Security Project

Emma Arfelt & Dennis Thinh Tan Nguyen & Thor Valentin Olesen

April 3, 2017

Contents

1 Risk Analysis and Security Measures

1.1 Information Assets and States

There has been identified several assets: the users and their credentials, the uploaded images and the site itself. All of these are logical assets, which can change into various states:

Asset	S_0	S_1	S_2	S_3
User credentials	Confidential	Public		
Images	Available	Correct	Confidential	Deleted
Website	Accessible	Down		

Moreover there are different assets that the are related to the system owners, such as:

Asset	S_0	S_1	S_2
Server	Restricted access	Down	Public access
Database	Restricted access	Correctness	

1.2 Threat Sources

As the service is running on a webserver, it is accessible to anyone who wishes to connect to it. It is therefore possible for any malicious user to attack the service.
The system is developed for peers in terms of vulnerability, and they are therefore the primary threat source for the system as they are assigned the task of breaking in.

1.3 Risk And Countermeasures

In this section the before mentioned risk will be used in calculating the risk. The risk calculations are based on Chapter 8 from Applied Information Security [1]. To measure the severity and likelihood of the events, the tables from page 140 are used showing the level definitions (Low, High and Medium) of impact, Likelihood and Risk.

1.3.1 Risk analysis for assets:

No	Threat	Countermeasure(s)	Likelihood	Impact	Risk
0	SQL Injection	mysqlitei prepared statements	H	H	H
1	XSS Injection	Input validation (sanitize user input)	H	M	M
2	Liberal File Permissions	Restricted access	M	M	M
3	System access control	System Hardening	H	H	H

1.3.2 Selected Countermeasures

The most effective countermeasure to the above mentioned threats are input validation and system hardening. Firstly, a clear distinction between code and data should be followed throughout the source code. Thus, any place where user input is given, it is either validated or sanitized. Further, any queries that build on user input from e.g. html input fields and forms are created using mysqlite that supports prepared statements. In this way, we can minimize the risk of injection attacks such as SQL and XSS.

Moreover, to harden the system, we have disabled unnecessary services on open ports like e.g. Telnet. This is to adhere to the minimum exposure security principle and minimize the attack surface of the adversary. For example, the adversary might scan and find vulnerabilities in the unencrypted Telnet server. The impact of this is high since an attacker may eavesdrop on a Telnet session and obtain credentials. To solve this, one might disable all other services other than the web application running on port 80 via HTTP.

Further, one might consider using HTTPS (SSL on top of HTTP) and redirect HTTP requests to avoid any communication from being unencrypted, thus being open for eavesdropping. Apart from disabling unwanted services in the system, the system has been updated, password been changed and the Apache configuration file has been modified to avoid HTTP 404 errors from exposing information about the underlying Apache server and version number.

[1] ISBN 978-3-642-24473-5

2 System Characterization

2.1 System Overview

The primary mission of the implemented system is to provide a simple platform for users to create an account and share their pictures with other users. The system should also allow users to comment on the photos they are shared with. The system is provided as a Web-based photo-sharing service that one can access through any devices with an Internet connection and a web-browser.

The boundaries in which the developer team is responsible for is to implement security safeguards in the components and also the environment that the system is executed in. An external party[2] has already implemented all of the basic required functionality relating to a Web-based photo-sharing service. Thus these core features are only to be optimized with security safeguards by the team. The environment is also provided by the third-party, so the team is only responsible for hardening it against vulnerabilities.

The overall architecture of the system is a web application that is mainly implemented in PHP. Each page is represented as HTML and its logic such as authentication, file-upload and so fourth, is implemented in PHP. Also, The system uses an SQL-database to store all its users, images, comments and their relations with each other. The environment which the system will be executed in is Linux/Ubuntu with an Apache server as a gateway for the users to access the service.

[2]The team is using the project template instead of implementing their own system.

2.2 System Functionality

The core system functionality is described in this section and are required by the stakeholders.

