RADICALLY OPEN SECURITY

Penetration Test Report

Open Tech Fund

V 0.2
Diemen, December 13th, 2019
Confidential

Document Properties

Client	Open Tech Fund
Title	Penetration Test Report
Target	https://github.com/iyouport-org/relaybaton
Version	0.2
Pentester	Pierre Pronchery
Author	Pierre Pronchery
Reviewed by	FirstName LastName
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	December 9th, 2019	Pierre Pronchery	Initial draft
0.2	December 13th, 2019	Pierre Pronchery	Import the results of the re-test.

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	John Sinteur
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between November 26, 2019 and December 9, 2019, Radically Open Security B.V. carried out a penetration test for Open Tech Fund.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- https://github.com/iyouport-org/relaybaton

(Go source code)

A breakdown of the scoped services can be found below:

- Code audit relaybaton, including scoping and reporting: 1 days
- Retest: 0.5 days
- **Total effort: 1.5 days**

1.3 Project objectives

The objective of this project is to perform a security review on the source code for Relaybaton, a pluggable transport to circumvent Internet censorship.

1.4 Timeline

The Security Audit took place between November 26, 2019 and December 9, 2019.

1.5 Results In A Nutshell

Some issues were found in the authentication routine for the service. Most importantly, it is vulnerable to replay attacks, as described in OTFRB-001 (page 10). Then, the use of SHA256 as the password hashing algorithm lets the password be more easily guessed than necessary, as explained in OTFRB-002 (page 11) and OTFRB-003 (page

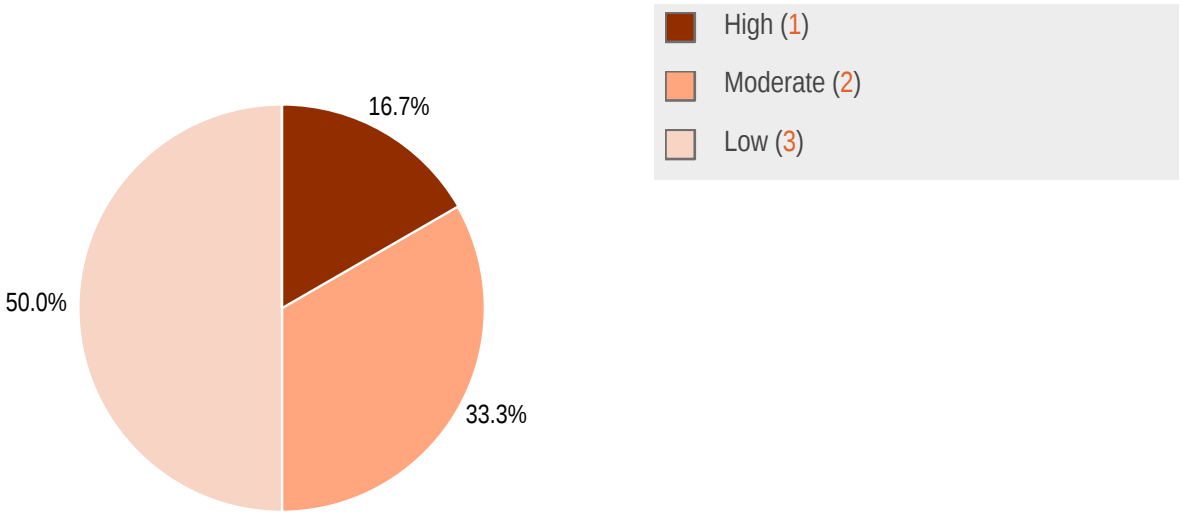
12). Finally, it will not work if the clocks of the server and client are not synchronized, as can be seen in OTFRB-004 (page 14).

In addition, minor concerns were expressed about relying on third-party infrastructure, when obtaining the source code in OTFRB-005 (page 15) or when using DNS in OTFRB-006 (page 16).

