Code Audit Report

Blueprint for free speech

V 1.1
Amsterdam, November 18th, 2022
Confidential

Document Properties

Client	Blueprint for free speech
Title	Code audit report
Target	https://github.com/blueprint-freespeech/gosling/
Version	1.1
Pentester	Daniel Attevelt (OSCP)
Authors	Daniel Attevelt, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	November 1st, 2022	Daniel Attevelt	Initial draft
0.2	November 16th, 2022	Marcus Bointon	Review
1.0	November 16th, 2022	Marcus Bointon	1.0
1.1	November 18th, 2022	Daniel Attevelt	Clarified recommendation f6

Contact

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

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1	**Executive Summary**	**4**
1.1	Introduction	4
1.2	Scope of work	4
1.3	Project objectives	4
1.4	Timeline	4
1.5	Results In A Nutshell	4
1.6	Summary of Findings	5
1.6.1	Findings by Threat Level	6
1.6.2	Findings by Type	6
1.7	Summary of Recommendations	6
2	**Methodology**	**8**
2.1	Planning	8
2.2	Risk Classification	8
3	**Findings**	**10**
3.1	GS-005 — DOS through malformed string	10
3.2	GS-001 — DoS through OOM condition	11
3.3	GS-002 — DoS through stale connections	12
3.4	GS-003 — Protocol is vulnerable to MITM attack	13
3.5	GS-004 — Protocol is not end-to-end encrypted	14
3.6	GS-006 — Path interception vulnerability	15
4	**Future Work**	**17**
5	**Conclusion**	**18**
Appendix 1	**Testing team**	**19**

1 Executive Summary

1.1 Introduction

Between October 24, 2022 and October 31, 2022, Radically Open Security B.V. carried out a security audit for Blueprint for free speech.

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

Some proofs of concept have been developed that should aid in the understanding of the issues, as well as help the developers to mitigate them. These can be found in the `poc.tar.gz` file in the report repository root. The archive contains a git repository, in which each commit is annotated with an issue number. Use `git log` for an overview.

1.2 Scope of work

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

- https://github.com/blueprint-freespeech/gosling/

The scoped services are broken down as follows:

- Code audit: 6 days
- Reporting: 1 days
- **Total effort: 7 days**

1.3 Project objectives

ROS will perform a code audit of the source repository in order to assess its security. To do so, ROS will analyze the codebase statically and dynamically and guide blueprint in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between October 24, 2022 and October 31, 2022.

1.5 Results In A Nutshell

We discovered 1 High, 1 Elevated, 2 Moderate and 2 Low-severity issues during this penetration test.

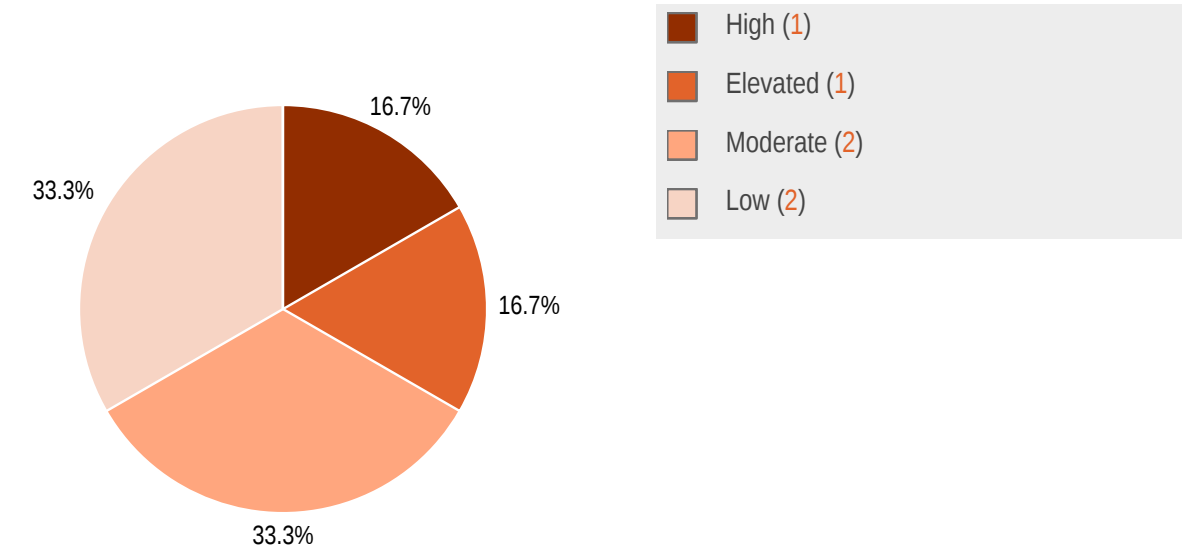
We were not able to breach the protocol that allows a user to obtain a communications channel on an endpoint server. Gosling's solution to the cyberstalking problem as described in the protocol specification is valid.

