Web App Security Assessment

Information Security Lab - Assignment 3

Manish Choudhary - November 1, 2014

Severity Rating Methodology

The OWASP risk rating methodology has been used to assign the severity rating to the vulnerabilities found. The risk associated with a vulnerability exploitation can be represented as the product of the likelihood of exploitation and the impact of a successful exploitation.

$$Risk = Likelihood * Impact$$

The most important factors governing the likelihood fall into two categories - Threat Agent Factors (capabilities and resources available to the the attacker) and Vulnerability Factors (how easy it is to discover and exploit the vulnerability). These factors have been rated from 1 to 10 for each vulnerability and represented in this form:

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall

The impact of a successful attack can be estimated based on technical and business impact factors. As we have not been given any information about the business environment, the business impact has not been considered here. Companies can use this application locally or it can be used as a public bug tracker. So, the business impact will vary depending on the usage. But, it can also be added easily following the guidelines on OWASP webpage.

The technical impact factors include the core security goals of confidentiality, integrity, availability and accountability. These have been rated from 1 to 10 and represented as following:

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall

The overall threat Agent, vulnerability and impact ratings have been used to decide the severity level of each vulnerability. An average of threat agent rating and vulnerability rating has been taken to get the rating for likelihood. Then, the likelihood rating and impact rating have been averaged to get the overall rating which would be between 1 and 10. This is different from OWASP as they don't assign a final numeric rating and just calculate severity in terms of low, medium, high and critical.

Note: It is difficult to figure out the exact rating for each factor exactly and thus, worst case possibility has been considered in such scenarios. Skill level has been considered to be that of security penetration skills in all cases. Finding the severity of a vulnerability depends upon the knowledge and expertise in the domain to see its likelihood and impact. So, I have tried to give the best possible rating as per my knowledge. In addition, OWASP top 10 list published in 2013 has been considered to identify if the vulnerability belongs to OWASP top 10.

Vulnerability 1

Vulnerability

Security Misconfiguration

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	9	9	7.75

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	9	6.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	0	0	7	2.25

Overall: 5

This discloses some information about the paths and admin directory and thus, impacts the confidentiality but doesn't have direct impact on the integrity and availability. The attacker has to find some actual exploitable vulnerabilities (or access the admin directory) using this information.

Vulnerability Description

Some warnings, notices or errors get displayed along with the path of the files on the server. For example:

1) On each page:
   Warning: ob_start(): function 'compress_handler' not found or invalid function name in /var/www/mantis/core.php on line 18
2) On each page:
   Notice: ob_start(): failed to create buffer in /var/www/mantis/core.php on line 18
3) On the login page:
   WARNING: Admin directory should be removed.
4) On accessing http://127.0.0.1/mantis/config_defaults_inc.php:
   Displays some notices along with some keywords and file paths on the server
5) On accessing http://127.0.0.1/mantis/print_all_bug_options_inc.php
   Fatal error: Call to undefined function config_get() in /var/www/mantis/print_all_bug_options_inc.php on line 13
6) On accessing http://127.0.0.1/mantis/view_all_inc.php
   Fatal error: Call to undefined function config_get() in /var/www/mantis/view_all_inc.php on line 13

So, this information is available to anyone. It provides some idea to the attacker about the paths on the server. The warning related to admin directory gives a hint that the admin directory has not been removed and thus, the attacker should try to access that. Using the

information provided, the attacker can further plan which areas he can explore to find the vulnerabilities.
First three examples are visible on the index page itself. The adversary would have to guess or should know the filenames in some way to request the php files in the last three examples.

Source and Mitigation

The reason is that the access to these files is not checked and thus, anyone knowing the name of the files can try to call them directly. The compiler throws some error/warning which gets displayed to the client. The admin directory warning might have been added to remind the administrator to remove that before making the application up and running. The admin directory check is done in login_page.php:

```
Check if the admin directory is available and is readable.
  $t_admin_dir = dirname( __FILE__ ) . DIRECTORY_SEPARATOR . 'admin' . DIRECTORY_SEPARATOR;
  if ( is_dir( $t_admin_dir ) && is_readable( $t_admin_dir ) ) {
            echo '<div class="warning" align="center">', "\n";
            echo '<p><font color="red"><strong>WARNING:</strong> Admin directory should be removed.</
font></p>', "\n";
            echo '</div>', "\n";
```

The ob_start('compress_handler') statement in core.php results in the first two errors.
To mitigate this, the access to the files should be checked. The errors thrown by the compiler shouldn't be displayed to the client and customized error messages should be displayed. The admin directory related message shouldn't be displayed. Otherwise, the admin should be more careful or some other way should be used to warn.

OWASP Top 10 Category

Yes. A5-Security Misconfiguration as the unnecessary features have been left enabled and the errors reveal the paths on the server.

Vulnerability 2

Vulnerability

Security Misconfiguration

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	9	9	7.75

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	3	7	7	5.75

Overall: 6

This can make much critical information available to the attacker and can also impact integrity and the availability in the worst case (when the database upgrade results in failure loosing all the data). This is visible to everyone and thus, no special efforts are required to find and exploit this. But, efforts would be required to figure out the ways to cause database upgrade failures.

Vulnerability Description

Admin directory was not removed after installation. An attacker can access the directory which gives some administrative options that can be exploited to gain some information or to undertake privileged activities.

"Check your installation" option reveals some information about the application environment and settings such as versions, paths, supported crypto and functions used.
"Upgrade installation" may affect the database and may even result is loss of complete database if the attacker can cause the upgrade to fail in some way or force the unexpected database update making it incompatible (although it has not been tested).
"Modify stylesheet" option lets the attacker to change the user interface and thus, affect the integrity in some sense. It also gives information about some pages and options in the application.
"System utils" provide some options such as viewing the database stats.

Attacker can use any of these functionalities to gather information which would help to plan attacks further or to try to affect the functionality of the existing system by updating the databases or modifying the style sheets in an unauthorized and unauthenticated way.

Source and Mitigation

The reason is the warning message getting displayed on the login page. But, the actual reason in the lapse at the administrator's end. The admin or the person setting up the application is responsible to remove this directory after installation.

To mitigate this, the admin directory should be made unavailable. The administrators responsible for installation should be more careful about such mistakes.

OWASP Top 10 Category

Yes. A5-Security Misconfiguration as an unnecessary feature has been made available to everyone.

Vulnerability 3

Vulnerability

HTTPs Enforcement / HTTP Strict Transport Security

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	9	7	9	8.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
9	9	6	9	8.25

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
9	9	1	7	6.5

Overall: 9

This seriously affects the confidentiality and the integrity of the traffic exchanged and can be performed by an attacker almost without being traced back. This doesn't have any additional direct affect on availability. The availability can be disrupted by other means also or even if HTTPs has been used. But, the important impact is the loss of confidentiality and integrity. Though the average comes out to be 8, I'll give it 9 rating as it is high priority.

