Radically Open Security

Penetration Test Report

Open Technology Fund

V 1.1
Amsterdam, March 4th, 2024
Confidential

Document Properties

Client	Open Technology Fund
Title	Penetration Test Report
Targets	Briar Android and Desktop client app Briar Android Protocols and Cryptography
Version	1.1
Pentester	András Veres-Szentkirályi
Authors	András Veres-Szentkirályi, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	November 1st, 2023	András Veres-Szentkirályi	Initial draft
0.2	November 3rd, 2023	Marcus Bointon	Review
1.0	November 21st, 2023	Marcus Bointon	1.0
1.0.1	February 27th, 2024	András Veres-Szentkirályi	Retest update
1.1	March 4th, 2024	Marcus Bointon	1.1

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

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

Between September 21, 2023 and October 31, 2023, Radically Open Security B.V. carried out a penetration test for Open Technology 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 targets:

- Briar Android and Desktop client app
- Briar Android Protocols and Cryptography

The scoped services are broken down as follows:

- Scoping and preparation: 1 days
- Pentest of Android and Desktop apps (incl. reporting): 8 days
- Protocol and cryptographic analysis (incl reporting): 10 days
- Retest and fix verification: 2-4 days
- PM and review: 2 days
- Infra and admin: 1 days
- **Total effort: 24 - 26 days**

1.3 Project objectives

ROS will perform a penetration test and code audit of the Briar protocol along with its Android and desktop clients with OTF in order to assess their security. To do so ROS will assess the targets, and guide OTF 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 September 21, 2023 and October 31, 2023.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate and 5 Low-severity issues.

The Briar protocol design and app implementation show a well-thought-out architecture that has been gradually improved by acting upon the results of previous reviews.

The Android application does not make enough use of the hardening features available in more recent versions of Android OTF-001 (page 12); a lesser concern is that support for older Android versions in non-finding NF-006 (page 22) results in exposure to other vulnerabilities for those users.

The PSK key used for Wi-Fi app sharing has insufficient entropy OTF-002 (page 13), making brute-force attacks on it feasible. The Tor layer is missing hardening measures on some architectures, particularly 32-bit ones OTF-004 (page 15). Consideration of remote peers as lower-quality sources of trust makes some options unavailable to users in OTF-005 (page 17). The Tor TCP service listens on the local loopback interface instead of a Unix socket, exposing it to malicious apps OTF-003 (page 14). Briar offers repudiation and deniability features, but these are not promoted to the user in OTF-006 (page 18).

By exploiting these issues, an attacker might be able to cause a local denial-of-service, and could cause minor problems in certain vulnerable configurations.

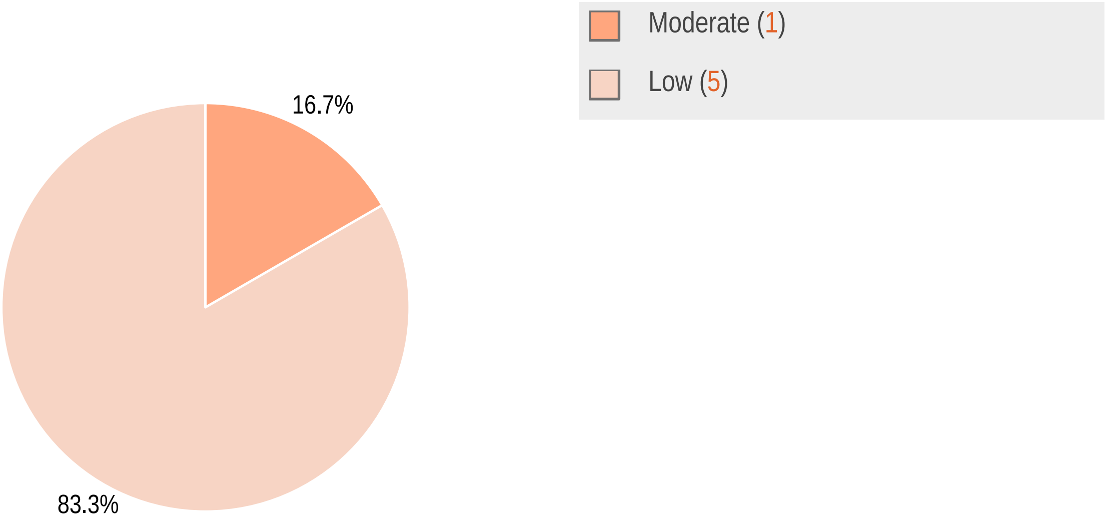
During the retest, we found that all the addressed issues were fixed properly, solving most of the issues. The PSK key got more entropy, the Tor layer received hardening measures, and the documentation got extended with sections regarding repudiation and deniability. These cover the most pressing and easy-to-fix vulnerabilities uncovered by the penetration test.