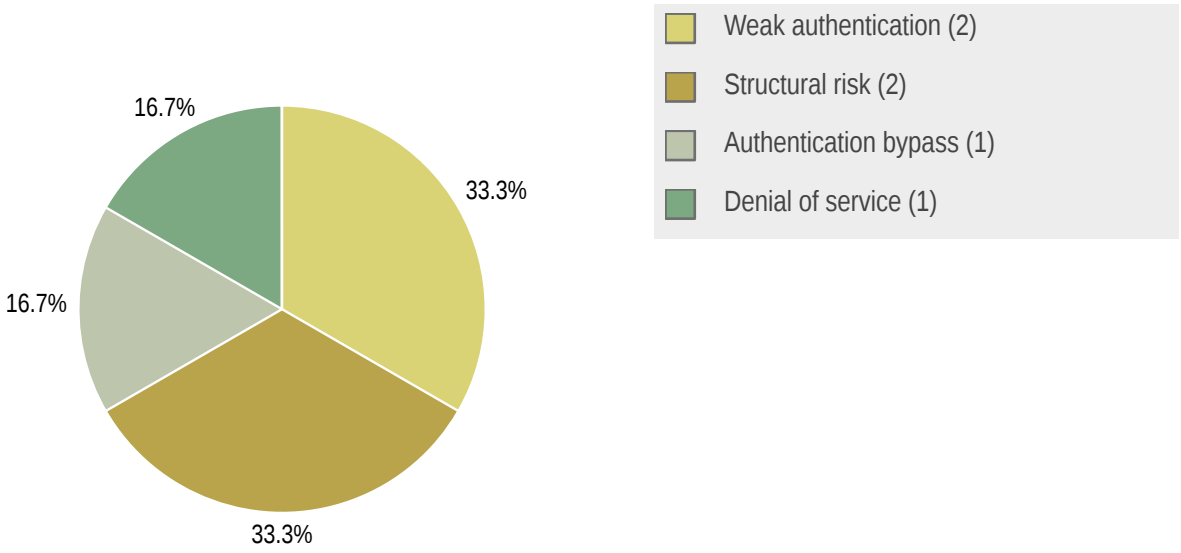
1.6 Summary of Findings

ID	Type	Description	Threat level
OTFRB-001	Authentication Bypass	The authentication token is valid for a given user for a total duration of 60 seconds. If revealed to an attacker, it is possible to re-use it, without knowledge of the password, for this entire duration.	High
OTFRB-002	Weak Authentication	The password used for authenticating users is transformed using the SHA256 hashing algorithm. While not reversible and cryptologically up to date, this algorithm is designed to avoid collisions while being computationally as efficient as possible. This is harmful when protecting secrets like passwords.	Moderate
OTFRB-003	Weak Authentication	Even though an IV is provided to protect the encrypted information, an exact SHA256 hash of the original password is used as the cryptographic key. As a consequence, it is easily possible to recover the original password through a dictionary attack.	Moderate
OTFRB-004	Denial of Service	The window of operation for the authentication to work is based on a shared clock, and it will only succeed within a window of 60 seconds at most. As a consequence, the authentication will fail blindly as soon as anyone does not use NTP with a skew of 60 seconds or more (server or client).	Low
OTFRB-005	Structural Risk	Most of the Go files audited in the project rely on third-party components, hosted on Github, without checks on the specific version to be used.	Low
OTFRB-006	Structural Risk	The resolution of DNS names to IPv6 or IPv4 addresses is performed with DNS over HTTP, and is hard-coded to use Cloudflare's infrastructure. There are both advantages and disadvantages in terms of security and privacy when using this solution.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
OTFRB-001	Authentication Bypass	• Consider using a challenge-response protocol instead.
OTFRB-002	Weak Authentication	• Use a hashing algorithm specialized for password-based authentication, such as bcrypt or argon2i.
OTFRB-003	Weak Authentication	• Use the IV (possibly in part) to salt the password. • Provide a number of iterations for the hashing algorithm.
OTFRB-004	Denial of Service	• Consider using a challenge-response protocol instead.
OTFRB-005	Structural Risk	• Fork every project used (and their dependencies) to a repository under OTF's control, or • Import the remote projects as Git submodules (thus enforcing the version used) • Consider hosting these projects in OTF's own infrastructure.
OTFRB-006	Structural Risk	• Let the administrator of the server instance configure which method should be used for DNS name resolution.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection, etc., afforded to the network. This usually involves trying to discover publicly available information by utilizing a web browser, visiting newsgroups, etc. An active form would be more intrusive and may show up in audit logs and may take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 OTFRB-001 — The Authentication Is Vulnerable to Replay Attacks

