Assignment Report

By: Eduardo Sorozabal

& Pravin Dayananda

Team: A8

Course Code: BTN 710

**Table of Contents**

**1.** [**Introduction**](#_o816z2by4snk) **2**

[**2. Approach**](#_61ou0int329h) **2**

[**3. The Secure Web Application**](#_t5a2kpr0eyd9) **3**

[3.1. Development Methodology](#_u7z6c0wm6pxn) 3

[3.2. Programming Language Selection](#_6a76aa1ez4lm) 3

[3.3. How the application was secured](#_zelk3gw3tpp) 3

[3.4. Application Diagram (data flow, class diagram)](#_uos42cn6ddbs) 5

[**4. The Back-end**](#_halfkp5ibs67) **6**

[4.1 Server](#_aq494g7ykdde) 6

[4.2. Web-server](#_7cypbwcth2oa) 7

[4.3. Database and Data](#_iz1h3qwvf6fn) 8

[**5. Challenges**](#_m5puegqkyy0f) **8**

[**6. Conclusion**](#_t21ft5u5h4d7) **9**

[**7. Bibliography**](#_tawsxv3uw24l) **11**

[**8. Appendix**](#_6mrliu5fgk2l) **12**

# Introduction

This assignment required us to create a secure web application using a programming language of our choice. The Web application needed to have a nice home page with 2 links. Each link prompted the user to log in to gain access to one of two simple data search pages. These pages searched for relevant data where the user can put in various simple form based queries, populating the list from an sql database.

This web app will showcase group A8’s cumulative security knowledge in securing a system. There are many ways to exploit a default implementation of an apache web server or even the host OS making our job to secure the system all the more complicated.

This report will cover our approach as a team on how to secure the system, the development methodologies and programing languages used to secure the web application as well as detailed information as to how the web app was secured.

We will then go over the back end, as to how the server, web server and database and data were secured. We will describe our greatest challenges during this assignment and wrap up with our concluding thoughts.

# 2.Approach

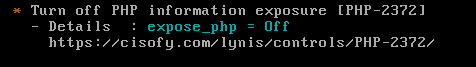
We started with no idea of how to use an Apache web server, so our approach was to learn as we went. We used many helpful tutorials to implement best practices in configuring and hardening the system.

Once the Web app was finished, we began following guides for web app, web server and centos hardening. These guides taught us the many of the flaws of default configurations as well as how to prevent many types of attacks and information leaks.

The work was iterative as we worked we had to constantly keep updating parts of the system we thought were done, as we learned about new security protocols.

We also used a helpful auditing tool to point out any additional vulnerabilities that we did not know about or missed. The auditing tool we used is Lynis. It is open source and helped us harden our config files.

Here is a sample of the Lynis Auditing tool suggestions, we had to research the tags to understand what they do, and mostly the auditing tool had very good suggestions.



The Auditing tool also suggested we download packages like mod\_evasive to protect against DDos/Dos attacks and Brute Force Attacks. It suggested a firewall but we were told that firewalls were to be left out of the assignment.

# 3. The Secure Web Application

### 3.1. Development Methodology

We planned how we would create the web app by looking at php tutorials. We separated tasks and completed them then checked over each others work. The work was very recursive and we continually returned to previous work to update or add security implementations. We coordinated tasks and communicated through skype.

### 3.2. Programming Language Selection

We used PHP because we both have past experience creating php web apps with Sql. Php is easy to use and works well with HTML. There is also lots of online documentation and tutorials available for php working with Sql.

### 3.3. How the application was secured

We secured the application by implementing best practices found on internet tutorials and the course notes. Each step we took hardened the web application.

The following is a list of the main steps taken to harden the web application to prevent many vulnerabilities such as sql Injection and information leakage.

**Secure Logins**

We secured the logins by preventing Sql Injections and malicious code. We used Sql functions that create an Sql plan, telling the database that the input is a parameter and not part of the query.

Passwords are hashed in the database, so when we validate, we compare the hash received from the database and the inputted password put through the same hashing function. This is done through the php password\_verify() function.

**Limit Users**

We created only three users, two for each search and one for an admin.

Each has their own role.

**Authentication**

Each user has a role attribute. This is used for authentication. At the start of every page that is protected: the IT Data Search and the Human Resource Data Search, we use sessions to check if the user is logged in and if not to redirect to logout.php. We also check for the proper role, and if the user does not have it they also get redirected to logout.php.

There are only three roles, one for each search and one for the admin that can access both. Sessions end when you go back to the index page or when you log out to make sure that a login is required each time and no sessions are overlapped.

**Strong Passwords**

We used strong passwords for all the users. The passwords are longer than 8 characters, and have lowercase, uppercase and special characters.