However, we did find some problems related to denial of service, end-to-end encryption, and the possibility of a machine-in-the-middle attack.

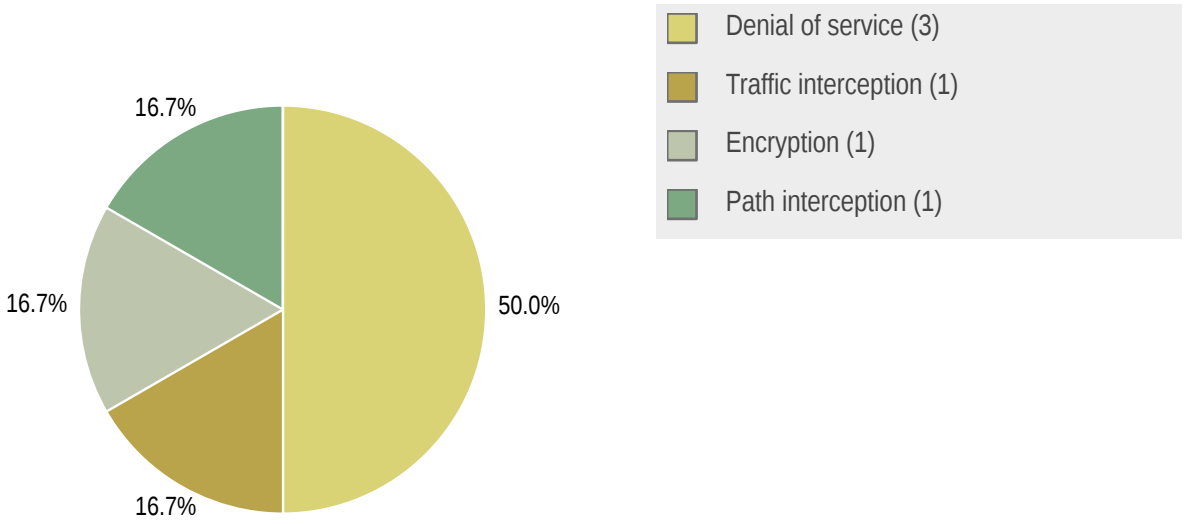
1.6 Summary of Findings

ID	Type	Description	Threat level
GS-005	Denial of service	The server can be crashed by submitting a malformed string.	High
GS-001	Denial of service	The application running the protocol can be crashed through manipulation of the BSON document size header.	Elevated
GS-002	Denial of service	The Gosling protocol does not close stale connections, resulting in resource exhaustion.	Moderate
GS-003	Traffic interception	The protocol is vulnerable to eavesdropping and message manipulation by a third party that is able to position itself between the communicating hosts.	Moderate
GS-004	Encryption	The link between the tor process and the application is not encrypted, allowing for plaintext extraction under certain circumstances.	Low
GS-006	Path interception	The method of spawning the tor process can be abused to spawn a malicious process.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
GS-005	Denial of service	• Sanitize and validate the `endpoint` parameter before using it.

