Security Research Labs

OpenVPN v2.7 Security Audit

Threat model and hacking assessment report

v1.0-FINAL, 23 March 2026

Prepared for:
OpenVPN Community

- Public - 51 pages -

Content

Version	v1.0-FINAL
Client	OpenVPN community
Date	23 March 2026
Assessment Team	Aarnav Bos Genco Altan Demir Marc Heuse Stephan Zeisberg

Version History

Date	Version
February 20, 2026	v0.1 – draft release
March 20, 2026	v0.3 – updated draft release
March 23, 2026	v1.0 – final release

Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the timeframe and scope detailed in Chapter 3.

Please note that this report does not guarantee that all existing security vulnerabilities have been exhaustively identified. Implementing the recommendations provided herein does not ensure that future code will remain free of bugs.

Integrity Notice

Assessment Timeline

Security Research Labs performed the OpenVPN 2.7 release security assessment over the course of 16 weeks, starting from 15th of October 2025.

Date	Event
October 15, 2025	Assessment kick-off
December 18, 2025	Intermediate report
February 20, 2026	Draft findings report
March 20, 2026	Updated draft findings report
March 23, 2026	Final findings report

Table 1: Audit timeline

1 Executive summary

1.1 Engagement overview

This report documents the results of Security Research Labs's security review of the OpenVPN source code from the 15th of October to the 20th of February, culminating in the release of version 2.7. OpenVPN is open-source and the most used VPN solution in the world.

During this study, the OpenVPN developers provided access to relevant documentation and effectively supported the research team. We verified the protocol design, concept documentation, and relevant available source code of OpenVPN.

This audit focused on assessing OpenVPN's codebase for resilience against hacking and abuse scenarios. Key areas of scrutiny included the TLS authentication implementation, packet handling and processing, VPN tunnel integrity and the configuration primitives. The testing approach combined static and dynamic analysis techniques, leveraging both automated tools and manual inspection. We prioritized reviewing critical functionalities and conducting thorough security tests to ensure the robustness of OpenVPN's server, interactive service and client implementations. We collaborated closely with the OpenVPN team which fully supported our work, utilizing full access to source code and documentation to perform a rigorous assessment.

1.2 Observations and risk

The research team identified several issues ranging from critical to informational level severity. Most of the issues concerned OpenVPN's Interactive Service on the Windows operating system, input parsing functionality, and configuration primitives. The OpenVPN developers have acknowledged the issues and is actively working on remediation in cooperation with us. By now, they have successfully remediated most of the identified critical issues.

We did not find any attacks against the OpenVPN transport security guarantees. Overall effective defensive in-depth development techniques were found in place.

Security Research Labs notes two key observations that pose a risk to the OpenVPN codebase: Firstly, since the OpenVPN protocol naturally has a long-standing legacy, the implementation supports several configuration and operational primitives. This extensive support has introduced complex conditional logic in the implementation and contradictions in the documentation. This poses a risk to the maintainability of the codebase. Secondly, a large number of findings were uncovered in the OpenVPN Interactive Service (for Windows), indicating that this component was not given appropriate scrutiny.

1.3 Recommendations

In addition to mitigating the remaining open issues, Security Research Labs recommends introducing state machine abstractions to better illustrate and maintain complex sequential code and utilizing static analysis to fix code quality issues and integrating newer fuzzing harnesses to ensure that critical functionality is thoroughly tested.

2 Evolution suggestions

To ensure that OpenVPN is secure against further unknown or yet undiscovered threats, we recommend considering the following evolution suggestions and best practices described in this section.

2.1 Business logic improvement suggestions

The business logic of the OpenVPN implementation and protocol is time and battle tested software. Security Research Labs' recommendations for business logic improvements do not call for change in functionality but for an investment to enhance code maintainability and maturity.

Introduce a state machine abstraction. On the OpenVPN server, the connection initiation and packet processing pipeline is a complex sequential process with several checks and side effects due to the large amount of configuration options available in the OpenVPN protocol. The checks and side effects are spread across several functions with large cyclomatic complexity, making the design and implementation of critical functionality hard to understand and maintain. We strongly recommend introducing a state machine abstraction where side effects, checks and state transitions are tied transparently to a state. This would make the process and implementation easier to reason, audit and maintain.

Expand and fix compiler warnings. OpenVPN has support for compiling time checks for common code quality issues, security vulnerabilities and best practice deviations via compiler warnings. These options may be toggled with `–enable-strict`. However, the options are currently limited to `-Wsign-compare` and `-Wuninitialized`. C compilers such as clang and GCC offer a plethora of options that could assert the security and quality of the code. During our audit, we found several violations of compiler warnings. In our static analysis section, we list the checks that we utilized and we strongly encourage their integration.

2.2 Secure development improvement suggestions

We recommend further strengthening the security of OpenVPN by implementing the following best practices:

Perform threat modeling. Threat modeling for all new features and major updates before coding promotes better code security. This practice lets developers identify potential security threats and vulnerabilities early in the design phase, enabling them to implement appropriate mitigations from the outset. Including the threat model in the pull request description ensures that the entire team is aware of the identified risks and the measures taken to address them, promoting a proactive security culture and enhancing the overall robustness of the codebase. Security Research Labs has shared a threat model document with the OpenVPN that the OpenVPN team could use for this purpose.

Use static analysis. Static analysis identifies vulnerabilities, coding errors, and compliance issues early in the development process allowing developers to proactively address potential security issues before they reach production. Tools such as CodeQL and cppcheck can run as part of a Continuous Integration (CI) pipeline ensuring that all code before merging is compliant. Security Research Labs is currently preparing to share the static analysis suite utilized in this assessment.

Perform dynamic analysis. OpenVPN has several fuzzing harnesses on Google's oss-fuzz project. However, some of the harnesses failed to compile during our audit, indicating a gap in testing. Additionally, these harnesses cover a very small subset of OpenVPN's functionality. Critical areas such as packet processing and options handling are not covered. Fuzz testing helps uncover bugs such as memory corruption, undefined behavior or assertion violations. These bugs are typically difficult to discover manually. Security Research Labs recommends updating and extending the fuzz testing suite to ensure an exhaustive dynamic analysis suite. Additionally, we recommend moving the harnesses to the OpenVPN repository and integrating them into a Continuous Integration pipeline to simplify maintenance, a process we are currently actively supporting.

2.3 Address currently open security issues

We recommend addressing already known security issues before the next major release. Even if an open issue has a limited impact, an attacker might use it as part of their exploitation chain, which may have a more severe impact on OpenVPN.

2.4 Further recommended best practices

Extensive test implementation. Currently, most functionality has relevant unit tests, but they are neither comprehensive nor exhaustive. For example, the options processing only tests the minimal mocked functionality, not the complete implementation. Another example is the auth token tests, which test a very limited subset of outcomes that should lead to failure. Although the essential tests are already implemented, they do not cover all scenarios. We recommend extending unit tests to provide a stronger focus on security. This includes testing for edge cases, potential vulnerabilities, and common attack vectors.

Documentation. The documentation occasionally lacked consistency with the code and sometimes contained contradictions. This made it challenging to understand the code without consulting the development team. Accurate documentation ensures that developers, auditors, and other stakeholders can comprehend the codebase without frequently consulting the development team. To achieve this, establish a practice of updating the documentation concurrently with any code changes. Incorporating documentation verification into the code review process can help detect discrepancies early.

3 Motivation and scope

OpenVPN is the world-wide most deployed open-source VPN solution. OpenVPN is a cross-platform client/server VPN with flexible configuration and multiple authentication methods (credentials, pre-shared keys, TLS certificates) using OpenSSL and other TLS libraries. It separates control and data channels, enabling kernel offloading via Data Channel Offload for better performance on Linux, FreeBSD and Windows. On other platforms that have no kernel implementation, like macOS, data channel packets are handled in user space, enabling easy software portability to new platforms.

Tasked by the Sovereign Technology Fund, we perform a thorough security audit of OpenVPN that we executed in two phases:

1. Focus on the new DCO implementation, Windows support, TLS integration, networking and caching.

2. Extended scope of TLS library integrations, DNS, auth tokens, SITNL, XKey and PKCS11.

Security Research Labs collaborated with the OpenVPN team to create an overview containing the runtime modules in scope and their audit priority. Table 2 reflects the primary components and Table 3 reflects the extended components. During the audit, Security Research Labs developed a threat model to guide efforts on exploring potential security flaws and realistic attack scenarios. The OpenVPN team also provided an architectural diagram, shown in Figure 1 of section 4.3, which aided SRLabs' assessment.

