Review of Group a by Group b

w x y z

...

Page limit: 18 pages.

Contents

1 Review of the External System

1.1 Background

Developers of the external system: *x', y', z', ...*

Date of the review: ...

1.2 Completeness in Terms of Functionality

Does the system meet the requirements given in the assignment?

Table 1: Completnes

No	Completness	
1	The system should allow the user to upload pictures	yes
2	The user can share his own pictures with other named users on a pictureby-picture basis	yes
3	The user can view his own pictures and pictures other has shared with him	yes
4	The user can comment on any picture he can view	yes
5	The user can view comments on any picture he can view	yes

1.3 Architecture and Security Concepts

Study the documentation that came with the external system and evaluation. Is the chosen architecture well suited for the tasks specified in the requirements? Is the risk analysis coherent and complete? Are the countermeasures appropriate?

1.4 Implementation

While looking at the solution, we found some vulnerabilities.

WEB SITE

- **Cleartext submission of password**
  /sec/views/login.html

  Issue background
  Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network

traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Issue remediation
Applications should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

- **Form does not contain an anti-CSRF token**
  - /sec/views/login.html
  - /sec/views/profile.html

Issue background
Cross-site Request Forgery (CSRF) is an attack which forces an end user to execute unwanted actions on a web application to which he/she is currently authenticated. With a little help of social engineering (like sending a link via email / chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and may allow an attacker to perform an account hijack. If the targeted end user is the administrator account, this can compromise the entire web application.

Issue remediation
The application should implement anti-CSRF tokens into all requests that perform actions which change the application state or which add/modify/delete content. An anti-CSRF token should be a long randomly generated value

unique to each user so that attackers cannot easily brute-force it.

It is important that anti-CSRF tokens are validated when user requests are handled by the application. The application should both verify that the token exists in the request, and also check that it matches the user's current token. If either of these checks fails, the application should reject the request.

- **Request vulnerable to Cross-site Request Forgery**
  - /sec/resources/account/create
  - /sec/resources/account/login
  - /sec/resources/protected/file
  - /sec/views/login.html

Issue background

Cross-site Request Forgery (CSRF) is an attack which forces an end user to execute unwanted actions on a web application to which he/she is currently authenticated. With a little help of social engineering (like sending a link via email / chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and may allow an attacker to perform an account hijack. If the targeted end user is the administrator account, this can compromise the entire web application.

Issue remediation

The application should implement anti-CSRF tokens into all requests that perform actions which change the application state or which add/modify/delete content. An anti-CSRF token should be a long randomly generated value unique to each user so that attackers cannot easily brute-force it.

It is important that anti-CSRF tokens are validated when user requests are handled by the application. The application should both verify that the token exists in the request, and also check that it matches the user's current token. If either of these checks fails, the application should reject the request.

- **Password submitted using GET method**
  - /sec/views/login.html
  - /sec/views/login.html

Issue background

Some applications use the GET method to submit passwords, which are transmitted within the query string of the requested URL. Sensitive information within URLs may be logged in various locations, including the user's browser, the web server, and any forward or reverse proxy servers

between the two endpoints. URLs may also be displayed on-screen, bookmarked or emailed around by users. They may be disclosed to third parties via the Referer header when any off-site links are followed. Placing passwords into the URL increases the risk that they will be captured by an attacker.

Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Issue remediation
All forms submitting passwords should use the POST method. To achieve this, applications should specify the method attribute of the FORM tag as method="POST". It may also be necessary to modify the corresponding server-side form handler to ensure that submitted passwords are properly retrieved from the message body, rather than the URL.

- **Open redirection (DOM-based)**

Issue detail
The application may be vulnerable to DOM-based open redirection. Data is read from document.location and passed to document.location via the following statements: var a = document.location.toString().substr(0,document.location.toString().length-1)+":8080/sec"; document.location = a;

Issue background
DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way.

