Pentest-Report Cupcake / Flashproxy 05.2015

Cure53, Dr.-Ing. Mario Heiderich, Dr. Jonas Magazinius, Fabian Fäßler, Alex Inführ

Index

Introduction

“Flash proxies are a new way of providing access to a censorship circumvention system such as Tor. A flash proxy is a miniature proxy that runs in a web browser. It checks for clients that need access, then conveys data between them and a Tor relay.

"Flash proxy" is a name that should make you think "quick" and "short-lived." Our implementation uses standard web technologies: JavaScript and WebSocket. (In the long-ago past we used Adobe Flash, but do not any longer.) If your browser runs JavaScript and has support for WebSockets then while you are viewing this page your browser is a potential proxy available to help censored Internet users.“

From https://crypto.stanford.edu/flashproxy/#overview

“Cupcake uses something called Flashproxy to create special Tor bridges that are harder to block. As with all circumvention projects, there's a lot more to it than that, but that is the jist. Flashproxy was created by David Fifield, and there is a lot of ongoing research in this area. You can learn more at the Stanford Flashproxy site. Cupcake exists as an easy way to distribute Flashproxy, with the goal of getting as many people to become bridges as possible.”

From https://github.com/glamrock/cupcake#what-even-is-cupcake

This test against the application compound Flashproxy & Cupcake was carried out by four testers of the Cure53 team over the course of eight days. The test only yielded two minor vulnerabilities and one general weakness. This is a rare and praiseworthy result for a software of this level of complexity, which showcases the amount of thought behind the endeavours as well as coding quality that has gone into the two projects.

The test entailed a creation of a setup which allowed the Cure53 team to get control over each part of the application and communication stack. This particular framework involved a custom facilitator set up on a publicly reachable VM[1], a custom version of the Flashproxy code, custom and modified versions of the Cupcake browser extensions, and, last but not least, a custom version of the Tor Browser Bundle[2], which permitted exclusive connectivity to our facilitator, forbidding any other instances.

Scope

- **Cupcake Source Code**
  - https://github.com/glamrock/cupcake
- **Cupcake Extension**
  - https://chrome.google.com/webstore/detail/cupcake/dajjbehmbnbppjkcnpdkaniapgdppdnc
- **Flashproxy Source Code**
  - https://crypto.stanford.edu/flashproxy/#source-code
  - https://fp-facilitator.org/

Test Parameters

This test was carried out with a specific threat model in mind. It involved several different parties that might solely or collaboratively try to attack other parts of the application and communication stack that are Cupcake and Flashproxy. The following list will describe the threat actors. It will further elaborate on their capabilities to harm the tested application-scope and hinder it from functioning correctly.

The aforementioned actors are a **proxy** (the browser of the user that is running the Flashproxy Iframe), a **client** (a TOR-enabled browser/TBB using Flashproxy bridges to bypass censorship), a **facilitator** (a server mediating between ***clients*** and ***proxies***), and lastly the owner of the network(s) that the communication is happening on (ISPs, state controlled entities etc.).

[1] https://cloud.google.com/
[2] https://www.torproject.org/projects/torbrowser.html.en

Consideration of a Malicious Proxy

- The proxy **is** essentially a legitimate MitM[3]
- The proxy **can** collect IPs of clients
- The proxy **can** collect statistics to correlate with other traffic
- The proxy **can** DoS[4] by not connecting
- The proxy ***could*** potentially modify the Diffie-Hellman exchange with the first relay[5]
- The proxy **cannot** decrypt/modify TOR traffic
- The proxy **cannot** attack the client since there is no other data than TOR traffic
- The proxy **cannot** attack the facilitator
- The proxy **cannot** attack the TOR relay

Consideration of a Malicious TOR Client

- The client **can** attempt to attack the facilitator via GET parameter
- The client **can** attempt to attack the facilitator via email
- The client **can** attempt to attack the facilitator via appspot
- The client **can** attempt to attack the proxy via IP parameter
- The client **can** attempt to attack the proxy via the proxy connection
- The client **can** attempt to DoS the facilitator by flooding client registration
- The client **cannot** attack the TOR relay

Consideration of a Malicious Facilitator

- The facilitator **can** attempt to DDOS a target by sending the IP to thousands of proxies
- The facilitator **can** attempt to attack the proxy via parameters
- The facilitator **can** deny a client access by not distributing the IP to proxies
- The facilitator **cannot** attack the client, since there is no input from facilitator to client
- The facilitator **cannot** attack the TOR relay

Consideration of an External Adversary (ISP, State-Level Actor)

- The state **can** block traffic to the facilitator
- The state **can** DoS the facilitator to prevent any connections
- The state **can** poll the facilitator to collect IPs of TOR users
- The state **cannot** block traffic to TOR relays
- The state **cannot** do a more efficient MitM than a proxy can

All communication-, network- and protocol-focused tests were carried out based on the above-described model. In addition, an audit was performed and covered the application's source code, comprising of the JavaScript, Python, PHP and other related code that was found in the Cupcake and Flashproxy repositories.

