Search ... SEARCH

OpenSIS Vulnerabilities

OpenSIS is an open source student information system. Recently, it was affected by several vulnerabilities such as SQL injections, local file inclusions and incorrect access controls (CVE-2020-13380, CVE-2020-13381, CVE-2020-13382, CVE-2020-13383). That is why I got interested and also had a quick look at the application.

As part of this investigation, I discovered two vulnerabilities, an XSS vulnerability (CVE-2020-27409) in the file SideForStudent.php that got quickly fixed after being reported (see commit edca085 for the details; the commit is included in release v7.5) and some incorrect (i.e. non-existent) access controls for the password change functionality (CVE-2020-27408). In this blog post, I would like to focus on the second vulnerability and describe the tedious disclosure process that – in the end – lead to nothing but the implementation of some ineffective obfuscation mechanism.

First, let us start with an explanation of the vulnerability. The vulnerability resides in the file ResetUserInfo.php. Within this file, the password change functionality is implemented.

```
if ($_REQUEST['new_pass'] != '' && $_REQUEST['ver_pass'] != '') {
    $get_vals = explode(",", $_REQUEST['user_info']);
    $flag = 'submited_value';

    $get_info = DBGet(DBQuery('SELECT COUNT(*) AS EX_REC FROM login_authentication WHERE user_id!=' . $get_vals[0] . ' AND profile_id!=' . $get_vals[1] . ' AND password=\'' . md5($_REQUEST['ver_pass'])
    if ($get_info[1]['EX_REC'] > 0) {
        $_SESSION['err_msg_mod'] = '<font color="red" ><b>Incorrect login credential.</b></font>';
    } else {
        DBQuery('UPDATE login_authentication SET password=\'' . md5($_REQUEST['ver_pass']) . '\' WHERE user_id=' . $get_vals[0] . ' AND profile_id=' . $get_vals[1] . ' ');
        $_SESSION['conf_msg'] = '<font color="red" ><b>Password updated successfully.</b></font>';
        echo'<script>window.location.href="index.php"</script>';
    }
}
```

As the above code snippet shows, the new password for the user should be contained within the new_pass parameter. However, this parameter is actually not really used and it is only checked if it is not empty. The more interesting parameter is ver_pass as it is later used to change the password of a user within the UPDATE query.

To select the user, whose password should be updated, the user_info parameter is required. This parameter consists of a list of two comma-separated values, where the first value corresponds to the user_id and the second value corresponds to the profile_id as can be seen form the UPDATE query. Together, these two values identify a user.

Before the part with the UPDATE command can be reached, the condition $get_info[1]['EX_REC'] > 0 has to evaluate to false. However, this condition only evaluates to true if some other user has the same password as specified via the ver_pass parameter. As we can choose this parameter arbitrarily, we can always set a password that is not used by any other user and therefore let the condition evaluate to false.

It should be noted that the code path above is not protected by any access controls. Therefore, an unauthenticated user can use the functionality to change the password of any user if he knows the user_id and profile_id values. However, these values are just integers and usually in a low range (at least for my test system). Hence, an attacker can just brute force existing values. It should also be noted that the described vulnerability is not the only thing that may be wrong with the above code as, for example, the supplied values a directly inserted into the SQL statement without using a safe mechanism to construct the query (e.g. prepared statements / parameterized queries).

To report the identified vulnerability, I opened an issue on July 8, 2020 on the respective OpenSIS GitHub page and asked the maintainers to contact me. They contacted me on the same day and asked for the details, which I send to them one day later. On August 5, they responded that they implemented a fix for the issue and asked me to validate it.

