Document Properties

<table>
<thead>
<tr>
<th>Client</th>
<th>Open Tech Fund</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Penetration Test Report</td>
</tr>
<tr>
<td>Target</td>
<td><a href="https://github.com/iyouport-org/relaybaton">https://github.com/iyouport-org/relaybaton</a></td>
</tr>
<tr>
<td>Version</td>
<td>0.2</td>
</tr>
<tr>
<td>Pentester</td>
<td>Pierre Pronchery</td>
</tr>
<tr>
<td>Author</td>
<td>Pierre Pronchery</td>
</tr>
<tr>
<td>Reviewed by</td>
<td>FirstName LastName</td>
</tr>
<tr>
<td>Approved by</td>
<td>Melanie Rieback</td>
</tr>
</tbody>
</table>

Version control

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>December 9th, 2019</td>
<td>Pierre Pronchery</td>
<td>Initial draft</td>
</tr>
<tr>
<td>0.2</td>
<td>December 13th, 2019</td>
<td>Pierre Pronchery</td>
<td>Import the results of the re-test.</td>
</tr>
</tbody>
</table>

Contact

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

<table>
<thead>
<tr>
<th>Name</th>
<th>John Sinteur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Overdijmenweg 28</td>
</tr>
<tr>
<td></td>
<td>1111 PP Diemen</td>
</tr>
<tr>
<td></td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Phone</td>
<td>+31 (0)20 2621 255</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:info@radicallyopensecurity.com">info@radicallyopensecurity.com</a></td>
</tr>
</tbody>
</table>

Radically Open Security B.V. is registered at the trade register of the Dutch chamber of commerce under number 60628081.
# Table of Contents

1  **Executive Summary**  
1.1 Introduction  
1.2 Scope of work  
1.3 Project objectives  
1.4 Timeline  
1.5 Results In A Nutshell  
1.6 Summary of Findings  
1.6.1 Findings by Threat Level  
1.6.2 Findings by Type  
1.7 Summary of Recommendations  

2  **Methodology**  
2.1 Planning  
2.2 Risk Classification  

3  **Findings**  
3.1 OTFRB-001 — The Authentication Is Vulnerable to Replay Attacks  
3.2 OTFRB-002 — The Authentication Relies on a Fast Hashing Algorithm  
3.3 OTFRB-003 — The Authentication Is Vulnerable to Dictionary Attacks  
3.4 OTFRB-004 — The Authentication Relies on a Shared Clock  
3.5 OTFRB-005 — Unmanaged Third-Party Dependencies  
3.6 OTFRB-006 — Name Resolution Defaults to Cloudflare’s DoH  

4  **Future Work**  

5  **Conclusion**  

Appendix 1  **Testing team**
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/youport-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
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

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Description</th>
<th>Threat level</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTFRB-001</td>
<td>Authentication Bypass</td>
<td>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.</td>
<td>High</td>
</tr>
<tr>
<td>OTFRB-002</td>
<td>Weak Authentication</td>
<td>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.</td>
<td>Moderate</td>
</tr>
<tr>
<td>OTFRB-003</td>
<td>Weak Authentication</td>
<td>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.</td>
<td>Moderate</td>
</tr>
<tr>
<td>OTFRB-004</td>
<td>Denial of Service</td>
<td>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).</td>
<td>Low</td>
</tr>
<tr>
<td>OTFRB-005</td>
<td>Structural Risk</td>
<td>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.</td>
<td>Low</td>
</tr>
<tr>
<td>OTFRB-006</td>
<td>Structural Risk</td>
<td>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.</td>
<td>Low</td>
</tr>
</tbody>
</table>
1.6.1 Findings by Threat Level

![Pie chart showing threat level distribution](chart1.png)

- High (1) - 16.7%
- Moderate (2) - 33.3%
- Low (3) - 50.0%

1.6.2 Findings by Type

![Pie chart showing type distribution](chart2.png)

- Weak authentication (2) - 33.3%
- Structural risk (2) - 33.3%
- Authentication bypass (1) - 16.7%
- Denial of service (1) - 16.7%
## 1.7 Summary of Recommendations

