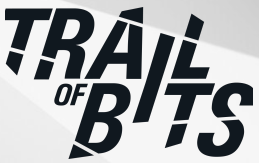

OpenVPN2

Security Assessment

August 25, 2023

Prepared for:
Robert Weiss
OpenVPN Inc.

Prepared by: **Artur Cygan, Hamid Kashfi, Anders Helsing, and Dominik Czarnota**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

OpenVPN, Inc. engaged Trail of Bits to review the security of the OpenVPN server (commit 77829be280). From October 24 to November 4, 2022, a team of two consultants conducted a security review of the OpenVPN source code, with four person-weeks of effort. Additionally, from December 12 to December 16, a team of two consultants performed additional analysis. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

In addition to this report, we are also providing pull request OpenVPN/openvpn#208 which fixes the broken build system of fuzzing harnesses from the oss-fuzz repository along with new fuzzing harnesses and changes that improve the fuzzing's code coverage.

Project Scope

The primary goal for this audit was to discover remotely exploitable vulnerabilities in the OpenVPN server that could result in a compromise of confidentiality, integrity, or availability of the server.

We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed static and dynamic analysis of the target system, and its codebase, using both automated and manual processes.

Summary of Findings

The audit did not uncover any significant flaws that could impact system confidentiality, integrity, or availability in the time provided.

EXPOSURE ANALYSIS

Severity	*Count*
Informational	7
Undetermined	3

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	3
Denial of Service	1
Documentation	1
Configuration	1
Cryptography	1
Testing	1
Undefined Behavior	2

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Artur Cygan, Consultant
artur.cygan@trailofbits.com

Hamid Kashfi, Consultant
hamid.kashfi@trailofbits.com

Anders Helsing, Consultant
anders.helsing@trailofbits.com

Dominik Czarnota, Consultant
dominik.czarnota@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 20, 2022	Pre-project kickoff call
November 2, 2022	Status update meeting #1
November 7, 2022	Delivery of report draft and report readout meeting
February 23, 2023	Delivery of final report
May 12, 2023	Report updates based on further analysis
August 25, 2023	Delivery of final report with fix review

Project Goals

The engagement was scoped to provide a security assessment of the OpenVPN server. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are there any memory corruption vulnerabilities that allow attackers to perform code execution, crash the program, or leak sensitive data?
- What is the state of fuzzing of the project? Are fuzzing harnesses implemented correctly in order to achieve maximum coverage?
- Are there any general issues related to C language that are easily detectable by automated static analysis tools? Can we construct rules that are tailored to OpenVPN specifically?
- What are the main modes in which OpenVPN operates, and which authentication and configuration methods are supported? Are user-controllable parameters properly handled and sanitized in the OpenVPN implementation?
- How are authentication modes implemented, are there any differences or gaps in their implementation?
- Are there any common patterns of bugs that have been identified in prior bug reports? Could static analysis be implemented to help prevent such bug patterns in future?
- How can OpenVPN improve its security in the long term? What are the biggest opportunities?

Project Targets

The engagement involved a review and testing of the following target:

OpenVPN2

Repository	https://github.com/OpenVPN/openvpn
Version	77829be280b3b280f35d1ca4947900f3c5e5dd26
Type	C
Platform	Multi-platform, native (Linux, Windows, macOS, *BSD)

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches include the following:

- Static analysis of the entirety of the code with scan-build, CodeQL, Weggli, and PVS-Studio
- Fixing, improving, and running the existing fuzzing harnesses
- Development of new fuzzing harnesses
- Manual review of the code nearby the issues reported from the status analysis tools and code covered by the fuzzing harnesses

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. A cursory review of the codebase revealed that the codebase lacked robust static analysis and fuzzing. We agreed with the OpenVPN team that we would focus on improving the current state of fuzzing and provide a best-effort source code review driven by static analysis results.

While this approach was well suited to the goals, time, and resources available for the engagement, manual review of the codebase is nonetheless important. In conjunction with the new static analysis and fuzzing support, we recommend additional manual code reviews preceded by threat modeling to uncover vulnerabilities that are hard to discover using other approaches. Additionally, while we strived to maximize coverage with our fuzzing efforts, some fuzzing harnesses could still be improved, as detailed in Appendix E.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in-house, to perform automated testing of source code and compiled software.

Tools Used

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
LibFuzzer	An in-process, coverage-guided, evolutionary fuzzing engine. LibFuzzer can automatically generate a set of inputs that exercise as many code paths in the program as possible.	E. Fuzzing OpenVPN
CodeQL	A code analysis engine developed by Semmle (now GitHub) to automate security checks.	D.1. CodeQL
Weggli	An open-source static analysis tool for finding bugs in C and C++ programs.	D.2. Weggli
Scan-build	A static analysis tool based on Clang Static Analyzer that finds bugs in C, C++ and Objective-C programs.	The tool was run as-is with: `scan-build make`.
PVS-Studio	A commercial static analysis tool for finding bugs in C and C++ programs.	All C++ rules were used.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We did not find any arithmetic-related issues such as integer overflows. However, the project does not take any specific measures to ensure arithmetic safety. Extended fuzzing efforts could uncover additional arithmetic issues. Different integral types are used, for example to represent lengths of buffers, and in many cases 64-bit integer arguments are truncated to 32-bit (TOB-OVPN-12). Although we have not found issues resulting from integer conversion, it is a common source of security vulnerabilities.	**Moderate**
Auditing	OpenVPN has implemented extensive logging and auditing features for critical code paths.	**Strong**
Authentication / Access Controls	We found no issues with the OpenVPN server authentication methods performed by clients. However, OpenVPN does not implement TLS for proxy authentication traffic, leaving the traffic, and the credentials contained in the traffic, vulnerable to disclosure and manipulation attacks (TOB-OVPN-11). Additionally, the code supports outdated, weak protocols and algorithms for proxy authentication, such as NTLMv1 and DES (TOB-OVPN-7). We recommend implementing TLS support for proxy authentication and adding warnings to the documentation about the risks of credential exposure if a proxy is used without TLS. We also recommend removing support for NTLMv1 and DES.	**Satisfactory**
Complexity	The code is split into reasonable functions and modules.	**Satisfactory**