1.6 Summary of Findings

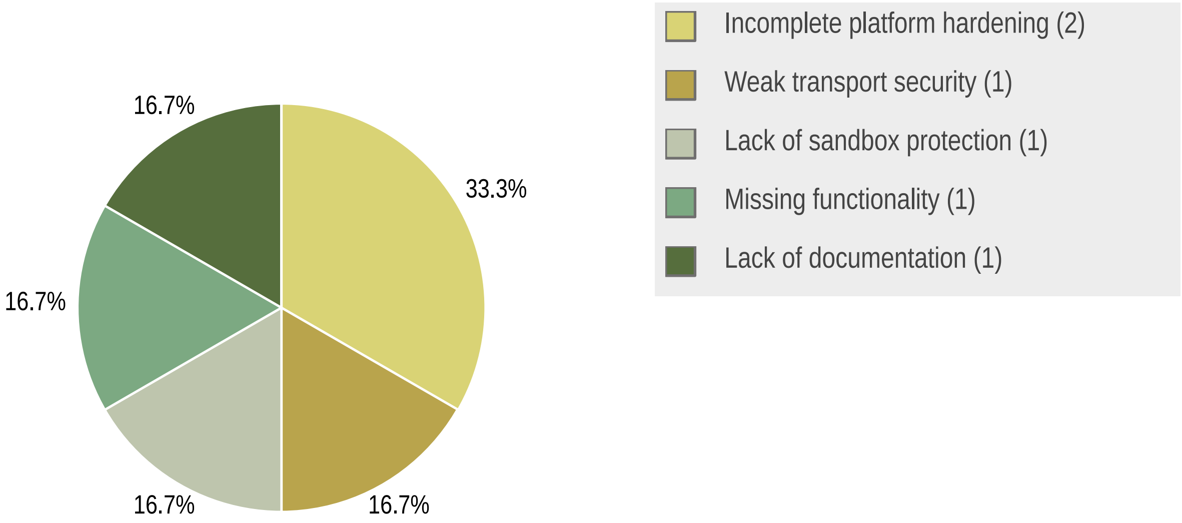
ID	Type	Description	Threat level
OTF-001	Incomplete Platform Hardening	The Android app has some protection against overlay attacks, but does not take full advantage of the protections offered by Android 12 and above.	Moderate
OTF-002	Weak Transport Security	The PSK used for sharing the application package has low entropy.	Low
OTF-003	Lack of Sandbox Protection	The application uses TCP services bound to the local loopback network interface. Since Android offers no sandbox for these services, other apps could abuse them.	Low
OTF-004	Incomplete Platform Hardening	Tor itself lacks fortified functions on 32-bit architectures, while obfs4-proxy and Snowflake lack both stack canaries and fortified functions on all architectures. While not a vulnerability in itself, it could make exploiting memory corruption issues easier for attackers.	Low
OTF-005	Missing Functionality	The application does not allow remote peers (i.e. not using the QR code) to verify each other's keypairs.	Low