Vulnerability Description

HTTP protocol has been used to communicate between the client and the server. The data packets are transmitted through an unencrypted channel. This is a really big hole in the security as the attacker can compromise the confidentiality and integrity by performing operations such as:

- Sniff the traffic to obtain all the data transferred between client and server specially confidential information such as credentials and session cookies.
- Establish a man-in-the-middle and modify the packets impacting the integrity of the data.
- Establish a fake server as no server authentication is enforced

This is really easy to demonstrate the impact of this vulnerability. Traffic capturing and analysis tools such as Wireshark can be used to capture the packets and obtain sensitive information.
But, the attacker need some kind of access to the target network. So, if we can provide an air-gap network, we can avoid this. But, such scenarios are not practical. Therefore, the attacker can use port mirroring/networking tapping to sniff the traffic. Attacker would have to work as a proxy to establish MITM.

Source and Mitigation

The reason for this vulnerability is that the use of HTTPs protocol has not been enforced. To mitigate this, the developers should provide the support for HTTPs and use the HSTS (HTTP Strict Transport Security) header mechanism to force the web browsers to establish connections only over secure channels and thus, ensuring that all the traffic is exchanged over HTTPs.

OWASP Top 10 Category

Yes. A6-Sensitive Data Exposure as sensitive data is transmitted in clear and security directives/headers haven't been used.

Vulnerability 4

Vulnerability

Improper Session Management - Session Token Reuse Vulnerability

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	9	6.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	5	7	5.75

Overall: 7

The adversary can hijack the session impacting the confidentiality, integrity and the availability of the system with respect to the compromised user. It is easy to discover and exploit this vulnerability as HTTP has been used and thus, traffic captures can reveal this information. Adversary can also observe this behavior as an application user if he can get an account created.

Vulnerability Description

Same cookie is getting used across the sessions for a user. Probably, the cookies are not getting unset properly.

cookies for cs6265-developer:
MANTIS_STRING_COOKIE=fdac11efe7cc7c1806aa22cf83225c3bab92ed9d491607badbc8da84a85b9aac
Same cookie is getting assigned to the user “cs6265-developer” on each login.

cookies for cs6265-reporter:
MANTIS_STRING_COOKIE=09929f0b02bd76c7e8034e7e626990c7b550c5835d2299c2446063b46683bdcd
Same cookie is getting assigned to the user "cs6265-reporter" on each login.

Attacker can use some XSS vulnerability or sniff the traffic to steal the cookie. Later on, attacker can hijack the user session (login as the user) as the same cookie would be used.
The attacker is assumed to have some way to steal the cookie once so that he can use it to log in as the user.

Source and Mitigation

The reason is that the cookies are not unset properly. The problem existing in "logout_page.php" page and the calls made from this code section to perform session logout. The session cookies should be destroyed along with the session data at logout.

OWASP Top 10 Category

Yes. A2-Broken Authentication and Session Management as the session cookies haven't been handled properly.

Vulnerability 5

Vulnerability

Cookie Attribute Vulnerability

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
5	3	4	8	5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	3	1	7	4.25

Overall: 5

This vulnerability makes it easier for an attacker to steal the cookies which can be used to hijack the session. But, it just helps the attacker and he still needs some way to sniff the cookies and thus, exploiting it is not straightforward. As it facilitates the cookie theft, its impact can be seen in terms of cookie compromise. This can impact the confidentiality, integrity and availability of the system limited by the access rights of the compromised user. But, the direct impact of this vulnerability is mainly the easy of cookie theft and thus, confidentiality.

Vulnerability Description

Secure Attribute is not used and thus, is not forced to send cookies over encrypted channel. So, the session cookies can be sniffed by the attacker which can lead to user account compromise. HTTPs has to be used to allow the use of secure attribute. Use of same cookie across the sessions makes it even worse. Here, the attacker is assumed to have some access to the network so that he can sniff the traffic between client and the server.

HttpOnly attribute is not used which should be used to provide some level of protection against cross site scripting attack. As the HTTPOnly attribute is not used, some XSS vulnerability can be exploited to steal the cookies. HTTPOnly prevents the client side script from accessing the cookies and thus, even though it doesn't provide protection against XSS, it prevents the access to the cookies.

Source and Mitigation

The mitigation is to set these attributes for the cookies. ***Secure*** Attribute will force the use of HTTPs and thus, encrypted exchange of cookies. The developers will have to enforce the use of HTTPs for this. Although ***HttpOnly*** protection is not supported by all the browsers and can be bypassed using cross site tracing, it provides some level of protection and increases the complexity for the attacker as he can't use the XSS vulnerability to execute some script on the client to steal the cookies.
The fix needs to be done in the code related to setting up the cookies so that these attributes can also be set along with the cookies.

OWASP Top 10 Category

As these are missing attributes when sensitive data is exchanged, it can be considered as A6-Sensitive Data Exposure.

Vulnerability 6

Vulnerability

Insecure Password Modification

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	3	6	8	6

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	3	5	7	4.25

Overall: 5
The vulnerability is easy to discover if the attacker can analyze the functionality of the web application. Attacker can't directly change the password for some user and thus, need some other attack (CSRF or session hijack) to achieve this. The direct impact is on integrity (change in password) and availability (denial of access). As attacker would have access to the account, he can also impact the confidentiality of the system limited by the compromised user's access rights.

Vulnerability Description

The web application allows the user to change the password without requiring him to enter the current password. If the attacker can hijack the session, he can change the password and

thus, the legitimate owner of the account would be denied access. A temporary denial of service can be launched while keeping the control to the account.
It is assumed that the attacker has some means to hijack the current session. For example, the session cookies can be stolen to login as the victim. Then, the attacker can change the password even if he doesn't know the current password. But, if the attacker can sniff the cookies, he can also see the credentials transmitted. However, it is still not recommended to allow password change without asking for current password.
A simpler attack would be CSRF. As the attacker doesn't need to know the current password, he can make the victim to post a password change request (if the user is logged in) without needing to hijack the session. In this way, attacker can take control of the account and can also cause denial of service for the victim.

Source and Mitigation

The source is the insecure password change functionality. The *account_page.php* page allows to change the password without asking for the current password. The developers should change this page so that the user is asked to enter the current password every time he wants to change the password. The current password should be checked before accepting the change. This would result in better and more secure password change mechanism.

OWASP Top 10 Category

As the credentials can be overwritten through weak account management functions, this can be considered as A2-Broken Authentication and Session Management vulnerability.