Management	Some files, such as `options.c` or `tun.c`, are extremely large and could benefit from splitting.	
Configuration	The OpenVPN project mainly leaves it up to the user to determine best practices related to security configurations and security hardening options. We recommend enabling compiler security mitigations in the build system (TOB-OVPN-13); adding guidance to the user documentation on how to securely set up OpenVPN; and communicating the risks involved in deviating from that guidance.	**Moderate**
Cryptography and Key Management	We found no issues in the implementation of cryptography or key management.	**Satisfactory**
Data Handling	While we found no significant issues with data handling, in some cases the APIs are not consistent. The reliance on implicit requirements (e.g., the size restriction of the `buffer` struct) could introduce bugs with future code changes.	**Satisfactory**
Documentation	The OpenVPN project lacks comprehensive documentation for the protocol and implementation of the code. A work in progress aims to improve protocol implementation and to create an RFC. Existing documentation is mainly based on doxygen, which is derived from code comments. Code comments in multiple areas also need improvement, as described in Appendix C: Non-Security-Related findings. Existing documentation also lacks recommendations and warnings regarding secure configurations, preferred encryption algorithms, supported (insecure) proxy authentication methods, and the potential risks related to them. In addition to improving the user documentation as outlined above, we also recommend continuing to pursue the OpenVPN wire protocol specification and RFC inclusion effort.	**Weak**

Maintenance	The build system is user friendly, extensible, and well adapted to work on multiple platforms. However, some minor improvements can be made to it (TOB-OVPN-10). Additionally, the project CI/CD should include static analysis tools to help detect potential issues with further code changes.	**Satisfactory**
Memory Safety and Error Handling	Errors are generally handled consistently within the codebase, though there were cases where a memory leak could happen due to incorrectly handled error paths (TOB-OVPN-2). We also uncovered an instance of a memory safety issue (TOB-OVPN-4). We recommend adding unit tests to cover unhappy paths and further extending of fuzzing to cover more code in the project. Additionally, we recommend using static analyzers in CI to help catch new problems.	**Moderate**
Testing and Verification	There are unit and integration tests in the project. Unit tests verify a smaller subset of implemented features and use an example-based approach with expected outcomes. Integration tests typically interact with the resulting `openvpn-binary` to verify functionality at a higher level, verifying user-facing features. Additionally, the fuzzing of the project, performed through the OSS-Fuzz project, is broken (i.e., it does not build), and fuzzing does not work on the reviewed code (TOB-OVPN-1). With a moderate fuzzing effort, we managed to find what can be considered relatively shallow bugs (TOB-OVPN-4). One of the existing fuzzing harnesses had a memory corruption bug itself that did not manifest until this audit (described in Appendix E). We recommend adopting a policy to ensure that the fuzzers are kept up to date and running.	**Weak**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Broken fuzzing harnesses	Testing	**Informational**
2	Certain error paths do not free allocated memory leading to memory leaks	Denial of Service	**Informational**
3*	<retracted>		
4	Stack buffer out-of-bounds read in command line options parsing	Undefined Behavior	**Informational**
5*	<retracted>		
6*	<retracted>		
7	Support of weak proxy authentication algorithms	Cryptography	**Informational**
8	Decoding username can silently cause truncated or empty username	Data Validation	**Informational**
9*	<retracted>		
10*	<retracted>		
11	Lack of TLS support by the HTTP and SOCKS proxy may lead to compromise of a user's proxy authentication credentials	Documentation	**Informational**
12	Implicit conversions that lose integer precision	Undefined Behavior	**Undetermined**

13	The OpenVPN build system does not enable compiler security mitigations	Configuration	**Undetermined**
14	The ntlm_phase_3 function does not verify if it received the correct length of the challenge data	Data Validation	**Informational**
15	The establish_http_proxy_passthru function proxy-authenticate header check is insufficient	Data Validation	**Undetermined**

* These findings had been previously reported in a provisional state. Further investigation determined that findings 5 and 6 were invalid; these have since been retracted. Findings 3, 9, and 10 were determined to be non-security-related and they have been moved either to Appendix C or Appendix G. These entries remain as placeholders in order to preserve the previously reported finding IDs.

Detailed Findings

1. Broken fuzzing harnesses	
Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-OVPN-1
Target: `google/oss-fuzz/projects/openvpn`	

Description

OpenVPN fuzzing is performed through the oss-fuzz project; however, the build has been broken since November 8, 2022, and the code has not been continuously fuzzed via oss-fuzz since that time (figure 1.1). We moved the fuzzing harnesses from the oss-fuzz project to the OpenVPN repository and fixed them in OpenVPN/openvpn#208. However, the oss-fuzz repository still needs to be updated in order to use the buildable fuzzing harnesses.

After measuring the code coverage these harnesses achieve, some were found to cover certain paths (case 0 in a switch statement) more than others, which can slow the process of obtaining proper code coverage. For example, the `fuzz_buffer` harness tests the `buf_clear` function substantially more often than any other cases it was supposed to test (figure 1.2). This happened due to bias in the return value of the `fuzz_randomizer_get_int` function, which we detail in Appendix E.