OTF-006	Lack of Documentation	Parts of the protocol allow repudiation/deniability, yet the threat model documentation lacks statements regarding such features.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
OTF-001	Incomplete Platform Hardening	• Check the SDK level and call `Window.setHideOverlayWindows(true)` on SDK levels 31 and above to prevent non-system overlay windows from being drawn on top of the Briar window.
OTF-002	Weak Transport Security	• Double the PSK length to make brute-force attacks prohibitively expensive.
OTF-003	Lack of Sandbox Protection	• The app and the bundled Tor service should use Unix domain sockets instead of TCP, as sockets are tied to the file system and thus can enjoy the protection of file system permissions offered by the Android sandbox.
OTF-004	Incomplete Platform Hardening	• Enable stack canaries and fortified functions for all supported architectures. Use the option `-fstack-protector-all` to enable stack canaries and `-D_FORTIFY_SOURCE=2` to fortify common insecure libc functions.
OTF-005	Missing Functionality	• Briar should give its users the opportunity to improve trust, either by allowing a later use of the already existing QR code route, or introducing a solution similar to the Socialist Millionaire Protocol (SMP) used in Off-the-Record Messaging (OTR).
OTF-006	Lack of Documentation	• The threat models should include the application developers' stance on repudiation/deniability.

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 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- nmap – http://nmap.org

4 Findings

We have identified the following issues:

4.1 OTF-001 — Incomplete protection against Android overlay attacks

Vulnerability ID: OTF-001	**Status:** Resolved
Vulnerability type: Incomplete Platform Hardening	
Threat level: Moderate	

Description:

The Android app has some protection against overlay attacks, but does not take full advantage of the protections offered by Android 12 and above.

Impact:

Malicious apps with permission to draw over other apps could trick the user if they target Briar.

Technical description:

Briar has code to detect if an app has been granted permission to draw over other apps, a technique used by malware in overlay attacks. However, the app merely displays a message box with the text *"Another app is drawing on top of Briar. To protect your security, Briar will not respond to touches when another app is drawing on top."* and allows the user to select *"Allow these apps to draw on top"*.

Since the above reaction already considers an attacker that has the ability to draw over the app, it could use some social engineering tactics to make the user tap on the option to allow this to proceed while continuing to use Briar. This is acceptable as a fallback on Android 11 and below, however, since Android 12 introduced the ability to prevent non-system overlay windows from being drawn on top of windows that opt into this behavior, this could and should be used by Briar to improve security for users with newer Android versions.

Recommendation:

- Check the SDK level and call `Window.setHideOverlayWindows(true)` on SDK levels 31 and above to prevent non-system overlay windows from being drawn on top of the Briar window.

Update :

Resolved: the codebase of version 1.5.9 contains the recommended fix.

4.2 OTF-002 — Low-entropy PSK used for Wi-Fi sharing

Vulnerability ID: OTF-002	**Status:** Resolved
Vulnerability type: Weak Transport Security	
Threat level: Low	

Description:

The PSK used for sharing the application package has low entropy.

Impact:

Attackers physically close to their target while this feature is in use could mount a brute-force attack and inject malware into the package as others download it from each other for the first time.

Technical description:

The app offers itself to be shared over Wi-Fi so that others in close physical proximity to a user can install the app without using the internet and/or trusting an app store. Since this involves opening up a network service to unauthenticated users, it was considered as a potential attack surface and thus was tested as part of this audit.

On the physical/link layer, the app uses standard Android APIs to create a Wi-Fi hotspot and randomizes both the SSID postfix and the WPA-PSK passphrase using a proper CSPRNG, thus given a proper amount of entropy, we could have said that only those able to view the screen of the host device (displaying the SSID and PSK as a QR code and a human-readable string) could interfere with the confidentiality and integrity of the network traffic. The PSK generator uses the `getRandomString` method with a parameter of `8`, which specifies the length of a random string generated from a set of 32 characters (see `static final String chars`). Since a 32-character alphabet provides 5 bits of entropy per character, the whole PSK has only 40 bits of entropy.