DOM-based open redirection arises when a script writes controllable data into the target of a redirection in an unsafe way. An attacker may be able to use the vulnerability to construct a URL that, if visited by another application user, will cause a redirection to an arbitrary external domain. This behavior can be leveraged to facilitate phishing attacks against users of the application. The ability to use an authentic application URL, targeting the correct domain and with a valid SSL certificate (if SSL is used), lends credibility to the phishing attack because many users, even if they verify these features, will not notice the subsequent redirection to a different domain.

Note: If an attacker is able to control the start of the string that is passed

to the redirection API, then it may be possible to escalate this vulnerability into a JavaScript injection attack, by using a URL with the javascript: pseudo-protocol to execute arbitrary script code when the URL is processed by the browser.

Burp Suite automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine whether this vulnerability is indeed present, or whether mitigations are in place that would prevent exploitation.

Issue remediation
The most effective way to avoid DOM-based open redirection vulnerabilities is not to dynamically set redirection targets using data that originated from any untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from introducing an arbitrary URL as a redirection target. In general, this is best achieved by using a whitelist of URLs that are permitted redirection targets, and strictly validating the target against this list before performing the redirection.

- **Password field with autocomplete enabled**
  - /sec/views/login.html
  - /sec/views/login.html

Issue background
Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

The stored credentials can be captured by an attacker who gains control over the user's computer. Further, an attacker who finds a separate application vulnerability such as cross-site scripting may be able to exploit this to retrieve a user's browser-stored credentials.

Issue remediation
To prevent browsers from storing credentials entered into HTML forms, include the attribute autocomplete="off" within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).

Please note that modern web browsers may ignore this directive. In spite of this there is a chance that not disabling autocomplete may cause problems obtaining PCI compliance.

- **Cross-site scripting (reflected)**

  Issue detail
  The value of the username request parameter is copied into the HTML document as plain text between tags. The payload z2ax5¡script¿alert(1)¡/script¿gm3p0 was submitted in the username parameter. This input was echoed unmodified in the application's response.

  This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

  The response does not state that the content type is HTML. The issue is only directly exploitable if a browser can be made to interpret the response as HTML. No modern browser will interpret the response as HTML. However, the issue might be indirectly exploitable if a client-side script processes the response and embeds it into an HTML context.

  Issue background
  Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

  The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

  Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

  The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and func-

tionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other applications that it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk.

Issue remediation
In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized. User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including ¡ ¿ " ' and =, should be replaced with the corresponding HTML entities (< > etc). In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

SYSTEM

- **phpinfo() output accessible**

Impact
Some of the information that can be gathered from this file includes: The username of the user who installed php, if they are a SUDO user, the IP address of the host, the web server version, the system version(unix / linux), and the root directory of the web server.

Solution
Delete them or restrict access to the listened files.

- **php Multiple Vulnerabilities**
  Installed Version: 5.5.9

  #1
  CVE: CVE-2015-4148, CVE-2015-4147, CVE-2015-2787, CVE-2015-2348, CVE-2015-2331
  Impact
  Successfully exploiting this issue allow remote attackers to obtain sensitive information by providing crafted serialized data with an int data type and to execute arbitrary code by providing crafted serialized data with an unexpected data type.

  Solution
  Upgrade to php 5.4.39 or 5.5.23 or 5.6.7 or later. For updates refer to http://www.php.net

- **#2**
  CVE: CVE-2015-4026, CVE-2015-4025, CVE-2015-4024, CVE-2015-4022, CVE-2015-4021

  Impact
  Successfully exploiting this issue allow remote attackers to cause a denial of service , bypass intended extension restrictions and access and execute files or directories with unexpected names via crafted dimensions and remote FTP servers to execute arbitrary code.

  Solution
  Upgrade to php 5.4.41 or 5.5.25 or 5.6.9 or later. For updates refer to http://www.php.net

- **#3**
  CVE: CVE-2015-3329, CVE-2015-3307, CVE-2015-2783, CVE-2015-1352

  Impact
  Successfully exploiting this issue allow remote attackers to cause a denial of service, to obtain sensitive information from process memory and to execute arbitrary code via crafted dimensions.