Vulnerability 7

Vulnerability

Cross Site Scripting Vulnerability in "view_all_set.php" page

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	3	4	7	5.25

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	1	7	4.75

Overall: 5

Authentication is not required to exploit this but some conditions should be met to enable exploitation. It can result in disclosure of some sensitive data and modification of the page displayed.

Vulnerability Description

The web app provides some filters which can be applied on the bug list. The apply filter request goes to http://127.0.0.1/mantis/view_all_set.php?f=3 with different filter values as the post parameters. Many of these filter values are vulnerable to cross site scripting attack.
For example, the PoC to exploit the view_type parameter is following:

- Go to the web page http://127.0.0.1/mantis/view_all_bug_page.php
- Run a proxy or plugin such as Tamper Data.
- Click on Apply Filters and check the request going to the server.
- The requested URL is http://127.0.0.1/mantis/view_all_set.php?f=3
- Set the post parameter "view_type" to *simple"><script>alert(document.cookie)</script>*
- Submit the request.
- This will result in loading of http://127.0.0.1/mantis/view_all_bug_page.php but with our script and thus, resulting in cookie display.
- This is a sort of stored XSS as the applied filter values get stored on the server and thus, every time we load this page, we can see the cookies getting displayed.
- If we reset the filter, the stored values will be changed to default and thus, the cookies won't get displayed.

Image tag can also be used by setting the view_type parameter to *simple'>"><img src=x onerror=alert(/Manish.here/)>)*

Other parameters on the same filter page which were tested and found vulnerable are:
relationship_bug, relationship_type, highlight_changed, view_state, reporter_id, handler_id, user_monitor, show_severity, show_priority, show_profile, show_status, show_build,

show_category, show_resolution, hide_status, start_month, start_day, start_year, end_month, end_day, end_year, match_type.

But, the impact of this vulnerability is not that severe. First, the filters are applied on per user basis. So, the filters stored for one user won't work for other users. Therefore, even if the attacker store some malicious data on the server in form of these filter values, he won't be able to affect other users. So, only the victim gets affected by this and not the other users and thus, we can say that it is not a typical stored XSS. Secondly, if the user resets the filters, the malicious data will get removed. Therefore, the impact duration is not much.

The attacker can make the victim to fill a form and can submit the request to view_all_set.php with filter values set to malicious data. In this way, he would be successful in injecting script/HTML. This can also be used to steal the sensitive data such as cookies.
It is also be possible to submit this request as GET and thus, the victim can be forced to click on a link such as (in this case, it can be considered as reflected XSS):

http://127.0.0.1/mantis/view_all_set.php?type=6&view_type="><script>alert(document.cookie)</script>

Source and Mitigation

The basic source of the vulnerability is that the input passed in these parameters is not properly sanitized before returning to the user. The vulnerable code starts from the file "view_all_set.php" where the application takes and processes the input parameters. The developers should properly validate or sanitize these input values.

For example:
The view type is read in this line:

```
$f_view_type    = gpc_get_string( 'view_type', 'simple' );
```

The gpc_get_string() function (in core/gpc_api.php) is called to get the value of view_type. It calls gpc_get() which calls gpc_strip_slashes() to remove the slashes based on magic quotes feature of PHP. But, this feature has been removed starting from PHP version 5.4 and thus, always returns false. So, even though it is not sufficient but some protection that could have worked before is useless with newer versions of PHP. The version used in this instance is 5.4.4.
The developers should validate this input. As the view_type is expected to be either simple or advanced, they can implement a whitelist to check that the input is one of these values otherwise it can be set to some default value.

The other input parameters are also read using gpc_get_string() in the same file. Code hasn't been shared here as there are a large number of related lines. Basically, the code takes all the

input values, does some processing and creates a filter string to store in into the database. Later on, when view_all_bug_page.php is requested, the filter string is read from the database to display the relevant bugs based on the filters applied.
For all the input values read, the developers should either validate using a whitelist if the all possible values are known for target parameter or perform escaping such as HTML, attribute or URL escaping depending upon where in the page these values are going to be displayed to the user. This would prevent the browser from considering these values as script when they are sent to the client from the server.

OWASP Top 10 Category

Yes. A3-XSS

Vulnerability 8

Vulnerability

Cross Site Scripting Vulnerability on "return_dynamic_filters.php"

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	4	7	5.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	1	7	4.75

Overall: 5

The exploitation is easy as the attacker has to just force the user to click on a link. The impact is the loss of sensitive data as the cookies can be stolen and possible modification of the page

rendered due to script/HTML injection. It doesn't have much direct impact on availability but the attacker can later perform some operations to make the application unavailable to the victim.

Vulnerability Description

The "filter_target" GET parameter on the "return_dynamic_filters.php" page is vulnerable to the XSS which can allow an attacker to inject some arbitrary script or html. This can be called a reflected XSS vulnerability. The attacker has to make the victim to request following:

http://127.0.0.1/mantis/return_dynamic_filters.php?filter_target=<script>alert(document.cookie);</script>

This can be achieved by sharing this link (or shortened URL) through e-mail or some blog. Once the request is made, server will think that the victim has requested this and the response will be sent to the victim resulting in the execution of the script in the victim's browser within the security context of the website. This script/HTML injection can be performed to change the page displayed, to run some arbitrary script or to steal cookies.
To get the good understanding of the impact, it should also be analyzed whether the attacker can use this vulnerability to successfully transfer the cookie to himself. The attacker is assumed to have some way to force the victim to request this URL.

Source and Mitigation

The source of the vulnerability is the improper sanitization of the input which the server sends back to the client. The attacker can craft the request in a way such that the script gets executed in the victim's browser on getting the response back. The code responsible for this vulnerability is in file return_dynamic_filters.php:

```
if(isset($_GET['filter_target'])){
  if ( ! headers_sent() ) {
        header( 'Content-Type: text/html; charset=' . lang_get( 'charset' ) );
    }
    $filter = $_GET['filter_target'];
    $t_functionName = 'print_filter_'. substr($filter,0,-7);
    echo "<!-- $filter -->";
    if(function_exists($t_functionName)){
        call_user_func($t_functionName);
    }elseif('custom_field' == substr($filter, 0, 12)){
        # custom function
        $t_custom_id = substr($filter, 13,-7);
        print_filter_custom_field($t_custom_id);
    }else {
        # error - no function to populate the target (e.g., print_filter_foo)
```

```
                    ?>
                    <span style="color:red;weight:bold;">
                            unknown filter (<?php echo $filter; ?>)
                    </span>
                    <?php
            }
} else {
            # error - no filter given
            ?>
            <span style="color:red;weight:bold;">
                    no filter selected
            </span>
            <?php
}
```

This page takes the GET parameter "filter_target", tries to find if there is an existing function corresponding to the input value and outputs *unknown filter (<?php echo $filter; ?>)* if there is no matching function. As the same input gets returned to the client, an attacker can put anything here to cause script/HTML injection.
Developers have different ways to mitigate this. One simple way is not to return the input value for the filter_target parameter. A simple message "filter not found" can be printed. But, this may not be desired in some scenarios. Therefore, we can do encoding (HTML Entity encoding) to encode all the special characters so that it doesn't get considered as the script by the browser.