Repository	Files(s)	Component
OpenVPN [1]	crypto.{c, h}	Openvpn
	crypto_epoch.{c, h}	Openvpn
	dco.{c, h}	Openvpn
	dco_freebsd.{c, h}	Openvpn
	dco_internal.h	Openvpn
	dco_linux.{c, h}	Openvpn
	dco_win.{c, h}	Openvpn
	forward.{c, h}	Openvpn
	mroute.{c, h}	Openvpn
	mtcp.{c, h}	Openvpn
	mudp.{c, h}	Openvpn
	multi.{c, h}	Openvpn
	multi_io.{c, h}	Openvpn
	options.{c, h}	Openvpn
	common.c	openvpnserv
	interactive.c	openvpnserv
	socket.{c, h}	Openvpn
	ssl.c	Openvpn
	ssl_ncp.{c, h}	Openvpn
	ssl_verify.{c, h}	Openvpn
	ssl_verify_openssl.c	Openvpn
	tls_crypt.{c, h}	Openvpn
	wfp_block.{c, h}	Openvpn

Table 2: Primary In-Scope OpenVPN's components

Repository	Files(s)	Component
OpenVPN [1]	ssl_backend.h	Openvpn
	ssl_common.h	Openvpn
	ssl.h	Openvpn
	ssl_mbedtls.{c,h}	Openvpn
	ssl_openssl.{c,h}	Openvpn
	ssl_pkt.{c,h}	Openvpn
	ssl_util.{c,h}	Openvpn
	ssl_verify_backend.h	Openvpn
	ssl_verify_mbedtls.{c,h}	Openvpn
	ssl_verify_openssl.h	Openvpn
	auth_token.{c, h}	Openvpn
	crypto_mbedtls.{c, h}	Openvpn
	crypto_openssl.{c, h}	Openvpn
	route.{c, h}	Openvpn
	cryptoapi.{c, h}	openvpnserv
	ntlm.{c, h}	openvpnserv
	proxy.{c, h}	Openvpn
	dns.{c, h}	Openvpn
	networking_sitnl.{c, h}	Openvpn
	openvpn.{c, h}	Openvpn
	packet_id.{c, h}	Openvpn
	openvpn.{c, h}	Openvpn
	pkcs11.{c, h}	Openvpn
	pkcs11_backend.h	Openvpn
	pkcs11_mbedtls.c	Openvpn
	pkcs11_openssl.c	Openvpn
	run_command.{c,h}	Openvpn
	service.{c,h}	Openvpnserv
	validate.{c,h}	Openvpnserv
	session_id.c	Openvpn
	xkey_common.h	Openvpn
	xkey_helper.c	Openvpn
	xkey_provider.c	Openvpn
	auth-pam/pamdl.c	PAM Plugin
	auth-pam/utils.{c,h}	PAM Plugin
	auth-pam/auth-pam.c	PAM Plugin

Table 3: Extended In-Scope OpenVPN's components

4 Methodology

The methodology used in this assessment follows a structured, attacker-centric approach aimed at identifying flaws in the OpenVPN project. The methodology consists of four steps: (1) threat modelling, (2) security design coverage checks, (3) implementation baseline check, and finally (4) remediation support. The threat modelling was conducted in alignment with the STRIDE threat modelling framework. The STRIDE framework is a widely used security model that helps identify potential threats in software systems. It stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege, each representing a different category of security risks.

4.1 STRIDE Threat modeling and attacks

The goal of the threat model framework is to determine specific areas of risk for OpenVPN, including both potential threats and concrete attack scenarios. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, as well as security testing.

Our threat modelling consists of 3 stages, each building upon the previous one, resulting in a list of attacks and their severities, graded from low to critical.

In the first stage, each STRIDE category is assigned a *STRIDE Hacking Value* which measures how valuable a threat in each category is to an attacker. The *STRIDE Hacking Value* is categorized into low, medium or high, with the following definitions:

- **Low:** Threats in this STRIDE category offer the hacker little to no gain.
- **Medium:** Threats in this STRIDE category offer the hacker considerable gains.
- **High:** Threats in this STRIDE category offer the hacker high gains.

In the second stage, concrete threat scenarios are identified and mapped to their corresponding STRIDE categories. Each threat is assigned a *Threat Impact* value, which represents the potential damage the threat could cause to the project. *Threat Impact* is categorized into low, medium or high, with the following definitions:

- **Low:** Threat scenario would cause negligible damage to the project.
- **Medium:** Threat scenario poses a considerable threat to the project.
- **High:** Threat scenario poses an existential threat to the project.

Based on the *Threat Impact* and the *STRIDE Hacking Value* of the associated category, a *Threat Risk* is derived according to Table 3. The *Threat Risk* represents the overall risk posed by each identified threat.

STRIDE Hacking Value/Threat impact	Low	Medium	High
Low	Low	Medium	Medium
Medium	Medium	Medium	High
High	Medium	High	High

Table 3: Threat Risk calculation matrix

In the third and final stage, concrete attack scenarios are identified and mapped to the previously defined threats. Each attack is assigned an *Attack Feasibility*, which measures how likely an attack is to succeed, considering complexity, required skill, and resources. *Attack Feasibility* is categorized into low, medium or high, with the following definitions:

- **Low:** Attack requires significant expertise and uncommon conditions.
- **Medium:** Attack requires reasonable effort and standard attacker capabilities.

- **High:** Attack requires minimal effort or widely available tools.

Based on the *Attack Feasibility* and the *Threat Risk* of the associated threat, an *Attack Severity* is derived according to Table 4.

Threat Risk / Attack Feasibility	Low	Medium	High
Low	Low	Medium	Medium
Medium	Medium	Medium	High
High	Medium	High	Critical

Table 4: Attack Severity calculation matrix

Together, these stages define a structured approach that progresses from abstract STRIDE threat categories and their value to an attacker, through concrete threat scenarios and their associated risk, and finally to specific attack scenarios with assigned severities.

Applying the framework to OpenVPN, different areas of risk were identified. Table 5 provides a high-level overview of the identified hacking risks relevant to the application, including each STRIDE category with its associated hacking value, example threat scenarios, their derived threat risk, and corresponding example attacks.

The complete set of identified threat scenarios, together with the attacks that enable them, is documented in the threat model deliverable. This list can serve as a starting point for developers when assessing security considerations for future feature development. By thinking in terms of threat scenarios and attacks during code review or feature ideation, many issues can be caught or even avoided altogether.

Scope Category	STRIDE Hacking Value	Example threat scenarios	Threat Risk	Example Attack Ideas
Spoofing	High	An attacker wants to impersonate a legitimate client or a server.	Low	An attacker can trick the TLS authentication implementation. An attacker can use a revoked certificate
Tampering	High	An attacker wants to manipulate traffic. An attacker wants to execute arbitrary code remotely	Medium	An attacker uses a memory corruption vulnerability to remotely execute arbitrary code on the server
Repudiation	Low	An attacker wants to initiate connections without appropriate logging.	Low	An attacker uses a memory corruption vulnerability to remotely execute arbitrary code on the server
Information disclosure	High	An attacker wants to leak session metadata. An attacker wants to leak server TLS keys	Low	An attacker can leak session keys and replay nonces An attacker is able to infer tls-auth/tls-crypt keys

Denial of service	High	An attacker wants to disrupt a VPN server.	Medium	An attacker can bypass TLS HMAC authentication An attacker can abuse –replay-persist for storage exhaustion
Escalation of Privilege	High	An attacker wants to use the running OpenVPN process to escalate privileges	Medium	An attacker uses the down-root plugin to escalate privileges. An attacker uses the Interactive service to escalate privileges.

Table 5: Risk overview

4.2 Security design coverage check.

Next, we reviewed the OpenVPN design for coverage against relevant hacking scenarios. For each scenario, we have investigated the following two aspects:

a. **Coverage**. Is each potential security vulnerability sufficiently covered by our audit?
b. **Underlying assumptions**. Which assumptions must hold true for the design to effectively reach the desired security goal?

The OpenVPN team provided us with an architectural diagram, shown in Figure 1 on the following page, with various components, their interactions and trust boundaries. This aided our thorough investigation.