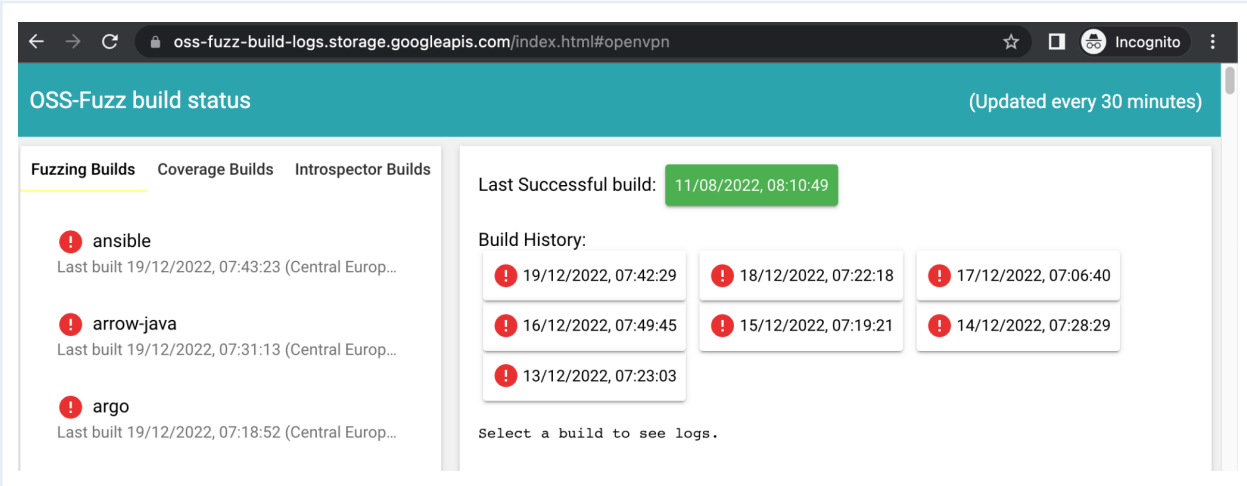

Figure 1.1: The OpenVPN project's oss-fuzz build status (https://oss-fuzz-build-logs.storage.googleapis.com/index.html#openvpn).

```
52  3.71k  #define NUM_TARGETS 32
53  3.71k        generic_ssizet = fuzz_randomizer_get_int(0, NUM_TARGETS);
54  3.71k        switch (generic_ssizet) {
55  1.86k        case 0:
56  1.86k          buf_clear(bufp);
57  1.86k          break;
58     52        case 1:
59     52          buf2 = clone_buf(bufp);
60     52          free_buf(&buf2);
61     52          break;
62     37        case 2:
63     37          buf_defined(bufp);
64     37          break;
65     89        case 3:
66     89          buf_valid(bufp);
67     89          break;
68     35        case 4:
69     35          buf_bptr(bufp);
70     35          break;
71     84        case 5:
72     84          buf_len(bufp);
73     84          break;
```

Figure 1.2: A screenshot of code coverage report for the `fuzz_buffer` harness. The columns represent: line number, number of hits by the fuzzer's corpus inputs, and the code lines. The first case (value 0) is hit more than 1,800 times, while the other cases are hit around tens of times.

Recommendations

Short term, build the updated fuzzing harnesses on the CI to ensure updates do not break their ability to build successfully. Update the OpenVPN project in the oss-fuzz repository to use the updated harnesses.

Long term, establish procedures for periodically reviewing and improving the coverage of existing fuzzing harnesses.

2. Certain error paths do not free allocated memory leading to memory leaks

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-OVPN-2
Target: `src/openvpn/{ssl_verify.c, console_systemd.c, lladdr.c}`	

Description

There are three cases where resources are not freed appropriately during error handling. These resources are allocated by either `argv_new` or `gc_new` (along with operations performed on the returned gc object which triggers the actual allocation since `gc_new` itself does not allocate):

- In the `verify_user_pass_script` function, the `gc`, `argv`, and `tmp_file` resources are not freed if the `key_state_gen_auth_control_files` call fails (figure 2.1). The code should jump to the `done` label after first setting the `retval` to `OPENVPN_PLUGIN_FUNC_ERROR`.
- In the `get_console_input_systemd` function, the allocation performed in `argv_new` will cause a memory leak due to lack of `argv_free` when the `openvpn_popen` call fails (figure 2.1).
- In the `set_lladdr` function, if the target is neither Linux or Solaris, the `argv` resource will be leaked due to lack of `argv_free` call (figure 2.3).

```
verify_user_pass_script(...) {
    struct gc_arena gc = gc_new();
    struct argv argv = argv_new();
    ...
    argv_parse_cmd(&argv, session->opt->auth_user_pass_verify_script);

    if (session->opt->auth_user_pass_verify_script_via_file) {
        struct status_output *so;

        tmp_file = platform_create_temp_file(session->opt->tmp_dir, "up", &gc);
        if (tmp_file) {
            ...
            argv_printf_cat(&argv, "%s", tmp_file);
        }
    }
    else { ... }

    /* generate filename for deferred auth control file */
```

```
    if (!key_state_gen_auth_control_files(&ks->script_auth, session->opt)) {
        msg(D_TLS_ERRORS, "TLS Auth Error (%s): "
            "could not create deferred auth control file", __func__);
        return OPENVPN_PLUGIN_FUNC_ERROR;
    }
    ...
done:
    if (tmp_file && strlen(tmp_file) > 0) {
        platform_unlink(tmp_file);
    }

    argv_free(&argv);
    gc_free(&gc);
    return retval;
}
```

Figure 2.1: openvpn/src/openvpn/ssl_verify.c#L1319-L1418

```
static bool get_console_input_systemd(...) {
    ...
    struct argv argv = argv_new();
    ...
    if ((std_out = openvpn_popen(&argv, NULL)) < 0) {
        return false;
    }
```