- A user must be able to create an account with a unique user name and password
- A user must be able to login with existing and valid credential information
- A user session is remembered when a sign-up/authentication was successful
- A user when logged in, must be able to upload an image.
- A user when logged in, can post a comment on his/her own images or images that has been shared with the given user.
- A user
- A user when logged in, can share any owned images to other users
- A user when logged in, can remove any users from an image which has been shared with.

All of these functionalities were already implemented by a third-party, therefore the team will mainly go through all the implemented functionality and implement safeguards against various malicious input and operations that can be sent and performed on the system.

2.3 Components and Subsystems

This section will list all system components subdivided into categories such as platforms, data records and application files. For each component, relevant properties have been stated.

2.3.1

Platform The web application is running on a virtual machine using an Apache server under Ubuntu 4. In this regard, one of the main concerns in securing the application is to ensure that the machine on which the application is deployed, has been hardened.

In other words, the Ubuntu environment should be secured by reducing its surface of vulnerabilities that may potentially be exploited by an adversary. In practice, one may disable all services that are not needed by the web server to fully function. For example, one may disable the ftp and telnet services running in the Ubuntu environment since the data transmission packet is unencrypted, allowing a potential adversary to easily sniff and obtain these information. Thus, the only port to be opened is the one used by the web server that resides on port 80 and uses the HTTP protocol for data communication.

However, HTTP is not encrypted and is vulnerable to man-in-the-middle and eavesdropping attacks, which can let the adversary gain access to website accounts and sensitive information or modify the webpage to inject malware. Thus, one might also want to use SSL on top of HTTP (i.e. HTTPS) on port 443 and redirect HTTP requests to port 443 using HTTPS to encrypt all data communication. By doing this, one will adhere to the security principle of minimum exposure and reduce the potential attack surface of the environment on which the application is running.

Finally, one might want to analyze the default file permissions used in the server environment and make sure that users only have the least privileges required for them to use the application. In this way, one will adhere to the least privilege security principle and minimize errors.

Applications Files The source code is comprised of a range of php files used to create the web application. The most noteworthy part of the program is the **ssas.php** file that constitutes the web server and all logic of the application. It is used for server side requests and handles users (create and register), authentication (login, logout), session management, images and comments. Further, it stores inner classes used to represent a User, Image and Comment. Further, the **index.php** file is used as the landing page of the web application. Finally, the **ssas.php** file used a range of php files (e.g. login, logout, register and image) to separate some of the functionality supported in the image sharing application.

Common to all of these files is the fact that user input is provided in many places without being validated or sanitized. Thus, to secure the web application one should make sure to make a clear distinction of actual code and data to avoid any injection based attacks. One approach to this would be to sanitize user input like 'username' or 'password' before using it in the code. Currently, the server only checks if the input is empty: textbfif($username == ""). To solve this, one might use a sanitize class in php with helper functions to validate it before actually using the data: textbfsanitize_text_field($_POST['username']);. Another example are the numerous places where SQL statements are created directly from the user input, thus being subject to sql injection attacks: textbf$query = "INSERT INTO user(username,password) VALUES ('".$username."','".$password."');";.
In this regard, one should either validate the user input first before constructing the query or used prepared statements to bind the values before executing the SQL query statement. To solve this with a prepared statement, one may bind values in the following way: if textbf($query = self::$mysqli -¿ prepare('INSERT INTO user(username,password) VALUES (?,?)')) $query -¿ bind_param('ss', $username,$hash); $query -¿ execute();.
Further, the error reporting is enabled by default in PHP allowing both the programmer and adversary to get useful information about the system. To secure the web application further, one might disable error reporting to avoid the adversary from getting information about the site, server and underlying information. Ultimately, this may be used to find exploits on the Internet for e.g. a given server and version number. Thus, the error reporting has been disabled in PHP by setting "error_reporting" to "0".

