Pentest-Report GreatFire Web Proxy 01.2015

Cure53, Dr.-Ing. Mario Heiderich, Johannes Dahse

Index

Introduction
Scope
Identified Vulnerabilities
GF-01-001 Unfiltered Callback parameter allows for Universal XSS (High)
GF-01-004 Fastly Servers do not enforce SSL Usage properly (High)
GF-01-006 Character Set conversion allows universal XSS (High)
GF-01-007 Unfiltered HTTP Request URI allows for HTML Injection (Medium)
Miscellaneous Issues
GF-01-002 Insecure Permissions for Log-Folder (Low)
GF-01-003 Static Window-ID allows for easy Proxy Detection (Medium)
GF-01-005 URL replacement enables browser XSS filter bypasses (Low)
GF-01-008 Overly verbose Logging risking User Privacy in index.php (Medium)
GF-01-009 Missing HTTP Security Headers for proxied Pages (Info)
Conclusion

Introduction

"Website-mirror-by-proxy is a server-side web proxy designed to host one or multiple dynamic mirror versions of any website. It is based on https://github.com/greatfire/redirect-when-blocked *(the full edition). Whereas redirect-when-blocked requires the source/origin website to be modified, website-mirror-by-proxy runs separately and does not need any modification of the source/origin website."*

From https://github.com/greatfire/website-mirror-by-proxy

This penetration test and code audit against the GreatFire censorship circumvention tool "Website-Mirror-By-Proxy" (later referred to as "Proxy") was carried out by two testers from the Cure53 team. The test lasted four days and resulted in the identification of four security vulnerabilities and five general weaknesses. Among all those issues discovered, none were classified as 'Critical' in severity. Importantly, however, three problems were ranked 'High' in regards to their significance and possible impact.

The test was targeted against both the open and freely available sources of the proxy tool, and the several available instances of the tool itself. In addition, a customized debug version was run on a shared cloud VM to test and reproduce the more intrusive attacks.

Scope

- **PHP Sources**
  - https://github.com/greatfire/website-mirror-by-proxy
- **Test-Server(s)**
  - https://github.com/greatfire/wiki

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact, which is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *GF-01-001*) for the purpose of facilitating any future follow-up correspondence.

GF-01-001 Unfiltered Callback parameter allows for Universal XSS *(High)*

The RWB software utilizes a callback parameter, which can be passed in via the HTTP request method GET. This allows an attacker to inject arbitrary payload that will then be deployed with the content type *application/javascript*. The reason for why the callback handling leads to a security issue is that there are no length or character restrictions applied to this parameter. Consequently, an attacker can take advantage of several techniques in order to cause XSS or cross-domain data exfiltration. The possible methods include:

- **Bypassing CSP.** With the use of forced same-domain JavaScript, the existing CSP protection for a website can be bypassed in a trivial manner[1]. Whenever a website is protecting itself by employing CSP, the attacker gains a capability to bypass the protection by using the callback. This signifies injecting a script element, which includes the URL with the malicious callback. The consequence would be XSS caused by the proxy, occurring in spite of properly securing the website.

- **Universal XSS in older MSIE versions:** Older MSIE versions, inclusive of IE8 and (in certain situations) IE9, do not recognize the MIME type *application/javascript* as a content handler and therefore apply MIME sniffing before actually deciding on how to render the requested data[2]. This can be abused to cause script execution via the unfiltered callback parameter despite content type being set to *application/javascript*.

- **Flash injections and Rosetta Flash attacks:** It is possible to abuse the callback to inject source code of a Flash file and then include the result from another domain[3]. This will lead to having access to contents from the attacked domain,

[1] https://developer.mozilla.org/en-US/docs/Web/Security/CSP
[2] http://blogs.msdn.com/b/ie/archive/2010/10/26/mime-handling-changes-in-internet-explorer.aspx
[3] http://quaxio.com/jsonp_handcrafted_flash_files/

regardless of having been originally embedded from a different domain. While the Rosetta Flash attack[4] (only printable ASCII word-characters) has been fixed in recent Flash player versions, injection of pure and unmodified SWF source code still works. As such, it can be used against the proxy.

- **PDF injections stealing same-domain data:** By injecting the content of a PDF file via callback parameter[5], an attacker can steal the data reflected on the same domain. This is possible due to the reported yet so far unpatched FormCalc behavior of the Adobe Reader 11[6].