Figure 2.2: openvpn/src/openvpn/console_systemd.c#L63-L78

```
int set_lladdr(...) {
#if defined(TARGET_LINUX)
    ...
#else /* if defined(TARGET_LINUX) */
    struct argv argv = argv_new();
#if defined(TARGET_SOLARIS)
    ...
#else  /* if defined(TARGET_SOLARIS) */
    msg(M_WARN, "Sorry, but I don't know how to configure link layer addresses on
this operating system.");
    return -1;
#endif /* if defined(TARGET_SOLARIS) */
```

Figure 2.3: openvpn/src/openvpn/lladdr.c#L35-L58

This issue can be found with custom static analysis queries with CodeQL or by querying the code with Weggli, as demonstrated in Appendices D.2 and D.3.

Recommendations

Short term, fix the code paths that leak memory by calling the appropriate memory-freeing functions. In the case of the `verify_user_pass_script` function, set `retval = OPENVPN_PLUGIN_FUNC_ERROR` and jump to the `done` label so that all the resources are

freed up before the function returns. In the other two cases, call `argv_free(&argv)` before the return statements.

Long term, create CodeQL rules to run on your CI/CD pipeline to ensure that your team is alerted if any new potential vulnerabilities surface during development of the project.

4. Stack buffer out-of-bounds read in command line options parsing

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OVPN-4
Target: `src/openvpn/options.h`	

Description

The `parse_argv` function contains an out-of-bounds read bug that can cause OpenVPN to crash or cause undefined program behavior. The function allocates a buffer `p` for command line parameters that is assumed to be null terminated, but may be filled in completely (figure 4.1). The `parse_argv` passes the buffer `p` to `add_option`, which passes it to `no_more_than_n_args`, which then passes it to `string_array_len`, where the out-of-bounds read occurs (figure 4.2).

This bug can be reproduced with the following command. We recommend compiling OpenVPN with AddressSanitizer (`-fsanitize=address` flag) because the problem may not otherwise be directly observable:

```
openvpn --tls-verify a a a a a a a a a a a a a a a.
```

The severity of this finding is informational because the issue does not seem to be exploitable.

We found this issue with the `fuzz_parse_argv` fuzzing harness that we created as described in Appendix E.

```
void parse_argv(struct options *options,
                const int argc,
                char *argv[],
                const int msglevel,
                const unsigned int permission_mask,
                unsigned int *option_types_found,
                struct env_set *es) {
    ...
    /* config filename specified only? */
    if (argc == 2 && strncmp(argv[1], "--", 2)) { ... }
    else {
        /* parse command line */
        for (i = 1; i < argc; ++i) {
            char *p[MAX_PARMS];  // MAX_PARMS is defines as 16 in options.h
            CLEAR(p);
            p[0] = argv[i];
```

```
            ...
            for (j = 1; j < MAX_PARMS; ++j) {
                if (i + j < argc) {
                    char *arg = argv[i + j];
                    if (strncmp(arg, "--", 2)) {
                        p[j] = arg;  // p is filled up to MAX_PARMS-1 index
                    }
            ...
            }
            // The add_option will eventually call string_array_len on p
            // which may access beyond the p buffer, e.g., its p[MAX_PARMS] index
            add_option(options, p, false, NULL, 0, 0, msglevel, permission_mask,
                       option_types_found, es);
            i += j - 1;
        }
    }
}
```

Figure 4.1: openvpn/src/openvpn/options.c#L5322-L5355

```
int string_array_len(const char **array) {
    int i = 0;
    if (array) {
        while (array[i]) {
            ++i;
        }
    }
    return i;
}
```

Figure 4.2: openvpn/src/openvpn/buffer.c#L715-L727

Recommendations

Short term, change the code highlighted in red in figure 4.1 to create a buffer of size `MAX_PARMS+1`.

Long term, integrate the fuzzing harness that covers options parsing into the project.

7. Support of weak proxy authentication algorithm

Severity: **Informational**	Difficulty: **Medium**
Type: Cryptography	Finding ID: TOB-OVPN-7
Target: `proxy authentication`	

Description

The OpenVPN code supports both the NTLMv1 and NTLMv2 proxy authentication methods. NTLM is an insecure and legacy authentication protocol that has been superseded by NTLMv2. NTLM (v1) uses cryptographically weak algorithms, such as MD4 and DES, to represent users' passwords. It is also vulnerable to relay attacks.

Note that other proxy authentication methods are also not safe when used through an unencrypted (HTTP) channel, as described in finding TOB-OVPN-11. However, a compromised NTLM credential can impact the other environments where it is used. Also, all currently supported versions of Windows support NTLMv2.

Furthermore, although the code supports NTLMv2, the documentation pages do not mention it as a valid authentication method.

```
bool ntlmv2_enabled = (p->auth_method == HTTP_AUTH_NTLM2);
...
if (ntlmv2_enabled)      { /* Generate NTLMv2 response */ }
else /* Generate NTLM response */ {
    unsigned char key1[DES_KEY_LENGTH], key2[DES_KEY_LENGTH];
    unsigned char key3[DES_KEY_LENGTH];

    create_des_keys(md4_hash, key1);
    cipher_des_encrypt_ecb(key1, challenge, ntlm_response);

    create_des_keys(&md4_hash[DES_KEY_LENGTH - 1], key2);
    cipher_des_encrypt_ecb(key2, challenge, &ntlm_response[DES_KEY_LENGTH]);

    create_des_keys(&md4_hash[2 * (DES_KEY_LENGTH - 1)], key3);
    cipher_des_encrypt_ecb(key3, challenge,
                           &ntlm_response[DES_KEY_LENGTH * 2]);
}
```

Figure 7.1: NTLM is considered insecure, as it relies to MD4 and DES ECB (`openvpn/src/openvpn/ntlm.c#L220-L374`)

Recommendations

Short term, deprecate the NTLMv1 proxy authentication mechanism and consider removing support for it in future OpenVPN versions. Additionally, improve the documentation to highlight the insecurity of the NTLMv1 authentication method and to advise users to use the more secure NTLMv2 method. If TLS proxy support is added, encourage users to use it as well.