Overall findings All together, the above findings showcase that the web application may be subject to SQL injections, XSS injections, error reporting issues and system hardening issues. The injection attacks arise due to the lack of input validation and sanitization of user input, which has been one of the major changes forced upon the secure version of the web application. Further, the system has a lot of services enabled by default, which may be disabled to harden the system. Finally, error reporting may be disabled to prevent the adversary from obtaining useful information about the system.

Data Records : The data model has been made in Sqlite and is comprised of four entities. Specifically, a user entity, image entity, shared_image entity and post entity. Firstly, the user entity is used to represent a user of the system that has a user name and password used for authentication. Secondly, an image entity is used to store pictures for a given user. Thirdly, a shared_image entity us used to store images shared by other users. Finally, a post entity exists to allow users to create a post with an image and text. All together, these entities constitute the data model of the image sharing application. In this regard, it is important to guard against SQL injections and make sure to validate user input where SQL queries are used server-side to retrieve data.

2.4 Interfaces

This section will specify all interfaces and information flow for system.

Technical The entry point of all data flows starts from each view components of the system. The data is retrieved from some text field when a submit button is pressed and then parsed down to the business logic. For this system the view is implemented in the following files and each has their own specific responsibility area.

- header.php
- image.php
- login.php
- register.php
- upload.php

The main business logic is located in **ssas.php** which main responsibility is to perform business logic on the data that it receives. For an example, authenticating a user based on the provided credential information and provide the next set of action based on whether the authentication was successful or not.

2.5 Backdoors

This section describes the implemented back doors. The section has been omitted in the version of the report that is handed over to the team that reviews our system.

2.6 Easy vulnerability

If the user tries to write '1' in the comment field of an image, the PHP code in ssas.pdf will open a shell and execute a command by using the remaining input of the comment. For example, the comment "1whoami" whill return "www-date" inside a ¡pre¿ html tag that will be displayed on top of the page. This may potentially be exploited by the adversary to gain root access through privilege escalation.

2.6.1 Difficult vulnerability

We have altered the file permissions of /etc/shadow, such that it is world readable. This way an adversary gets access to current passwords, and is able to try to perform password cracking on the root password. The root password is very easy to hack: `sunshine`.

We used the following commands: `chmod =r /etc/shadow`

2.7 Additional Material

Security Goal

- Confidentiality: prevent unauthorized access to information
- Availability: authorized users should have access to system
- Integrity: prevent unauthorized altering of information
- Accountability: actions of a principal may be traced uniquely to that principal

Security Principles

- **Simplicity: keep it simple**
  - Simple systems contain lss potential flaws
- **Open design: security should not depend on secrecy of its protection mechanisms**
  - Example: not realistic to maintain secrecy for any system
- **Compartmentalization: organize resources into isolated groups of similar needs**
  - Example: a submarine has multiple isolated rooms to avoid sinking entire vessel upon a leak
- **Minimum Exposure: minimize attack surface a system presents to adversary**
  - Example: a castle only has one or two entry points to minimize potential intruder access
- **Least privilege: any component should operate using least set of privileges**
  - Example: assistant only needs key to own office but boss needs to office and main door
- **Minimum trust and maximum trustworthiness: minimize trust and turn assumptions into validated properties**
  - Trustworthy system: satisfies user expectations
  - Trusted system: user assumes that system satisfy expectations but it may "misbehave"
  - Example: system should not rely on valid inputs but verify input and take corrective actions (e.g. countries do not trust individuals at borders but evidence like passports)
- **Secure fail-safe defaults: system should start in and return to secure state in event of failure (recovery)**
  - Example: doors to building are locked when closed and cannot be opened from the outside
- **Complete Mediation: access to any object must be monitored and controlled**
  - Example: airport ensures that subject is authenticated before entering sensitive areas of airport or boarding a plane (solved by controlling passenger flow)
- **No single point of failure: build redundant security mechanism whenever feasible!**
  - Example: security should not rely on a single mechanism so e.g. credit card and PIN are delivered in separate letters (loss of one does not compromise the other)
- **Traceability: log security-relevant system events**
  - A trace is a sign or evidence of past events
- **Generating Secrets: maximize entropy (order) of secrets**
  - Example: a bike combination lock is easier to open with smaller key spaces (e.g. 10^3 than 10^5 options