OWASP Top 10 Category

Yes. A3-XSS

Vulnerability 9

Vulnerability

Cross Site Scripting Vulnerability on view_filters_page.php

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	4	7	5.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	1	7	4.75

Overall: 5

The exploitation is easy as the attacker has to just make the victim (logged in) to click on the shared link. The major impact is on the confidentiality (cookie theft) and the integrity (injection).

Vulnerability Description

The "target_field" GET parameter on the view_filters_page is vulnerable to XSS

http://127.0.0.1/mantis/view_filters_page.php?
target_field="><script>alert(document.cookie);</script>

The "view_type" GET parameter on the view_filters_page is vulnerable to XSS

http://127.0.0.1/mantis/view_filters_page.php?
view_type="><script>alert(document.cookie);</script>

The exploitation technique is same as described in the last vulnerability. The attacker has to make the victim to click on the maliciously crafted link and the response from the server would result in execution of the script in the victim's browser within the security context of the website. This can be used to steal cookies or inject script/HTML.
The attacker is assumed to have some resources to share the link with the victim making him to request the desired URL.

Source and Mitigation

The basic reason is the same as discussed in last vulnerability. The application doesn't perform proper input validation/sanitization and the response contains the input passed as it is and thus, gets executed in the victim's browser. The vulnerability can be fixed by

modifying the handling of the input data in view_filters_page.php. The key code lines responsible are:

```
$t_target_field = gpc_get_string( 'target_field', '' );
..
$f_view_type = gpc_get_string( 'view_type', '' );
..
$f_view_type = gpc_get_string( 'view_type', $f_default_view_type );
..
<input type="hidden" name="view_type" value="<?php PRINT $f_view_type; ?>" />
..
<?php
        $f_switch_view_link = 'view_filters_page.php?target_field=' . $t_target_field .
'&amp;view_type=';
 if ( ( SIMPLE_ONLY != config_get( 'view_filters' ) ) && ( ADVANCED_ONLY !=
config_get( 'view_filters' ) ) ) {
            if ( 'advanced' == $f_view_type ) {
                print_bracket_link( $f_switch_view_link . 'simple',
lang_get( 'simple_filters' ) );
            } else {
                print_bracket_link( $f_switch_view_link . 'advanced',
lang_get( 'advanced_filters' ) );
```

The developers should properly sanitize the inputs received before sending them back in the response. The view_type value can be handled easily using a whitelist as there are only two possible values for this parameter. The target_field can be validated using a whitelist of known possible values (and set accordingly for output) or escaping can be performed using the escaping rules such as HTML entity escaping and URL escaping as discussed on OWASP website.

OWASP Top 10 Category

Yes. A3-XSS

Vulnerability 10

Vulnerability

Directory Traversal/File Include Vulnerability

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	9	7	6	7.75

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
7	5	5	7	6

Overall: 7

The vulnerability is easy to exploit but the attacker needs some authenticated account to compromise the confidentiality by reading the sensitive data stored on the server. There can also be partial impact on the availability and the integrity.

Vulnerability Description

The "account_prefs_page.php" page provides a form which can be used to update the preferences of the user. The language parameter takes the preferred language as the input. The submission sends these parameters to "account_prefs_update.php". Based on the language preference input some specific file gets included. But, as there is no check on the user input, arbitrary files can be included within a PHP require_once() statement. Therefore, it allows reading of arbitrary files on the web server.

Denial of Service can be also launched for a victim by forcing him to set the language parameters to some wrong file as the application displays an error on including wrong/non-existing file and the user can't access the application functionalities after that. The error message also reveals the installation path.

To reproduce this,

- Go to the page http://127.0.0.1/mantis/account_prefs_page.php
- Start some proxy or tamper data plugin
- Click on Update Prefs
- Modify the post parameter Language to spanish/../../../../etc/passwd%00
- Submit the request

The attacker can exploit this vulnerability to read the sensitive data present on the server if he has an account so that he can ask to include the arbitrary files. The post parameter needs to be modified on the fly as the form has a list of values to choose from. Attacker can also force some other user to send a post request (by making him fill and submit some fake form) containing some wrong language preference which would make his account inaccessible. The victim should be logged in to perform the later.

Note: This seems to be a directory traversal vulnerability but I was not able to reproduce it exactly. I was getting error resulting in all the account functionalities to be inaccessible. But, this is also an issue as the application should handle the wrong input and shouldn't reflect the error to the client.

Source and Mitigation

The file gets included by this statement:

require_once($t_lang_dir . 'strings_' . $p_lang . '.txt')

at line 37 in lang_api.php

To mitigate this vulnerability, some sort of validation should be performed using a white list of all the allowed existing languages. The new preferred language file should be loaded only if the input value matches one of the languages in the whitelist. Otherwise, the current loaded file should be kept or it should fall to some default language such as English.

OWASP Top 10 Category

No.

Vulnerability 11

Vulnerability

Cross Site Scripting Vulnerability in the filename of the file uploaded

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	6	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	4	8	6

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	1	4	4

Overall: 5

The attacker needs to be authenticated to upload the files related to some issue. The exploitation is easy and it can lead to compromise of sensitive data such as cookies or injection in to the page rendered. The accountability is possible as the attacker should be logged in and thus, it is possible to trace him.

Vulnerability Description

There exists an XSS vulnerability in the filename of the files uploaded for some issue. This is a stored XSS vulnerability as the filename gets stored and the script would get executed in any user's account who can access that issue's page.
PoC: The attacker can use the code to be injected such as

<img src=x onerror=alert(document.cookie)>

as the name of a file and upload it which would result in stored XSS. The Attacker can exploit this vulnerability to inject some arbitrary script/HTML in the users' browser or steal the cookies (for which he would have to transfer the fetched cookie).
The attacker is supposed to have some account and the permission to upload the files for some issue. It may also be possible to force some victim user to make some file upload but this possibility has not been explored.

Source and Mitigation

The reason is that the filename is not sanitized and is returned as it is to the user when that issue's page gets loaded. Therefore, if there is a script in the filename, the victim's browser would execute it considering it as code. The file add request goes to "bug_file_add.php" which calls the function "gpc_get_file()" in "core/gpc_api.php" to get the filename. Then, it uses the "file_add()" function defined in "file_api.php" to add the file.