It is strongly recommended to exclusively allow usage of the word-characters (a-Z, 0-8) in the callback parameter. Furthermore, the length should be restricted to a maximum of 16 characters or less. Ideally, the callback function name is not dynamic at all, but rather hard-coded and impossible to change via a GET parameter. This particular setup would eliminate the entire range of possible attacks. The affected code can be found below:

Affected File:
/public/rwb/RedirectWhenBlockedFull.inc #181ff

```
/*
 * This request comes from another base url (mirror or source host).
 * We take the normal output and turn it into a jsonp response.
 */
if (RedirectWhenBlockedFull::getOutputType() ==
    RedirectWhenBlockedFull::OUTPUT_TYPE_JSONP) {
    print
        $_GET['callback'] . '(' . json_encode(
            array(
                'html' => utf8_encode_current_charset($html)
            )) . ');';
}
```

A possible fix could be composed as follows:

```
if (RedirectWhenBlockedFull::getOutputType() ==
    RedirectWhenBlockedFull::OUTPUT_TYPE_JSONP) {
    if(preg_match('/\W/', $_GET['callback']) or strlen($_GET['callback']) > 16){
        die('/* invalid callback: max. 16 word characters are allowed */')
    }
    print
        $_GET['callback'] . '(' . json_encode(
            array(
                'html' => utf8_encode_current_charset($html)
            )) . ');';
}
```

[4] https://miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/
[5] http://www.cse.chalmers.se/~andrei/ccs13.pdf
[6] http://insert-script.blogspot.ca/2014/12/multiple-pdf-vulnerabilites-text-and.html

However, if special characters are for some reason necessary for the callback name, it is recommended to use Unicode escapes (*uXXXX*) for the sake of better security[7]. This will ensure that the JavaScript can still recognize method's name, while handling dangerous characters - such as parenthesis or lesser- and greater-than -- in a way that they are masked properly and cannot cause any damage.

GF-01-004 Fastly Servers do not enforce SSL Usage properly *(High)*

During the tests against the available live instances of the proxy tool, it was noticed that a group of servers responds with both SSL and plaintext HTTP. The latter of the two is dangerous and should be avoided. Per recommendation, all these servers should make use of HTTP security headers as described in GF-01-009.

- http://pp3.global.ssl.fastly.net/
- http://bbc2.global.ssl.fastly.net/
- http://fw.global.ssl.fastly.net/
- http://pt.global.ssl.fastly.net/
- http://rmjdw5.global.ssl.fastly.net/

It is advised to use HSTS on the application layer[8] and strictly avoid deploying any content from port 80 (or unencrypted HTTP). Nevertheless, keep in mind that this protection is not perfect: in case the problem is rooted at the webspace provider's side, it should be considered to either contact the provider with a fix request, or, alternatively cease to include them on the providers' list for users' security sake.

GF-01-006 Character Set conversion allows universal XSS *(High)*

The proxy's redirect-when-blocked (RWB) software allows configuring alternative proxies in case the proxy cannot fetch the proxied website's response itself. For this purpose, it uses the JSON mode (activated by the string "rwb3498472=2") to retrieve data from the alternative proxy via JSON.

In this mode, the RWB software searches for the "charset=" string in the proxied website's HTML response in order to determine the used character set. If found, it performs a character set conversion from the character set specified behind the string "charset=" (meaning UTF-7 if the strings says "charset=UTF-7", for example) into the used UTF-8 character set through PHP's native multi-byte conversion. When the proxied website does not specify its character set via expected HTML *meta* tag, an attacker is able to perform universal XSS attacks.

Attack Example:
We assume that the proxied URL reflects the GET parameter *reflect* at some point of the website, for example in a search field. An attacker can inject the string "charset=**UTF-7'**"

[7] https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Values,_variables,_and_literals
[8] http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

(note the single-quote) followed by an UTF-7 encoded XSS payload[9] into the website's HTML response by requesting:

URL:
```
http://proxy1/http://proxied.url/index.php?reflect=charset=UTF-7'%2bADw-
img+src=x+onerror%2bAD0-alert(1)%2bAD4-
```

The proxy will then invoke an AJAX request to an alternative proxy (if configured) in order to fetch the website via this proxy (here: *proxy2*). The payload is transferred in the following AJAX request to proxy2:

