, while additional header files are acceptable. Specifically, the default compilation command used by the *SASE SUBMISSION SYSTEM* is `gcc main.c submission.c -o submission`.

Questions:

- (6.a) ☐ (2 points) Find a way to get the C compiler to divulge the content of the `tasks/ws/src/sase.submission.system/db-init.sql` file during *compilation*.

💡 Depending on the method chosen, you may need more than one "submission" to the web-application to extract the information. Use the `clang` compiler (which is the default setting of the *SASE SUBMISSION SYSTEM*) for this task.

References

- [1] VMWare. <http://www.vmware.com/>.
- [2] Open Web Application Security Project. Owasp top 10 - 2013 the ten most critical web application security risks. Technical report, Open Web Application Security Project, 2013.