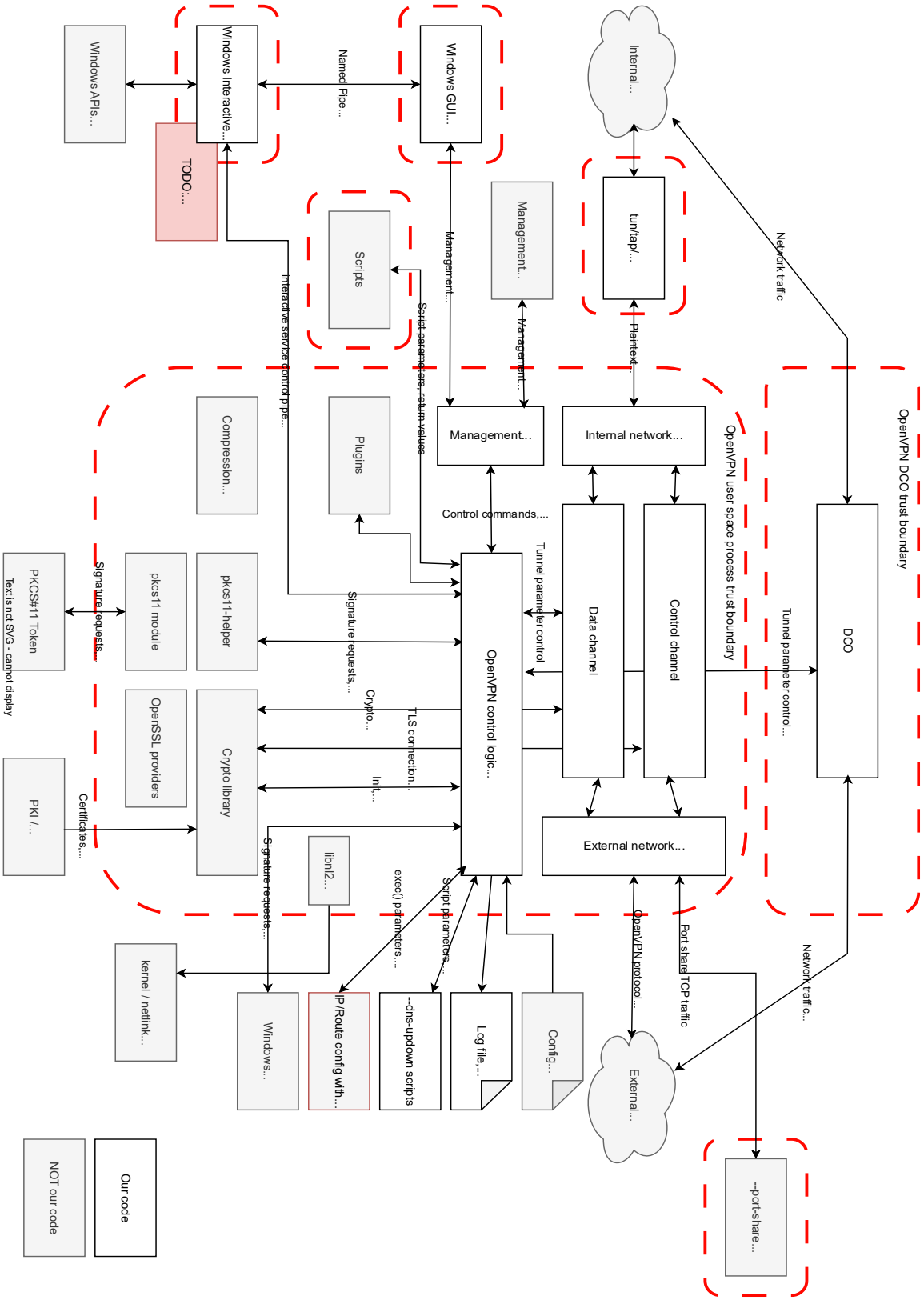

Figure 1: OpenVPN architectural diagram

4.3 Implementation check

As a third step, we tested the current OpenVPN implementation for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the OpenVPN codebase, we derived our code review strategy based on the threat model that we established as the first step. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.1.

Prioritizing potential risk for the network, the code was assessed for present protections against the respective threats and attacks as well as the vulnerabilities that make these attacks possible. For each threat, we:

1. Identified the relevant parts of the codebase, for example, the pre-authentication packet processing and the interactive service
2. Identified viable strategies for the code review. We performed manual code audits, fuzz testing, and static analysis where appropriate
3. Ensured the code did not contain any vulnerabilities that could be used to execute the respective attacks. Otherwise, we ensured that sufficient protection measures against specific attacks were present
4. Immediately reported any vulnerability that was discovered to the development team along with suggestions around mitigations

We carried out a hybrid strategy utilizing a combination of code review, static tests, and dynamic tests (e.g., fuzz testing) to assess the security of the OpenVPN codebase.

While static and dynamic testing establishes a baseline assurance, the focus of this audit was on manual code review of the OpenVPN codebase to identify logic bugs, design flaws, and best practice deviations. We reviewed the OpenVPN repository which contains OpenVPN implementation up to commit *33a700d6* from the 15th of October 2025. We aimed to trace the intended functionality of the files in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected or malicious behavior on the VPN due to logic bugs or missing checks. Since the OpenVPN codebase is entirely open source, it is realistic that an adversary could analyze the source code while preparing an attack.

Fuzz testing is a technique to identify issues in code that handles untrusted input. In OpenVPN‘s case these are remote authentication primitives such as TLS certificates, data and control channel packets and configurable options.

Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz testing methods written for this assessment utilized custom harnesses alongside oss-fuzz harnesses [2].

4.4 Remediation support

The final step is supporting OpenVPN with the remediation process of the identified issues. Each finding was documented and published with mitigation recommendations. Once the mitigation solution is implemented, the fix is verified by us to ensure that it mitigates the issue and does not introduce other bugs.

During the audit, findings were shared via a dedicated GitHub repository [3]. We also used a private Signal group for asynchronous communication and status updates. In addition, biweekly jour fixe meetings were held to provide detailed updates and address open questions.

5 Static analysis assessment

In our static analysis assessment, we utilized weggli-rs, semgrep and cppcheck. Semgrep is a closed-source static code analysis, cppcheck and weggli-rs are open source. At the kick-off of our manual review phase we applied a set of custom-developed and open-source C/C++ rules designed to identify common vulnerabilities and best practices mitigations.

All tools yielded no findings and only minor non-security issues which were communicated in our Signal group chat.

Upon completion of manual review phases, we further subjected the codebase to analysis using a curated set of reasoning-capable large language models (LLMs). These models were selected and benchmarked for their performance in identifying nuanced security risks and latent attack surfaces specifically in C, thereby serving as a safeguard against human oversight or tunnel vision.

This methodology aims to complement and bolster the manual audit process by encapsulating it between two automated analysis stages: an initial rule-based scan to rapidly surface straightforward vulnerabilities, and a final LLM-driven assessment to highlight areas warranting deeper manual scrutiny.

During our inspection of the build process, we found that OpenVPN allowed compilation with compiler warnings via `–enable-strict`. These warnings were limited to `-Wsign-extension` and `-Wuninitialized`. We found several violations of `-Wsign-extension`.

Modern versions of clang and GCC support many more security related warnings, which we recommend enabling. In our audit, we utilized additional warnings shown in Table 4. We found several additional violations which we have shared with the OpenVPN team.

Warning	Description
float-warnings	Warns when a value is implicitly converted between signed and unsigned integer types, which is undefined behavior
float-equal	Warns when floating-point values are compared using exact equality, which is unreliable due to precision limitations
switch-enum	Warns when a value is implicitly converted between signed and unsigned integer types, which may alter its meaning
implicit-fallthrough	Warns when a case in a switch statement falls through to the next case without an explicit annotation or break.
duplicate-enum	Warns when multiple enumerators within the same enum are assigned the same constant value.
conditional-uninitialized	Warns when a variable may be used uninitialized depending on the control flow path taken at runtime.
shift-overflow	Warns when a bit-shift operation overflows the width of the result type, causing undefined or implementation-defined behavior.
null-derefence	when the compiler detects a code path that dereferences a null pointer.
bool-conversion	Warns when a non-boolean value such as a pointer or integer is implicitly converted to a boolean in a potentially unintended way.

string-plus-int	Warns when an integer is added to a string literal, which performs pointer arithmetic rather than concatenation.
array-bounds	Warns when array accesses use indices that can be statically determined to be out of bounds.

6 Dynamic analysis assessment

During the audit, we utilized fuzz testing to identify bugs in several core components of OpenVPN. By applying fuzz testing, we aim to uncover issues that may go undetected through manual code review. For this purpose, we utilized existing harnesses for OpenVPN on oss-fuzz and additionally built custom harnesses to test uncovered functionality. The oss-fuzz harnesses required patching as they had compilation errors, indicating an existing testing gap. Thus, we are actively supporting the OpenVPN in the migration of the harnesses to the OpenVPN repository. This would ensure continuous maintenance and integration of fuzzing the OpenVPN lifecycle.

Introduction of new harnesses. Security Research Labs implemented three new fuzzing harnesses for the audit: `fuzz_tls_pre_decrypt_lite`, `fuzz_tls_pre_decrypt` and `fuzz_options`. These harnesses fuzz-test previously uncovered remotely accessibly entrypoints.

Usage of sanitizers. As OpenVPN is implemented in C, we additionally utilized Address Sanitizer (ASAN) and Undefined Behavior sanitizer (UBSAN) in our fuzzing campaigns to ensure that we caught various memory corruption or undefined behavior bugs.

6.1 OpenVPN Fuzzing Suite

Component	Functionality Under Test	Source	Coverage Achieved
fuzz_base64	Base64 encoding/decoding	oss-fuzz	100%
fuzz_buffer	Buffer and string utility routines	oss-fuzz	31.20%
fuzz_packetid	Packet ID processing	oss-fuzz	2.14%
fuzz_dhcp	DHCP message parsing	oss-fuzz	0.89%
fuzz_proxy	HTTP Proxy auth/setup	oss-fuzz	1.7%
fuzz_list	OpenVPN's hash table implementation	oss-fuzz	1.6%
fuzz_tls_pre_decrypt_lite	Server-side encrypted packet validation and processing	custom	4.8%
fuzz_tls_pre_decrypt	Client-side encrypted packet validation and processing	custom	8.13%
fuzz_options	Processing of openvpn configuration primitives	custom	16.56%

Due to the monolithic nature of the static archive (*`libopenvpn.a`*), the coverage remains rather low across most harnesses. However, Security Research Labs has verified that the intended functionality was covered. Additionally, the audit's fuzzing campaign exhibits higher functionality covered when compared to OpenVPN's fuzzing suite on oss-fuzz [4].

7 Assessment Findings Overview

We identified 18 issues during our analysis of the runtime modules in scope in the OpenVPN codebase that enabled some of the attacks outlined above. In summary, we found 1 critical-severity, 4 medium-severity, 9 low-severity, and 4 information-level issues. An overview of all findings can be found in 7.1.

Critical	1	
High	0	
Medium	4	
Low	9	
Informational	4	
Total Issues	18	

Note: In our methodology, "critical-severity issues" are high-severity issues that could be exploited immediately by an attacker on already deployed infrastructure.