  Solution
  Upgrade to php 5.4.40 or 5.5.24 or 5.6.8 or later. For updates refer to http://www.php.net

- **#4**
  CVE: CVE-2015-6831, CVE-2015-6832, CVE-2015-6833

  Impact
  Successfully exploiting this issue allow remote attackers to execute arbitrary code and to create or overwrite arbitrary files on the system and this may lead to launch further attacks.

  Solution
  Upgrade to php version 5.4.44 or 5.5.28 or 5.6.12 or later. For updates refer to http://www.php.net

- **#5**
  CVE: CVE-2015-3330

  Impact
  Successfully exploiting this issue allow remote attackers to cause a denial of service or possibly execute arbitrary code via pipelined HTTP requests.

  Solution
  Upgrade to php 5.4.40 or 5.5.24 or 5.6.8 or later. For updates refer to http://www.php.net

- **php Multiple Remote Code Execution Vulnerabilities**
  CVE: CVE-2015-0273, CVE-2014-9705

  Impact
  Successfully exploiting this issue allow remote attackers to execute arbitrary code via some crafted dimensions.

  Solution
  Upgrade to php 5.4.38 or 5.5.22 or 5.6.6 or later. For updates refer to http://www.php.net

- **php Use-After-Free Remote Code Execution Vulnerability**
  CVE: CVE-2015-2301

  Impact
  Successfully exploiting this issue allow remote attackers to execute arbitrary code on the target system.

Solution
Upgrade to php 5.5.22 or 5.6.6 or later. For updates refer to http://www.php.net

- **php Use-After-Free Denial Of Service Vulnerability**
  CVE: CVE-2015-1351

  Impact
  Successfully exploiting this issue allow remote attackers to cause a denial of service or possibly have unspecified other impact.

  Solution
  Upgrade to php 5.5.22 or 5.6.6 or later. For updates refer to http://www.php.net

- **php 'serialize_function_call' Function Type Confusion Vulnerability**
  CVE: CVE-2015-6836

  Impact
  Successfully exploiting this issue allow remote attackers to execute arbitrary code in the context of the user running the affected application. Failed exploit attempts will likely cause a denial-of-service condition.

  Solution
  Upgrade to php version 5.4.45, or 5.5.29, or 5.6.13 or later. For updates refer to http://www.php.net

- **php 'phar_fix_filepath' Function Stack Buffer Overflow Vulnerability**
  CVE: CVE-2015-5590

  Impact
  Successfully exploiting this issue allow remote attackers to execute arbitrary code in the context of the PHP process. Failed exploit attempts will likely crash the webserver.

  Solution
  Upgrade to php version 5.4.43, or 5.5.27, or 5.6.11 or later. For updates refer to http://www.php.net

- **php Multiple Denial of Service Vulnerabilities**
  CVE: CVE-2015-7804, CVE-2015-7803

Impact
Successfully exploiting this issue allow remote attackers to cause a denial of service (NULL pointer dereference and application crash).

Solution
Upgrade to php 5.5.30 or 5.6.14 or later. For updates refer to http://www.php.net

- **php Out of Bounds Read Memory Corruption Vulnerability**
  CVE: CVE-2016-1903

  Impact
  Successfully exploiting this issue allow remote attackers to obtain sensitive information or cause a denial-of-service condition.

  Solution
  Upgrade to php version 5.5.31, or 5.6.17 or 7.0.2 or later. For updates refer to http://www.php.net

- **Apache HTTP Server Multiple Vulnerabilities**
  CVE: CVE-2015-3185, CVE-2015-3183

  Impact
  Successful exploitation will allow remote attackers to bypass intended access restrictions in opportunistic circumstances and to cause cache poisoning or credential hijacking if an intermediary proxy is in use.

  Solution
  Upgrade to version 2.4.14 or later, For updates refer to http://www.apache.org

1.5 Backdoors

If defining a backdoor as a way to gain root access to the system. Then we didn't manage to exploit any.
One thing we do know, is that the server is running "WildFly", and that comes with a webinterface, that will give you access to a terminal to the system. This is of course proteced with authentication. We have tryed an attempt to break this, with the top 500 most used password, and the username admin - But with no luck.

1.6 Comparison

Compare your system with the external system you were given for the review. Are there any remarkable highlights in your system or the external system?