URL:
```
http://proxy2/http://proxied.url/index.php?reflect=charset=UTF-7'%2bADw-
img+src=x+onerror%2bAD0-alert(1)%2bAD4-
&rwb3498472=2http://proxy1/&callback=jsonpCallback0&_=1420741850163
```

Next, Proxy2 requests the proxied URL with our payload in the *reflect* parameter. The activated JSON mode of proxy2 (activated by parameter *rwb3498472=2*) will now trigger the character transformation in the response page into the character set found behind the string *charset=* in the HTML. Because the attacker injected *charset=UTF-7*, their injected payload is transformed from UTF-7 into UTF-8. Proxy1 then loads the **converted** HTML response of Proxy2 into its iframe where the XSS payload triggers.

This will bypass any XSS protection mechanism, such as server-side input sanitation or client-side XSS filter, as those operate on UTF-8 strings. As a result, universal XSS without commonly filtered characters such as "<" and ">" is now possible. Furthermore, the technique does not recognize the more modern way of setting character sets by employing the *<meta>* tag with *charset* attribute[10]. This Report advises against using this kind of re-encoding based on HTML-provided data, proposing to rely instead on the HTTP headers for the purpose of retrieving (and subsequent processing of) this information.

GF-01-007 Unfiltered HTTP Request URI allows for HTML Injection *(Medium)*

When a website is blocked, the RWB software displays the targeted proxied URL in an iframe. The iframe source is built from the proxied request URL. While the request URL is URL-encoded in certain browsers, for instance in Firefox and in Chrome, the Internet Explorer browser does not URL-encode meta-characters like "<", ">" and similar.

These characters can be used to break the existing iframe's *src* attribute and inject further HTML or JavaScript code. It can possibly lead to a HTML injection vulnerability for the MSIE users among the members of the GreatFire proxy community. For example, the following query can be appended to the proxy URL and will execute JavaScript in the victim's browser:

[9] http://openmya.hacker.jp/hasegawa/security/utf7cs.html
[10] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta

Sample Attack (working on older IE9/10 and 11):

```
https://proxy/?xss="></iframe><svg><script/xlink:href=data:,alert%281%29><p>
```

Resulting HTML:

```
<iframe frameBorder="0" scrolling="no" verticalscrolling="no" name="rwb3498472i"
seamless="seamless" height="600px" src="/?
xss="></iframe><svg><script/xlink:href=data:,alert
%281%29><p>&rwb3498472=1"></iframe>
<script src="../../rwb/jquery-1.11.1.min.js"></script>
```

Nevertheless note that in the most recent MSIE versions (fully patched MSIE11) an XSS attack is not operational without additional measurements and the impact is limited to a HTML injection. In addition, it is possible for a Man-In-The-Middle attacker, such as an ISP, to append URL-decoded payloads to a proxy request that will then trigger a JavaScript payload in the victim's browser. This attack would function across all browsers, including Chrome and Firefox.

Affected File:

In *public/rwb/RedirectWhenBlockedFull.inc*, line 124,
the iframe source is set to the current request URI.

```
if (! isset($_GET[self::QUERY_STRING_PARAM_NAME])) {
       $iframe_src = $_SERVER['REQUEST_URI'];
       ...
       require 'substitute-page.php';
       exit();
}
```

This iframe source is then used in *substitute-page.php*, line 30:

```
<iframe frameBorder="0" scrolling="no" verticalscrolling="no"
   name="<?php print self::IFRAME_WINDOW_NAME ?>"
   seamless="seamless" height="600px" src="<?php print $iframe_src ?>"></iframe>
```

It is strongly recommended to properly escape every location where user-controlled data is being printed. This includes several data fields from the *_SERVER* array[11]. The method to perform proper escaping for such data would be with PHP's native *htmlentities()*[12] or *htmlspecialchars()*[13] functions.

[11] http://php.net/manual/en/reserved.variables.server.php
[12] http://php.net/htmlentities
[13] http://php.net/htmlspecialchars

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

GF-01-002 Insecure Permissions for Log-Folder *(Low)*

It was discovered during the audit of the proxy software's PHP code that the logging mechanism is using insecure permissions for file and folder creation[14]. This finding does not mean that a directly exploitable vulnerability exists, but rather indicates particular situational conditions. Under this premise, an attacker can lever an existing attack (i.e. arbitrary logfile access caused by forgotten *.htaccess* settings or similar) to gain code execution privileges stemming from the lax permissions. In the current setup, both log files and folder are applied with a permission octet set to 777, which is anything but ideal.