Vulnerability ID: OTFRB-001 **Retest status:** Resolved

Vulnerability type: Authentication Bypass

Threat level: High

Description:

The authentication token is valid for a given user for a total duration of 60 seconds. If revealed to an attacker, it is possible to re-use it, without knowledge of the password, for this entire duration.

Technical description:

In `server.go`, function `authenticate()`:

```
155 func (handler Handler) authenticate(header http.Header) error {
[...]
185         if time.Since(time.Unix(plaintext, 0)).Seconds() > 60 {
186                 err = errors.New("authentication fail")
187                 log.Error(err)
188                 return err
189         }
190         return nil
191 }
```

In `client.go`, function `buildHeader()`:

```
305 func buildHeader(conf Config) (http.Header, error) {
[...]
310         plaintext := []byte(strconv.FormatInt(time.Now().Unix(), 2))
311         block, err := aes.NewCipher(key)
312         if err != nil {
313                 log.Error(err)
314                 return nil, err
315         }
[...]
322         stream := cipher.NewCFBEncrypter(block, iv)
323         stream.XORKeyStream(cipherText[aes.BlockSize:], plaintext)
324
325         header.Add("username", conf.Client.Username)
326         header.Add("auth", hex.EncodeToString(cipherText))
327         return header, nil
```

```
328 }
```

Status: RESOLVED

Feedback from Pentest Team:

This issue was fixed with the addition and tracking of nonce values.

Impact:

Authentication can be bypassed for a limited duration.

Recommendation:

- Consider using a challenge-response protocol instead.

3.2 OTFRB-002 — The Authentication Relies on a Fast Hashing Algorithm

Vulnerability ID: OTFRB-002 **Retest status:** Resolved

Vulnerability type: Weak Authentication

Threat level: Moderate

Description:

The password used for authenticating users is transformed using the SHA256 hashing algorithm. While not reversible and cryptologically up to date, this algorithm is designed to avoid collisions while being computationally as efficient as possible. This is harmful when protecting secrets like passwords.

Technical description:

In `server.go`, function `authenticate()`:

```
155 func (handler Handler) authenticate(header http.Header) error {
[...]
163         h := sha256.New()
164         h.Write([]byte(handler.getPassword(username)))
[...]
191 }
```

In `client.go`, function `buildHeader()`:

```
305 func buildHeader(conf Config) (http.Header, error) {
```

```
306         header := http.Header{}
307         h := sha256.New()
308         h.Write([]byte(conf.Client.Password))
309         key := h.Sum(nil)
[...]
328 }
```

See also https://en.wikipedia.org/wiki/Bcrypt, https://www.usenix.org/legacy/events/usenix99/provos/provos_html/, and https://en.wikipedia.org/wiki/Argon2.

Status: RESOLVED

Feedback from Pentest Team:

This issue was fixed with the use of Argon2 as password cipher, together with a random salting value.

Impact:

Computational attacks on the password are faster than they could be.

Recommendation:

- Use a hashing algorithm specialized for password-based authentication, such as bcrypt or argon2i.

3.3 OTFRB-003 — The Authentication Is Vulnerable to Dictionary Attacks

Vulnerability ID: OTFRB-003 **Retest status:** Resolved

Vulnerability type: Weak Authentication

Threat level: Moderate

Description:

Even though an IV is provided to protect the encrypted information, an exact SHA256 hash of the original password is used as the cryptographic key. As a consequence, it is easily possible to recover the original password through a dictionary attack.