GS-001	Denial of service	• Pick a lower limit for the BSON document size.
GS-002	Denial of service	• Implement state tracking logic that kills the connection if no bytes are present on the stream after reading the BSON length header. • Enforce handshake completion within a reasonably short period, or kill the connection.
GS-003	Traffic interception	• Make use of a secure protocol to establish trust after the communications channel has been opened.
GS-004	Encryption	• Negotiate a shared symmetric key through deployment of ecdh-25519.
GS-006	Path interception	• Use a suitable package to look up the path to the executable, and determine whether the path isroot/admin writable only. This gives some degree of protection. • Alternatively, hash the executable and compare to a known list of hashes for `tor`. This is the most secure, but precludes the use of custom `tor` versions.

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 afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might 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. This step also consist of manually testing the application against the latest (2017) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

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 GS-005 — DOS through malformed string

Vulnerability ID: GS-005

Vulnerability type: Denial of service

Threat level: High

Description:

The server can be crashed by submitting a malformed string.

Technical description:

A client requests an endpoint on the identity server. As the protocol allows for multiple endpoint types to be supported, the client needs to supply the type. It does this by providing the `endpoint` parameter as part of the challenge/response request.

The server handles this request in `gosling.rs:send_response_impl()`. In this function the validity of the endpoint is checked:

```
self.client_requested_endpoint_valid = self.handshake.endpoint_supported(&self.requested_endpoint);
```

For the `IdentityServerHandshake` trait, this function is implemented in `ffi.rs:906-`.

The following line is problematic:

```
let endpoint0 = CString::new(endpoint).unwrap();
```

The `.unwrap()` method indicates that casting the `endpoint` parameter from `&str` to `Cstring` will succeed without a problem.

However, the function's documentation states that if a null byte is found, an error is thrown.

Since an adversary has control over the `endpoint` parameter, they can insert a null byte, which causes a panic because the error condition is not handled.

This problem also occurs in `build_endpoint_challenge():932`, `build_challenge_response:817` and `verify_challenge_response):974`.

Impact:

An adversary can crash the application if the `service_id` of the identity server is known.

Recommendation:

The cause of this issue is missing input validation. As a general rule, all external input should be regarded as untrusted. This means that as soon as possible after receiving data, validation should take place to determine if it can be processed by subsequent logic. If any data fails to meet to the requirements, the handshake should be considered failed.

3.2 GS-001 — DoS through OOM condition

Vulnerability ID: GS-001

Vulnerability type: Denial of service

Threat level: Elevated

Description:

The application running the protocol can be crashed through manipulation of the BSON document size header.

Technical description:

The protocol uses the BSON message format for communication between nodes using the protocol.

After a node starts up, a TOR hidden service is initialised (the identity server), and listens for handshake message to allow the client access to an endpoint server.

Ultimately, the communication between the server and client is done over a TCP stream, over which the BSON messages are exchanged.

The logic that deals with reading from this stream is implemented in `honk_rpc.rs`, particularly in `Session::read_message(...)`.

The logic starts with the expectation that the first 4 bytes contains the length of the BSON document. As such, it will enter the arm on line 535. Here, some error handling and 0 condition handling takes place, but if all is well, `self.remaining_byte_count` will contain the number of bytes that should be read from the TCP stream.

The `self.read_message()` is called again, and now enters the top arm defined on line 502.

Here is where we run into trouble. Line 503 contains the following instruction:

```
let mut buffer = vec![0u8; remaining]
```

The largest number `remaining` can contain is `2147483647 - 4`, due to the fact this number is derived from the `i32` datatype.

The `remaining` variable is under the control of an adversary, since it is sent from the client. If an adversary picks this large number, the server will try to allocate more than 2GiB of RAM. If the kernel cannot provide that much, the program will crash.

Impact:

An unauthenticated user with knowledge of the node's `service_id` could crash the node's process if the system has insufficient RAM available.

Recommendation:

- Since this logic is only used in the handshake protocols used to connect to the ident server and endpoint server, we recommend selecting a much lower limit for the BSON document size.

3.3 GS-002 — DoS through stale connections

Vulnerability ID: GS-002

Vulnerability type: Denial of service

Threat level: Moderate

Description:

The Gosling protocol does not close stale connections, resulting in resource exhaustion.

Technical description:

The Gosling protocol uses a polling scheme to update the state of incoming connections to the identity server. For every connection, the stream is polled for data and the state of the handshake is determined.