We also used a website to test password strength to make sure that they are strong passwords: passwordmeter.com.

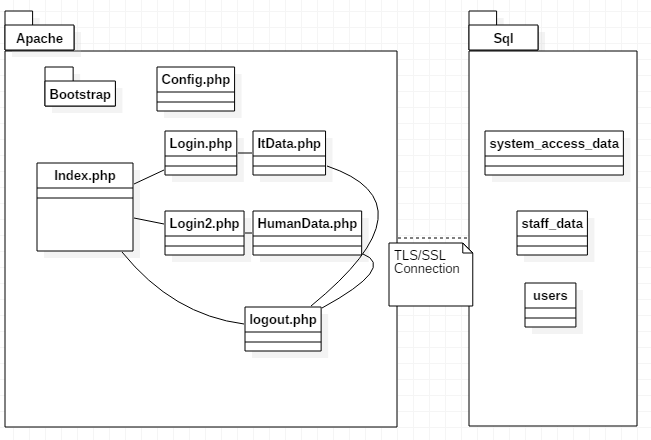
**Data Serialization**

We use Sql functions that sterilize input such as Filter\_var() and mysqli\_real\_escape\_string(). They remove some special characters or unwanted characters. We also set max length limits to the HTML inputs.

**Login Screen Bypass Prevention**

We prevent login screen bypass attacks by checking sessions at the beginning of protected pages and redirecting to the index page if the session shows that the user isn not logged in or has the proper authorization.

### 3.4. Application Diagram (data flow, class diagram)

**

# 4. The Back-end

### 4.1 Server

We added TLS/SSL and created a self signed certificate to communicate securely between the server and the web app. The SSL private key is a 2048 bit key, which is the size most big companies use. We also edited SSL Cipher to only use strong Ciphers.

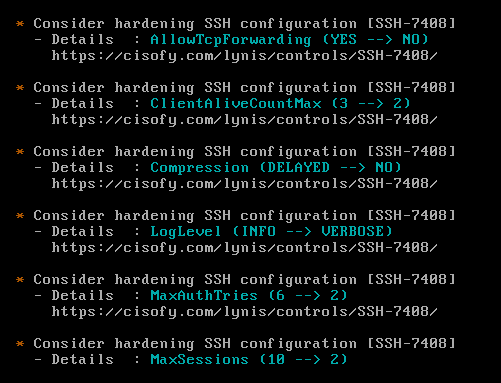
The connection to the site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE\_RSA with P-256 (a strong key exchange), and AES\_128\_GCM (a strong cipher).

We removed all unnecessary user accounts and services. We changed the root password and made it strong. We also gave single user mode and the bootloader a password.

Mod\_evasive was installed to prevent DDos/Dos attacks and Brute Force Attacks. We are aware that there are firewall and antivirus packages that can be downloaded but we did not because we were told they were not part of the assignment.

By using an audit tool called lynis we configured many SSH tags to harden the server,

the image below shows a few of the default SSH tags that we changed. There are many more all hardening the system by limiting connections and forwarding types.



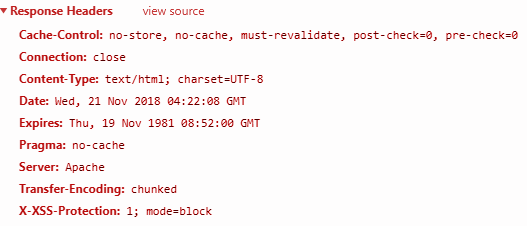
We also have backups of the vm in the form of VMware snapshots and a copy of the vm stored in a secure pc.

### 4.2. Web-server

In the default Apache configuration you would have many sensitive information disclosures, which can be used to prepare for an attack.

To reduce sensitive information we turned off several httpd.conf options that show server information.

We removed the Server Version Banner that shows the web server type and version by switching the ServerTokens to Prod and ServerSignature Off. As shown in the image below, there are no critical information leaks. The server attribute for example used to expose the version of apache being used and the os. Scouting for system information is the first step hackers take in finding system vulnerabilities so it is important to prevent information leaks.



We disabled the directory browser listing that shows the filesystem in the browser by setting the directory options to none.

Disable the Etag header to prevent lots of server information from being exposed by setting FileETag to none and disable Trace HTTP Requests and enable X-XSS-Protection to prevent cross Site Tracking attacks.

SSL is set to default, if http is used it is redirected to https.

We made sure that apache was running as a separate user and group that has minimum privileges to prevent people for being able to use this user to do much with the web server.

### 4.3. Database and Data

We removed all users, especially those without passwords and left only one localhost root and one localhost user called lessPower with select permission for the assignment database. The web app uses the lessPower user to connect and only has the SELECT permission, minimizing the harm of the user being compromised somehow.

