Codoforum 4.8.7: Critical Code Vulnerabilities Explained

BY DENNIS BRINKROLF | AUGUST 26, 2020

Security

In the SonarSource R&D team we are equally driven by studying and understanding real-world vulnerabilities, then by helping the open-source community secure their projects. This recently led us to uncover and report multiple security vulnerabilities in Codoforum, an open source forum software developed in PHP. The vulnerabilities enable different attack vectors for a complete take over of any Codoforum board with version <4.9 and are rated as critical. No prior knowledge or privileges are required by a remote attacker. We reported all issues responsibly to the affected vendor who released a security patch immediately.

In this blog post we analyze the technical root cause of three vulnerabilities, what security measures were found and bypassed, and how to correctly prevent these in your code. We will look at the vulnerabilities from an attacker's perspective and demonstrate how various exploitation techniques are used in an attack to sharpen your defender's mindset.

SQL Injection (CVE-2020-13873)

We found two SQL Injection vulnerabilities and one of these can be exploited as an unauthenticated forum user to extract data from the database. These allow an attacker to fully compromise an administrator account by retrieving a password reset token. Once an administrator account is accessed, the attacker can gain Remote Code Execution on the targeted web server and compromise the system's host and data.

For demonstration purposes we've created a short video that shows the most critical SQL injection vulnerability and its impact.

Technical Analysis

The vulnerability hides within the API call for fetching forum posts. Its code is defined in the *routes.php* file. Here, a dispatch function maps a route to a function that processes certain URL parameters. As shown in line 165, a topic ID `$tid (:tid)` is processed from the route that can be modified by a malicious user.

routes.php

```
165 function dispatch_get('Ajax/topic/:tid/:from/get_posts', function ($tid, $from) {
⋮
168     $topic = new \CODOF\Forum\Topic(\DB::getPDO());
169     $topic_info = $topic->get_topic_info($tid);
179 }
```

This route has neither CSRF protection nor a permission check and can be accessed from any visitor without authentication. The user input $tid is then passed to the get_topic_info() function without any sanitization.

In the `get_topic_info()` function, the user controlled variable `$tid` is concatenated directly into a SQL query in line 462 which is executed in line 464. This is a textbook SQL injection that allows an attacker to malform the SQL query in order to access other SQL tables and columns than intended. Erroneously, the developer assumed that the parameter `$tid` is an integer before it is included into the query as we can see from the comment in line 461.

sys/CODOF/Forum/Topic.php

```
458 public function get_topic_info($tid) {
⋮
461     // $tid is converted to integer so its safe
462     $qry = "SELECT t.redirect_to,t.topic_id,t.post_id, t.no_posts, t.no_views,t.uid," . "t.title, c.cat_name,t.post_id, c.cat_alias, c.cat_id," .
```

SonarSource SA's websites use cookies to distinguish you from other users of our websites. This helps us to provide you with a good experience when you browse our websites and also allows us to improve them.

via an uncaught `PDOException` because error reporting is enabled by default in Codoforum. This requires less HTTP requests and the data can be extracted in chunks.

The MySQL function `extractvalue()` can be abused during a SQL injection attack for this purpose. It constructs an XPath query and checks for correct syntax. When we define a faulty XPath that includes information that we want to read, e.g. the MySQL version number, then this is leaked as part of the error message.

```
EXTRACTVALUE(RAND(),CONCAT(0x3a,(SELECT VERSION() LIMIT 0,1))
```

(!) Fatal error: Uncaught PDOException: SQLSTATE[HY000]: General error: 1105 XPATH syntax error: ':5.7.30-0ubuntu0.18.04.1' in /var/www/html/apps/codoforum/sys/CODOF/Forum/Topic.php on line *464*

(!) PDOException: SQLSTATE[HY000]: General error: 1105 XPATH syntax error: ':5.7.30-0ubuntu0.18.04.1' in /var/www/html/apps/codoforum/sys/CODOF/Forum/Topic.php on line *464*

Call Stack

#	Time	Memory	Function	Location
1	0.0002	359168	{main}( )	.../index.php:0
2	0.0055	764056	require('/var/www/html/apps/codoforum/routes.php')	.../index.php:22
3	0.0095	1105184	CODOF\Access\Request::start( )	.../routes.php:971
4	0.0095	1105184	run(???)	.../Request.php:94
5	0.0097	1111240	{closure:/var/www/html/apps/codoforum/routes.php:165-177}(string(72), string(2))	.../limonade.php:429
6	0.0099	1117632	CODOF\Forum\Topic->get_topic_info(string(72))	.../routes.php:169
7	0.0099	1118080	query (string(377))	.../Topic.php:464

With the help of this error-based technique, an attacker can extract data from the database quickly and efficiently. For example, the attacker could extract all passwords from the users table. This is very inefficient though because Codoforum stores only the hashes of all passwords using the bcrypt algorithm. The attacker would need to make the effort of cracking these hashes in order to login.