Affected Code:

/public/Log.inc #46ff

```
if (! isset($fh)) {
    $dir = __DIR__ . '/log';
    if (! is_dir($dir)) {
        mkdir($dir);
        chmod($dir, 0777);
    }
    $filename = tempnam($dir,  __CLASS__ . '.' . time() . '.' .
       urlencode($_SERVER['SCRIPT_NAME']));
    touch($filename);
    chmod($filename, 0777);
    $fh = fopen($filename, 'w');
}
```

It is recommended to use far stricter permissions in hopes of making sure that an attacker cannot abuse the log folder to execute arbitrary code. This means that the execution flag for each user group should henceforth be removed. Furthermore, it should be considered to similarly cease read-access to the log files for any user, with the exception of those specifically authorized to access the file. Ideally, the folder and file settings are set to 600, assuming that avoiding any risk of abuse is a goal here. Ultimately, the PHP parser should be excluded from parsing files in this directory[15].

GF-01-003 Static Window-ID allows for easy Proxy Detection *(Medium)*

The window's ID, which is fundamentally important for the iframe and window detection of proxied pages, is nothing but a static key hard-coded in the sources at present. This should be avoided and the static key / ID should be replaced by a dynamically generated, secured and random identifier that varies from user to user, or at least for

[14] http://en.wikipedia.org/wiki/File_system_permissions

[15] http://stackoverflow.com/questions/2336573/php-protecting-site-from-folders-with-777-permissions

each and any proxy instance. Otherwise an attacker can easily spot the used ID by checking the client's side code, and therefore block all proxy instances, regardless of whether URL or host-name are changed.

Affected Code:
RedirectWhenBlockedFull.inc #7ff
```
// This can't be used elsewhere on the website.
const QUERY_STRING_PARAM_NAME = 'rwb3498472';

// Same as above, used for anchors targeting top window.
const TOP_WINDOW_NAME = self::QUERY_STRING_PARAM_NAME;

// Used to identify main iframe.
const IFRAME_WINDOW_NAME = 'rwb3498472i';
```

It is recommended to allow users to set a strong seed value in the config file, and subsequently necessitate a creation of the ID based on that seed value with the use of hashing algorithms. The seed should be unique for each proxy instance and the computed value should be different each and every time that the proxy is being used.

Careful consideration should be given to designing and implementing a system that changes the ID upon every request and communicates the change between iframes. This would help avoid capability to use the ID to pinpoint specific users and track their browsing history. Note that the value of *window.name* is accessible across navigation to and from different names, further remaining in place until actively being reset[16]. Randomization of that value is therefore recommended as well.

GF-01-005 URL replacement enables browser XSS filter bypasses *(Low)*

The GreatFire proxy strips occurrences of alternative base URLs in the request URI that is appended to the proxied URL. This enables an attacker to intentionally use these alternative base URLs during an XSS attack on the proxied URL, with the goal of obfuscating the payload. This way, XSS filters of the browser can be circumvented.

The alternative base URL is publicly visible in the JavaScript source code displayed to the user. By default, it is set to *http://localhost/website-mirror-by-proxy/public/*. For example, while Internet Explorer's XSS filter is able to detect the payload *<script>alert(1)</script>*, it still can be bypassed with the use of the following string:

```
<scrhttp://localhost/website-mirror-by-proxy/public/ipt>alert(1)</script>
```

That very string will later be transformed to the original payload again, by the proxy which strips the alternative base URL. The affected code resides in the method *applyReverse()* of class *TextExternalUrlFilters*.

```
public static function applyReverse(&$body)
{
```

[16] https://developer.mozilla.org/en-US/docs/Web/API/Window.name

```
        foreach (RedirectWhenBlockedFull::getAltBaseUrls() as $alt_base_url) {
                $body = str_replace($alt_base_url, '', $body);
        }
}
```

It is called in the method *getUrl()*, which determines the Proxy request URL.

```
public function getUrl()
{
        static $url;
        if (! isset($url)) {
             $url =
RedirectWhenBlockedFull::getRequestUriWithoutQueryStringParam();
             $this->removeThisScriptDirFromUrl($url);
        }
        $url = Conf::$default_upstream_base_url . $url;
        TextExternalUrlFilters::applyReverse($url);
        return $url;
}
```

It is recommended to minimize the processing of incoming strings that are later being shown in the generated HTML markup in regards to replacements and stripping. While this is not a vulnerability per se, it makes it possible for the attacker to easily bypass a browser's XSS filter by removing certain sub-strings from the later printed and unfiltered data.

