Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Tor Android VPN & Arti Interface 06.2025

Cure53, Dr.-Ing. M. Heiderich, Dr. A. Pirker, Dr. D. Bleichenbacher, MSc. D. Weißer,
Dr. N. Kobeissi, J. Ginesin

Index

- Introduction
- Scope
- Identified Vulnerabilities
  - TTP-04-007 WP2: Targeted DNS DoS by saturating allocated IPv4 space (Low)
  - TTP-04-008 WP2: DNS server DoS via resolution request flooding (Medium)
  - TTP-04-009 WP2: DoS of Tor VPN app via IPv4 allocation flooding (Medium)
  - TTP-04-011 WP2: Absent TCP parsing checks (Medium)
- Miscellaneous Issues
  - TTP-04-001 WP1: Android application supports outdated SDK versions (Low)
  - TTP-04-002 WP1/2: Unencrypted shared preferences & configuration files
  - (Low) TTP-04-003 WP1: Insufficient bridge line validation (Low)
  - TTP-04-004 WP1: Lack of certificate pinning in bridge HTTPS request
  - (Medium) TTP-04-005 WP1: Insecure randomness for bridge selection (Low)
  - TTP-04-006 WP2: DNS resolution may re-occur for same domain name (Low)
  - TTP-04-010 WP2: DoS via zero-size packet parsing (Medium)
  - TTP-04-012 WP2: TURN client DoS due to endless loop by rogue server (Medium)
  - TTP-04-013 WP1: Public IP resolution hardening considerations (Info)
  - TTP-04-014 WP2: DNS poisoning via lack of cache expiration (Low)
  - TTP-04-015 WP2: Apple FFI vulnerable to OOB memory read (High)
  - TTP-04-016 WP2: Onionmasq-mobile FFI vulnerable to double fd close
  - (Low) TTP-04-017 WP2: Tor VPN Android app lacks root detection (Low)
  - TTP-04-018 WP2: TCP socket listeners could lead to DoS or hijacking
- (Low) Conclusions

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

"We believe everyone should be able to explore the internet with privacy. We are the Tor Project, a 501(c)(3) US nonprofit. We advance human rights and defend your privacy online through free software and open networks."

From https://www.torproject.org/

This document details the verdict of a penetration test and source code audit against the Tor VPN for Android, as well as the Tor Tunnel Interface for Arti.

The summer 2025 engagement was requested by The Tor Project in October 2024 and duly performed across a two-week period (CW25-CW26). Twentyfour work days were allocated by the client for ample coverage levels.

Two separate work packages (WPs) were created for this procedure, entitled *WP1: Crystal box pen.-tests & source code audits against Tor VPN for Android* and *WP2: Crystal-box pen.-tests & source code audits against Tunnel Interface for Arti*, respectively.

The crystal-box tasks were conducted by a team comprising six security specialists, who were granted access to sources and other beneficial materials in advance. Preliminary actions were undertaken shortly before the test commencement (CW24 June 2025) in order to encourage a hindrance-free examination period.

The communications method of choice was Signal, whereby a dedicated and private group was established for direct contact between the Tor and Cure53 teams. The collaborative efforts were seamless, contributing to an optimal working setup. The testers rarely asked questions regarding the scope or aims, since these were clear from the outset. No delays were encountered at any stage, testament to the groundwork laid by all participating personnel.

Live reporting was deemed unnecessary for this iteration, although the audit team ensured to keep the Tor maintainers in the loop, furnishing progress updates and pertinent insights when appropriate. The Cure53 consultants detected eighteen findings after achieving thorough reviews across the system in focus. Positively, only four of those were classified as

exploitable security vulnerabilities, while the remaining fourteen pertained to best practice suggestions and minor hardening initiatives.

The findings from this assessment confirm that the core Tor integration is sufficiently robust, demonstrating no fundamental flaws in its tunnel establishment or traffic routing. The modular architecture also effectively confines potential compromises.

cure53
Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Nevertheless, Cure53 identified two primary areas of concern: absent input data validations and vulnerabilities related to DNS resolver/Denial-of-Service attack vectors. The majority of findings originate from these two aspects; as such, Cure53 highly recommends prioritizing the remediation of these tickets, alongside a comprehensive investigation to address the underlying root causes.

In summary, the assessment revealed that the Tor Android VPN and Tunnel Arti interface require specific enhancements and general modifications, despite the generally commendable security posture.

The report will now shed light on the *Scope* and testing setup, as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

To close, the *Conclusions* chapter summarizes Cure53's estimation of the components from a security perspective, pinpointing key areas for improvement and other beneficial aspects for the client to consider.

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration-tests & code audits against Tor Android VPN & Tunnel Arti interface** ◦ **WP1:** Crystal-box pen.-tests & source code audits against Tor VPN for Android ▪ **Source code:**
  - https://gitlab.torproject.org/tpo/applications/vpn/-/tree/ef9f87e80ab2cdae481db7dadfeafe2ab89f424a
  - ▪ **Branch:**
    - main
  - ▪ **Commit ID:**
    - ef9f87e80ab2cdae481db7dadfeafe2ab89f424a
  - ◦ **WP2:** Crystal-box pen.-tests & source code audits against Tunnel Interface for Arti ▪ **Source code:**
    - https://gitlab.torproject.org/tpo/core/onionmasq/-/tree/Tor Android VPN & Tunnel Arti Interface d80a33be0bd12ea2467f7342ef4133f6a8f28d89
  - ▪ **Branch:**
    - main
  - ▪ **Commit ID:**
    - d80a33be0bd12ea2467f7342ef4133f6a8f28d89

- **Test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53**

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *TTP-04-001*) to facilitate any future follow-up correspondence.