There is a more clever way. By requesting a password reset for a user, for example the forum's administrator, a password reset token is generated and stored in the database. Although the attacker does not have access to the admin's email to receive this token, he can now abuse the SQL injection to extract that token directly from the database. As a result, the attacker can reset the admin's password with that token and then login as administrator. From here, the attacker can abuse administrator features to compromise the server as we will see in the last section of this post.

Patch

By using prepared statements or an integer typecast it is prevented that an attacker can inject arbitrary SQL syntax and mix user input with the SQL query. This way, the attacker cannot modify the SQL query to its advantage anymore.

sys/CODOF/Forum/Topic.php

```
458  public function get_topic_info($tid) {
461      $tid = (int)$tid;
462      $qry = "SELECT t.redirect_to,t.topic_id,t.post_id, t.no_posts, t.no_views,t.uid," . "t.title, c.cat_name,t.post_id, c.cat_alias, c.cat_id,"
463
464      $res = $this->db->query($qry);
```

But there were alternative ways for an attacker to compromise the Codoforum board software.

Path Traversal (CVE-2020-13874)

The second vulnerability type found was a Path Traversal that allows an unauthenticated attacker to download arbitrary files from the server, such as sensitive configuration files. Although the developers tried to prevent this vulnerability with input sanitization, the filter could be bypassed. Let's have a look at the details.

Technical Details

The vulnerability resides in the file attachment feature of the forum. Via the route *serve/attachment* the `attachment()` function is called as shown in the code below. Here, in line 64, it calls the constructor `Serve()`. Note that there are several routes that lead to the vulnerable `Serve()` function and this route can be used as an unauthenticated user.

routes.php

```
63  dispatch_get('serve/attachment', function () {
```

SonarSource SA's websites use cookies to distinguish you from other users of our websites. This helps us to provide you with a good experience when you browse our websites and also allows us to improve them.

We now inspect what happens in this `serve()` function. In line 37, a user controlled input `$_GET['path']` is retrieved and sanitized. It is concatenated with other strings and then used as a file path in line 42 to open a file that is offered for download. The whole security is based on the `sanitize()` function in line 37 which is supposed to prevent a path traversal attack.

sys/Controller/Serve.php

```
35  private function serve($path) {
36
37      $name = $this->sanitize($_GET['path']);
38      $dir = DATA_PATH . $path;
39
40      $path = $this->setBasicheaders($name, $dir);
41      header('Content-Disposition: attachment; filename="' . $this->getRealFileName($name) . '"');
42      @readfile($path);
43      exit;
44  }
```

The following code listing shows the sanitization approach. In line 123, the characters `..` are removed and then the url encoded representation `%2e%2e` is also removed in the next line.

sys/Controller/Serve.php

```
121  private function sanitize($name) {
122
123      $name = str_replace("..", "", $name);
124      $name = str_replace("%2e%2e", "", $name);
125
126      return $name;
127  }
```

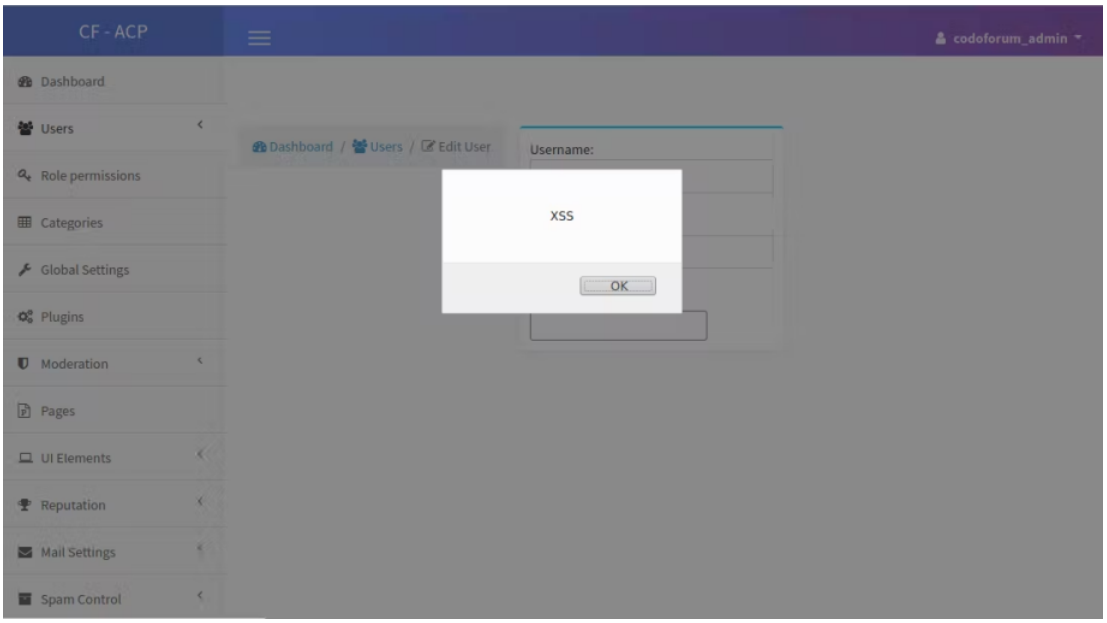
The problem is that the PHP function `str_replace()` does **not** replace the string recursively and is only processed once from left to right. This means that if the variable `$name` contains something like `/.%2e%2e./` it will be replaced back to `/../` and the sanitization is bypassed.