The SSID contains 2 + 10 randomized characters generated the same way, resulting in 60 bits of entropy, which is big enough to make precomputing rainbow tables prohibitively expensive because of the storage space required. This leaves targeted real-time attacks using GPUs, for example `p3.2xlarge` instances on AWS EC2 with a single Tesla V100 running a current release of Hashcat would try around 850 kilohashes per second. For 40 bits of entropy, this would need about 15 days to crack a single PSK, which could be lowered in a linear fashion by using larger instances such as `p3.16xlarge` with 8 such GPUs, or launching more than one instance and sharding the keyspace between them.

While these values are not small, they are well within the realm of practicality, and as such are not strong enough for the intended purpose.

Recommendation:

- Double the PSK length to make brute-force attacks prohibitively expensive.

Update :

Resolved: the codebase of version 1.5.9 contains a fix that doubled the PSK length.

4.3 OTF-003 — TCP listeners not protected by Android sandbox

Vulnerability ID: OTF-003 **Status:** Not Retested

Vulnerability type: Lack of Sandbox Protection

Threat level: Low

Description:

The application uses TCP services bound to the local loopback network interface. Since Android offers no sandbox for these services, other apps could abuse them.

Impact:

Denial of service, potential further attacks.

Technical description:

The application starts the Tor service on TCP ports 59050 and 59051 (SOCKS and control port, respectively) bound to the local loopback network interface, and since TCP sockets are not sandboxed on Android for applications with the permission `android.permission.INTERNET` (a permission given to practically all apps nowadays; the OS does not even ask about it during installation). This could be abused in two ways: either by connecting to it from other apps in order to exhaust resources such as the maximum number of file descriptors, or if a malicious app starts before Briar does, it could start listening on these hardcoded ports and prevent Briar from accessing the Tor network, resulting in a limited (as Bluetooth and local Wi-Fi connections are not affected) denial of service.

Similar attacks affect hidden services, where the Briar app runs services bound to randomly numbered TCP ports on the local loopback network interface, which the local Tor daemon connects to when a hidden service receives an incoming connection. Because of the random choice of ports, occupying the port beforehand by a malicious app is not possible – there is not even a race condition, as the random port allocation is done by the OS itself. However, the resource exhaustion attacks could apply here as well if a malicious app occupies all available ports.

Another potential angle to exploit for these loopback-bound services would be a malicious app connecting to these services with the intent to actually send and receive data. In case of the SOCKS port, this could mean that the connections might use the same circuit as those used for fetching RSS feeds, which could be abused for identifying the exit node used for such a purpose. For hidden services ports, malicious apps might start to participate in protocols that use them, such as BHP, but these could also be accessed by anyone who knows the public key, which, by its very nature, is not considered a secret.

Recommendation:

- The app and the bundled Tor service should use Unix domain sockets instead of TCP, as sockets are tied to the file system and thus can enjoy the protection of file system permissions offered by the Android sandbox.

Update :

This was not ready for retesting.

4.4 OTF-004 — Lack of fortified functions and stack canaries in Tor and transport libraries

Vulnerability ID: OTF-004	**Status:** Resolved
Vulnerability type: Incomplete Platform Hardening	
Threat level: Low	

Description:

Tor itself lacks fortified functions on 32-bit architectures, while obfs4-proxy and Snowflake lack both stack canaries and fortified functions on all architectures. While not a vulnerability in itself, it could make exploiting memory corruption issues easier for attackers.

Impact:

Easier exploitation of memory corruption issues.

Technical description:

The Android application includes Tor and two transport libraries, obfs4-proxy and Snowflake, as native code linked as shared object (SO) files. Both 32- and 64-bit ARM and Intel architectures are supported, and since most of these codebases are in C, are prone to memory corruption vulnerabilities (stack overflow, heap overflow, off-by-one errors, dangling pointers, etc.). These libraries work directly with untrusted network traffic, so we checked for the presence of binary hardening measures. Testing was done on version 1.5.7 retrieved from F-Droid; the libraries all had the NX bit enabled.