TTP-04-007 WP2: Targeted DNS DoS by saturating allocated IPv4 space *(Low)*

The Tor VPN client rolls its own DNS client to ensure that DNS queries cannot be leaked between different apps. Upon resolving a DNS query, the DNS client allocates a unique local IP address for the client-specific query (as stratified via an assigned *isolation_token)* by calling the *make_for_v4* function. This function performs one of two actions. Firstly, it either generates a new IPv4 address, assigns it into the *map_v4* hashmap, and returns it. Secondly, it returns *None* if the *map_v4* object offers 2^{24} - 2 items (corresponding to the 10.0.0.0/8 range).

The *isolation_token* is derived from the *app_id*, as assigned directly by the operating system. However, its uniqueness is not necessarily guaranteed. As mentioned in the documentation, applications signed by the same vendor may indeed share an *app_id* in certain situations.[1] In this case, one of the apps sharing the *app_id* may repeatedly invoke *make_for_v4 and* intentionally saturate the *10.0.0.0/8* address space. This will cause *make_for_v4* to only return *None* for any future DNS query for a given *isolation_token*, effectively denying service for all apps with the same *isolation_token*.

The excerpt below highlights the underlying flaw, whereby 2^{24} - 2 entries in *map_v4* will force the function to return *None*, effectively denying the service.

Affected file:
onionmasq/crates/onion-tunnel/src/dns.rs

Affected code:

```
fn make_for_v4<S: TunnelScaffolding>(
 &mut self,
 #[allow(unused)] arti: Option<&TorClient<OnionTunnelArtiRuntime<S>>>,  mut hostname: String,0
 ) -> Option<Ipv4Address> {
 Self::canonicalize_hostname(&mut hostname);
 if let Some(a) = self.map_v4.get_by_right(&hostname) {
 return Some(*a);
 }
```

[1] https://developer.android.com/guide/topics/manifest/manifest-element.html#uid

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
 #[cfg(feature = "udp-tunnel")]
 if let Some(arti) = arti {
 self.begin_resolve(arti.clone(), hostname.clone());  }

 // Make a random IPv4 address in the range 10.0.0.0/8.
 let mut ret = Ipv4Address([10, 0, 0, 0]);

 // 16777214 = 2^24 - 2, the total amount of IPv4 addresses in  10.0.0.0/8.
 if self.map_v4.len() == 16777214 {
 return None;
 }
```

To mitigate this vulnerability, Cure53 advises ensuring that multiple applications cannot hold the same *isolation_token*, which can be achieved by sanitizing the *app_id* received from the operating system, rather than simply accepting it as is. Moreover, connections should be identified with a tuple including the *app_id* and IP endpoints (as well as the app name if possible), which will offer enhanced uniqueness guarantees. Lastly, the Tor team should consider employing battle-tested in-process DNS libraries, as opposed to home-rolling a DNS server.[2]

TTP-04-008 WP2: DNS server DoS via resolution request flooding *(Medium)*

Testing confirmed that Onionmasq's built-in DNS resolver lacks rate limiting and fails to clear cached entries from *resolver_map*, creating a memory exhaustion vulnerability. An attacker can flood the system with DNS queries for unique hostnames, each adding a permanent entry to the cache. Since entries are never evicted or expired, memory usage will expand unbounded until the system's resources are exhausted.

The defect is compounded by the fact that both the main DNS resolver and the UDP tunnel implementation maintain separate DNS caches without size limits or cleanup mechanisms. This dual-caching architecture doubles the memory consumption, as each unique hostname occupies space in both caches indefinitely.

Affected file #1:
onionmasq/crates/onion-tunnel/src/dns.rs

Affected code #1:

```
fn resolve_result(&mut self, hostname: &str) -> Option<&ResolverResult> {  while let
Ok((hostname, result)) = self.resolver_recv.try_recv() {  self.resolver_map.insert(hostname,
result);
 }
 self.resolver_map.get(hostname)
}
```

[2] https://github.com/hickory-dns/hickory-dns

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file #2:
onionmasq/crates/onion-tunnel/src/udp_tunnel/allocation.rs

Affected code #2:

```
resolver_memo.insert(resolver_key, resolver_result);
```

To mitigate this vulnerability, Cure53 advises implementing cache size limits with LRU eviction when the cache reaches maximum capacity. Periodic cleanup tasks should also be performed to remove old entries based on the TTL or previous access time. Moreover, rate limiting should be enforced to prevent rapid cache filling. Lastly, cache management should be coordinated between the main DNS resolver and UDP tunnel to assert global memory limits across both implementations.

TTP-04-009 WP2: DoS of Tor VPN app via IPv4 allocation flooding *(Medium)*

While investigating the pitfall detailed in ticket TTP-04-007, Cure53 confirmed that it can be abused to mount DNS DoS attacks on both apps sharing the same *app_id* and the entire Tor VPN app. In fact, the DNS client at most assigns 16.777.213 IPv4 addresses for DNS queries originating from apps using a single *app_id*. Since one IP address consumes 4 bytes

and a hostname also contains more than a single byte, the memory consumed by saturating the IPv4 address space allocation will certainly exceed 66MB. Pertinently, the DNS client will never manage these allocations. Furthermore, since the DNS client stores the IP address in a hashmap and the hostname, an average hostname of 10 characters will lead to a total memory consumption of 660MB for a single app if the IPv4 space is saturated. This scenario reflects only a single *app_id* and scales up accordingly to multiple *app_id* values.

If an adversary is able to flood the DNS client with DNS requests offering lengthy hostnames or via multiple attacker apps, the process memory will be consumed, leading to a crash. This activity results in a DoS situation for the Tor VPN app, which differs from the situation outlined in ticket TTP-04-007, whereby the attacker denies access to all apps.

The excerpt below verifies the detected drawback. The *DnsCookies* implementation populates the hashmap *cookies* with new entries if the *isolation_key* remains unknown. Furthermore, the *IsolatedCookies* implementation adds new entries to the hashmap *map_v4* without removing items.

Affected file:
onionmasq/crates/onion-tunnel/src/dns.rs

Affected code:
```
impl IsolatedCookies {
 [...]
 fn make_for_v4<S: TunnelScaffolding>(
```


Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
 &mut self,
 #[allow(unused)] arti:
Option<&TorClient<OnionTunnelArtiRuntime<S>>>,
 mut hostname: String,
 ) -> Option<Ipv4Address> {
 [...]
 // Make a random IPv4 address in the range 10.0.0.0/8.  let mut ret =
Ipv4Address([10, 0, 0, 0]);
 [...]
 let _ = self.map_v4.insert_no_overwrite(ret, hostname);  Some(ret)
 }
 [...]
}
[...]
impl DnsCookies {
 [...]
 fn make_for_v4<S: TunnelScaffolding>(
 &mut self,
 arti: Option<&TorClient<OnionTunnelArtiRuntime<S>>>,
 isolation_key: Option<u64>,
 hostname: String,
```

```
) -> Option<Ipv4Address> {
self.cookies
.entry(isolation_key)
.or_insert_with(IsolatedCookies::new)
.make_for_v4(arti, hostname)
}
[...]
}
```

Mitigating this vulnerability is challenging, as any imposed constraint on the number of DNS entries of a single *app_id* could induce DoS via exhaustion. Nonetheless, Cure53 recommends significantly limiting the number of IPv4 addresses and introducing a FIFO scheme in the event that the IPv4 address space has been exhausted. The Tor team should also consider applying a timeout for these entries that automatically removes them if they are not actively in use.

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

TTP-04-011 WP2: Absent TCP parsing checks *(Medium)*

Cure53's review of the Onionmasq codebase revealed that the TCP packet parser lacks several critical validation checks required by RFC standards. The parser accepts TCP packets without verifying checksums, sequence numbers, window sizes, or implementing SYN flood protections. These missing validations allow malformed or malicious packets to propagate through the tunnel system.

This could enable attackers to flood the system with invalid TCP packets that consume processing resources, create invalid socket states, or cause upstream applications to misbehave. While the immediate parser may not crash, invalid packets reaching upstream components could trigger undefined behaviors, memory exhaustion from unchecked socket creation, or degraded performance from malformed data processing.

Affected file:
onionmasq/crates/onion-tunnel/src/parser.rs

Affected code:
```
fn handle_tcp_packet(...) -> Option<ParseResult> {
```

```
// No checksum validation
// No sequence number validation
// No window size validation
// No SYN flood protection

if tcp.flags.syn && !tcp.flags.ack {
return self.maybe_make_tcp_socket(...)
}
// Direct socket creation without rate limiting
}
```

The following TCP validations are absent from the current construct:

- TCP checksum verification to detect corrupted packets.
- Sequence number validation to prevent sequence number attacks.
- Window size sanity checks to prevent window-based attacks.
- SYN cookie implementation or rate limiting for SYN flood protection.
- TCP flags combination validation (illegal flag combinations).
- Maximum segment size validation.

To mitigate this vulnerability, Cure53 suggests implementing comprehensive TCP validations that conform to the RFC 793 requirements.[3]

Specifically, this includes integrating checksum verification; validating sequence numbers against connection state; incorporating SYN cookies or rate limiting for new connections;

[3] https://www.rfc-editor.org/rfc/rfc793.html

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

and rejecting packets with invalid flag combinations or parameters. The dev team should also consider adopting established TCP parsing libraries that handle these validations correctly, rather than applying custom parsing logic.

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

TTP-04-001 WP1: Android application supports outdated SDK versions *(Low)*

While dynamically testing the Tor VPN Android app, Cure53 noted that the outdated SDK version 24 is supported. This was discovered by extracting the manifest from the app installed on an Android phone using the *Drozer* tool.[4] This introduces a security risk to users with mobile phones operating older versions of Android, as the outdated SDK versions no longer receive security patches or updates that mitigate known vulnerabilities.

The excerpt below presents the extracted manifest of the Tor VPN app, confirming that Android SDK version 24 is supported.