Thus a path traversal attack is possible and an unauthenticated attacker can read arbitrary files from the server by traversing in the file system and accessing sensitive files (`…/…/…/other/path/file`). This can lead to a full takeover of certain servers hosted with Codoforum.

The faulty sanitization can be fixed by first using `urldecode()` and then using `str_replace("..")` or by removing the second replacement altogether.

Persistent Cross-Site Scripting (CVE-2020-13876)

Last but not least, we uncovered a Persistent XSS vulnerability in Codoforum. It enables a low privileged, malicious user to inject a JavaScript payload into the admin backend. When an admin then visits an infected user profile, the XSS payload is executed and the attacker can perform any action as authenticated admin, including the execution of arbitrary code on the targeted web server.



an e-mail address which is then reflected in the admin backend. In the following code

SonarSource SA's websites use cookies to distinguish you from other users of our websites. This helps us to provide you with a good experience when you browse our websites and also allows us to improve them.

```
676
677     if (Request::valid($_POST['token'])) {
678        $user = new \Controller\user();
679        $user->register(true);
680
681        CODOF\Smarty\Layout::load($user->view, $user->css_files, $user->js_files);
682     }
683  });
```

The following code shows the simplified `register()` function. Here, in line 3, the user input `$_REQUEST['mail']` is retrieved and checked with other information in line 4. If there is no error, such as an invalid or already existing email address, the user will be registered.

sys/Controller/user.php

```
1  public function register($do) {
⋮
3     $register->mail = $_REQUEST['mail'];
4     $errors = $register->get_errors();
5
6     if (empty($errors)) {
7        //register user
8     }
⋮
```

For this purpose, the user controlled `$mail` variable is checked with the PHP built-in function `filter_var()` and its filter option `FILTER_VADLIDATE_EMAIL` in line 108. If `$mail` is a valid email and does not exist, the registration will work without problems (see above).

According to the PHP documentation, the `FILTER_VADLIDATE_EMAIL` generally validates the email address against the syntax defined in *RFC 822*. Hence, malicious HTML characters that can be used to construct a JavaScript payload can be used within an email address. Something like `"><script>alert(1)</script>x@foo.com` is valid and will not be rejected by the filter.

sys/CODOF/Constraints/User.php

```
105  public function mail($mail) {
106
107     $errors = array();
108     if (!filter_var($mail, FILTER_VALIDATE_EMAIL)) {
109        $errors[] = _t("email address not formatted correctly");
110     }
111
112    if (\CODOF\User\User::mailExists($mail)) {
113       $errors[] = _t("email address is already registered");
114    }
115
116     $this->errors = array_merge($errors, $this->errors);
⋮
```

admin/layout/templates/users/edit.tpl

```
50  Email:<br>
51  <input type="text" name="email"  value="{$user.mail}" class="form-control" placeholder=""      required />
52  <br/>
```

Since Codoforum uses the PHP template engine Smarty, the escape modifier of Smarty can be used as a patch by replacing line 51 with the following content:

```
51  <input type="text" name="email"  value="{$user.mail|escape:'html'}" class="form-control" placeholder="" required />
```

If an admin visits the user profile to edit (block/delete) our registered user, the XSS payload is rendered in the admin's web browser and we can perform any action as admin on the page. For example, administrator features can be abused to upload a PHP shell and to execute arbitrary code on the server. A similar XSS issue was found earlier that affected the user name.

As a result, an attacker can smuggle an XSS payload within the email address of a new user which is reflected unfiltered in the HTML response of the admin backend.

Summary

In this blog post we analyzed three different security vulnerabilities in Codoforum, a popular board software. Each of these issues could lead to a complete takeover of the application. We've learned that a malicious user can take multiple paths when attacking an application and that finding only one single vulnerability is enough to fully compromise its security. Hence it is our task to make our applications as robust and secure as possible. Checking all user inputs properly and leveraging existing and proven sanitization and validation mechanisms is the first step towards a solid defense.

Since the release of a fix in March. If you are hosting a Codoforum and didn't update your system, a patch version was quickly released after our reports.

SonarSource SA's websites use cookies to distinguish you from other users of our websites. This helps us to provide you with a good experience when you browse our websites and also allows us to improve them.

DENNIS BRINKROLF
Security Researcher

In-IDE

IDE extension that lets you fix coding issues before they exist!

Discover SonarLint →

In-Cloud

Setup is effortless and analysis is automatic for most languages

Discover SonarCloud →

On-premise

Fast, accurate Code Quality and Code Security analysis for most languages

Discover SonarQube →

Sonar blog delivered directly to your inbox!
We respect your privacy.

Email Subscribe Now

© 2008-2022, SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE, and SONARCLOUD are trademarks of SonarSource SA.
All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
Privacy Policy | Terms and Conditions

SonarSource SA's websites use cookies to distinguish you from other users of our websites. This helps us to provide you with a good experience when you browse our websites and also allows us to improve them.