****

We also changed the name of root to edy because root is a common name for the admin user and gave the users strong passwords.

We removed the test database that is by default accessible by all users and deleted the Sql shell history file to prevent anyone from seeing us setting up user passwords.

The database connection information for the web app is in a file called Config.php in the app folder, to keep it out of the root directory and a little safer.

All the sql data is backed up in the form of .sql files and the VM copy stored in another secure pc. We decided not to encrypt the entire database because

# 5. Challenges

We had many challenges during this assignment because we did not know how to use the required software and were experienced in security implementation. Although we had challenges they made us much better and confident at using linux systems and securing web apps.

1. Learning how to use centos, apache and php was one of our biggest challenges. We began the assignment with very little idea of how to use any of the tools required. We spent a lot of time working with tutorials and reading through hardening guides as well as the course notes to be able to secure the system. Although it was a challenge to learn so much about security and linux based servers, it was very informative and useful for our futures.
2. Learning to use the Httpd.conf file to start apache was a challenge. We could not get the web app to load on the browser. We had to re learn how to use the vi editor and use tutorials to learn how to configure apache, but for some reason it would not work. We then found out that you needed to start apache as a service for it to work.
3. Using VMware’s shared directory is easy to do, but it took us days to be able to get it to work. We needed to enable a shared folder between the vm and windows, so that we could add the .sql files and create the database tables. Using a shared folder connecting Centos to windows also made editing the web application files much easier and faster than it would have been by using the vi editor. The documentation around how to enable VMware’s shared folder feature was lacking and there were many centos specific problems. The first problem was that we needed to download VM tools, a package of drivers from VMware, but it was difficult to find the package and even harder to get it to work. Ater many tutorials on the internet we managed to get it to work, and then after more internet tutorials we managed to get the shared folder to work.

# 6. Conclusion

Security is an important part of Software Development and probably the hardest to keep up with because new security measures are always emerging and there are always people finding new ways to exploit systems. In this assignment team A8 created a secure web application on centos using apache. We chose to program in php because we had experience in php from other classes.

Our approach was one of learning as we had to research how to do most things, making the assignment all the more challenging. We secured the web app by putting all of our security knowledge into practice, and by following many best practice guides. Securing the web application included creating secure logins, creating back ups, limiting the number of users, implementing authorization in secured pages, using strong passwords, preventing sql injections using data validation and sterilization, and preventing login bypass attacks.

Securing the back end was also challenging because we had to learn a lot about how apache, centos and sql work. We secured the server by enabling TSL/SSL, using strong encryptions and cyphers. We removed many default configurations to harden the server as well as installed packages to protect from known vulnerabilities like DDos/Dos attacks. We used auditing software to show us additional vulnerabilities to fix as well as created backups of the VM. We limited users and services running and used strong passwords.

We secured the web server by configuring a lot of default apache configurations that left vulnerabilities and information leaks. We also restricted users and privileges for apache and made SSL the only way to access the web application.

We secured the sql databases by restricting users and their permissions. We made sure that each user had strong passwords, deleted the test database and created backups.

All of this posed many challenges like learning to use centos and how to best secure a web application. This assignment has taught us how complex system security is and how much we still have to learn to to keep up with the ever changing security protocols.

# 

# 7. Bibliography

“File System Partitioning.” *HowTos/SELinux - CentOS Wiki*, wiki.centos.org/HowTos/OS\_Protection.

Nagy, Tibor. “SQL Server Security Checklist.” *What Is a Virtual Log File (VLF)?*, MSSQLTips, 5 Feb. 2014, www.mssqltips.com/sqlservertip/3159/sql-server-security-checklist/.

Kumar, Chandan, et al. “Apache Web Server Hardening and Security Guide.” *Geekflare*, 30 Sept. 2018, geekflare.com/apache-web-server-hardening-security/.

# 8. Appendix

**Centos:**

**Bootloader:**

GalaxyX!@#

**Centos login:**

Root : P0pCornTime!23 (P[Number zero]pCorn...)

**SQL:**

**Enter Sql Mode as root:**

mysql -u edy -p

**Root SQL Password :**

Or@nge!23

**SQL Users:**

edy@localhost : Or@nge!23 (root)

lessPower:G00gleCrome!23 (only select permission on assignment database)

(G[Number zero][Number zero]gle)

**Database:**

assignment

**Web App:**

**Web App Files:**

/var/www/html

**Admin User:**

Admin : Pot@to!23

**It Data Form Login:**

easorozabal : T@ngerine!23

**Human Data Form Login:**

Smith : Hum@nData!23