The current implementation does not drop stale connections, i.e. streams that are asked for data, but do not send any for a long time.

This behavior is implemented in `honk_rpc.rs::Session::read_message()` on lines 505 and 542:

```
Err(err) => if err.kind() == ErrorKind::WouldBlock || err.kind() == ErrorKind::TimedOut {
 Ok(None)
} else {
 bail!(err);
```

```
}
```

If no data is presented, this arm returns `Ok(None)` and the polling loop continues.

This behavior could be abused by an adversary creating a large number of connections to the identity server. Not only would this result in CPU overhead, since all these connections need to be covered in the loop, but if there are more than ~64k connections, no new connections can be made.

If the adversary drops the connection, the server detects this correctly and the connection is no longer polled.

Impact:

An adversary could prevent legitimate users from connecting and obtaining an endpoint `service_id`.

Recommendation:

- Implement state tracking logic that kills the connection if no bytes are present on the stream after reading the BSON length header; A BSON message should be sent atomically.
- Enforce handshake completion within a reasonable (short) amount of time, or kill the connection.

3.4 GS-003 — Protocol is vulnerable to MITM attack

Vulnerability ID: GS-003

Vulnerability type: Traffic interception

Threat level: Moderate

Description:

The protocol is vulnerable to eavesdropping and message manipulation by a third party that is able to position itself between the communicating hosts.

Technical description:

Consider the following scenario: Alice wants to set up a connection over TOR with Bob. To this end, Bob has published his service id `sid_Bob` on an untrusted platform. Charlie controls the untrusted platform and replaces `sid_Bob` with his own service id `sid_Charlie`. Alice, believing she is connecting to Bob will set up a connection with Charlie instead. Alice will complete the handshake with Charlie's identity server, and open a channel on Charlie's endpoint service. Charlie will now open a connection with Bob using `sid_Bob` and will end up with a channel on Bob's endpoint service.

At this point Charlie can eavesdrop on communication between Alice and Bob and even inject and or alter messages of his own to both Alice and Bob.

A proof of concept is included in the report's GitLab repository.

The reason this is possible is that there is no system in place to establish trust between the communicating parties.

Impact:

An adversary who is able to successfully insert themselves between the communicating parties is able to eavesdrop on and tamper with their data without their knowledge.

Recommendation:

To mitigate this issue we recommend implementing a provision that establishes trust after the communication channel has been established.

One could consider the socialist millionaire protocol (SMP) to achieve this. It requires Alice and Bob to share a secret through a *secure* side channel. The knowledge of this secret is verified by the protocol without transmitting any information about the secret itself.

The SMP is implemented in the Off-The-Record (OTR) secure communications protocol. One could decide on implementing OTR for secure communication and leave Gosling to negotiate a channel to set up a secure channel over.

Sources

- https://blog.securegroup.com/the-socialist-millionaire-protocol-how-secure-chat-prevents-mitm-attacks
- https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html
- https://robertheaton.com/otr1

3.5 GS-004 — Protocol is not end-to-end encrypted

Vulnerability ID: GS-004

Vulnerability type: Encryption

Threat level: Low

Description:

The link between the `tor` process and the application is not encrypted, allowing for plaintext extraction under certain circumstances.

Technical description:

To be able to communicate over the TOR network, the client spawns a `tor` process, which opens a socket the client connects to. The TOR client encrypts any data going out over the TOR network and decrypts incoming data. The communication between the `tor` process and the client application is necessarily unencrypted as so far control messages go. However, the Gosling protocol encrypt neither its handshake messages, nor the resulting communication channel. Below is a packet capture of a client requesting a channel on the server's endpoint service, which shows plaintext data.

Impact:

The impact is very low. An adversary would need to have access to the host the `tor` process runs on, on top of that they would need to have permissions to do packet dumps, which is typically granted only to root. If these conditions are met, an adversary could eavesdrop on communication, and learn of private information.

When combined with GS-006 (page 15), this issue could allow an adversary to exfiltrate plaintext data to a remote host.

Currently, the protocol spawns a local `tor` process. If in the future the protocol allows for the selection of a TOR daemon socket, this issue will gain more importance, as the attack surface would increase if a user has a dedicated host running as a TOR host somewhere on a local network. An adversary would then have more options for eavesdropping on communications, through exploitation of a network device for instance.