Extracted manifest:

```
<manifest versionCode="11"
 versionName="0.9.0"
 compileSdkVersion="35"
 compileSdkVersionCodename="15"
 package="org.torproject.vpn"
 platformBuildVersionCode="35"
 platformBuildVersionName="15">
 <uses-sdk minSdkVersion="24"
 targetSdkVersion="35">
[...]
```

To mitigate this issue, Cure53 advises increasing the minimum supported SDK version to 33, which will ensure that the Tor VPN Android app only runs on devices that receive regular security updates.

[4] https://github.com/ReversecLabs/drozer/tree/develop

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

TTP-04-002 WP1/2: Unencrypted shared preferences & configuration files *(Low)*

The Tor VPN Android app persists its configuration to files stored locally on the device app's folder. The configuration files include general settings, such as the specific apps designated for Tor VPN protection and whether the protection should initiate on boot, as well as network information such as the identity and addresses of relay nodes. While investigating these files, Cure53 discovered that the Tor VPN Android app stores them in plaintext on the Android device.

In the unlikely event that an attacker is able to access and manipulate these files, e.g., on a rooted phone with physical access, the files can be altered without the app noticing. This severely damages user data integrity and introduces myriad other implications. For example, the attacker may disable the flag to start the VPN on boot or amend the list of protected apps.

The excerpt below demonstrates the plaintext storage of the *tor-vpn.xml* file.

Extracted *tor-vpn.xml* file:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
 <string name="bridge_type">Obfs4</string>
 <boolean name="use_bridge" value="false" />
 <set name="protected_apps">
 <string>com.android.chrome</string>
 </set>
 <string
name="cached_apps">[{&amp;quot;isRoutingEnabled&amp;quot;[...]]</string>  <boolean
name="should_show_guide" value="false" />
 <boolean name="protect_all_apps" value="false" />
 <set name="exit_node_countries">
 <string>DE</string>
 [...]
 </set>
 <boolean name="start_on_boot" value="false" />
</map>
```

The excerpt below underscores the plaintext storage of relay information.

Extracted relay info from *files/arti-data/state/guards.json*:

```
{"default": { "guards": [ { "id": { "ed25519":
"wy8MgXzblrJNRY7sCsRlQ0nf7QACwS00ozIDSEjZTSg", "rsa":
"037b6c60dad4df32fda29bf458d5c8c816aa8f3e" }, "orports":
[ "91.205.230.113:443" ], [...]
```


Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 advises generating encryption keys in the Android Keystore

and encrypting all app files before persisting them, enforcing one encryption key per file.

TTP-04-003 WP1: Insufficient bridge line validation *(Low)*

Testing confirmed that bridge line validation in the Tor VPN application relies solely on non blank input checks. This minimal validation approach fails to verify the actual format and validity of Tor bridge lines, which follow specific syntax patterns for different transport types.

As such, an attacker or malicious user can input malformed bridge lines that cause application crashes, connection failures, or potentially inject unexpected configurations into the Tor client. Invalid bridge configurations may also leak connection attempt information via error messages.

Affected file:
vpn/app/src/main/java/org/torproject/vpn/ui/bridgesettings/BridgeLinesFragment.kt

Affected code:

```
private fun isValidBridgeLine(value: String): Boolean {
 // TODO: evaluate input
 return value.isNotBlank()
}
```

To mitigate this issue, Cure53 suggests implementing optimal validation for bridge line formats based on the transport type. Bridge lines should match expected patterns such as *obfs4 IP:PORT FINGERPRINT cert=CERT iat-mode=MODE* for obsf4 bridges and *snowflake FINGERPRINT* for Snowflake bridges. Regular expressions could also be utilized to validate the format and sanitize inputs before processing. Additionally, the error handling protocols could be augmented for invalid bridge configurations to prevent application crashes and information leakage.

TTP-04-004 WP1: Lack of certificate pinning in bridge HTTPS request *(Medium)*

Testing confirmed that the Tor VPN application performs HTTPS requests to the Tor Project's bridge API without implementing certificate pinning. The *OkHttpClient* is instantiated with default settings, lacking any form of certificate validation beyond standard system trust store verification.

Accordingly, a threat actor with network-level access can perform Man-in-the-Middle (MitM) attacks against bridge configuration requests. In censorship-heavy environments whereby users rely on bridge configurations, adversaries could intercept and modify bridge responses, potentially providing malicious bridge addresses or preventing users from obtaining valid bridges. This is particularly concerning given that users requesting bridges are often in hostile network environments where such attacks are feasible.

Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
vpn/app/src/main/java/org/torproject/vpn/circumvention/CircumventionApi.kt

Affected code:
```
object ServiceBuilder {
 private val client = OkHttpClient.Builder().build()
 private val retrofit = Retrofit.Builder()
 .baseUrl("https://bridges.torproject.org/moat/circumvention/")
.addConverterFactory(GsonConverterFactory.create())
 .client(client)
 .build()
 fun<T> buildService(service: Class<T>): T = retrofit.create(service) }
```

To mitigate this issue, Cure53 advises installing certificate pinning for the *bridges.torproject.org* domain. The Tor Project's certificate public key pins should be added to the *OkHttpClient* configuration with *CertificatePinner*. This will ensure that only legitimate Tor Project certificates are accepted, effectively preventing MitM attacks even when adversaries control the network or have installed malicious CA certificates on the device.

TTP-04-005 WP1: Insecure randomness for bridge selection *(Low)*

While reviewing the Android app's source code, Cure53 determined that two bridge types are supported, namely *obfs4* and *snowflake*, both of which serve to obfuscate user traffic. If a user configures multiple bridges of both types, the app randomly selects the bridge and its type via the insecure random number generator *java.util.Random*.

This results in predictable bridge selections, which could aid attackers in achieving their malicious objectives. For instance, a DoS attack could be mounted on the next bridge server used by a victim app, or an MitM attack could be attempted by placing itself between the Tor VPN app and the subsequent user-selected bridge.

The excerpt below verifies the adoption of the insecure random number generator *java.util.Random* for selecting the bridge(s) and type.

Affected file:
onionmasq/android/OnionmasqAndroid/onionmasq/src/main/java/org/torproject/onionmasq/BridgeLineParser.java

Affected code:
```
import java.util.Random;
 public void selectSingleBridgeConfig() {
 if (bridgeConfigs.isEmpty()) {
 return;
 }
 Random random = new Random();
```


Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
 selectedBridgeConfig = random.nextInt(bridgeConfigs.size());  }
[...]
static class IPtConfig {
 [...]
 public void selectSingleBridgeConfig() {
 if (bridgeConfigs.isEmpty()) {
 return;
 }
 Random random = new Random();
 selectedBridgeConfig = random.nextInt(bridgeConfigs.size());  }
 [...]
}
[...]
public static IPtConfig getIPtConfigFrom(String bridgeLines) {  [...]
 int index = 0;
 if (bridgeTypeConfigsMap.size() == 0) {
 return null;
 } else if (bridgeTypeConfigsMap.size() > 1){
 // we have more than 1 bridge type, let's randomize which type of  bridgelines to use
 Random random = new Random();
 index = random.nextInt(bridgeTypeConfigsMap.size());
 }

 Object[] keys = bridgeTypeConfigsMap.keySet().toArray();
 String key = (String) keys[index];
 IPtConfig config = bridgeTypeConfigsMap.get(key);
 [...]
}
```