Later on, the "view.php" page is requested to get the bug page which includes return of "string_get_bug_view_page()" defined in "string_api.php". The return is either "bug_view_page.php" or "bug_view_advanced_page.php"which prepares the page to be displayed. The function file_list_attachments() defined in "file_api.php" is called to get the list of the files to be displayed on the user's page. If the stored filename is some malicious script, it will be sent to the client from this point and thus, the client browser will execute it resulting in XSS attack.

The best way is to do escaping on the filename when it is read using the gpc_get_file(). All the characters in the filename except the alphanumeric characters should be replaced. Otherwise, escaping should be performed before storing the filename in the database or before delivering the filename as part of the the rendered webpage.

OWASP Top 10 Category

Yes. A3-XSS

Vulnerability 12

Vulnerability

Unaudited Copying of Issues

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	6	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	5	5	7	4.75

Overall: 6

It is easy to find and exploit this vulnerability. The attacker copying the issue can already see the bug and thus, there is not much impact on confidentiality. But, the unaudited copying can impact the integrity and availability as it may result in confusion due to object duplication.

Vulnerability Description

The application doesn't audit when a user copies an issue allowing him to copy issues without being spotted.

Steps to reproduce:

- Login as cs6265-reporter and report a new issue (I got id= 0000015)
- Add a note to it (I got note id = 0000011) and logout from this session
- Login as cs6265-developer
- Go to view issues, select this newly reported issue and choose copy option (can move to a different project also)
- New issue got created (I got id=0000016)
- The note id gets changed to 0000012
- Check the history. There should be no clue of this copy operation and the related changes

This can be considered as security breach as an attacker can copy the issues and issue notes without being detected. The attacker is assumed to have the privileges to copy the issues.

Source and Mitigation

The reason is that the issue history is not updated when a bug gets copied. The problem is in bug_copy() function in "bug_api.php".
To mitigate this, the history should be updated to reflect whenever a issue gets copied showing the original and new bug id. This code can be added in the bug_copy() function once the copy operation has been completed so that any copy operation will have a record.

OWASP Top 10 Category

No.

Vulnerability 13

Vulnerability

Unvalidated Redirects

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	9	6.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	5	1	7	3.75

Overall: 5

Attacker doesn't need any special access and just has to make the authenticated user to click on a link. It compromises the integrity of the website as the user trusts the website but lands into a dangerous domain. The confidentiality of the user may get affected depending upon what the attacker wants him to do.

Vulnerability Description

On login, the request is sent to the server where authentication is done and session cookie is set. After that, request is sent for "http://127.0.0.1/mantis/login_cookie_test.php?return=my_view_page.php". The server performs cookie test and then the user gets redirected to "my_view_page.php". But, the server doesn't perform any check on the return parameter value. An attacker can make a logged in user to click on a URL such as:

http://127.0.0.1/mantis/login_cookie_test.php?return=https://bitcoin.org/bitcoin.pdf
OR
http://127.0.0.1/mantis/login_cookie_test.php?return=http://sagamusix.de/sample_collection/bass.zip

Many users will click on the URL considering it as legitimate one as it seems to be in the trusted domain and will get redirected to a URL of attacker's choice.

The attacker can exploit this vulnerability to make to users download some malware or redirect to some fake webpage where he can ask them to enter some sensitive information. The attacker would have to make the user click on this URL in some way. The attacker is assumed to have skills/resources required to perform this task.

Note: There is a CVE for CRLF but I couldn't reproduce that. I tested for CRLF using this:

http://127.0.0.1/mantis/login_cookie_test.php?return=%0D%0ALocation:%20http://www.google.com

But, I got this message:

SYSTEM WARNING: Header may not contain more than a single header, new line detected

However, unvalidated redirect is still a vulnerability.

Source and Mitigation

The vulnerability exists because the redirection URL is not validated by the server.

The login.php sets the return parameter after authenticating the user (if return is not empty, default_home_page value is used):

```
$f_return = gpc_get_string( 'return', config_get( 'default_home_page' ) );
…
if ( auth_attempt_login( $f_username, $f_password, $f_perm_login ) ) {
        $t_redirect_url = 'login_cookie_test.php?return=' . urlencode( $f_return );
    }
```

After this, the "login_cookie_test.php" tests the cookie and sets the redirect_url to the value of the GET return parameter.

```
<?php
    $f_return = gpc_get_string( 'return', config_get( 'default_home_page' ) );

    if ( auth_is_user_authenticated() ) {
        $t_redirect_url = $f_return;
    } else {
        $t_redirect_url = 'login_page.php?cookie_error=1';
    }

    print_header_redirect( $t_redirect_url );
?>
```

As there is no validation on the value of the return parameter, attacker can make the user to go to any URL of his choice.

The developers can fix this vulnerability by validating the return value. Implementation can be changed so that they don't need to pass this in the URL. Otherwise, it can be checked to make sure that it doesn't take the user out of the trusted domain. Some kind of mapping can also be implemented so that the actual paths are not passed as part of parameters and this, mapping can be used to validate the parameter values.

Owasp Top 10 Category

Yes. A10-Unvalidated Redirects and Forwards

Vulnerability 14

Vulnerability

Missing Function Level Access Control

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	9	9	7.75

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
3	9	4	8	6

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
7	0	0	7	3.5

Overall: 5

The discovery of the vulnerability is difficult as the attacker has to know about the existence of such file. The exploitation is really easy as he has to just request this. The majority of the impact is on the confidentiality as a lot of critical data gets revealed. There is no direct impact on the integrity and availability.

Vulnerability Description

"phpinfo.php" is present in this instance of the application running on the local server.

http://127.0.0.1/mantis/phpinfo.php

This can be executed directly and there is not need to login. It provides a lot of information about the environment such as the version of the components, configuration/installation paths, various settings and other information.

If the attacker can get this information, he can get aware of the environment and thus, can plan the attacks very systematically.

The attacker would have to brute force or guess the presence of this file. He should be aware of the existence of this file to execute it as it is not visible to the normal users because it is not part of any normal communication between clients and server.

Source and Mitigation

The source is the sole existence of this file as there is no need to keep such file and even if there is a requirement, only the administrator should have access to such a file. Only authenticated user and that too if he has the access to this function should be allowed to request it.

OWASP Top 10 Category

This can be considered as A7-Missing Function Level Access Control as unauthenticated and unauthorized user can execute privileged functions/files.