Recommendation:

Negotiate a shared symmetric key through deployment of ecdh-25519. This should happen once a connection is made and before any handshake messages have been exchanged. Note that though this protects against eavesdropping, it does not imply trust. In short, DH is vulnerable to active MITM, as presented in GS-003 (page 13).

3.6 GS-006 — Path interception vulnerability

Vulnerability ID: GS-006

Vulnerability type: Path interception

Threat level: Low

Description:

The method of spawning the `tor` process can be abused to spawn a malicious process.

Technical description:

A Gosling client starts a `tor` process, by using rust's `Command` API. (`tor_controller.rs`:132). The path to the executable is fetched by `system_tor()`. This function returns a relative path to the executable.

This causes Gosling to rely on the system's `$PATH` environment variable. This exposes Gosling to the risk of path interception, in which an adversary places a bogus `tor` executable in a path that is resolved before the path where `tor` actually resides.

The image below illustrates the problem.

An attacker with a code execution primitive as the user could alter the `$PATH` environment variable by editing the shell's `rc` file, to an inconspicuous path and place a malicious `tor` executable that notifies some remote host that `tor` is being launched, or perhaps even intercept plaintext communication.

For linux, code execution abilities are needed, since the file needs to be made executable, so a simple write primitive will not suffice. Windows, however, is notorious for its security issues. It is not unthinkable that an adversary using a spearphishing attack might be able to have a target install a malicious payload that puts a malicious `tor` somewhere on the filesystem where it is loaded before the legitimate executable.

Impact:

If an adversary can place a malicious `tor` executable in a custom path, it will be run in place of the legitimate `tor`. This could result in complete breakdown of the protection that TOR offers.

Recommendation:

- Use a suitable package to look up the path to the executable, and determine whether the path isroot/admin writable only. This gives some degree of protection.
- Alternatively, hash the executable and compare to a known list of hashes for `tor`. This is the most secure, but precludes the use of custom `tor` versions.

4 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

5 Conclusion

We discovered 1 High, 1 Elevated, 2 Moderate and 2 Low-severity issues during this penetration test.

We focussed on breaking the protocol by testing the solution proposed in the protocol spec. We found that by turning off the identity server it is no longer possible to detect the presence of the server on the network. As stated in the added note, it is still possible to detect and track a user if the endpoint server assigned to a peer runs when the application using the protocol is active.

We tested for protocol step bypass, and replay attacks. This is not possible due to the generation of random server and client cookies for each connection context. This forces the steps in the protocol to be followed sequentially.

Impersonation of service ids is not possible since strict possession checks are in place. The protocol utilizes TOR's authentication capability that allows access to an endpoint service only when the client possesses the authorized `x25519` keypair. This makes sure that only the designated client can access that endpoint. This prevents cyberstalking on the endpoint by clients other than the authorized one.

We tested the `FFI` boundary between the rust library and the C++ interface for possible memory corruption vulnerabilities; there do not seem to be any.

The protocol does suffer from an active MitM vulnerability, which allows an adversary that has successfully inserted herself between to communicating nodes to eavesdrop on the communication between the nodes and send messages on behalf of the connected parties.

Furthermore, though the traffic between `tor` processes is encrypted, communication between the application and the `tor` process is not. This would allow an adversary (if able), to either packet dump the plaintext, or otherwise intercept communications through a malicious `tor` process (see GS-006 (page 15)).

Lastly, there are some denial-of-service attack vectors present in GS-001 (page 11), GS-002 (page 12), and GS-005 (page 10), which would allow an adversary to terminate the application's process remotely. These issues have been given a relatively high risk factor, due to their ease of execution.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a 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 corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Daniel Attevelt (OSCP)	Daniel started programming at a young age, writing demos and games in assembly. He then began developing hardware interfaces and control software for home brewn hardware in C++ . Daniel studied Cognitive Neuroscience at Utrecht University but chose to follow a more practical path into software development. After completing a career in software development, he switched to the security field and is now using his skills to help protect society's information systems.
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.