To mitigate this issue, Cure53 advises utilizing performant cryptographic random number generators for any security sensitive operation requiring randomness, such as *SecureRandom* for Java.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

TTP-04-006 WP2: DNS resolution may re-occur for same domain name *(Low)*

Testing confirmed that the hostname canonicalization function of Onionmasq's built-in DNS stack is incomplete and fails to handle case normalization entirely. The function only removes trailing dots while ignoring case differences, causing the same domain with different casing to be treated as distinct entries in the DNS cache.

As such, *DoMaiN.com*, *domain.com*, and *DOMAIN.COM* can trigger separate DNS resolutions and occupy different cache slots, despite constituting the same domain. The incomplete implementation leads to redundant DNS queries, increased memory usage, potential cache exhaustion, and privacy implications from unnecessary DNS lookups that could aid traffic correlation attacks.

Affected file:
onionmasq/crates/onion-tunnel/src/dns.rs

Affected code:

```
fn canonicalize_hostname(hostname: &mut String) {
 // TODO: This is incomplete
 if hostname.ends_with('.') {
 hostname.pop();
 }
}
```

To mitigate this issue, Cure53 recommends implementing complete DNS hostname canonicalization in adherence to the RFC 4343 requirements. Case normalization should be integrated by converting all ASCII characters to lowercase, handling internationalized domain names (IDN) properly with Unicode normalization, and implementing a comprehensive hostname validation procedure. Furthermore, the dev team should consider replacing the incomplete function with adequate canonicalization logic before performing any DNS operations to ensure consistent cache behavior and prevent redundant lookups.

TTP-04-010 WP2: DoS via zero-size packet parsing *(Medium)*

During a source code review of the *onionmasq* crate, Cure53 observed that version identifiers are extracted from packets without checking their lengths. In fact, the *parser* implementation reads the element at location 0 of a *packet* type instance (which corresponds to an alias for byte arrays), without checking that the packet length is strictly greater than 0. Reading out-of-bounds in Rust results in a panic and process crash if unhandled.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin

To elaborate, in the event that an attacker is able to send zero-sized packets to the *onionmasq* implementation, a panic situation will be induced. Since the process fails to handle these situations, the process will crash and a DoS situation will be forced for the entire app.

The excerpt below demonstrates the missing check. The *parse* function accesses the contents of the *packet* variable without checking its length, resulting in an out-of-bounds access for null packets that triggers a panic situation.

Affected file:
onionmasq/crates/onion-tunnel/src/parser.rs

Affected code:
```
pub fn parse(packet: &Packet) -> Option<ParseResult> {
 let vers = packet[0] >> 4;
 let (src, dst, protocol, payload) = match vers {
 [...]
 };
 [...]
}
```

To mitigate this issue, Cure53 advises incorporating checks on the *packet* variable lengths and dropping all packets with unexpected values.

TTP-04-012 WP2: TURN client DoS due to endless loop by rogue server *(Medium)*

The *turn-tcp-client* crate contains an implementation for a TURN client, which is utilized by the UDP tunnel feature of the *onionmasq* crate. This client contains the *transaction* function; this is applied to read a response from a TURN server after sending a message. As such, the client reads from the server until a response is sent with the correct *transaction_id*.