7.1 Findings summary

ID	Issue	Severity	Status
S4-1	Session-id replay firewall always accepts bogus values, enabling blind handshake spoofing	Critical	Mitigated
S2-2	Reading Heap Buffer Overflow in pkcs11h_certificate_deserializeCertificateId	Medium	Mitigated
S2-3	AEAD usage limit undercounted on receive path delays key rollover	Medium	Mitigated
S2-4	Inconsistent byte–character accounting causes out-of-bounds memory writes in DNS domain conversion	Medium	Mitigated
S2-5	Flooding unauthenticated UDP packet allocates instance and commits IP:port to real-address hash resulting in DoS	Medium	Open
S1-6	mroute_learnable_address() allows pre-auth state pollution with 0.0.0.0/255.255.255.255 when port is present	Low	Acknowledged
S1-7	Named pipe accepts remote clients and uses predictable tid in pipe name during startup	Low	Mitigated

S1-8	DeleteWfpBlock restores metrics to msg->iface.index instead of the blocked interface, enabling routing corruption and DoS	Low	Mitigated
S1-9	Case-insensitive ccd hijack	Low	Mitigated
S1-10	FormatMessageW uses byte length -> stack overflow in BlockDNSErrHandler	Low	Mitigated
S1-11	Attacker can trivially bypass reject pull filter	Low	Mitigated
S1-12	multi_process_file_closed unsafe loop boundary on failed read	Low	Mitigated
S1-13	Plugin trusted-directory check allows prefix path bypass	Low	Acknowledged
S1-14	The iproute2 networking backend ignores route table ID, and doesn't qualify gateway/interface when deleting routes	Low	Acknowledged
S0-15	Invalid incrementing of addrs offset in HandleDNSConfigMessage	Info	Mitigated
S0-16	Incorrect undo logic in HandleDnsConfigMessage may lead to DNS resolution failures	Info	Mitigated
S0-17	Null pointer dereference in dns-updown option handling	Info	Mitigated
S0-18	PAM blocks the whole openvpn thread if not configured otherwise	Info	Acknowledged

Table 6: Findings overview

8 Detailed findings

8.1 S4-1: Session-id replay firewall is vulnerable to handshake spoofing

Attack scenario	**An attacker wants to perform a denial-of-service attack on a server**
Classification	CWE-697: Incorrect Comparison
Component	OpenVPN server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/88
Attack impact	T-1: Prevent or degrade VPN connectivity
Severity	Critical
Status	Mitigated
Fix Commit	18c483dd6031d86eb393527855734e8cd62fea19

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

During UDP server packet processing, the server performs a check to verify that the client echoed the server's session-ID MAC.

The check inverts the comparison from `memcmp_constant_time()` (OpenSSL's `CRYPT_memcmp`) which would return 0 on match. The implementation incorrectly returns success If a non-zero value is returned.

```
struct session_id expected_id =
            calculate_session_id_hmac(state->peer_session_id, from, hmac,
handwindow, offset);

if (memcmp_constant_time(&expected_id, &state->server_session_id, SID_SIZE))
{
     return true;
}
```

Due to the incorrect comparison, any spoofed packet whose session ID differs from the expected value passes immediately, while legitimate packets that match the expected session ID do not. `do_pre_decrypt_check()` treats the true return value as "packet authenticated" (src/openvpn/mudp.c:155) and proceeds to call `multi_create_instance()`, allocating a `tls_mutli` struct and inserting it into the peer map before any TLS handshake completes.

Risk

An attacker can send spoofed UDP packets from an arbitrary IP and have them accepted by the HMAC/session-id firewall, causing the server to create half-open sessions and emit P_CONTROL_HARD_RESET_SERVER_V2/V3 replies toward the spoofed address. This enables state session exhaustions on the server, without possessing any credentials.

Mitigation

Fix the logic to accept only equality: change the check to return true when `memcmp_constant_time(...) == 0` and return false otherwise and ensure packets lacking ACK/session-id data are rejected and logged.

8.2 S2-2: Reading Heap Buffer Overflow in pkcs11h_certificate_deserializeCertificateId

Attack scenario	**An attacker wants to execute arbitrary code or scripts**
Classification	CWE-122: Heap-based Buffer Overflow
Component	PKCS11 backend
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/85
Attack impact	T-5: Violate traffic integrity T-1: Prevent or degrade VPN connectivity
Severity	Medium
Status	Mitigated
Fix Commit	3d0d4b1cf6c77f107af323e0ec81b0273c211867

Background

OpenVPN implements support for the PKCS11 interface, which enables the usage of hardware cryptographic tokens such as smart cards and hardware keys.

Issue description

OpenVPN implements `PKCS11` functionality to interface with hardware devices to perform cryptographic operations. During TLS context initialization, OpenVPN attempts to de-serialize a `certificate_id` through `pkcs11h_certificate_deserializeCertificateId`.

The `_pkcs11h_util_unescapeString` has a bug where it jumps 4 bytes if it encounters a `\\` character. This loop would then continue until it encounters a NULL byte or terminates due to reaching the max size.

```
while (*s != '\x0') {
        if (*s == '\\') {
            if (t != NULL) {
                if (n+1 > *max) {
                    rv = CKR_ATTRIBUTE_VALUE_INVALID;
                    goto cleanup;
                }
                else {
                    char b[3];
                    unsigned u;
                    b[0] = s[2];
                    b[1] = s[3];
                    b[2] = '\x0';
                    sscanf (b, "%08x", &u);
                    *t = (char)(u & 0xff);
                    t++;
                }
            }
            s+=4;
        }
```

Risk

An out of bounds read with the possibility to read large chunks of adjacent memory could leak secrets from the OpenVPN process. Additionally, this could crash the openvpn process.

Mitigation

As mitigation, we recommend implementing appropriate bounds checking in `_pkcs11h_util_unescapeString` before reading memory.

8.3 S2-3: AEAD usage limit undercounted on receive path delays key rollover

Attack scenario	**An attacker wants to weaken a connection's encryption guarantees**
Classification	CWE-320: Key Management Errors
Component	OpenVPN server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/86
Attack impact	T-9: Downgrade protocol security
Severity	Medium
Status	Mitigated
Fix Commit	5e6d478fb6246465fb81060e60348bb0061a94fa

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

The function `openvpn_decrypt_aead()` incorrectly accounts for plaintext data when updating the `plaintext_blocks` usage counter during decryption. Specifically, it only increments this counter based on the output length from `cipher_ctx_final_check_tag()`, which often returns zero. It omits the byte count produced by the main decryption step in `cipher_ctx_update()`, leading to an underestimation of the total decrypted data. As a result, the receive-side usage limit for AEAD ciphers is not accurately tracked. This causes `aead_usage_limit_reached()` and the subsequent `should_trigger_renegotiation()` logic to fail to initiate key renegotiation based on received traffic volume alone.

Risk

A remote peer can send continuous encrypted AEAD traffic without triggering a key rollover, especially in receive-heavy environments such as VPN gateways or data ingestion servers. This weakens the cryptographic nonce reuse protection and increases the risk of compromising the cipher's integrity guarantees, particularly for AEAD modes like AES-GCM that depend on strict usage limits.

The risk is mitigated in symmetric traffic environments or where transmit or receive side limits dominate, asymmetric workloads expose the flaw more acutely. The issue is low severity in general deployments but rises to medium or high in scenarios where a server primarily receives data. Additionally, this risk is somewhat mitigated when the remote peer is a recent OpenVPN version that also has AEAD usage checking, as counting the transmitted bytes was implemented correctly, thus the remote peer would notice reaching the usage limit, initiating a TLS re-keying from the remote side.

Mitigation

We recommend adjusting `openvpn_decrypt_aead()` to accumulate plaintext lengths from both `cipher_ctx_update()` and `cipher_ctx_final_check_tag()` and use their sum to increment `plaintext_blocks`. This approach aligns receive-side accounting with the transmit path and restores correct AEAD usage tracking.

8.4 S2-4: Invalid byte/char accounting causes out-of-bounds writes in DNS domain conversion

Attack scenario	**An attacker wants to execute arbitrary code**
Classification	CWE-787: Out-of-bounds Write
Component	OpenVPN Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/93
Attack impact	T-8: Execute arbitrary code or scripts
Severity	Medium
Status	Mitigated
Fix Commit	cd533c483aa516a60d12a132761ab349f9b8b399

Background

The OpenVPN interactive service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

The function `GetItfDnsDomains` mixes character counts and byte counts while converting registry values into a `MULTI_SZ` search-domain list. The function additionally confuses buffer capacity (`buf_size`) with actual used length (`*size` from `RegGetValueW`) and uses `buf_size` when calculating `memmove` length, potentially copying beyond the valid string region and over uninitialized data.

The function also performs an invalid size check. `size == 0` is invalid `size` is a pointer, thus making the check a `NULL` pointer check instead of a value check.

```
GetItfDnsDomains(HKEY itf, PCWSTR search_domains, PWSTR domains, PDWORD size)
{
    if (domains == NULL || size == 0)
  {
        return ERROR_INVALID_PARAMETER;
  }
}
```

Similar issues are present in `RemoveDnsSearchDomains`.

Risk

Attackers who control or influence interface registry values can trigger memory corruption and crash the process. In-place expansion with incorrect lengths can copy attacker-chosen data past the end of the buffer and create a write-what-where primitive within the current heap allocation. Depending on allocator metadata layout and surrounding objects, this enables denial of service, information disclosure through over-reads, or structured control-flow manipulation that can lead to code execution under the process account. Even without explicit exploitation, inconsistent size reporting and missing termination can poison downstream DNS resolution logic and route traffic to attacker-controlled domains.