Reviewing the issue, I found that they included a new file called AuthCryp.php within the ResetUserInfo.php. Looking at the AuthCryp.php file, we find the following code snippet in the beginning of the file:

```
if(isset($_REQUEST['calling']))
{
        if(function_exists($_REQUEST['calling']))
        {
                [...]

                $encoded_string = $_REQUEST['calling']($_REQUEST['telling'], $_REQUEST['motive'], $_REQUEST['jsc']);
        }
}
```

This code snippet allows calling arbitrary PHP functions via the calling parameter and three arguments can be passed via the telling, motive, and jsc parameters. To extend this to arbitrary code execution, we can just use the exec function of PHP by using some optional parameters to increase the number of arguments to exactly three. So yes, they actually made the vulnerability worse with the patch as they introduced a remote code execution vulnerability… I even do not know what the code snippet was meant to be used for and I guess the maintainers also did not know as they just removed the code after I told them about it (see this commit).

Coming back to the initial vulnerability, the question remains whether it is fixed with the update. The first thing that we can note is that they do not use the plain user_id and profile_id values now, but put some 'cryptor' around them:

```
$get_vals[0] = cryptor($get_vals[0], 'DEC', '');
$get_vals[1] = cryptor($get_vals[1], 'DEC', '');
```

Here $get_vals[0] corresponds to the user_id and $get_vals[1] corresponds to the profile_id. Looking at the implementation of the cryptor, we see that it does some voodoo that consists of remapping values, i.e. the value 6 is remapped to value 0, the value 4 is remapped to value 1, the value 7 is remapped to value 2 and other shenanigans.

```
[...]
                $decryp_array_numbers = array(
                        // NUMERICS
                        "6" => "0",
                        "4" => "1",
                        "7" => "2",
                        "8" => "3",
                        "2" => "4",
                        "9" => "5",
                        "3" => "6",
                        "0" => "7",
                        "1" => "8",
                        "5" => "9"
```

Moreover, values cannot be of arbitrary size anymore but the length has to be a multiple of four characters. In the sim
second, and fourth character do not matter and just the third character is taken and remapped. To illustrate this, let u
is taken (which is mapped to 1).

Overall, therefore, the fix consisted of implementing some kind of obfuscation and did not address the issue. To actually address the issue, proper authorization controls would need to be implemented. These controls should guarantee that only authorized users (e.g. administrators or the user to which the password belongs) can change the respective password.

There was some further email exchange, where I tried to explain the issue in more detail (you can find the disclosure timeline below), but ultimately the maintainers stopped responding such that the 90 days disclosure deadline expired.

Finally, I would like to provide two takeaways. The first is that – in my opinion – one should nearly always try to implement web applications based on one of the available web frameworks. There exist a number of them and I do not want to argue about which one to use, but most of them simplify the development process and come with built-in security features that may be used. In the present case, however, much of the application's functionality has been implemented from scratch.

Second, obfuscation is nearly always a bad counter measure. Try understanding the origin of the vulnerability to fix it instead of just making it more complicated to exploit by putting some other functions around it. In the present case, this would mean implementing proper access controls for the affected endpoint.

If you are using OpenSIS, it is recommended to block access to the ResetUserInfo.php as a temporary countermeasure until a fix is available.

Oliver

——————————

Disclosure Timeline

2020-07-08 Maintainer notified via his GitHub page
2020-07-08 Maintainer contacts us via email and asks for details
2020-07-09 Details of identified vulnerabilities are sent to maintainer and disclosure deadline is set to the date 2020-10-07 (2020-07-09 + 90 days)
2020-07-22 Asking the maintainer whether he was able to reproduce the issue
2020-07-26 Maintainer responds that they are working on the issue
2020-08-05 Maintainer publishes a fix and asks to validate it
2020-08-05 Analyzing the fix and responding that the XSS is fixed, but the other issue still remains and an RCE is introduced
2020-08-13 Maintainer publishes a fix and asks to validate it
2020-08-14 Analyzing the fix and responding that the RCE has been removed but the original vulnerability still remains and can only be fixed by implementing proper access controls
2020-09-22 Asking the maintainer whether there is some update on the issue as the disclosure deadline is approaching
2020-10-07 Disclosure deadline (but no response from vendor)
2020-10-15 Publication of blog post