As a result, a rogue TURN server could allow the TURN client to loop indefinitely and attempt to exhaust the CPU resources of the *onionmasq* process by sending responses with an unexpected *transaction_id*. This will ultimately result in a DoS situation for the Tor VPN app in the worst-case scenario.

The excerpt below highlights the underlying limitation. The *transaction* function reads from the TURN server until the incoming response's *transaction_id* matches the expected message's *transaction_id*.

Affected file:
onionmasq/crates/turn-tcp-client/src/proto.rs

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106

Affected code:

```
pub async fn transaction<W: AsyncWrite + Unpin, R: AsyncRead + Unpin>(  writer: &mut
TurnTcpWrite<W>,
 reader: &mut TurnTcpRead<R>,
 msg: &Message,
 ) -> Result<Message, Error> {
 log::trace!("STUN transaction -> {:?}", msg);
 writer.write_stun(msg).await?;
 loop {
 if let TurnMessageOwned::Stun(incoming) = reader.read().await? {  if incoming.transaction_id
== msg.transaction_id {  log::trace!("STUN transaction reply <- {:?}", incoming);  return
Ok(incoming);
 }
 }
 }
}
```

To mitigate this issue, Cure53 advises aborting the read after several attempts and returning an error from the *transaction* function. Alternatively, a hard timeout could be enforced and an error returned from the *transaction* function if the timeout elapses.

TTP-04-013 WP1: Public IP resolution hardening considerations *(Info)*

Cure53's analysis confirmed that the Tor VPN application's utilities employ a third-party service to verify VPN connectivity by checking the public IP address. The implementation leverages standard HTTPS connections without certificate pinning and relies on an untrusted external service, which could raise concerns in high-risk settings.

While this functionality appears to be limited to test code, similar patterns in production code could expose users to privacy risks. Third-party IP resolution services can track request patterns, potentially correlating VPN usage with real IP addresses if connections fail or during the initial connection phase.

Affected file:
app/src/androidTest/java/org/torproject/vpn/utils/NetworkUtils.kt

Affected code:

```
private fun getJsonFromGeoIPService(): JSONObject {
 val url = URL("https://wtfismyip.com/json")
 val connection = url.openConnection() as HttpsURLConnection  val response =
StringBuilder()
 // ... standard HTTPS connection without pinning
}
```


Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106

To mitigate this issue, Cure53 advises utilizing multiple IP check services with failover to prevent single points of failure. Moreover, request headers should be integrated to identify the application and version for optimal service compatibility. Furthermore, timeout and retry logic should be applied with exponential backoff to handle service unavailability. Lastly, if the aforementioned pattern is used in production code, the dev team should ensure that IP checks only occur after VPN establishment to prevent IP correlation.

TTP-04-014 WP2: DNS poisoning via lack of cache expiration *(Low)*

Testing confirmed that Onionmasq's built-in DNS resolver implementation caches DNS responses indefinitely without implementing Time To Live (TTL) expiration or cache invalidation mechanisms. Once a hostname is resolved, the mapping remains in the cache for the entire session duration, ignoring the authoritative TTL values from DNS responses.

Consequently, attackers that are able to successfully poison the DNS cache can maintain persistent control over hostname resolution. If malicious DNS responses are cached via temporary network compromise or DNS hijacking, victims will continue to use attacker controlled IP addresses even after the initial attack vector is remediated. Additionally, legitimate DNS changes for load balancing or failover are ignored, potentially directing users to outdated or decommissioned servers.

Affected file:
onionmasq/crates/onion-tunnel/src/dns.rs

Cure53 identified that the *onionmasq* repository fails to implement the following safeguards:

- TTL extraction from DNS responses.
- Expiration timestamps stored with cached entries.
- Periodic cache cleanup mechanisms.
- Maximum cache size limits.
- Cache invalidation on network modifications.

To mitigate this issue, Cure53 suggests asserting DNS cache expiration by storing TTL values from DNS responses alongside cached entries. Timestamps can also be introduced to track cached entries and perform periodic removal of expired entities. Furthermore, it is advised to implement a maximum cache size with LRU eviction and provide mechanisms to flush the cache on network changes or user requests. Following RFC standards for minimum and maximum TTL bounds will prevent cache poisoning and excessive DNS queries.

Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

TTP-04-015 WP2: Apple FFI vulnerable to OOB memory read *(High)*

Cure53's evaluation of Onionmasq's Apple-specific FFI interface confirmed that the exposed *receive* function performs unsafe memory operations without any validation. The function accepts raw pointers for packet data and lengths, then blindly dereferences them based on an untrusted *packet_count* parameter, creating multiple memory safety vulnerabilities.

An attacker in control of the FFI interface (via a malicious iOS app or compromised *NetworkExtension*) could trigger out-of-bounds memory reads, potentially exposing sensitive data from adjacent memory regions or causing application crashes. The lack of bounds checking may permit adversaries to read arbitrary memory by providing extensive *packet_count* values or invalid pointers.