Mitigation

We recommend rewriting `GetItfDnsDomains` and `RemoveDnsSearchDomains` and unify all internal accounting in character units and convert to bytes only when calling external APIs or when assigning the final `*size`, which eliminates unit mismatches in comparisons and memmove lengths. Validate inputs at the top by rejecting null pointers and refusing buffers smaller than two `wchar_t` for the required double terminator, which prevents empty-buffer underflows. Prefer building the normalized

list into a separate temporary buffer and then copying it back only if it fits, which simplifies bounds checks and removes complex in-place shifts.

8.5 S2-5: Flooding unauthenticated UDP packet allocates and stores instances, resulting in DoS

Attack scenario	**An attacker wants to degrade or prevent VPN connectivity**
Classification	CWE-400: Uncontrolled Resource Consumption
Component	OpenVPN Server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/102
Attack impact	T-1: Prevent or degrade VPN connectivity
Severity	Medium
Status	Open
Fix Commit	-

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

In `multi_get_create_instance_udp()` the server allocates a new multi_instance and commits its real address tuple (IP, port, protocol) into the real-address hash table before completing TLS handshake or authentication. This occurs when an incoming UDP packet does not match an existing instance and passes a lightweight pre-decrypt check. The function `do_pre_decrypt_check()` allows processing of packets that contain valid tls-auth/tls-crypt signatures or even no authentication at all, explicitly permitting unauthenticated connection attempts to trigger instance creation.

```
/* This is an unknown session but with valid tls-auth/tls-crypt
 * (or no auth at all). */
/* ... if (!mi) { ... */
else if (do_pre_decrypt_check(m, &state, real))
{
    if (frequency_limit_event_allowed(m->new_connection_limiter)) {
        reflect_filter_rate_limit_decrease(m->initial_rate_limiter);
        mi = multi_create_instance(m, &real, sock);
        if (mi) {
            hash_add_fast(hash, bucket, &mi->real, hv, mi);
            mi->did_real_hash = true;
            multi_assign_peer_id(m, mi);
            /* ... */
        }
    }
}
```

Upon passing this gate, the server immediately creates a new instance using `multi_create_instance()` and inserts the address into the hash using `hash_add_fast()`. This behavior exposes the server to spoofed packets that can trigger memory allocation and consume connection slots, all without successful authentication.

Risk

An unauthenticated attacker capable of spoofing source IP and port can send crafted UDP packets that pass the pre-decrypt gate, causing the server to allocate connection state prematurely. This results in unnecessary memory use, fills the instance hash table, and can exhaust rate-limited connection slots governed by --connect-freq. On networks without source address validation, this enables a resource-based denial-of-service window. While --tls-auth or --tls-crypt significantly mitigate this risk by requiring a valid HMAC, the code path explicitly allows unauthenticated packets under certain

configurations (when no –tls-auth or –tls-crypt is set), leaving the server exposed on permissive or misconfigured setups.

Mitigation

One option is to defer instance creation and hash insertion until after successful TLS handshake or authentication, ensuring no pre-auth state commits occur. This change eliminates the DoS surface but requires restructuring connection initiation logic to delay state tracking.

Another option is to harden the pre-decrypt gate to only allow creation for packets with verified HMAC (i.e., only with --tls-auth or --tls-crypt), rejecting packets without authentication earlier in the pipeline. This preserves current architecture but limits flexibility for configurations that intentionally that intentionally allow connections without the extra protective layer around the TLS handshake.

A third option is to implement a temporary quarantine mechanism where initial state exists but is not committed to global structures until authentication completes. This reduces exposure but increases complexity and may still consume memory under high-load spoofing conditions.

8.6 S1-6: mroute_learnable_address() allows pre-auth state pollution with certain addresses

Attack scenario	**An attacker wants to pollute routing information**
Classification	CWE-20: Improper Input Validation
Component	OpenVPN Server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/82
Attack impact	-
Severity	Low
Status	Acknowledged
Fix Commit	-

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

The function `mroute_learnable_address()` inappropriately accepts addresses such as `0.0.0.0:<non-zero port>` and `255.255.255.255:<port != 65535>` due to a flaw in its filtering logic. Specifically, it scans the entire `raw_addr` buffer to determine if the address is all-zeros or all-ones. When the address includes a port, as in the `MR_WITH_PORT` context, the raw_addr buffer includes both the IP and the 2-byte port. If the port is non-zero or not 0xFFFF, the scan fails to match the all-zero or all-one patterns, allowing these otherwise invalid or reserved addresses to be marked as "learnable." This behavior permits pre-authentication state creation or retention for these tuples, which can be passed to --learn-address hooks, effectively polluting the authentication state with addresses that should be considered invalid.

The same is true for IPv6 addresses with `MR_WITH_PORT`.

Risk

While the issue is classified as low severity due to the limited exploitation impact and lack of direct code execution or privilege escalation, it still introduces unnecessary state complexity and potential attack surfaces. In environments where state tracking or address learning affects policy decisions or logging, this could result in misleading data, increased resource consumption, or reduced visibility of legitimate events.

Mitigation

We recommend modifying `mroute_learnable_address()` to validate only the IP portion of `raw_addr` when determining all-zeros or all-ones patterns, ignoring the port entirely. This option is straightforward to implement and preserves existing logic for legitimate addresses.

8.7 S1-7: Named pipe accepts remote clients and uses predictable tid in pipe name

Attack scenario	**An attacker wants to deploy an unauthorized configuration**
Classification	CWE-340: Generation of Predictable Numbers or Identifiers
Component	OpenVPN Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/83
Attack impact	T-8: Deploy unauthorized configuration
Severity	Low
Status	Mitigated
Fix Commit	05d0808ee65d68691b0133f5fc3c09bfdba5259d

Background

The OpenVPN interactive service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

In `interactive.c`, the Windows component creates and connects to a named pipe at `.\pipe\service_` during process startup. The pipe name includes the thread identifier, which an attacker could predict or brute force. The pipe creation does not specify `PIPE_REJECT_REMOTE_CLIENTS`, which allows remote systems to attempt connections if they can reach the host. The vulnerable window exists only during initialization and remains short, but an attacker who guesses the threat ID and wins the race could interfere with the expected handshake. The main service does not verify that the expected workerthread connected and executes all messages that the attacker sends.

Risk

Because the per-thread pipe is created with a `NULL` security descriptor and `cMaxInstances = 1`, there exists a narrow timing window where an attacker could race the spawned client process. Because the named pipe is missing the `PIPE_REJECT_REMOTE_CLIENTS` flag, this attack can be performed from remote.

There are two primary risks.

Denial of Service:

If the attacker connects to the before the worker calls `CreateFile`**,** the worker's `CreateFile` fails, which is treated as a fatal error and thus results in a process exit.

Spoofing / message injection:

If an attacker connects to the pipe after the worker calls `CreateFile`, then the worker will accept and process messages. In that case the attacker could send messages that HandleMessage() will execute with the worker's privileges (route adds, DNS changes, WFP blocks, etc.).

Mitigation

As mitigation, we propose three options:

1. Set `PIPE_REJECT_REMOTE_CLIENTS` when creating the pipe to restrict connections to the local machine; this change reduces exposure to remote clients and keeps the trust boundary local. Then, apply a restrictive security descriptor via `SECURITY_ATTRIBUTES` that grants access only to `SYSTEM` and the intended service account. This descriptor blocks unauthorized local clients

and hardens the endpoint, though it adds configuration complexity and can cause deployment friction if service identities vary.

2. Replace the tid-based suffix with a high-entropy, cryptographically random identifier or include an unguessable nonce in the pipe name or handshake; this closes the predictability gap at the cost of additional state management and logging complexity.
3. Validate client identity after connection and before processing messages from `ovpn_pipe`. Call `GetNamedPipeClientProcessId(ovpn_pipe, &pid)` and compare the PID of the `proc_info` to the launched child. If it doesn't match, close the pipe and fail the session.

8.8 S1-8: DeleteWfpBlock restores metrics to incorrect interface, enabling routing corruption and DoS

Attack scenario	**An attacker wants to disrupt VPN connection**
Classification	CWE-706: Use of Incorrectly-Resolved Name or Reference
Component	OpenVPN Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/84
Attack impact	T-1: Prevent or degrade VPN connectivity
Severity	Low
Status	Mitigated
Fix Commit	a1d4a79487ec0089ee01ab635921761933e4e1d6

Background

The OpenVPN interactive service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

In src/openvpnserv/interactive.c, `DeleteWfpBlock()` removes the first `wfp_block_data_t` via `RemoveListItem` with `CmpAny(NULL)` and then restores interface metrics using `msg->iface.index` rather than the interface index recorded in the popped `block_data`:

```
static DWORD
DeleteWfpBlock(const wfp_block_message_t *msg, undo_lists_t *lists)
{
    DWORD err = 0;
    wfp_block_data_t *block_data = RemoveListItem(&(*lists)[wfp_block], CmpAny,
NULL);

    if (block_data)
    {
        err = delete_wfp_block_filters(block_data->engine);
        if (block_data->metric_v4 >= 0)
        {
            set_interface_metric(msg->iface.index, AF_INET, block_data-
>metric_v4);
        }
        if (block_data->metric_v6 >= 0)
        {
            set_interface_metric(msg->iface.index, AF_INET6, block_data-
>metric_v6);
        }
        free(block_data);
    }
    // …
}
```

A malicious or corrupted message can set `msg->iface.index` to an arbitrary interface. The function then writes the stored metrics from a different adapter to that arbitrary interface while the actually blocked adapter remains pinned to `WFP_BLOCK_IFACE_METRIC`. This behavior corrupts routing state, because traffic continues to prefer the VPN interface and another interface unexpectedly inherits the old metric values. The function also frees the popped `block_data` after restoring metrics on a mismatched interface, which discards the only correct record for the originally blocked interface and prevents a proper restore later.