We searched for stack canaries, an important countermeasure against stack overflow attacks, by looking at symbol names typically associated with the panic function called by a function epilogue encountering an invalid canary value caused by a stack overflow overwriting it. Only Tor itself (in `libtor.so`) has such a symbol.

```
lib % for i in */*; do echo '===' $i '==='; strings $i | grep -E '(__stack_chk_fail|
__intel_security_cookie)'; done
=== arm64-v8a/libobfs4proxy.so ===
=== arm64-v8a/libsnowflake.so ===
=== arm64-v8a/libtor.so ===
__stack_chk_fail
=== armeabi-v7a/libobfs4proxy.so ===
=== armeabi-v7a/libsnowflake.so ===
=== armeabi-v7a/libtor.so ===
__stack_chk_fail
=== x86/libobfs4proxy.so ===
=== x86/libsnowflake.so ===
=== x86/libtor.so ===
__stack_chk_fail
=== x86_64/libobfs4proxy.so ===
=== x86_64/libsnowflake.so ===
=== x86_64/libtor.so ===
__stack_chk_fail
```

We searched for fortified functions, a stopgap measure wrapping common easy-to-misuse libc functions in wrappers that perform security checks, by looking for symbol name suffixes typically associated with such wrappers. Only Tor itself (in `libtor.so`) has such a symbol, but that is only included on 64-bit ARM (`arm64-v8a`) and Intel (`x86_64`) architectures, and not on 32-bit ones.

```
lib % for i in */*; do echo '===' $i '==='; strings $i | grep '_chk$'; done
=== arm64-v8a/libobfs4proxy.so ===
=== arm64-v8a/libsnowflake.so ===
=== arm64-v8a/libtor.so ===
__FD_SET_chk
__read_chk
__strlcpy_chk
__FD_CLR_chk
__FD_ISSET_chk
__memcpy_chk
__strlen_chk
__strchr_chk
__strlcat_chk
__strrchr_chk
__memset_chk
=== armeabi-v7a/libobfs4proxy.so ===
=== armeabi-v7a/libsnowflake.so ===
=== armeabi-v7a/libtor.so ===
=== x86/libobfs4proxy.so ===
=== x86/libsnowflake.so ===
=== x86/libtor.so ===
=== x86_64/libobfs4proxy.so ===
=== x86_64/libsnowflake.so ===
=== x86_64/libtor.so ===
__strlen_chk
__FD_SET_chk
__FD_CLR_chk
__FD_ISSET_chk
__strlcpy_chk
__memcpy_chk
__memset_chk
__read_chk
__strchr_chk
__strlcat_chk
__strrchr_chk
```

Recommendation:

- Enable stack canaries and fortified functions for all supported architectures. Use the option `-fstack-protector-all` to enable stack canaries and `-D_FORTIFY_SOURCE=2` to fortify common insecure libc functions.

Update :

Resolved: the published APK of version 1.5.9 updated libtor on 32-bit platforms to include the recommended hardening measures, and the case of transport libraries was clarified, the latter needing no such additional protection.

4.5 OTF-005 — Lack of remote verification

Vulnerability ID: OTF-005 **Status:** Not Retested

Vulnerability type: Missing Functionality

Threat level: Low

Description:

The application does not allow remote peers (i.e. not using the QR code) to verify each other's keypairs.

Impact:

A missed opportunity for security-conscious users to raise their security level.

Technical description:

When using BQP, the application trusts that users are reading each other's QR codes with their cameras, thus the chance of an attacker swapping the public keys for their own is minimal. This is even reflected in the UI, indicating a lower level of trust for peers added remotely, either by introduction or using `briar://` links.

However, the app does not make it possible for users of the latter two types to improve this situation, even if they could later meet or have alternate means of verifying their trust. This results in peers getting "stuck" at a lower level of trust if they cannot meet up and/or use the QR code feature upon establishing their contact on Briar.