8. Decoding username can silently cause truncated or empty username

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OVPN-8
Target: `src/openvpn/ssl_verify.c`	

Description

In the `get_auth_challenge` function, the separate parts of the `auth_challenge` are decoded and verified. The `user` field is encoded using Base64, and the `openvpn_base64_decode` function is used to decode it. The return value is not checked for success (figure 8.1). Failure to decode the `work` string into ac->`user` will silently pass. Since the `ac->user` field is allocated with the "clear" allocation flag set, this would leave no or partially decoded content in the `ac->user` field.

```
struct auth_challenge_info *
get_auth_challenge(const char *auth_challenge, struct gc_arena *gc) {
    ...
        ac->user = (char *) gc_malloc(strlen(work)+1, true, gc);
        openvpn_base64_decode(work, (void *)ac->user, -1);
```

Figure 8.1: Failed Base64-decoding silently corrupts the stored username. (src/openvpn/misc.c#457–458)

The `get_auth_challenge` function is called in the `get_user_pass_cr` function when OpenVPN is compiled with the management interface enabled and when the username/password are provided from standard input (figure 8.2).

Although this finding does not seem a direct security risk, we include it so that similar mistakes can be avoided in the future.

```
bool get_user_pass_cr(..., const char *auth_challenge) {
  ...
  // Get username/password from standard input?
  if (username_from_stdin || password_from_stdin || response_from_stdin) {
#ifdef ENABLE_MANAGEMENT
    if (auth_challenge && (flags & GET_USER_PASS_DYNAMIC_CHALLENGE) &&
response_from_stdin) {
        struct auth_challenge_info *ac = get_auth_challenge(auth_challenge, &gc);
```

Figure 8.2: Code that calls the `get_auth_challenge` (src/openvpn/misc.c#L283-L291).

Recommendations

Short term, verify the return value of the `openvpn_base64_decode` function in the `get_auth_challenge` function and take necessary action on failure.

Long term, consider marking functions that can fail with `[[nodiscard]]` attribute—a C23 feature—to require handling the return value.

11. Lack of TLS support by the HTTP and SOCKS proxy may lead to compromise of a user's proxy authentication credentials

Severity: **Informational**	Difficulty: **High**
Type: Documentation	Finding ID: TOB-OVPN-11
Target: `http proxy`	

Description
OpenVPN allows clients to connect to the OpenVPN server through a HTTP and SOCKS proxies. However, it does not implement TLS for authentication to the proxy server. Without TLS, an attacker who observes or intercepts the authentication traffic between a user and proxy can compromise the proxy authentication credentials, as all of the available authentication methods (Basic, Digest, and NTLM) have weaknesses. These risks are not documented by OpenVPN nor is the user warned of the same.

Recommendations
Short term, alert users of the risks of using unencrypted HTTP proxy authentication in the documentation.

Long term, consider adding support for using HTTPS (TLS) for proxy authentication. Additionally,

12. Implicit conversions that lose integer precision

Severity: **Undetermined**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-OVPN-12
Target: Various files	

Description
The code contains many cases where variables of the integer class are implicitly converted from one type to another in a way that risks altering the value. For example, building with `clang` and `-Wshorten-64-to-32` flags 128 places where a 64-bit variable is passed to a function using a 32-bit type for the argument.

Each such case is a potential problem if there is ever a case where the value before conversion exceeds the maximum value for the corresponding receiving type, or if the signedness is changed by the conversion.

Exploit Scenario
Mallory finds a place in the code where such a truncation causes a too-small memory allocation and uses this memory write primitive to corrupt application memory.

Recommendations
Short term, triage the conversion warnings to eliminate those where the conversion is deemed safe, and update the types used where it is not. Using CodeQL rules and value analysis can facilitate this somewhat. For example, create a rule that returns a conversion warning only if the value analysis cannot prove that the value is safe.

Long term, as the codebase is updated, incrementally introduce explicit casts and, unless it is clear from the adjacent code, add a note why the cast is safe. This will help future code audits, as it makes the developer's intention clear.

13. The OpenVPN build system does not enable compiler security mitigations	
Severity: **Undetermined**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-OVPN-13
Target: build system	

Description

The OpenVPN build system on Linux does not explicitly enable modern compiler security mitigations, which may cause users to use less secure configurations if they build the OpenVPN binaries themselves without extra caution. This would make it easier for an attacker who finds a low-level vulnerability to exploit a bug and gain control over the process. Modern compilers support exploit mitigations including the following:

- NX (non-executable data)
- PIE (a position-independent executable, which is position-independent code for address space layout randomization (ASLR))
- Stack canaries (for buffer overflow detection)
- RELRO (for data section hardening)
- Source fortification (for buffer overflow detection and format string protection)
- Stack clash protection (for the detection of clashes between a stack pointer and another memory region)
- CFI (control flow integrity)
- SafeStack (for stack overflow protection)

For details on these exploit mitigation technologies, see Appendix F: Compiler Mitigations.

The severity of this finding is undetermined as compilers and package maintainers enable some of these mitigations by default, and we have not investigated if this issue actually concerns OpenVPN users; it is possible that it does. Additionally, we have not reviewed the security hardening flags on MacOS or Windows, and we recommend doing this for OpenVPN clients on those platforms.