Risk

A local attacker who can trigger `DeleteWfpBlock` with a crafted message can redirect metric restores to an arbitrary Network Interface Controller (NIC) and keep the actually blocked interface stuck at the forced metric. This causes persistent routing preference skew, breaks failover logic, and produces a simple denial of service by degrading or blackholing traffic over the targeted paths. Because the function releases the mismatched record, operators lose the ability to cleanly unwind the block for the original interface without manual repair. Systems that rely on stable adapter metrics for VPN enforcement, split tunneling, or management plane access face elevated impact.

Mitigation

We recommend changing the removal to match by interface so the function restores and frees the same record it removes: Replace `CmpAny(NULL)` with a comparator keyed by the interface index and call `RemoveListItem` with that key, for example:

```
RemoveListItem(&(*lists)[wfp_block], CmpByIface, &msg->iface.index)
```

Additionally, we recommend trusting the record, not the message - by store `iface_index` in `wfp_block_data_t` and restore via `block_data->iface_index` instead of msg->iface.index.

8.9 S1-9: Case-insensitive ccd hijack

Attack scenario	**An attacker wants to impersonate another user**
Classification	CWE-178: Improper Handling of Case Sensitivity
Component	OpenVPN Server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/91
Attack impact	T-4: Impersonate a participant
Severity	Low
Status	Mitigated
Fix Commit	8d278223df96e74e9b7ad8ae962ac28761a6fb19

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

OpenVPN derives the `client-config-dir` path from the certificate common name without case normalization. `platform_gen_path()` copies the Common Name (CN) and replaces path separators only, which leaves `Alice` and `alice` as distinct strings while producing paths that resolve to the same file on case-insensitive filesystems such as default HFS+ or APFS and Windows NTFS. During connect, `multi_client_connect_source_ccd()` probes `<ccd_dir>/<CN>` and then default and imports whichever file exists into the client's option set via `options_server_import()`. On a case-insensitive filesystem, `ccd/Alice` and `ccd/alice` refer to the same on-disk file, so an attacker who presents a cert with a case-variant CN inherits the victim's Client Config Directory (CCD) defined settings, including static IP and iroute ownership. The duplicate-session guard in `multi_delete_dup()` relies on `strcmp()`, which too treats `Alice` and `alice` as different users, so the server keeps both sessions alive while both consume the same CCD data and virtual IP. CCD-exclusive authorization checks in `verify_final_auth_checks()` also call `platform_gen_path()` and therefore accept case-variant CNs when the filesystem claims the CCD file exists, which collapses CCD-based admission to case-insensitive matching.

Risk

An attacker who obtains or mints a certificate with a case-variant CN can impersonate a targeted user's CCD profile on servers that host CCDs on case-insensitive filesystems. The attacker can receive the victim's static address, appear as the route owner for the victim's networks, intercept or misdirect traffic for those routes, and confuse monitoring or status output. Because duplicate detection does not recognize the conflict, the server allows both sessions to persist, which enables traffic interception, address collision, and privilege inheritance without access to the victim's private key. Deployments that gate access using CCD-existence or CCD-execute semantics also degrade, since a case-variant CN passes the same existence check and bypasses intended per-identity admission control.

As this vulnerability requires bad practices on the part of the certificate issuer, we consider this a low severity.

Mitigation

Normalize and bind identity consistently at every boundary that uses the CN.

Apply a single canonicalization strategy, such as Unicode casefolding to lower case, when constructing CCD paths in `platform_gen_path()`, when performing CCD lookups in

`multi_client_connect_source_ccd()`, and when comparing identities in `multi_delete_dup()` and related duplicate checks.

8.10 S1-10: FormatMessageW uses byte length -> stack overflow in BlockDNSErrHandler

Attack scenario	**An attacker wants to execute arbitrary code as a privileged user**
Classification	CWE-121: Stack-based Buffer Overflow
Component	OpenVPN Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/92
Attack impact	T-6: Execute Arbitrary Code or Scripts
Severity	Low
Status	Mitigated
Fix Commit	b39bed539f218f6165d79224adbc343a63a4514d

Background

The OpenVPN interactive service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

BlockDNSErrHandler in src/openvpnserv/interactive.c (lines 725–742) calls `FormatMessage` with a stack buffer declared as `WCHAR buf[256]` but passes `sizeof(buf)` as the size parameter. In a Unicode build `FormatMessageW` expects a character count, while `sizeof(buf)` reports bytes (512). This lets `FormatMessageW` write up to 512 `WCHAR`s into a 256-`WCHAR` buffer when the system error message is long:

```
WCHAR buf[256];
[...]
if (FormatMessage(FORMAT_MESSAGE_IGNORE_INSERTS | FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_ARGUMENT_ARRAY, NULL, err, 0, buf, sizeof(buf), NULL))
[...]
```

An attacker who can send crafted `wfp_block_message_t` requests to the OpenVPN interactive service pipe and force `add_wfp_block_filters` to fail with an attacker-influenced Win32 error can trigger a long system message that overruns buf, corrupts the stack of the SYSTEM service process, and causes crashes or potentially arbitrary code execution.

The risk is low since an attacker should not be able to send such messages, and executing the attack depends on other reported security issues reported by us. Despite this, we recommend fixing such issues.

Risk

An attacker with access to the OpenVPN interactive service pipe and the ability to submit `WFP_BLOCK` requests can induce a controlled failure and provoke a long system error string resulting in a memory corruption issue.

Mitigation

Correct the `FormatMessage` usage by supplying a character count rather than byte size and replace the `sizeof(buf)` argument with the buffer element count. For example: `sizeof(buf)`/`sizeof(buf[0])` or a platform-safe `_countof` macro.

8.11 S1-11: Attacker can trivially bypass reject pull filter

Attack scenario	**An attacker wants to deploy an unauthorized configuration**
Classification	CWE-693 – Protection Mechanism Failure
Component	OpenVPN Client
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/99
Attack impact	T-8: Deploy unauthorized configuration
Severity	Low
Status	Mitigated
Fix Commit	d3e03b9a97177f62d31697f2b4b453295ee30e60

Background

The OpenVPN client is the client implementation of the OpenVPN protocol.

Issue description

OpenVPN allows connection options to be pushed from the server or pulled by the client, enabling dynamic configuration options. Through pull filters, a client can have fine-grained access control for server pushed configurations. The server allows `reject`, `ignore` and `accept` filters for certain configuration options. These filters also allow for parameters to be specified, allowing nuanced filtering strategies.

Due to the usage of `strncmp` in apply_pull_filter, the `reject` filters with parameters are trivially bypassed.

For example: here's a an example route filter

```
--pull-filter  "reject route 10.0.0.0"
```

This can be trivially bypassed by adding an extra space before `10.0.0.0`

```
PUSH_UPDATE,route  10.0.0.0
```

Risk

Despite the fact that pull-filters are not typically used as a security measure, this is not discouraged in the documentation. If `reject` pull-filters are utilized in a security context, then an attacker can trivially bypass them. This may result in information leakage, if an attacker can push `route` or `dns`, or denial of service if they can configure `keepalive`.

Mitigation

As mitigation, we recommend explicitly discouraging the use of nuanced pull-filters as a security measure and instead, encourage an allowlist based approach; where all pushed options are `reject` or `ignore` by default and only explicit options are `accept`.

8.12 S1-12: multi_process_file_closed unsafe loop boundary on failed read

Attack scenario	**An attacker wants to prevent or degrade VPN connectivity**
Classification	CWE-195: Signed to Unsigned Conversion Error
Component	OpeVPN Server
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/101
Attack impact	T-8: Prevent or degrade VPN connectivity
Severity	Low
Status	Mitigated
Fix Commit	18d1b1fe008a4bbfd5a56ca7bb59f6f8cb58114a

Background

The OpenVPN server is the server implementation of the OpenVPN protocol.

Issue description

The function `multi_process_file_closed` reads from an `inotify` file descriptor into a fixed-size buffer and stores the return value of `read` in the signed variable `r`. The code then uses `buffer_i`, an unsigned `size_t`, as the loop index with the condition `buffer_i < r`:

```
void
multi_process_file_closed(struct multi_context *m, const unsigned int mpp_flags)
{
    char buffer[INOTIFY_EVENT_BUFFER_SIZE];
    size_t buffer_i = 0;
    int r = read(m->top.c2.inotify_fd, buffer, INOTIFY_EVENT_BUFFER_SIZE);

    while (buffer_i < r)
    {
        /* parse inotify events */
        struct inotify_event *pevent = (struct inotify_event
*)&buffer[buffer_i];
        size_t event_size = sizeof(struct inotify_event) + pevent->len;
        buffer_i += event_size;

        msg(D_MULTI_DEBUG, "MULTI: modified fd %d, mask %d", pevent->wd, pevent-
>mask);

        struct multi_instance *mi =
            hash_lookup(m->inotify_watchers, (void *)(unsigned long)pevent->wd);

        if (pevent->mask & IN_CLOSE_WRITE)
        {
            ...
        }
        else if (pevent->mask & IN_IGNORED)
        {
            ...
        }
    }
}
```

When read fails and returns a negative value, the implicit comparison between an unsigned and a signed integer causes `r` to undergo integer promotion into a large unsigned value. The loop condition then evaluates as true, and the code attempts to parse events from uninitialized memory. The loop

will most likely crash when it casts invalid buffer contents to `inotify_event` structures and advances `buffer_i` beyond the buffer boundary.