Technical description:

In `server.go`, function `authenticate()`:

```
155 func (handler Handler) authenticate(header http.Header) error {
[...]
```

```
163         h := sha256.New()
164         h.Write([]byte(handler.getPassword(username)))
165         key := h.Sum(nil)
166         block, err := aes.NewCipher(key)
[...]
178         stream := cipher.NewCFBDecrypter(block, iv)
179         stream.XORKeyStream(cipherText, cipherText)
180         plaintext, err := strconv.ParseInt(string(cipherText), 2, 64)
[...]
185         if time.Since(time.Unix(plaintext, 0)).Seconds() > 60 {
186                 err = errors.New("authentication fail")
187                 log.Error(err)
188                 return err
189         }
190         return nil
191 }
```

In `client.go`, function `buildHeader()`:

```
305 func buildHeader(conf Config) (http.Header, error) {
306         header := http.Header{}
307         h := sha256.New()
308         h.Write([]byte(conf.Client.Password))
309         key := h.Sum(nil)
310         plaintext := []byte(strconv.FormatInt(time.Now().Unix(), 2))
311         block, err := aes.NewCipher(key)
[...]
322         stream := cipher.NewCFBEncrypter(block, iv)
323         stream.XORKeyStream(cipherText[aes.BlockSize:], plaintext)
324
325         header.Add("username", conf.Client.Username)
326         header.Add("auth", hex.EncodeToString(cipherText))
327         return header, nil
328 }
```

The target plaintext is known to be an ASCII string of ones and zeroes (the '1' and '0' ASCII characters). Once this target hit, it is trivial to generate new authentication headers, based on the current time as expected.

While we believe that Rainbow tables could be created for this authentication scheme, they would have to take the current time (in seconds) into account, thereby making them very complex.

See also https://csrc.nist.gov/publications/detail/sp/800-132/final.

Status: RESOLVED

Feedback from Pentest Team:

This issue was fixed with the use of Argon2 as password cipher, together with a random salting value.

Impact:

It is easily possible to bruteforce passwords.

Recommendation:

- Use the IV (possibly in part) to salt the password.
- Provide a number of iterations for the hashing algorithm.

3.4 OTFRB-004 — The Authentication Relies on a Shared Clock

Vulnerability ID: OTFRB-004 **Retest status:** Not Retested

Vulnerability type: Denial of Service

Threat level: Low

Description:

The window of operation for the authentication to work is based on a shared clock, and it will only succeed within a window of 60 seconds at most. As a consequence, the authentication will fail blindly as soon as anyone does not use NTP with a skew of 60 seconds or more (server or client).

Technical description:

In `server.go`, function `authenticate()`:

```
155 func (handler Handler) authenticate(header http.Header) error {
[...]
185         if time.Since(time.Unix(plaintext, 0)).Seconds() > 60 {
186                 err = errors.New("authentication fail")
187                 log.Error(err)
188                 return err
189         }
190         return nil
191 }
```

In `client.go`, function `buildHeader()`:

```
305 func buildHeader(conf Config) (http.Header, error) {
[...]
310         plaintext := []byte(strconv.FormatInt(time.Now().Unix(), 2))
[...]
328 }
```

Some ways to harm proper operation through this attack vector could include:

- corrupting the time source used by either parties (e.g. NTP, FM, GPS...)

- blocking the NTP traffic if it is in use.

Impact:

The protocol may fail to work without any obvious reason for the user.

Recommendation:

- Consider using a challenge-response protocol instead.

3.5 OTFRB-005 — Unmanaged Third-Party Dependencies

Vulnerability ID: OTFRB-005 **Retest status:** Not Retested

Vulnerability type: Structural Risk

Threat level: Low

Description:

Most of the Go files audited in the project rely on third-party components, hosted on Github, without checks on the specific version to be used.