In their March 2023 paper, Yuanming Song also identified this as a problem in section 9.2 *Active man-in-the-middle attack*, noting that *"it could still be a good idea to at least provide some form of out-of-band verifications and let the users decide"*.

Recommendation:

- Briar should give its users the opportunity to improve trust, either by allowing a later use of the already existing QR code route, or introducing a solution similar to the Socialist Millionaire Protocol (SMP) used in Off-the-Record Messaging (OTR).

Update :

This was not ready for retesting.

4.6 OTF-006 — Lack of documentation regarding repudiation

Vulnerability ID: OTF-006	**Status:** Resolved
Vulnerability type: Lack of Documentation	
Threat level: Low	

Description:

Parts of the protocol allow repudiation/deniability, yet the threat model documentation lacks statements regarding such features.

Impact:

Users are not made aware of repudiation/deniability features in the application.

Technical description:

The BHP's design provides repudiation and deniability, where contacts can plausibly deny having had an exchange even if the transcript is saved by an adversary. However, the same is not true for forums, blog posts, and group chats, as the messages are signed.

Since the application already has a detailed and publicly available threat model, these features should be mentioned to help security-conscious users achieve their aims.

Recommendation:

- The threat models should include the application developers' stance on repudiation/deniability.

Update :

Resolved: the documentation has been updated with an explicit declaration of threat models and details of the guarantees of the app regarding repudiation/deniability.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

5.1 NF-001 — Analysis of the modified BTP handshake

In their March 2023 paper, Yuanming Song identified the lack of forward security in the handshake part of the Bramble Handshake Protocol. Although the impact of the vulnerability is limited due to the additional protective layer provided by Tor being used as a transport layer below the protocol, Briar 1.5.3 replaced the handshake with a more secure version. One of the first steps of this audit was to check whether this change had actually solved the problem.

As it can be seen in commit `462f57c9669e80509612203da0563c11da5770a9`, the handshake was changed in a way that section 5.2 of the aforementioned paper suggested – incorporating a DH involving the ephemeral keypairs, and removing the DH that involved the long-term handshake keypairs. The resulting protocol is similar to the former 3DH approach of Signal documented at https://signal.org/blog/simplifying-otr-deniability/ and has similarities with the (as of October 2023) current Signal implementation of X3DH. Although the documentation of the latter has a section regarding signatures (https://signal.org/docs/specifications/x3dh/#signatures), comparing the two protocols and the differences in context reveals that a signature is not needed in Briar because the protocol is fully synchronous, and the resulting key material is only ever used for encryption *after* a mutual exchange of proofs has succeeded.

5.2 NF-002 — Analysis of HTML sanitization

Since the application works so hard to make local network traffic analysis difficult by sending all internet-bound requests (including DNS) over the Tor network, we analyzed local HTML sanitization. The first step was to identify potentially vulnerable sinks that would interpret HTML tags. The Briar app does not use WebViews, however, its text widgets could parse and display HTML if such a `Spanned` instance was passed.

Fortunately, the codebase only creates these in a dedicated static function called `getSpanned` and only the class `BlogPostViewHolder` ever invokes it. Even then, it was only used to parse the body of a blog post. However, blog posts have their bodies sanitized using the JSoup library at least once in the `BaseViewModel` class. In case of blog posts fetched from an RSS feed, another round of the same sanitization occurs *before* storing the post into the database. While this may seem redundant, this could have defense-in-depth benefits, and lowers storage and (in case of reblogging) network utilization.

The actual sanitization in the JSoup library defines a reasonable list of allowed tags (headings from 1 to 6 and the basic safelist of JSoup) and strips all others. The library itself has had CVEs related to this cleansing but at the time of this report, none of them affect version 1.15.3 which the Briar app depends on. Although images and scripts would not have been interpreted by the widgets used, this level of protection seems like a wise defense-in-depth measure for an application with such a threat model.

5.3 NF-003 — Analysis of random number generators

Since the application implements its own cryptographic protocol, it generates random numbers in multiple places. Since the effective entropy of such values have an important role in mitigating brute force attacks, the relevant code paths were analyzed in this regard as well.