Recommendations

Short term, enable security mitigations for OpenVPN binaries using the compiler and linker flags described in Appendix F: Compiler Mitigations. Although compilers often enable certain mitigations by default, explicitly enabling them will ensure that they will be used regardless of a compiler's defaults.

Long term, enable security mitigations for all OpenVPN binaries and add a scan for them with checksec.rs or BinSkim Binary Analyzer into the CI/CD pipeline to ensure that certain options are always enabled. This will make it more difficult for an attacker to exploit any bugs found in the binaries. For additional assurance, consider verifying whether the ASLR system-wide setting is enabled during startup by checking the value stored in the `/proc/sys/kernel/randomize_va_space` file; if the value is below 2, designate it for future investigation. Also track the development of the Linux kernel configuration aimed at making the `randomize_va_space` setting read-only; update the kernel and use that option when it becomes available.

We also recommend reviewing possible security hardening options on Windows and MacOS builds.

References

- "Getting the maximum of your C compiler, for security"
- Debian hardening recommendations
- GCC man page
- LD man page (see -z keywords)

14. The ntlm_phase_3 function does not verify if it received the correct length of the challenge data

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OVPN-14
Target: `src/openvpn/ntlm.c`	

Description

The `ntlm_phase_3` function decodes a Base64 payload into the `buf2` buffer, from which the NTLM challenge bytes are then extracted (figure 14.1). However, the code does not verify that the decoded data is big enough to include the challenge bytes. As a result, if the decoded Base64 data is shorter than expected, the code will copy the previous data stored in the `buf2` buffer (which are all zeroes due to the `CLEAR(buf2)` call beforehand).

The severity of this finding is informational since the `buf2` buffer is cleared before it is copied from. If that were not the case, the challenge extraction would copy uninitialized data and could leak sensitive information such as memory addresses to the proxy server this code talks to.

```
const char* ntlm_phase_3(/* (...) */) {
    // (...)
    uint8_t buf2[128]; /* decoded reply from proxy */

    // (...)
    CLEAR(buf2);
    // (...)

    ret_val = openvpn_base64_decode(phase_2, buf2, -1);
    if (ret_val < 0) {       // no check for the size of data decoded in buf2
        return NULL;
    }

    /* we can be sure that phase_2 is less than 128
     * therefore buf2 needs to be (3/4 * 128) */

    /* extract the challenge from bytes 24-31 */
    for (i = 0; i<8; i++) {
        challenge[i] = buf2[i+24];
    }
```

Figure 14.1: openvpn/src/openvpn/ntlm.c#L256-L269

Recommendations

Short term, change the `ntlm_phase_3` code to verify that the length of the decoded buffer matches the expected length of the challenge data, which is eight bytes. The length of the decoded buffer is the result of the `openvpn_base64_decode` call and so is stored in the `ret_val` variable (if the decoding succeeds and does not return -1).

Long term, create unit tests to test unhappy paths which contain truncated HTTP request/response data. This will help to prevent similar issues in the future.

15. The establish_http_proxy_passthru function proxy-authenticate header check is insufficient

Severity: **Undetermined**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-OVPN-15
Target: `src/openvpn/proxy.c`	

Description

The `establish_http_proxy_passthru` function's parsing of the "`Proxy-Authenticate: NTLM …`" header is incorrect. Instead of matching the exact header name, the function reads the request lines until it finds a line that matches the "`%*s NTLM %128s`" format. This may result in processing an incorrect header that would contain the "NTLM" string as the "`Proxy-Authenticate`" header.

Additionally, the function does not take into account a possible case where the `Proxy-Authenticate` header is duplicated. In such a case, the code will use the first header, while maybe it should not process such a request.

The severity of this finding is undetermined as we haven't fully analyzed the impact of this issue due to time constraints.

```
bool establish_http_proxy_passthru(/* (...) */) {
  // (...)

  /* look for the phase 2 response */
  while (true) {
    if (!recv_line(sd, buf, sizeof(buf), /* (...) */)) {
      goto error;
    }
    chomp(buf);
    msg(D_PROXY, "HTTP proxy returned: '%s'", buf);

    openvpn_snprintf(get, sizeof get, "%%*s NTLM %%%ds", (int) sizeof(buf2) - 1);
    nparms = sscanf(buf, get, buf2);
    buf2[128] = 0; /* we only need the beginning - ensure it's null terminated. */

    /* check for "Proxy-Authenticate: NTLM TlRM..." */
    if (nparms == 1){
      /* parse buf2 */
      msg(D_PROXY, "auth string: '%s'", buf2);
      break;
    }
  }
```

Figure 15.1: openvpn/src/openvpn/proxy.c#L759-L781

Recommendations

Short term, investigate this issue and fix it. Consider changing the `establish_http_proxy_passthru` function to: 1) process the NTLM data only in case the correct `Proxy-Authenticate` header is received and 2) to account for the situation when this header may be duplicated, in which case the function should probably reject such a request.

Summary of Recommendations

Trail of Bits recommends that OpenVPN Inc. address the findings detailed in this report and take the following steps to further enhance the security of the OpenVPN2 project:

- Improve and extend existing testing and fuzzing harnesses for the project to increase the code coverage. Current code coverage for fuzzing is rather limited, and some of the existing harnesses are broken. Integrate the fuzzing efforts into the project's repository so they are easily accessible to any researcher and are built along with the code. This will ensure that the fuzzing harnesses' code does not break and is always ready to be used and extended. Update the OSS-Fuzz project to use the OpenVPN's integrated fuzzing harnesses.