Technical description:

```
$ grep -F 'github.com' client.go config.go connection_pool.go main/main.go peer.go server.go
 websocket_writer.go
client.go:      "github.com/gorilla/websocket"
client.go:      "github.com/iyouport-org/doh-go"
client.go:      "github.com/iyouport-org/doh-go/dns"
client.go:      "github.com/iyouport-org/socks5"
client.go:      log "github.com/sirupsen/logrus"
main/main.go:   "github.com/iyouport-org/relaybaton"
main/main.go:   log "github.com/sirupsen/logrus"
main/main.go:   "github.com/spf13/viper"
peer.go:        "github.com/gorilla/websocket"
peer.go:        log "github.com/sirupsen/logrus"
server.go:      "github.com/gorilla/websocket"
server.go:      "github.com/iyouport-org/doh-go"
server.go:      "github.com/iyouport-org/doh-go/dns"
server.go:      "github.com/iyouport-org/socks5"
server.go:      log "github.com/sirupsen/logrus"
websocket_writer.go:    "github.com/gorilla/websocket"
websocket_writer.go:    "github.com/iyouport-org/socks5"
```

```
websocket_writer.go:    log "github.com/sirupsen/logrus"
```

There is no warranty for these repositories to freeze their API, remain consistent, or even available over time. They may also introduce vulnerabilities, either directly or indirectly (e.g. through malicious pull-ups or other compromise).

Impact:

There is a risk of Denial of Service (unavailability of the repository), erroneous behaviour (breach of implementation contract) or remote code execution (malicious content) when using these repositories.

Recommendation:

- Fork every project used (and their dependencies) to a repository under OTF's control, or
- Import the remote projects as Git submodules (thus enforcing the version used)
- Consider hosting these projects in OTF's own infrastructure.

3.6 OTFRB-006 — Name Resolution Defaults to Cloudflare's DoH

Vulnerability ID: OTFRB-006 **Retest status:** Resolved

Vulnerability type: Structural Risk

Threat level: Low

Description:

The resolution of DNS names to IPv6 or IPv4 addresses is performed with DNS over HTTP, and is hard-coded to use Cloudflare's infrastructure. There are both advantages and disadvantages in terms of security and privacy when using this solution.

Technical description:

In file `server.go`, function `nsLookup()`:

```
232 func nsLookup(domain string) (net.IP, byte, error) {
233         var dstAddr net.IP
234         dstAddr = nil
235
236         ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
237         defer cancel()
238         c := doh.New(doh.CloudflareProvider)
```

Naturally, when deployed directly within Cloudflare's infrastructure, this finding does not apply. This is recommended in the documentation in order to *« [support] ESNI features and [hide] the IP address of the server from interception »*.

Status: RESOLVED

Feedback from Pentest Team:

This issue was fixed by allowing an additional DoH provider, Quad9. We still recommend to add support for additional providers as they are implemented upstream.

Impact:

Corporate and political interests may take advantage of this position to eavesdrop or tamper with the network connections of this server and its clients.

Recommendation:

- Let the administrator of the server instance configure which method should be used for DNS name resolution.

4 Future Work

- **Audit the external dependencies**
  The code base depends on some third-party components, from public repositories. As explained in this report, the security of this project relies on the quality and stability of these repositories. It would therefore make sense to audit their code as well.

5 Conclusion

In this project, the authentication routine drew the most attention. It was implemented through a custom algorithm, where some cryptological weaknesses were identified. Most were addressed within days after their discovery. Both of the remaining issues identified have a lower impact, where the protocol still requires both the client and server to operate with a common date and time reference, and the Go language does not allow pinning specific versions for third-party components.

Finally we want to emphasize that security is a continuous process; this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of yoursecurity. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully. Do not hesitate to let us know if you have any further questions or need further clarification of anything in this report.

Appendix 1 Testing team

Pierre Pronchery	Pierre Pronchery is a Senior IT-Security Consultant and an accomplished developer. Freelancing for over a decade now, he can be found auditing major companies in the Telecommunications, Finance and Retail sectors, or supporting the Open Source Software and Hardware movements. He is currently serving as Vice-President for the NetBSD Foundation, and the founder and CEO of Defora Networks GmbH in Germany.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.