Affected file:
onionmasq-apple/src/nepackettunnelflow.rs

Affected code:

```
pub unsafe fn receive(
 packets: *const *const u8,
 packet_lengths: *const usize,
 packet_count: usize,
 ) {
 let mut data = BUFFER.lock().expect("buffer poisoned");

 for i in 0..packet_count {
 let packet = *packets.add(i);
 let len = *packet_lengths.add(i);
 let slice = slice::from_raw_parts(packet, len);

 data.push_back(slice.to_vec());
 }
}
```

To mitigate this issue, Cure53 advises applying comprehensive safety checks prior to performing pointer operations, such as null pointer validations, bounds checking, and safer abstractions where possible.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

TTP-04-016 WP2: Onionmasq-mobile FFI vulnerable to double fd close *(Low)*

The *onionmasq-mobile* crate's *run_proxy* function, which is exposed to the application code via the *Java_org_torproject_onionmasq_OnionMasqJni_runProxy* FFI function, assumes a file descriptor from the FFI client and carries it through when creating the Tor VPN virtual device. The virtual device will eventually close the file descriptor when the *Java_org_torproject_onionmasq_OnionMasqJni_closeProxy* FFI function is called. However, since the same file descriptor passed from the FFI client is carried through for use within the Tor VPN virtual device, a malicious FFI client may preemptively close the file descriptor. This leads to *close* being called on the same file descriptor twice, causing undefined behavior and potentially leading to Rust FFI host crashes or file descriptor hijacking (among other implications).

Albeit, the current Tor VPN Android app correctly avoids calling *close* on the file descriptor passed to *run_proxy*. As such, RCE on the Tor VPN Android app (or any SDKs the app employs) is required for a successful exploit.

Affected files:

- *onionmasq/crates/onionmasq-mobile/src/ffi.rs*
- *onionmasq/crates/onionmasq-mobile/src/lib.rs*

To mitigate this issue, Cure53 suggests modifying the *run_proxy* function in the *onionmasq mobile* crate to immediately call the *dup* syscall (or any variant) on the received file descriptor passed from the FFI client. This will prevent the FFI client from calling *close* on the file descriptor passed into *run_proxy* and avoid any undefined behaviors.

TTP-04-017 WP2: Tor VPN Android app lacks root detection *(Low)*

Dynamic testing confirmed that the Tor VPN Android app fails to implement a root detection mechanism. This poses a security threat, since attackers with root access to the device can bypass most security measures, modify files, or tamper with the app's functionalities.

According to the recommendations outlined in the *OWASP Mobile Application Security Testing Guide* (MASTG)[5], mobile applications must incorporate root/jailbreak detection and anti-debugging mechanisms. These measures will enhance the app's anti-tampering resiliency, rendering the manipulation of application code and its data significantly difficult for an attacker to achieve. Moreover, the application will be hardened in general and the potential attack surface minimized.

[5] https://github.com/OWASP/owasp-mastg

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 advises adopting the freely available *jail-monkey*[6] library for Android to notify root users of all risks associated with using the app on these devices.

TTP-04-018 WP2: TCP socket listeners could lead to DoS or hijacking *(Low)*

While auditing the *onion-tunnel* crate, Cure53 discovered that the *handle_tcp_packet* function of the *parser.rs* file opens a socket listener during the TCP packet handling process. Due to the lack of a timeout for incoming connections, the likelihood of DoS attacks is increased. An application could purposefully keep the listeners open without ever establishing a connection in order to consume resources and induce a DoS condition.

Retaining open listeners also introduces the risk of connection hijacking if an attacker is able to connect to the listener before the intended application. However, this scenario is considered unlikely, as the attacker would need to guess the listener's address and port, then connect faster than the legitimate application to successfully hijack the connection. The excerpt below highlights the open listener on handling TCP packets, which fails to enforce a timeout for open connections.

Affected file:
onionmasq/crates/onion-tunnel/src/parser.rs

Affected code:

```
fn handle_tcp_packet(
 src_addr: IpAddress,
 dst_addr: IpAddress,
 payload: &[u8],
) -> Option<ParseResult> {
 match TcpPacket::new_checked(payload) {
 Ok(tcp_packet) => {
 [...]
 if let Some(mut socket) =
Self::maybe_make_tcp_socket(&tcp_packet) {
 let src_port = tcp_packet.src_port();
 let dst_port = tcp_packet.dst_port();
 [...]
 if let Err(e) = socket.listen((dst_addr, dst_port)) {  [...]
 }
 [...]
 }
 }
 [...]
 }
 None
}
```

[6] https://github.com/GantMan/jail-monkey

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 recommends implementing a timeout mechanism that closes dangling socket listeners. This revised approach will ensure that open listener resources are released in a timely manner, effectively preventing DoS situations.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This pentest's key priority was to estimate the security proficiency of the Tor VPN app for

Android (WP1) and the tunnel interface for Arti (WP2).

The internal and auditing teams utilized a dedicated Signal group for open questions. Throughout the engagement, Cure53 provided frequent status updates on the assessment's progress. Communication was excellent and assistance was provided whenever requested.

To conform to the desired crystal-box methodology, Tor shared the URLs to the GitLab repositories for the Tor VPN Android app and Onionmasq tunnel interface, as well as the branch names subject to evaluation. Cure53 fixed the commit at the start of the pentest and consequently audited the repositories at these commit hashes. To conduct dynamic testing, a debug build of the Tor VPN Android app for the chosen commit hash was utilized.

The Tor VPN Android app is primarily written in Kotlin, whereas the Onionmasq tunnel interface comprises several Rust crates. Both components are astutely organized, allowing for straightforward comprehension of the implemented functionality.