Risk

The code risks out-of-bounds memory access, undefined behavior, and process termination when read returns a negative value. A local attacker who can trigger read failures may force the service into a crash, causing a denial of service. The issue does not expose memory contents or enable privilege escalation, but it reduces reliability and increases the attack surface for further exploitation attempts.

Mitigation

The code should explicitly check the return value of `read` before entering the loop and should exit early when `r` is less than or equal to zero.

Another option would be changing `buffer_i` to a signed integer `ssize_t`.

8.13 S1-13: Plugin trusted-directory check allows prefix path bypass

Attack scenario	**An attacker wants to execute arbitrary code**
Classification	CWE-20: Improper Input Validation
Component	OpenVPN Plugin Interface
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/109
Attack impact	T-6: Deploy unauthorized configuration
Severity	Low
Status	Acknowledged
Fix Commit	-

Background

OpenVPN provides a plugin interface for external developers to extend OpenVPN's functionality in a modular manner.

Issue description

The `HKLM\SOFTWARE\OpenVPN\plugin_dir` registry key is an allow-list for where plugins can be loaded from, and is meant as a security measure. If the directory path specified does not end with a trailing backslash (e.g. the registry key is set to `C:\\openvpn_plugins`), because the check is only performed as a prefix match, plugins can be loaded from a sibling directory with the same prefix (e.g. `C:\\openvpn_plugins_evil`).

```
    /* Attempt to retrieve the trusted plugin directory path from the registry,
     * using installation path as a fallback */
    if (!get_openvpn_reg_value(L"plugin_dir", plugin_dir, _countof(plugin_dir))
        && !get_openvpn_reg_value(NULL, plugin_dir, _countof(plugin_dir)))
    {
        msg(M_WARN, "Installation path could not be determined.");
    }

    [...]

    WCHAR normalized_plugin_dir[MAX_PATH] = { 0 };

    /* Normalize the plugin dir */
    if (wcslen(plugin_dir) > 0)
    {
        if (!GetFullPathNameW(plugin_dir, MAX_PATH, normalized_plugin_dir,
NULL))
        {
            msg(M_NONFATAL | M_ERRNO, "Failed to normalize plugin dir.");
            return false;
        }
    }

    /* Check if the plugin path resides within the plugin/install directory */
    if ((wcslen(normalized_plugin_dir) > 0)
        && (wcsnicmp(normalized_plugin_dir, plugin_path,
wcslen(normalized_plugin_dir)) == 0))
    {
        return true;
    }
```

Risk

An attacker who can run OpenVPN with arbitrary configuration on a setup where a `plugin_dir` is specified but missing a trailing slash and who is unable to write to the directory specified but is able to write to a sibling directory with the specified directory's name as a name prefix may exploit this vulnerability to achieve code execution in OpenVPN's context and possible privilege escalation.

Mitigation

We recommend using the `PathIsPrefixW` API to determine whether the plugin to be loaded is in the allow-list directory, rather than prefix matching via `wcsnicmp`, in the `plugin_in_trusted_dir` function in `src/openvpn/win32.c`.

8.14 S1-14: iproute2 networking backend ignores route table ID; doesn't qualify gateway/interface when deleting routes

Attack scenario	**An attacker wants to leak VPN traffic**
Classification	CWE-440: Expected Behavior Violation
Component	OpenVPN Client
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/110
Attack impact	T-5: Violate traffic integrity
Severity	Low
Status	Acknowledged
Fix Commit	-

Background

The OpenVPN Client is the client implementation of the OpenVPN protocol.

Issue description

In src/openvpn/networking_iproute2.c, the `iproute2` networking backend contains two issues regarding parameter handling:

- The `table` parameter is ignored across the entire backend; all `net_route_*` functions fail to append `table <id>` to the executed `ip` command, causing all routes to default to the main table even when a custom table is requested.
- Specifically within `net_route_v4_del`, the gateway (`gw`) and interface (`iface`) parameters are ignored. While the route addition logic correctly appends `via <gw>` and `dev <iface>`, the deletion logic omits them, resulting in a generic `ip route del <network>` command.

It should be noted that iproute2 is not the default configuration and needs to be enabled at compile-time (with --enable-iproute2). It is mainly provided for compatibility reasons with older setups depending on the “ip” command execution for local purposes. As such, new features like the routing table ID - introduced in 2.7.0 - do not always propagate immediately to older mechanisms.

Risk

The lack of `table` support means OpenVPN configurations using policy-based routing (e.g., `--route-table`) will silently fail to isolate VPN traffic, installing routes into the system's main table instead. This can lead to routing conflicts or traffic leaking outside intended policy scopes.

Additionally, the failure to qualify route deletion with `via` or `dev` creates ambiguity. In complex network setups, OpenVPN may inadvertently delete a matching route belonging to a different interface or gateway, causing a Denial of Service for legitimate non-VPN traffic.

Mitigation

We recommend updating `src/openvpn/networking_iproute2.c` to fully implement the `iproute2` command spec, updating all `net_route_*` functions to check if `table` is non-zero and append `table %u` to the command arguments; and updating `net_route_v4_del` specifically to conditionally append `via %s` (if `gw` is set) and `dev %s` (if `iface` is set), matching the existing logic in `net_route_v4_add`.

8.15 S0-15: Invalid incrementing of addrs offset in HandleDNSConfigMessage

Attack scenario	-
Classification	CWE-682: Incorrect Calculation
Component	Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/75
Attack impact	-
Severity	Info
Status	Mitigated
Fix Commit	235aa8858b9999c14603ff1d12c91c855d5dcf94

Background

The OpenVPN Interactive Service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

OpenVPN's interactive service is a Windows service that is run with maximum local privileges. It interfaces with OpenVPN-GUI to allow non privileged users to run openvpn.exe.

The `HandleDnsConfigMessage` takes in an array of DNS messages and adds them to the list of DNS servers available to the client.

While iterating over the list of addresses, the offset is incremented by `strlen(addrs)` instead of just the newly appended data.

```
CHAR addrs[_countof(msg->addr) * 64]; /* 64 is enough for one IPv4/6 address */
size_t offset = 0;
for (int i = 0; i < addr_len; ++i)
{
    if (i != 0)
    {
        addrs[offset++] = ',';
    }
    if (msg->family == AF_INET6)
    {
        RtlIpv6AddressToStringA(&msg->addr[i].ipv6, addrs + offset);
    }
    else
    {
        RtlIpv4AddressToStringA(&msg->addr[i].ipv4, addrs + offset);
    }
    offset += strlen(addrs);
}
```

Risk

While the incrementing offset is invalid, it is safe due to the limit of the number of address (4) and the fact that the maximum size of an ipv6 address is 38 + 1 bytes. However, this issue can have consequences if APIs or configurations change in the future.

Firstly, if the number of addresses is increased, this will likely result in a heap-buffer-overflow. Additionally, if the `RtlIpv6AddressToStringA` function removes the truncation functionality, or if it is replaced by another address to string function, this will also likely result in a heap buffer overflow.

Mitigation

For mitigation, we recommend assigning `offset` to `strlen(addrs)`, instead of incrementing it with `strlen(addrs)`.

8.16 S0-16: Incorrect undo logic in HandleDnsConfigMessage may lead to DNS resolution failures

Attack scenario	-
Classification	CWE-697: Incorrect Comparison
Component	OpenVPN Interactive Service
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/78
Attack impact	-
Severity	Info
Status	Mitigated
Fix Commit	59fed2ad127bda2986f4c70d128453740ca63d34

Background

The OpenVPN interactive service allows un-privileged windows users to perform limited privileged actions that are required to initiate and configure their VPN connection.

Issue description

OpenVPN's Interactive Service on Windows provides an interface to handle VPN configuration as a way to allow users without administrative access to use OpenVPN. The DNS configuration handler supports two types of messages:

1. allows clearing DNS name servers of a given family (IPv6/IPv4)
2. allows setting new name servers of a given family (IPv6/IPv4). This also explicitly clears existing name servers.

Since OpenVPN supports both IPv4 and IPv6 DNS name servers and since name servers for both are stored in different Registry fields, the service must differentiate when assigning and clearing name servers.

However, the ternary to select which entry to clear is inverted, resulting in ipv6 addresses being cleared when ipv4 addresses are being configured and vice-versa.

```
undo_type_t undo_type = (msg->family == AF_INET6) ? undo_dns4 : undo_dns6;
```

Risk

This issue could result in failed DNS resolutions in a scenario where both IPv6 and IPv4 DNS name servers are utilized. Any DNS name-server configuration of either IPv4 or IPv6 would override the other.