[3] http://en.wikipedia.org/wiki/Man-in-the-middle_attack
[4] http://en.wikipedia.org/wiki/Denial-of-service_attack
[5] https://svn.torproject.org/svn/projects/design-paper/tor-design.html#subsubsec:constructing-a-circuit

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. *CFP-01-00X)* for the purpose of facilitating any future follow-up correspondence.

CFP-01-001 Malicious user can collect TOR Users' IPs / Facilitator DoS *(Medium)*

By design the users that want to get access to TOR[6] through flashproxy will register on the *fp-facilitator.*[7] Relays can query the fp-facilitator for clients to connect to. The *fp-facilitator* will then return a client's IP. This makes it possible for a malicious user to collect IP addresses from users seeking to get access to TOR. The vulnerability can also be used to drain all clients from the fp-facilitator, while the real working relays would no longer find clients.

PoC:

```
curl -k https://130.211.133.236\?r\=1
```

Note that the given IP points to the custom facilitator we set up for this test, meaning that it may no longer be active at the time of reading. The facilitator already implements a poll interval via the check-back parameter. This interval is only enforced in the Cupcake extensions but not on the server side. By implementing this check on the server side to force an IP to cease and wait for a configured period of time, meaning before it can issue request for new clients, this attack could be prevented.

CFP-01-003 Inner Protocol of Client-Transport Parameter not validated *(Low)*

Flashproxy clients can specify both an inner and an outer protocol via the *client-transport* parameter ie. `client-transport=string|websocket`. [8] While the facilitator parses the inner and the outer protocol, only the outer protocol is validated against a list of valid protocols.

This makes it possible to spam the facilitator with clients' registrations which have random inner protocols. There will never be a relay supporting the same inner and outer protocol, so the clients will never be pulled from the internal clients list and fill up the facilitators memory. The functionality of the inner protocol could be dropped because it appears to be an unused feature.

However, it would be even more optimal if the inner protocol was properly handled and validated against a list of valid protocols.

[6] https://www.torproject.org/
[7] https://gitweb.torproject.org/flashproxy.git/tree/facilitator
[8] https://www.torproject.org/docs/pluggable-transports.html.en

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid attackers 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.

CFP-01-002 “Client” Parameter is not being validated correctly *(Low)*

The facilitator does not properly validate the “client” parameter sent by Flashproxy clients. The only check is done at line 412 in the executable file *fp-facilitator*[9], where the function `addr_af()`[10] is called. It passes the client's address to the Python method `socket.getaddrinfo()`[11]. Although an invalid address should normally cause an error,it was discovered during the test that the function respects null bytes. An address such as `1.1.1.1\0Definitely not an address`, will be parsed without an error and the complete address will be stored in the clients table.

This is just a minor issue because it was not possible to spoof new parameters or cause any application to malfunction. Nevertheless it is recommended to check the client parameters for any additional characters other than digits and dots.

[9] https://gitweb.torproject.org/flashproxy.git/tree/facilitator/fp-facilitator#n412
[10] https://gitweb.torproject.org/flashproxy.git/tree/facilitator/fp-facilitator#n393
[11] https://docs.python.org/2/library/socket.html

Conclusion

It is a very rare situation for a thorough team-penetration test against a complex software-compound to yield as few vulnerabilities and weaknesses as Cupcake and Flashproxy did. The result speaks to the thoughtfulness and experience that have gone into the code of both Flashproxy and Cupcake. It also illuminates a praiseworthy approach to the very specific goal and scope of this software. We found no major threats in the sources of the tested projects.

A setup was tailor-made in order to acquire a desired capability to test the software compound properly and enable the team to explore each and every of the aforementioned threat actors and re-enact their capabilities throughout the test accordingly. Said setup included a running facilitator on a publicly accessible VM, which aided possibilities for modifying the code at runtime. Further, we altered the JavaScript delivered by Flashproxy and optimized it for debugging purposes. Lastly, a modified TOR Browser was created and compiled in the hopes of only connecting to those proxies that were offered by our manipulated facilitator in the first instance. Later, it was also adopted to use and examine the proxy of the client's data through the dedicated IPs our team controlled. This way, with the the help of a custom facilitator, proxy and client, we could control each and every part of the threat model and get a deeper insight into the inner workings of Flashproxy and Cupcake.

Despite the complex setup only a handful of issues was spotted and reported to the maintainers. None of those issues were of notable severity and, given Flashproxy’s threat model, none of them required immediate fixes or optimizations. No exploit that would reliably hinder the communication or affect any of the involved parties in a significantly severe or negative way was discovered.

Cure53 would like to thank Griffin Boyce and the Cupcake / Flashproxy Team for awarding us this interesting project, as well as for their continuously good support and assistance during this assignment.