- Integrate static analysis tools (such as CodeQL and scan-build) into the development process of the project. As soon as they are introduced, these tools continuously uncover a wide range of vulnerabilities in the codebase. In particular, CodeQL can be integrated with and used as a GitHub code scanning action. We also recommend extending the CodeQL query set with custom queries based on OpenVPN functions. An example is provided in figure D.1.2.

- Improve the OpenVPN documentation, especially in areas where best-practice security recommendations are required, or where default configurations are not secure. Users should be made aware about potential risks related to less secure authentication or encryption algorithms. It is also worth noting that OpenVPN Inc. has an ongoing project to improve the protocol documentation and create an RFC for OpenVPN, which is available on GitHub.

- Enhance code readability and robustness as detailed in Appendix C: Non-Security-Related findings.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

Inconsistent use of types by APIs. This includes internal APIs such as the widely used buffer type. A good example is the return value of the `buf_write`, `buf_write_prepend`, `buf_write_u8`, `buf_write_u16`, and `buf_write_u32` functions, which all return a Boolean indicating success. This contrasts with the corresponding read-functions, which return different types and must be compared to different constants to check for success. This increases the cognitive load on the developers and enables mistakes.

Another aspect of consistency is the type of a length argument; in some cases it is an `int`, while in others it is a `uint32_t`. In C, many conversions happen implicitly, giving the appearance of code just working, while the compiler actually introduces casts. And these conversions often lead to security vulnerabilities when they involve unexpected input, such as negative values or values of excessive range. Further, signed integers are prone to operations with undefined behavior, such as overflows. Using only an unsigned type to represent the length prevents these risks and clearly indicates that the type is a length.

The code contains several uses of `strcpy` and other potentially unbounded functions. Although their use may be correct and correct length checks and proper string termination are in place, a coding mistake can easily happen and open up a vulnerability. Consider changing to bounded versions, such as `strncpy`.

Some `switch` statements are missing `break` statements (figures C.1-2). If this is intentional, mark the statement with a `fallthrough` attribute (if possible, since this feature was added in C23) or a comment. Otherwise, add the `break` statement.

```
switch (auth_retry_get()) {
    case AR_NONE:
        msg(M_FATAL, "Error: private key password verification failed");
        break;

    case AR_INTERACT:
        ssl_purge_auth(false);

    case AR_NOINTERACT:
```

Figure C.1: `switch` case missing `break` statement (`openvpn/src/openvpn/init.c#L2879-L2886`)

```
case AR_INTERACT:
```

```
    ssl_purge_auth(false);

case AR_NOINTERACT:
```

Figure C.2: `switch` case missing `break` statement (openvpn/src/openvpn/push.c#100–101)

There are cases where a `gc_arena` object is created as a local variable but is never used apart from being freed at the end of the function. Either remove these variables, or, if this is intended, consider adding a comment that clarifies why the variable is needed.

- openvpn/src/openvpn/forward.c#L1830-L1917
- openvpn/src/openvpn/multi.c#L3480-L3566

The code contains empty blocks. Either refactor the code to remove the empty code blocks or annotate them with a comment that the body is intentionally blank. This will make the code more clear.

- openvpn/src/openvpn/comp-lz4.c#L238-L240
- openvpn/src/openvpn/crypto.c#L1299-L1301
- openvpn/src/openvpn/lzo.c#L251-L253
- openvpn/src/openvpn/options.c#L2973-L2975
- openvpn/src/openvpn/ssl_openssl.c#L1879-L1881
- openvpn/src/openvpn/ssl_openssl.c#L1903-L1905
- openvpn/src/openvpn/ssl_openssl.c#L1829-L1831

The compiler may omit the use of `memset` for deleting data. Use `memset_s` to avoid this. Long term, use `memset_explicit` (a C23 feature).

- openvpn/src/openvpn/ssl_mbedtls.c#L1383
- openvpn/src/openvpn/ssl_mbedtls.c#L1233

Certain code paths use the `strtok` function, which is not thread-safe. Consider refactoring the use of `strtok` to either use the `strsep` function, or, if POSIX conformance is required, `strtok_r`. While OpenVPN 2.x is single threaded, using thread-safe functions allows avoiding bugs if the code is ever used in a threaded context. The `strtok` function is used in the following code paths:

- openvpn/src/openvpn/options_util.c#L52
- openvpn/src/openvpn/options_util.c#L85
- openvpn/src/openvpn/ssl_ncp.c#L104
- openvpn/src/openvpn/ssl_verify.c#L892

A file descriptor is not closed if the authentication fails. To prevent this issue, close the `fp` file pointer in the `get_user_pass_cr` function when it returns `false`.

```
bool get_user_pass_cr(...) {
    ...
    fp = platform_fopen(auth_file, "r");
    // if/else ...
        if (!auth_user_pass_mgmt(up, prefix, flags, auth_challenge)) {
            return false;
        }
```

Figure C.3: openvpn/src/openvpn/misc.c#L217-261

The `verify_cert_export_cert` function creates a peer-cert file with 0600 permissions through the `platform_create_temp_file` function and later reopens it with the `fopen` call with a `"w+"` mode. Because of this mode, if the peer-cert file is removed before the `fopen` call, it will be created with broader permissions of 0666 (which may be influenced by `umask`). Consider refactoring this function or/and the `platform_create_temp_file` so that created temporary files are kept open instead of re-opening them, which may lead to issues.

```
static const char *verify_cert_export_cert(...) {
    FILE *peercert_file;
    const char *peercert_filename = "";

    /* create tmp file to store peer cert */
    if (!tmp_dir
        || !(peercert_filename = platform_create_temp_file(tmp_dir, "pcf", gc))) {
        msg(M_NONFATAL, "Failed to create peer cert file");
        return NULL;
    }

    /* write peer-cert in tmp-file */
    peercert_file = fopen(peercert_filename, "w+");
```