Mitigation

For mitigation, we recommend inverting the ternary operator to make sure that if the family is `AF_INET6`, that it chooses undo_dns6.

For example:

```
undo_type_t undo_type = (msg->family == AF_INET6) ? undo_dns6 : undo_dns4;
```

8.17 S0-17: Null pointer dereference in dns-updown option handling

Attack scenario	-
Classification	CWE-476: NULL Pointer Dereference
Component	OpenVPN Client
Tracking	https://github.com/OpenVPN/openvpn-private-issues/issues/108
Attack impact	-
Severity	Info
Status	Mitigated
Fix Commit	62a17417de26735e04cb527c5df8137e4d50454a

Background

The OpenVPN client is the client implementation of the OpenVPN protocol.

Issue description

An OpenVPN client can configure `dns-updown` script to run to handle DNS configurations.

Within `dns-updown` there are three variants available: `disable`, `force` and a custom script.

The `disable` variant sets the value of `dns->updown` to `NULL`

```
if (streq(p[1], "disable")) {
    dns->updown = NULL;
    dns->updown_flags = DNS_UPDOWN_NO_FLAGS;
}
```

If a configuration file configures `dns-updown` as `disabled`, which sets `dns->updown` to `NULL` and later configures it to a `custom script`, the string comparison as of the `dns-updown` value will result in a null pointer dereference.

```
if (streq(dns->updown, DEFAULT_DNS_UPDOWN)) {
}
```

`streq` is an alias for `strcmp`

```
#define streq(x, y) (!strcmp((x), (y)))
```

Risk

A null pointer dereference would result in the termination of the process without an error message, confusing client operators.

Mitigation

We recommend adapting the conditional to handle the case where `updown` could be `NULL`.

```
if (dns->updown != NULL && streq(dns->updown, DEFAULT_DNS_UPDOWN))
```

8.18 S0-18: PAM blocks entire openvpn thread if not configured otherwise

Attack scenario	-
Classification	CWE-1322: Use of Blocking Code in Single-threaded, Non-blocking Context
Component	PAM Plugin
Tracking	https://github.com/OpenVPN/openvpn/issues/893
Attack impact	T: Prevent or degrade VPN connectivity
Severity	Info
Status	Acknowledged
Fix Commit	-

Background

Pluggable Authentication Module (PAM) is an interface which allows an application to implement support for authentication in a provider agnostic manner. OpenVPN implements PAM support for connection authentication.

Issue description

OpenVPN implements PAM support as a plugin which allows server administrators to implement support for various third-party authentication schemes. When an authentication reuqest is initiated, OpenVPN blocks the server thread, limited by a timeout, when awaiting a response from the PAM authentication provider. This is the default case. If an administrator wishes to perform the PAM authentication asynchronously, they must set `deferred_auth_pam` to `yes` in their OpenVPN configuration.

Third party providers may become unresponsive or malicious due to a fault in their service. In case an administrator forgets or is unaware of `deferred_auth_pam`, they risk a denial-of-service attack due to the blocking of the server thread.

Risk

The OpenVPN server risks denial of service due to an unresponsive or malicious third-party PAM server.

Mitigation

As mitigation, we recommend setting `deferred_auth_pam` to `true` by default.

9 Bibliography

[1] [Online]. Available: https://github.com/OpenVPN/openvpn/tree/v2.7_rc1.

[2] [Online]. Available: https://github.com/google/oss-fuzz/tree/master/projects/openvpn.

[3] [Online]. Available: https://github.com/openvpn/openvpn-private-issues.

[4] [Online]. Available: https://introspector.oss-fuzz.com/project-profile?project=openvpn.

Appendix A: Threat categories

Category	Description
T-1: Prevent or degrade VPN functionality	VPN traffic is disrupted or throttled, causing denial of service that prevents legitimate users from establishing or maintaining secure tunnel connections.
T-2: Exploit the VPN process to escalate privileges locally	Vulnerabilities in the OpenVPN process are leveraged to gain elevated system privileges beyond those intended for the VPN service account.
T-3: Gain unauthorized information from the VPN server	Sensitive data such as credentials, session keys, configuration details, or client metadata is extracted from the VPN server through exploitation or misconfiguration.
T-4: Impersonate a participant	A legitimate VPN client, server, or certificate authority is impersonated to intercept communications or gain unauthorized network access.
T-5: Violate traffic integrity	Data within the VPN tunnel is tampered with, modified, or injected without detection, compromising the trustworthiness of transmitted information.
T-6: Execute arbitrary code or scripts	Remote or local code execution is achieved on the VPN server or client by exploiting input handling flaws, plugin mechanisms, or script hooks.
T-7: Perform actions without appropriate logging	Malicious activities are carried out while evading detection through disabled, bypassed, or manipulated audit logs and monitoring mechanisms.
T-8: Deploy unauthorized configuration	VPN server or client configurations are pushed or modified to weaken security controls, redirect traffic, or introduce backdoor access.
T-9: Downgrade protocol security	The VPN connection is forced to negotiate weaker ciphers, older protocol versions, or less secure authentication methods, making traffic easier to intercept or decrypt.

Appendix B: Technical services

Security Research Labs delivers extensive technical expertise to meet your security needs. Our comprehensive services include software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. We aim to equip your organization with the security knowledge essential for achieving your objectives.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of applications. Our work includes design and architecture reviews, data flow and threat modelling, and code analysis with targeted fuzzing to find exploitable issues.

BLOCKCHAIN SECURITY ASSESSMENTS We offer specialized security assessments for blockchain technologies, focusing on the unique challenges posed by decentralized systems. Our services include smart contract audits, consensus mechanism evaluations, and vulnerability assessments specific to blockchain infrastructure. Leveraging our deep understanding of blockchain technology, we ensure your decentralized applications and networks are secure and robust.

POLKADOT ECOSYSTEM SECURITY We provide comprehensive security services tailored to the Polkadot ecosystem, including parachains, relay chains, and cross-chain communication protocols. Our expertise covers runtime misconfiguration detection, benchmarking validation, cryptographic implementation reviews, and XCM exploitation prevention. Our goal is to help you maintain a secure and resilient Polkadot environment, safeguarding your network against potential threats.

TELCO SECURITY We deliver specialized security assessments for telecommunications networks, addressing the unique challenges of securing large-scale and critical communication infrastructures. Our services encompass vulnerability assessments, secure network architecture reviews, and protocol analysis. With a deep understanding of telco environments, we ensure robust protection against cyberthreats, helping maintain the integrity and availability of your telecommunications services.

DEVICE TESTING Our comprehensive device testing services cover a wide range of hardware, from IoT devices and embedded systems to consumer electronics and industrial controls. We perform rigorous security evaluations, including firmware analysis, penetration testing, and hardware-level assessments, to identify vulnerabilities and ensure your devices meet the highest security standards. Our goal is to safeguard your hardware against potential attacks and operational failures.

CODE AUDITING We provide in-depth code auditing services to identify and mitigate security vulnerabilities within your software. Our approach includes thorough manual reviews, automated static analysis, and targeted fuzzing to uncover critical issues such as logic flaws, insecure coding practices, and exploitable vulnerabilities. By leveraging our expertise in secure software development, we help you enhance the security and reliability of your codebase, ensuring robust protection against potential threats.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated adversaries. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team a sense of the overall security posture of your organization.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology that lets us find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This gives your team a stronger assurance that the significant security-impacting flaws have been found and corrected.

SECURITY DEVELOPMENT LIFECYCLE CONSULTING We guide organizations through the Security Development Lifecycle to integrate security at every phase of software development. Our services include secure coding training, threat modelling, security design reviews, and automated security testing implementation. By embedding security practices into your development processes, we help you proactively identify and mitigate vulnerabilities, ensuring robust and secure software delivery from inception to deployment.

REVERSE ENGINEERING We assist clients with reverse engineering efforts that are not associated with malware or incident response. We also provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products to core networking equipment that powers Internet backbones.

VULNERABILITY PRIORITIZATION We streamline vulnerability information processing by consolidating data from compliance checks, audit findings, penetration tests, and red team insights. Our prioritization and automation strategies ensure that the most critical vulnerabilities are addressed promptly, enhancing your organization's security posture. By systematically categorizing and prioritizing risks, we help you focus on the most impactful threats, ensuring efficient and effective remediation efforts.

SECURITY MATURITY REVIEW We conduct comprehensive security maturity reviews to evaluate your organization's current security practices and identify areas for improvement. Our assessments cover a wide range of criteria, including policy development, risk management, incident response, and security awareness. By benchmarking against industry standards and best practices, we provide actionable insights and recommendations to enhance your overall security posture and guide your organization toward achieving higher levels of security maturity.

SECURITY TEAM INCUBATION We provide comprehensive support for building security teams for new, large-scale IT ventures. From Day 1, our ramp-up program offers essential security advisory and assurance, helping you establish a robust security foundation. With our proven track record in securing billion-dollar investments and launching secure telco networks globally, we ensure your new enterprise is protected against cyberthreats from the start.

HACKING INCIDENT SUPPORT We offer immediate and comprehensive support in the event of a hacking incident, providing expert analysis, containment, and remediation. Our services include detailed forensics, malware analysis, and root cause determination, along with actionable recommendations to prevent future incidents. With our rapid response and deep expertise, we help you mitigate damage, recover swiftly, and strengthen your defenses against potential threats.