Vulnerability 15

Vulnerability

Cross Site Request Forgery Vulnerability in "account_update.php" Page

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	9	7.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	3	6	8	6

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	3	5	7	4.25

Overall: 5

Discovering the vulnerability is easy but to exploit the attacker has to make some authenticated user to fill a form on some webpage and post the request with hidden parameters. This would impact the integrity as password would get changed and would result in denial of access for the victim. This may later result in data exposure and the overall impact can be more severe depending on the attacker's motives (as he can now log into the account).

Vulnerability Description

CSRF vulnerability exists on the page "account_update.php". The "account_page.php" page provides the functionality to update the password, real name and email address. The update request goes to account_update.php page. This can be exploited by an attacker to modify the victim's password without his consent.

The attacker can design some fake webpage and make the victim to submit a request. The attacker can attract the victim by constructing some interesting page. Victim can be asked to enter some information such as real name and email. The attacker can make the password and password_confirm fields hidden. The form action attribute can be set to "http://127.0.0.1/mantis/account_update.php". A sample html page can be:

```
<html>
<head>
<title>Fun Page</title>
</head>
<body>
<form action = "http://127.0.0.1/mantis/account_update.php" method="POST">
<!--<input type="text" size="25" name="realname" value="Enter your name here!" /> -->
```

```
<input type="hidden" name="email" size="32" maxlength="64" value="cs6265@localhost" />
<input type="hidden" name="password" value="newpassw0rd" />
<input type="hidden" name="password_confirm" value="newpassw0rd" /> -->
<div class="button">
    <button type="submit">Start Fun</button>
</div>
</form>
</body>
```

In this way, the attacker can make the victim to submit a password change request to the server. The attacker would know the new password and the victim can be locked out of the his account. The attacker is assumed to have an understanding of this process. To exploit this vulnerability, the victim should be authenticated. Attacker should also be aware of the parameters which should be submitted in this POST request.

Source and Mitigation

A simple reason for this vulnerability is the insecure way of password update. The application doesn't ask the user to enter his current password to change the password. So, the attacker doesn't need to know the current password. To mitigate this, the developers should add another field in the form asking for the current password. This password should be verified before accepting the password change.
The other solution is to provide protection against CSRF attacks. A per-session or per-request CSRF token can be used to verify that the user is the one actually requesting some resource/ update. The server can provide a random CSRF token which would be submitted as a hidden field in a POST request for all the sensitive operations. The server can authenticate the request by checking this token. Other alternatives such as double submit cookies and referrer header check can also be implemented.

Owasp Top 10 Category

Yes. A8-CSRF

Vulnerability 16

Vulnerability

CSRF Vulnerability in "account_prof_edit_page.php" page

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	1	7	9	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	3	4	8	5.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
1	3	3	7	3.50

Overall: 4

The attacker has to make the authenticated victim to submit a form so that the hidden request can be sent. The integrity gets impacted as the attacker is able to delete some profile. It can have some impact on availability related to the that profile.

Vulnerability Description

The "account_prof_menu_page.php" page provides functionality to manage profiles. The attacker can make the victim to submit request to delete some profile. A sample form can be:

```
<html>
<head>
<title>Fun Page</title>
</head>
<body>
<form action = "http://127.0.0.1/mantis/account_prof_edit_page.php" method="POST">
<input type="hidden" name="action" value="delete" />
<input type="hidden" name="profile_id" value="3" />
<div class="button">
      <button type="submit">Start Fun</button>
 </div>
</form>
</body>
</html>
```

On submitting the request on this page, the request to delete the profile with id=3 will be made. In this way, the attacker can affect the integrity of the system and can also disrupt some service.

The victim should be authenticated for the success of this attack. The attacker should have the knowledge of the fields and would have to guess the profile ids. Otherwise, the attacker can get some of this knowledge if he has an account in the system.

Source and Mitigation

The reason is that the web application allows the attacker to predict all the details of this action and doesn't implement any technique to distinguish between forged and legitimate requests.
This can be mitigated using the same technique as explained in the last vulnerability. CSRF token functionality can be implemented. The server can provide a random CSRF token to the client in each session. Client has to provide this token to the server through a hidden field for each sensitive operation. The server can use this token to verify that the request is legitimate one.

Owasp Top 10 Category

Yes. A8-CSRF

Vulnerability 17

Vulnerability

Using Components with Known Vulnerabilities

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	9	7	9	8.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
9	9	6	8	8

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	5	5	7	5.75

Overall: 7

Using old components don't have any direct impact but they allow exploitation of known vulnerabilities and thus, the impact should be judged based on the exploitation of the vulnerabilities. In general, the confidentiality, integrity and availability of the system gets compromised.

Vulnerability Description

This instance of the application uses some old vulnerable components such as:

- PHP
  Used version: PHP 5.4.4-14
  Latest available version: 5.6.2
  Some known vulnerabilities in used version: CVE-2012-2688, CVE-2013-1635, CVE-2011-4718, CVE-2013-2110

- Apache:
  Latest available version: 2.4.10
  Used version: 2.2.22
  Some known vulnerabilities in used version: CVE-2014-0098, CVE-2013-5704, CVE-2013-1862, CVE-2012-3499

The attacker can do some information gathering to know the used components and can exploit the known vulnerabilities in these old components. Automated tools are available to achieve this.

Source and Mitigation

The reason is that the components used are not updated/patched. To mitigate this, the administrators of the application should update the components to the latest available versions.

OWASP Top 10 Category

Yes. A9-Using Components with Known Vulnerabilities

Vulnerability 18

Vulnerability

Insecure Crypto/Hashes

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	9	4	9	7.75

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
3	3	4	5	3.75

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
7	5	5	7	6

Overall: 6

Discovery and exploitation of the vulnerability is not that easy as the attacker needs some other attack to get the hashes so that passwords can be retrieved. Immediate impact is the loss of confidentiality due to sensitive data disclosure. But, impact can be more severe depending upon attacker's motive and thus, the integrity and the availability can be impacted by using the passwords to enter into the system, to modify the data or to cause denial of service.

Vulnerability Description

The application uses MD5 to hash the passwords which get stored on the server. MD5 is not a secure hashing algorithm any more. In case of database compromise, the attacker can get the actual passwords from the MD5 hashes.
For example:

"0d107d09f5bbe40cade3de5c71e9e9b7" is the MD5 hash of "letmein"

Online tools are available which have huge database of MD5 hashes. These can be used to get the password from the hash. The attacker is supposed to have some way to get the hashes to exploit this vulnerability.

Source and Mitigation

The reason is the use of insecure/broken hashes. The MD5 hashes shouldn't be used. To mitigate this, the developers can use some strong password hash algorithm such as bcrypt to hash the passwords before storing in the database. It will also provide protection against brute force attacks.