Program Findings

Whitebox

Filename	Line	Vulnerability	Principles	Goal	Solution
ssas.php	355 Function comment	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injections in comments	Sanitize input Or user SQL prepared statements
ssas.php	116 Function createUser	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injections when creating an account. (User/pass fields)	Sanitize username, password input Or user SQL prepared statements
ssas.php	129 Function login	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injections when logging in. (User/pass fields)	Sanitize username and password input Or user SQL prepared statements
ssas.php	176 Function uploadImage	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injections when uploading images	Use prepared statements and sanitize input
ssas.php	192 Private function getUserId	SQL injection	Minimum Trust - Maximum Trustworthiness	Private function uses input in sql query → Avoid injection	Sanitize username input Or user SQL prepared statements
ssas.php	210 Function removeShare	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when removing share of photo with users	Sanitize iid and username input Or user SQL prepared statements

ssas.php	229 Function shareImage	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when sharing image	Sanitize iid input Or user SQL prepared statements
ssas.php	246 Function getUsersToShareWith	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when retrieving a list with all users one can share a picture with	Sanitize input
ssas.php	268 Function shareWith	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when sharing a picture with a specific user	Sanitize iid input
ssas.php	288 Function save_image	File permission	Least privilege	Integrity - Confidentiality Prevent adversary to use max privilige	Function uses chmod(... , 0777) to set file permissions. By using 777 tag, the file/dir has full read,write and executable for the whole world
----------					
Header.php	1-5	Error Reporting	Minimum exposure	Confidentiality	Disable error tracing on client side Link: https://secure.php.net/manual/en/function.ini-set.php https://secure.php.net/manual/en/function.error-reporting.php
image.php	1-5	Error Reporting (not disabled)	Minimum Exposure	Confidentiality	ini_set('display_errors', 0); error_reporting(~0);
index.php	1-5	Error	Minimum	Confidential	ini_set('display_errors', 0);

		Reporting (not disabled)	Exposure	ity	error_reporting(~0);
login.php	1-5	Error Reporting (not disabled)	Minimum Exposure	Confidentiality	ini_set('display_errors', 1); Remove this in final version ?

logout.php	1-5	Error Reporting (not disabled)	Minimum Exposure	Confidentiality	ini_set('display_errors', 1);
					error_reporting(~0);
					Remove this in final version ?

upload.php	1-5	Error Reporting (not disabled)	Minimum Exposure	Confidentiality	ini_set('display_errors', 1); error_reporting(~0);
ssas.php	350 - function comment	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when adding a comment to a picture	Sanitize iid input and use prepared statements https://stackoverflow.com/questions/548986/mysql-vs-mysqli-when-using-php
ssas.php	364 - function getComments	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when fetching all comments for a given image	Sanitize iid input and use prepared statements https://stackoverflow.com/questions/548986/mysql-vs-mysqli-when-using-php
ssas.php	390 - function isOwner	SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection when finding owner of image	Sanitize iid input and use prepared statements https://stackoverflow.com/questions/548986/mysql-vs-mysqli-when-using-php
ssas.php	401 - verifyInput	???	???	???	Should it be there?
ssas.php	408 - function - verifyShare	SQL injection	Minimum Trust - Maximum Trustwort	Avoid injection when checking if	Sanitize uid and iid input and use prepared statements https://stackoverflow.com/questions/548986/mysql-vs-mysqli-when-usin

			hiness	user has access or is owner of image	g-php
ssas.php	Username and password throughout all code	XSS/SQL injection	Minimum Trust - Maximum Trustworthiness	Avoid injection of code when using username or password input	Sanitize username and password and use prepared statements Link: https://secure.php.net/manual/en/function.htmlspecialchars.php

Primary vulnerabilities

- SQL injection
- XSS injection
- Error reporting (not disabled)
- System: harden it by disabling services on open ports and minimize file writes