The coverage included critical parsing and validation components across both work packages. The testing efforts revealed negative patterns regarding input validation, memory safety, and cryptographic implementations that warrant attention. Other areas of concern were also noted.

Onionmasq builds the foundation for forwarding network traffic to the Tor network, operating on the network layer 3 (IP layer) and handling the low-level TCP/UDP protocols, as well as managing DNS request querying and caching. Viable attack scenarios entail malicious servers that trigger unintended behavior via specially crafted response data and vulnerabilities that can be triggered via a manipulated web site. In addition, local attacks from different user accounts and malicious apps were considered.

As network packets are parsed manually by Onionmasq, Cure53 honed in on the individual protocol parsers for TCP/UDP and the associated handlers. The testers quickly confirmed that the premise lacks validation checks, leading to several corresponding issues. Specifically, the tested components offer inadequate input validation on multiple levels. For example, the TCP packet parser accepts packets without implementing fundamental RFC mandated checks such as checksum verification or sequence number validation (TTP-04-011). Similarly, the DNS canonicalization function remains incomplete, handling only trailing dots while ignoring case normalization entirely (TTP-04-006). This suggests that input validation may have been an afterthought rather than a security-by-design consideration.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The pattern of partial or absent validation creates a potential attack surface that sophisticated adversaries could later chain into impactful attacks. The handling of TCP packets was also explored, confirming that Onionmasq creates a new listener for each TCP

packet, which could result in DoS or hijacking (TTP-04-018).

The bridge line validation protocols attest to this subpar approach, whereby the application merely checks if the input is non-blank, rather than validating against expected bridge line formats (TTP-04-003).

The included DNS resolver runs a local DNS server and requests all queries via the Tor network. At present, A (ipv4) and AAAA (ipv6) requests are supported exclusively. DNS queries are only sent once, as responses are cached locally indefinitely in the current process. Plausible considerations here include poisoning, which could allow returning incorrect addresses for a given host name, and DoS due to improper resource management. The DNS resolver implementation revealed several flaws in these areas. The lack of rate limiting and permanent cache retention create a trivial DoS vector whereby adversaries could exhaust system memory via flooding attacks, as well as raises the probability of DNS poisoning (TTP-04-008 and TTP-04-014). Furthermore, the DNS resolver suffers from two additional DoS pitfalls, owing to the use and handling of IPv4 address allocations on queries (TTP-04-007 and TTP-04-008). Moreover, the TCP handling is affected by a potential DoS vulnerability (TTP-04-010) as well as specific risks involving the TURN client utilized by the UDP tunnel feature (TTP-04-012).

The FFI interfaces were also reviewed. Here, the discovery of an out-of-bounds memory read vulnerability in the Apple FFI interface (TTP-04-015) highlights unsafe memory operations in platform-specific integration code, although this particular Rust crate (*onionmasq-apple*) was considered out-of-scope for this project. Memory-safe parsing libraries should be adopted over Rust's *unsafe*. The *onionmasq-mobile* crate is also susceptible to a double fd close (TTP-04-016).

The absence of certificate pinning for critical bridge API communications (TTP-04-004) and the use of predictable randomness for bridge selection (TTP-04-005) suggest that cryptographic best practices have not been consistently applied during the development of the Tor VPN Android app. For an application designed to operate in censorship-heavy or otherwise high-risk environments, these oversights are troubling and could facilitate the type of MitM attacks that users seek to avoid. Conducting a systematic review of all external API interactions to implement proper certificate pinning would strengthen this deficiency. Furthermore, the public IP address resolution would benefit from improvement (TTP-04-013).

Elsewhere, Cure53 unearthed common limitations concerning mobile app security, including the support of outdated SDK versions (TTP-04-001), unencrypted shared preferences and configuration files (TTP-04-002), and a lack of root detection (TTP-04-017).

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Subsequently, the deployed dependencies were appraised to verify their project suitability and maturity, as well as pinpoint any improvement opportunities. A code review of selected

areas with a focus on anonymity was performed to determine if the requirements for the Tor VPN Android app are satisfied. Third-party libraries are generally employed in an effective manner. Their selection is appropriate for the given project, while the selected libraries are actively maintained and appropriately safeguarded.

The review team achieved satisfactory coverage across both work packages. Towards the end of the engagement, the customer provided an additional archive, slightly expanding the scope. However, no noteworthy findings were identified within it.

Despite the uncovered issues, testing revealed that the core Tor integration appears sound, exhibiting no fundamental flaws in its tunnel establishment or traffic routing mechanisms. The modular architecture also assists in containing potential compromises, meaning that weaknesses in the DNS resolver do not directly affect VPN tunnel integrity.

In summary, the team identified two primary areas of concern regarding vulnerabilities and faults: missing input data validations and DNS resolver/DoS attack vectors. Cure53 suggests prioritizing the implementation of comprehensive input validation frameworks rather than ad-hoc checks, as well as establishing resource limits, cleanup, and timeout mechanisms across all (caching) components. The recurring nature of these validation and resource management issues suggests that an internal security review process could benefit from formalized checklists or automated tooling to catch these patterns earlier in development.

Cure53 would like to thank Micah and Bella from the Tor Project team for their excellent project coordination, support, and assistance, both before and during this assignment.