Since Briar runs on Android as well, some people might remember that back in 2013 the Android platform had issues with weak random number generators, affecting API levels 18 and below. However, the app declares in its manifest that it requires at least API level 21 to run, so this issue does not apply. Regardless of this, the app installs its own `SecureRandomProvider` if it detects a Linux kernel (e.g. in the Briar Android app and Briar Desktop for Linux), which uses `/dev/urandom` as a source of entropy, which can be considered secure enough for this purpose.

All modules of the app use `SecureRandom` to generate cryptographic keys and nonces, which is best practice in Java cryptography.

5.4 NF-004 — Handling of messages that are too long

In their March 2023 paper, Yuanming Song identified improper handling of incoming messages that are too long for the implementation to handle. Briar 1.4.22 modified the parsing code to behave in a more robust way, ignoring such messages instead of crashing the application. One of the first steps of this audit was checking whether this change solved the problem.

As it can be seen in commit `c92ee0458e6c4c136a0f341e95c2eb7c1ecf576a`, the parsing code was changed in a way that section 8.2 of the aforementioned paper suggested – checking whether the payload length is larger than the maximum allowed message length, and if so, throwing a `FormatException` instead of the previously thrown `IllegalArgumentException`. This makes the behavior consistent with other sanity checks such as that of the header length just before the new check, and that of the timestamp immediately after it.

5.5 NF-005 — Tor configuration

As described in OTF-004 (page 15), the application starts the Tor service on TCP ports 59050 and 59051 (SOCKS and control port, respectively) bound to the local loopback network interface, and since TCP sockets are not sandboxed on Android for applications with the permission `android.permission.INTERNET` (a permission given to practically all apps nowadays; the OS does not even ask about it during installation). This means that the Tor configuration needs to be checked to see whether the control port could be abused by a malicious app.

The Tor configuration contains the line `CookieAuthentication 1` which results in Tor creating a cookie file within the `app_tor` subdirectory of the Android sandbox. As it can be seen below, this was already owned by the app's UID, and access to group members and others was disabled. So only apps run with this UID can control the Tor process through this socket.

```
emu64xa:/data/data/org.briarproject.briar.android # ls -l
total 64
drwx------ 2 u0_a157 u0_a157       4096 2023-10-18 13:18 app_db
drwxrwx--x 2 u0_a157 u0_a157       4096 2023-10-18 13:18 app_dev-reports
drwx------ 2 u0_a157 u0_a157       4096 2023-10-18 13:18 app_key
drwx------ 4 u0_a157 u0_a157       4096 2023-10-18 13:18 app_mailbox
drwx------ 3 u0_a157 u0_a157       4096 2023-10-18 13:18 app_tor
```

5.6 NF-006 — Janus vulnerability on Android

The Android app defined the minimum SDK level as 21 (Android 5.0), and Android versions below API level 27 (Android 8.1) are susceptible to attacks where the APK file is modified in a way that the system still accepts the package as if its signature was still valid. This is called the Janus vulnerability, and can be found easily using automated tools which tout such vulnerabilities, giving them more weight than their severity deserves.

Since the application uses both v1 and v2 APK signature schemes, the Janus vulnerability cannot be exploited on Android 8 devices, and this way, the app has done everything it can in this regard. The only way to avoid Android 5-7 users being vulnerable to Janus exploits would be to raise the minSDK level and/or remove v1 signatures, which would simply make it impossible for such users to install the app in the first place. Ultimately, it is a decision left to the app developers knowing their audience and making the decision regarding this tradeoff.

5.7 NF-007 — Android hardening

The app uses many ways to harden the attack surfaces as a result of using Android as a platform. One of these is the registered URL scheme `briar://` and its associated share target for `text/plain` content. In this case, a strict regular expression validates the payload, and the same expression is reused later to parse it, preventing attacks based on parser differentials.