GF-01-008 Overly verbose Logging risking User Privacy in index.php *(Medium)*

The logging facilities employed by the proxy tool are overly verbose and log data that, in case the server gets compromised, could potentially be used to target all past users. It should be considered to avoid logging user data in unencrypted way, and, even more so, directly in the webroot. It should be guaranteed that sensitive data and any form of PII[17] in plain text is removed, replaced or otherwise made unreadable.

Affected Code Example:

index.php #9ff

```
Log::add($_SERVER);
Log::add(new Conf());
...

Log::add($request);$message = $request->send();
Log::add($message);
```

Additional options for handling this mishap would be to either log into a database on a different server, or, alternatively, completely disable logging of PII by ceasing all processes wherein the *_SERVER* array and similar sensitive data sources' are logged.

[17] http://en.wikipedia.org/wiki/Personally_identifiable_information

GF-01-009 Missing HTTP Security Headers for proxied Pages *(Info)*

It is recommended to set HTTP security headers to enhance the client-side security of the proxied website's users. Those include the specific:

- `X-Content-Type-Options: nosniff`
- `X-Download-Options: noopen`
- `X-Frame-Options: samorigin`
- `X-XSS-Protection: 1; mode=block`
- `Strict-Transport-Security: max-age=31536000; includeSubdomains`

These headers should not affect the functionality of the proxied websites, yet ought to significantly decrease the chance for carrying out successful attacks against them.

Foremost important is the X-Frame-Options header, the proper handling and securing of which would assure that the proxy still works. While it would still frame websites from same origin, an attacker could not have framed the proxy tool itself. Conducting exploratory attacks to identify vulnerabilities for further harming the user would no longer be possible.

Conclusion

Summing up this four-day long test against the "Website-Mirror-By-Proxy" tool written and published by the GreatFire, one must reiterate that several vulnerabilities were found. Broadly speaking, the findings predominantly pertain to cross-site scripting or related attacks. No critical issues were spotted, and so none of the problems described here would allow an attacker to orchestrate a direct takeover of a server responsible for running this tool. Thus, from a security point of view, the tool appears robust, affected only by minor non-critical issues that are generally rather easy to fix.

At the same time, it must be noted that the code regrettably relies[18] on outdated PECL[19] packages, some of which have known vulnerabilities. This should be addressed immediately. Main focus should be placed on making sure that under no circumstances a developer is forced to install vulnerable packages when wishing to run the tool for privacy- and security-critical operations in the first place. During the process of code auditing it was noticeable that the code quality is not of the very high level. Oftentimes code comments are missing, there is no unified choice made and followed in regards syntax features, and sometimes crude regular expressions are used to get a feature working "just well-enough" rather than properly.

In addition, there is a total lack of test cases, unit tests, functional tests and the like - simply meaning that recommendations and requirements for quality assurance are by no means followed. This should be fixed in a timely manner to arrive at the overall better code quality, which will further facilitate maintaining more appropriate level. In that sense, those effort help to guarantee that the core functionality of the whole tool is not at

[18] https://github.com/greatfire/website-mirror-by-proxy/blob/master/install/install.sh
[19] http://pecl.php.net/

risk of simply breaking down upon introducing changes. The phase following a security audit is often a good moment to start implementing unit tests, just to make sure that the fixed vulnerabilities stay fixed and the fixes work as expected.

More attention should also be given to the installer script and the documentation thereof. It is crucial for a system using the tool to be as secure and well-hardened as possible. When this layer is not sufficient, an attacker can harvest massive amounts of information about users and use the tool for deploying arbitrary HTML markup, potentially containing browser exploits or tracking scripts. In its current state, the installer script should be deemed problematic and requiring attention. Consequently, its documentation should contain more links that would point to resources focused on effective server hardening. For future versions, it should be considered to offer a pre-configured, pre-hardened VM as a deliverable.

Cure53 would like to thank the team of GreatFire for their project coordination, support and assistance before and during this assignment.