Owasp Top 10 Category

As old/weak cryptography is used, it will fall in the category A6-Sensitive Data Disclosure.

Vulnerability 19

Vulnerability

SQL Injection in "manage_user_page.php"

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	6	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
7	5	7	7	6.5

Overall: 7

Exploitation needs some privileges to request the vulnerable page. This can compromise the confidentiality of multiple users allowing an attacker to steal a lot of sensitive information. That can be used to compromise the integrity and availability of the system affecting multiple users.

Vulnerability Description

SQL injection vulnerability exists in "manage_user_page.php" page which can be exploited by an attacker to execute SQL injection attack on the database resulting in exposure of sensitive data such as password hashes, cookies etc.

PoC:

Requesting http://127.0.0.1/mantis/manage_user_page.php?sort=password' results in:

> *http://127.0.0.1/mantis/manage_user_page.php?sort=password'*
>
> *APPLICATION ERROR #401*
>
> *Database query failed. Error received from database was #1064: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\' ASC' at line 4 for the query: SELECT * FROM mantis_user_table WHERE (1 = 1) ORDER BY password\' ASC*
>
> *Please use the "Back" button in your web browser to return to the previous page. There you can correct whatever problems were identified in this error or select another action. You can also click an option from the menu bar to go directly to a new section.*

This shows that the code on the server doesn't validate the input parameter and passes it exactly to the SQL interpreter and error is received on executing this query.

Similar problem exists in handling the input parameter 'prefix'.
Requesting http://127.0.0.1/mantis/manage_user_page.php?prefix=admin' results in:

> *APPLICATION ERROR #401*
>
> *Database query failed. Error received from database was #1064: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '%')*
> *ORDER BY username ASC' at line 3 for the query: SELECT **

FROM mantis_user_table
WHERE (username like 'ADMIN'%')
ORDER BY username ASC

Please use the "Back" button in your web browser to return to the previous page. There you can correct whatever problems were identified in this error or select another action. You can also click an option from the menu bar to go directly to a new section.

Here also, the input parameter prefix is not validated and passed as it is in the SQL query.

An attacker can exploit this vulnerability to access the sensitive data from the database. An example of SQL injection attack that can be performed exploiting this vulnerability is:

Lets assume that the attacker want to know some user's password hash stored in the database. He can extract the the password character by character by checking each character in the password against all possible ASCII characters. The page displayed will change depending upon whether it is a match or not. If it is a match, attacker knows one more character and thus can move forward to the next character in the password. A comparison with NULL should also be made each time to check if the end of the string has been reached. Some sample queries (only successful ones to keep it short) are:

This query tells that first character is 0 (bugs displayed sorted by last visit on a match):
http://127.0.0.1/mantis/manage_user_page.php?sort=%28CASE%20WHEN%20%28SELECT%20ASCII%28SUBSTRING%28password,%201,%201%29%29%20FROM%20mantis_user_table%20where%20username%20=%200x61646d696e6973747261746f72%29%20=%2048%20THEN%20last_visit%20ELSE%20username%20END%29

This query tells that second character is d:
http://127.0.0.1/mantis/manage_user_page.php?sort=%28CASE%20WHEN%20%28SELECT%20ASCII%28SUBSTRING%28password,%202,%201%29%29%20FROM%20mantis_user_table%20where%20username%20=%200x61646d696e6973747261746f72%29%20=%20100%20THEN%20last_visit%20ELSE%20username%20END%29

This query tells that third character is 1:
http://127.0.0.1/mantis/manage_user_page.php?sort=%28CASE%20WHEN%20%28SELECT%20ASCII%28SUBSTRING%28password,%203,%201%29%29%20FROM%20mantis_user_table%20where%20username%20=%200x61646d696e6973747261746f72%29%20=%2049%20THEN%20last_visit%20ELSE%20username%20END%29

This query tells that fourth character is 0:
http://127.0.0.1/mantis/manage_user_page.php?sort=%28CASE%20WHEN%20%28SELECT%20ASCII%28SUBSTRING%28password,%204,%201%29%29%20FROM%20mantis_user_table%20where%20username%20=%200x61646d696e6973747261746f72%29%20=%2048%20THEN%20last_visit%20ELSE%20username%20END%29

This query tells that fourth character is 7:

http://127.0.0.1/mantis/manage_user_page.php?sort=%28CASE%20WHEN%20%28SELECT%20ASCII%28SUBSTRING%28password,%205,%201%29%29%20FROM%20mantis_user_table%20where%20username%20=%200x61646d696e6973747261746f72%29%20=%2055%20THEN%20last_visit%20ELSE%20username%20END%29

…

Similarly, all the characters can be retrieved by checking against all possible characters until a NULL is matched.

Now, the password hash stored in the database for the user administrator is:

0d107d09f5bbe40cade3de5c71e9e9b7

We can see that the characters retrieved are correct.

The attacker is assumed to have the rights to make such requests or should have some technique to force a legitimate user to make these requests. The attacker should have knowledge of SQL and some idea about the position where these vulnerable parameters are going to fit in the SQL query.

Source and Mitigation

The vulnerability exists in "manage_user_page.php" page and the input validation/sanitization has not been performed on the parameters before passing them in the SQL query. The relevant code is:

```
$f_sort     = gpc_get_string( 'sort', 'username' );
..
$f_prefix = strtoupper( gpc_get_string( 'prefix', config_get( 'default_manage_user_prefix' ) ) );
...
$t_where = "(username like '$f_prefix%')";
..
# Get the user data in $c_sort order
   if ( 0 == $c_hide ) {
            $query = "SELECT *
                              FROM $t_user_table
                              WHERE $t_where
                              ORDER BY $c_sort $c_dir";
   } else {
            $query = "SELECT *
                              FROM $t_user_table
                              WHERE (" . db_helper_compare_days(db_now(),"last_visit","<
'$days_old"') . ") AND $t_where
                              ORDER BY $c_sort $c_dir";
   }
   $result = db_query($query);
   $user_count = db_num_rows( $result );
```

To mitigate this vulnerability, the developers can use parameterized queries in which they can define the query first and then pass the parameters later. Otherwise, escaping can be performed to escape all the special characters in the input so that it is not treated as code by the SQL interpreter. ESAPI can also be used to achieve this. Whitelisting can be for sort parameter as all the possible values should be known.

OWASP Top 10 Category

Yes. A1-Injection

Vulnerability 20

Vulnerability

Remote Code Execution via sort parameter in "manage_proj_page.php" page

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	9	7	8	8.25

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
9	9	9	7	8.5

Overall: 9

Exploitation of this vulnerability is easy as the code has to passed in the parameter. The impact is very severe and can compromise the confidentiality, integrity and availability of all the users as the attack can execute code on the server to read sensitive information and to modify or delete files. The average comes out to be around 8 but the overall rating has been set to 9 as this vulnerability sounds critical.