The APK is signed using its SHA-256 digest, which can be considered secure for this purpose. The built-in backup functionality of the Android system is instructed to exclude every file within the sandbox, and then some files such as the database are excluded explicitly as well for good measure. `FLAG_SECURE` is enabled for all windows, making it harder for screen capturing malware to gain access to sensitive information displayed on the app screens.

On Android versions and devices with support for Android KeyStore, the database key is derived using an HMAC operation performed on the output of the PBKDF2 function; the resulting digest is used as a database key. This way, hardware-backed security modules could be involved, which would make key extraction and brute force attacks harder.

The resource `res/xml/network_security_config.xml` opts out of the default Android security policy of disallowing cleartext traffic for all subdomains of the top level domain `onion`. This is sensible since Tor itself adds a layer of encryption for hidden services served over such hostnames, and BTP offers a similar level of security without using TLS.

5.8 NF-008 — Analysis of TCP/IP layer of Wi-Fi sharing

Although we found that the PSK has low entropy in OTF-002 (page 13), we also audited the TCP/IP layer. The only two services available to Wi-Fi client stations are DNS provided by the Android OS, and an HTTP service on a high port (9999). This service is run using NanoHTTPd, and it only serves two files: the static HTML page that describes what is being served, and the APK file itself, which delivered a bit-perfect copy of the installed package.

Although NanoHTTPd has had some CVEs, none of them affected the usage in the Briar Android app. No untrusted input is processed by the application, and both responses (the HTML and the APK) are generated correctly regardless of what the HTTP client asked.

5.9 NF-009 — Cross-protocol attacks

Although the Briar apps use multiple protocols which attackers could target, thorough analysis of cryptographic signing using *Semgrep* showed that all such signings are properly namespaced, so signatures can't be reused across protocols. BHP only exchanges a single hash, which can only be computed correctly if both users have access to the keypairs they claimed to have access to.

All other protocols handled by the application rely on MACs and signatures matching correctly, otherwise operations are aborted immediately.

6 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.
- **Perform a similar assessment on dependencies that haven't had one so far**

  Since this assessment treated dependencies as being out of scope, we recommend checking whether they have had any recent security assessments. To ensure supply chain security, those without proper assessments should receive one, prioritizing components that interact directly with untrusted content.
- **Set up automated code analysis for the issues uncovered and potential future ones**

  Automated code analysis and unit tests could make sure that the issues uncovered in this report are not reintroduced with future merges or developments. This could also include those described in non-findings to keep those areas in check, such as taint analysis proving that all signatures are namespaced, HTML displayed on UI coming from untrusted sources is sanitized, etc.

7 Conclusion

We discovered 1 Moderate and 5 Low-severity issues during this penetration test.

The Briar protocol design and app implementation show a well-thought-out architecture that has been gradually improved by acting upon the results of previous reviews such as the Cure53 assessment and the March 2023 paper of Yuanming Song.

We did not find any serious issues in the apps or underlying protocol. Even though we confirmed that a previously known issue was still present, and we identified some new ones, none of these result in direct danger to most users. Exploiting these issues would require a significant amount of investment, and in most cases, either depends on users making opsec mistakes and/or discovering vulnerabilities in dependencies.

Fixing these issues would make Briar even more resilient and hopefully the next such assessment will only reveal issues of similar or lower severity as a result of further refinements to the layers of effective hardening measures already in place.

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.

The retest confirmed that addressed issues were fixed properly, and as a result, the most pressing and easy-to-fix subset of vulnerabilities were resolved.

Appendix 1 Testing team

András Veres-Szentkirályi	András Veres-Szentkirályi has CISSP, OSCP, GWAPT, and SISE certifications in addition to an MSc. in Computer Engineering, and has been working in IT Security since 2009.
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.

Front page image by Slava (https://secure.flickr.com/photos/slava/496607907/), "Mango HaX0ring",
Image styling by Patricia Piolon, https://creativecommons.org/licenses/by-sa/2.0/legalcode.