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTFRB-001</td>
<td>Authentication Bypass</td>
<td>- Consider using a challenge-response protocol instead.</td>
</tr>
<tr>
<td>OTFRB-002</td>
<td>Weak Authentication</td>
<td>- Use a hashing algorithm specialized for password-based authentication, such as bcrypt or argon2.</td>
</tr>
<tr>
<td>OTFRB-003</td>
<td>Weak Authentication</td>
<td>- Use the IV (possibly in part) to salt the password.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Provide a number of iterations for the hashing algorithm.</td>
</tr>
<tr>
<td>OTFRB-004</td>
<td>Denial of Service</td>
<td>- Consider using a challenge-response protocol instead.</td>
</tr>
<tr>
<td>OTFRB-005</td>
<td>Structural Risk</td>
<td>- Fork every project used (and their dependencies) to a repository under OTF's control, or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Import the remote projects as Git submodules (thus enforcing the version used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Consider hosting these projects in OTF's own infrastructure.</td>
</tr>
<tr>
<td>OTFRB-006</td>
<td>Structural Risk</td>
<td>- Let the administrator of the server instance configure which method should be used for DNS name resolution.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Vulnerability ID: OTFRB-001</th>
<th>Retest status: Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability type: Authentication Bypass</td>
<td></td>
</tr>
<tr>
<td>Threat level: High</td>
<td></td>
</tr>
</tbody>
</table>

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():

```go
155 func (handler Handler) authenticate(header http.Header) error {
156     [...]
157     if time.Since(time.Unix(plaintext, 0)).Seconds() > 60 {
158         err = errors.New("authentication fail")
159         log.Error(err)
160         return err
161     } else {
162         return nil
163     }
```

In client.go, function buildHeader():

```go
305 func buildHeader(conf Config) (http.Header, error) {
306     [...]
307     plaintext := []byte(strconv.FormatInt(time.Now().Unix(), 2))
308     block, err := aes.NewCipher(key)
309     if err != nil {
310         log.Error(err)
311         return nil, err
312     }
313     [...]
314     stream := cipher.NewCFBEncrypter(block, iv)
315     stream.XORKeyStream(cipherText[aes.BlockSize:], plaintext)
317     header.Add("auth", hex.EncodeToString(cipherText))
318     return header, nil
```
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

<table>
<thead>
<tr>
<th>Vulnerability ID: OTFRB-002</th>
<th>Retest status: Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability type: Weak Authentication</td>
<td></td>
</tr>
<tr>
<td>Threat level: Moderate</td>
<td></td>
</tr>
</tbody>
</table>

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()`:

```go
355 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()`:

```go
395 func buildHeader(config Config) (http.Header, error) {
```

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  
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():

```go
155 func (handler Handler) authenticate(header http.Header) error {
[...]
```
In `client.go`, function `buildHeader()`:

```go
305 func buildHeader(config Config) (http.Header, error) {
306     header := http.Header{
307         h := sha256.New()
308         h.Write([]byte(config.Client.Password))
309         key := h.Sum(nil)
310         plaintext := []byte(time.Now().Unix(), 2)
311         block, err := aes.NewCipher(key)
312         stream := cipher.NewCFBEncrypter(block, iv)
313         stream.XORKeyStream(cipherText, plaintext)
315         header.Add("auth", hex.EncodeToString(cipherText))
316         return header, nil
317     }
318
time.Since(time.Unix(plaintext, 0)).Seconds() > 60 {
319         err = errors.New("authentication fail")
320         log.Error(err)
321         return err
322     }
323     return nil
324 }
```

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.


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

<table>
<thead>
<tr>
<th>Vulnerability ID: OTFRB-004</th>
<th>Retest status: Not Retested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability type: Denial of Service</td>
<td></td>
</tr>
<tr>
<td>Threat level: Low</td>
<td></td>
</tr>
</tbody>
</table>

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()`:

```go
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()`:

```go
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

<table>
<thead>
<tr>
<th>Vulnerability ID: OTFRB-005</th>
<th>Retest status: Not Retested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerability type: Structural Risk</td>
<td></td>
</tr>
<tr>
<td>Threat level: Low</td>
<td></td>
</tr>
</tbody>
</table>

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"
```
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

<table>
<thead>
<tr>
<th>Vulnerability ID: OTFRB-006</th>
<th>Retest status: Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Vulnerability type:</strong> Structural Risk</td>
<td></td>
</tr>
<tr>
<td><strong>Threat level:</strong> Low</td>
<td></td>
</tr>
</tbody>
</table>

**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()`:

```go
232 func nsLookup(domain string) (net.IP, byte, error) {
233     var dstAddr net.IP
234     dstAddr = nil
235     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 your security. 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pierre Pronchery</td>
<td>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.</td>
</tr>
<tr>
<td>Melanie Rieback</td>
<td>Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.</td>
</tr>
</tbody>
</table>