Vulnerability Description

Any arbitrary code can be passed as the value of the sort parameter to "manage_proj_page.php" which gets executed on the server.

The exploit provided at http://www.exploit-db.com/exploits/6768/ was used to test this vulnerability. The important part is:

```
$code    = "']);}error_reporting(0);print(_code_);passthru(base64_decode(\
$_SERVER[HTTP_CMD]));die;%%23";
$packet  = "GET {$path}manage_proj_page.php?sort={$code} HTTP/1.0\r\n";
```

Code variable is passed as the value of the sort parameter which gets executed as the code. The attacker is assumed to have access to this page for which he should be authenticated.

Source and Mitigation

The related source code is:

In manage_proj_page.php:

```
$f_sort       = gpc_get_string( 'sort', 'name' );
..
$t_projects = multi_sort( $t_full_projects, $f_sort, $t_direction );\
..
```

In core/utility_api.php:

```
# Sort a multi-dimensional array by one of its keys
 function multi_sort( $p_array, $p_key, $p_direction=ASC ) {
   ...
            $t_function = create_function( '$a, $b', "return $t_factor * strnatcasecmp( \
$a['$p_key'], \$b['$p_key'] );" );
            uasort( $p_array, $t_function );
            return $p_array;
   ...
 }
```

"manage_proj_page.php" takes "sort" as a parameter which is passed to the "multi_sort()" function defined in "core/utility_api.php". The "multi_sort()" function passes this "sort" value to "create_function()" which is a PHP function to create anonymous function. As no validation/sanitization has been performed, any arbitrary code can be passed via sort to get it executed on the server.

To mitigate this, whitelisting can be implemented by the developers. As all the possible values which can be passed in sort parameter should be known, the input can be checked against this list. If found, the input value should be passed to create function otherwise some default value should be passed.

OWASP Top 10 Category

Yes. As the code is getting injected, it can be considered as A1-Injection.

Vulnerability 21

Vulnerability

Application Platform Configuration Problem - allows any file type upload

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	8	7

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
2	3	1	7	3.25

Overall: 4

The discovery and exploitation is really easy as the attacker just need to try to upload malicious files. The direct impact is not that severe as it depends what type of files are getting shared by the attacker.

Vulnerability Description

The application doesn't handle file uploads properly. It allows the users to upload files of any type. An attacker can upload some kind of executable/script which can get executed on the other user's machine when the user downloads it (and clicks open/allow).

The attacker should have rights to upload a malicious file or should be able to force some other user to do so. It will also depend how careful the user downloading such files is.

Source and Mitigation

The vulnerability exists as the application doesn't check the type of the file and allows any arbitrary files to be uploaded. The user should be restricted to upload only certain types of file. If executable or potentially malicious files have to be uploaded, they should be allowed only in archived form. Therefore, a whitelist can be maintained for the type of files allowed.

OWASP Top 10 Category

Yes. A5-Security Misconfiguration

Vulnerability 22

Vulnerability

Missing Function Level Access Control

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	6	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	1	1	7	3.75

Overall: 5

This bug is easily to discover and exploit as it is directly visible to the logged in users. It affects the confidentiality as the unauthorized user is able to see the private bugs.

Vulnerability Description

view_all_bug_page.php allows the user to view information related to private bugs for which he is not authorized.
Suppose "cs6265-developer" reports a new bug and marks it as private. The user "cs6265-reporter" should not be able to view or know any information about this page.

But, if the user "cs6265-reporter" goes to http://127.0.0.1/mantis/view_all_bug_page.php and just click on apply filter, all the bugs including the private ones (which shouldn't be visible to this user) get displayed to the user.

The authenticated but unauthorized user can exploit this vulnerability to get the information about private bugs and keep track of their status.

Source and Mitigation

The bug is in the request to "view_all_set.php" where the passed filter values are used to decide what content should be displayed to the user. It doesn't make a check to figure out which user is requesting this page and what access rights he has.

The developers should implement a check on the user requesting the page and should display only those bugs after filtering for which he has the access rights.

OWASP Top 10 Category

Yes. A7-Missing Function Level Access Control

Vulnerability 23

Vulnerability

Insecure Direct Object Reference Vulnerability

Severity Rating

Threat Agent:

Skill Level	Motive	Opportunity	Size	Overall
9	4	7	6	6.5

Vulnerability:

Ease of Disovery	Ease of Exploit	Awareness	Intrusion Detection	Overall
7	5	6	8	6.5

Impact(Losses):

Confidentiality	Integrity	Availability	Accountability	Overall
6	1	1	7	3.75

Overall: 5

Any authenticated user can comment on some bug and thus, can exploit this vulnerability by tagging the target private bugs. It compromises the confidentiality as some information is visible to unauthorized user.

Vulnerability Description

Any authenticated user with rights to comment on any bug can directly access some information related to the bugs for which he is not authorized.

PoC:

- Login as "cs6265-developer" and report a bug marking it as private.
- Login as "cs6265-reporter" and view any bug to which you have access.
- As the bugs are assigned IDs in a sequence, you can predict the possible private bug IDs. For example, if you can view bugs 0000001, 0000002 and 0000004, then 0000003 can be a bug ID assigned to a private bug for which you don't have access.
- So, add a note to the bug you opened referencing the possible bug ID(#0000003).
- Hover over the bug ID shown in the note added. You should be able to see the status and the summary of the private bug.

In this way, an attacker can exploit this vulnerability to directly access the information related to the private bugs.
The attacker is assumed to be authenticated as a user and has some access so that he can add and view notes to some bug.

Source and Mitigation

The problem is in "bugnote_add.php" to which the bug content gets submitted. It calls gpc_get_string() defined in gpc_api.php and then, calls bugnote_add() function defined in bugnote_api.php to perform bug addition.
A check should be made to see if the user is trying to reference a bug for which he doesn't have access. As the user shouldn't have any information about the private bug, he shouldn't be able to reference this bug. This can be performed before inserting the bug note in the database.

If the above above is not preferred, an alternative is not to display the information on referencing the bug. User should be able to just click on the link so that the access can be checked. bugnote_view_inc.php calls string_display_links() defined in core/string_api.php which in turn calls string_process_bug_link() which in turn calls string_get_bug_view_link() which adds status and summary as the title attribute of anchor tag.

```
$t_link .=  ' title="[' . $t_status . '] ' . $t_summary . '"';
```

Here, this title can be changed so that this information is not visible to the user.

OWASP Top 10 Category

Yes. A4-Insecure Direct Object References as the objects (bug information) can be access directly without authorization as the access checks are missing.