Figure C.4: openvpn/src/openvpn/ssl_verify.c#L503-L510

The assignment to `seq` (shown in figure C.5) is superfluous, as `seq` is never read and can be omitted.

```
payload->nlmsg_seq = seq = time(NULL);
```

Figure C.5: Assignment to `seq` is superfluous (openvpn/src/openvpn/networking_sitnl.c#261)

The assignment to `e` on line 686 in figure C.6 is redundant, as it has already been assigned on line 684. Consider removing the assignment on line 686.

```
struct push_entry *e = push_list->head;

e = push_list->head;
```

```
while (e)
```

Figure C.6: Redundant assignment to variable e (openvpn/src/openvpn/push.c#684–687)

The assignment to `i` on line 71 in figure C.7 is redundant, as it will be assigned zero in the for loop on line 72. Consider removing either of the assignments.

```
i = 0;
for (i = 0; i < size; )
```

Figure C.7: Redundant assignment to variable `i`. (openvpn/src/openvpn/base64.c#71–72)

The expression `tmp - options` is cast to `int`. This is unnecessary, as the `size` argument of `buf_write` is a `size_t`. Because the value of `buf_write` is not checked, the cast could lose information, causing incorrect reads or silent failures.

```
buf_write(&buf, options, (int)(tmp - options));
```

Figure C.8: Incorrect cast to `int` when a `size_t` is expected (openvpn/src/openvpn/ssl_util.c#108)

D. Automated Analysis Tool Configuration

As part of this assessment, we performed automated testing on the OpenVPN codebase using tools such as CodeQL, and other internal tools developed by Trail of Bits. Testing details are provided below.

D.1. CodeQL

We used CodeQL to analyze the OpenVPN codebases, using both public rulesets and private query suites developed by Trail of Bits. If OpenVPN Inc. intends to run CodeQL, we recommend reviewing CodeQL's licensing policies for commercial use. Since OpenVPN is an open-source project, GitHub likely offers special licensing terms for the project.

Due to the nature of OpenVPN's codebase and various customizations in implementation of memory allocator or garbage collector functions, standard CodeQL queries will naturally produce a large number of false positives. Existing rules can be extended or customized to adapt to such functions. Investment into learning how to write and use custom CodeQL queries will be very beneficial for the OpenVPN project in the long term. Moreover, CodeQL can and should be also integrated with the project's development pipeline, for instance via GitHub code scanning actions.

```
# Create the C/C++ database
codeql database create codeql.db --language=cpp --source-root=..

# Run all C/C++ queries
codeql database analyze codeql.db --format=sarif-latest --output=codeql_all.sarif --
{ruleset}
```

Figure D.1.1: Commands used to run CodeQL

We commonly use CodeQL to perform variant analysis of bugs. From the finding TOB-OVPN-2, we created a rule to scan the code for other instances where the code flows from a call to `argv_new`, to a return, without hitting an `argv_free` call along the way. Although these rules did not highlight any additional issues, CodeQL is still a strong candidate to include in the CI pipeline, as it will help to ensure that future changes to the code do not reintroduce the problem.

```
import cpp
import semmle.code.cpp.dataflow.DataFlow
import semmle.code.cpp.controlflow.StackVariableReachability

class CustomAllocatorLeak extends StackVariableReachabilityWithReassignment {

  CustomAllocatorLeak() {
    this = "CustomAllocatorLeak"
```

```
  }

  override predicate isSourceActual(ControlFlowNode node, StackVariable var) {
    // var = argv_new() or T* var = argv_new();
       node.(FunctionCall).getTarget().getName() = "argv_new" and (
        var.getAnAssignedValue() = node or     // var = argv_new();
        var.getInitializer().getExpr() = node  // T* var = argv_new();
    )
  }

  override predicate isBarrier(ControlFlowNode node, StackVariable var) {
    // argv_free(var);
    node.(FunctionCall).getTarget().getName() = "argv_free" and
    node.(FunctionCall).getAnArgument().getAChild() = var.getAnAccess()
  }

  override predicate isSinkActual(ControlFlowNode node, StackVariable var) {
    // A return statement which doesn't return var.
    var.getFunction() = node.(ReturnStmt).getEnclosingFunction() and not
    node.(ReturnStmt).getExpr() = var.getAnAccess()
  }
}

from
  CustomAllocatorLeak leak,
  ControlFlowNode source,
  ControlFlowNode sink,
  StackVariable var
where
  leak.reaches(source, var, sink)
select
  source.getLocation(),
  "The variable `" + var + "` allocated here may leak when returning on line " +
sink.getLocation().getStartLine() + "."
```

Figure D.1.2: A rule used to scan for memory leaks tied to `argv_new`

D.2. Weggli

We used Weggli, a semantic grep search tool, to quickly look for potential problems in the codebase. Apart from the queries shown in the Weggli repository and Julien Voisin's blog post, we also used the queries shown below.

```
# check for potential memory leaks with argv_new not being freed before return
weggli '{ $x = argv_new(); not: argv_free(&$x); return; }' ./

# check for potential memory leaks with gc_new not being freed before return
# note that the first query may result in lots of false positives
weggli '{ $x = gc_new(); not: gc_free(&$x); }' ./
weggli '{ $x = gc_new(); not: gc_free(&$x); return; gc_free(&$x); }' ./

# check for potential memory leaks with hmac_ctx_new not being freed
```

```
weggli '{ $x = hmac_ctx_new(); not: hmac_ctx_free($x); return; hmac_ctx_free($x); }'
./

# check for potential memory leaks with buffer_list_new not being freed
weggli '{ $x = buffer_list_new(); not: buffer_list_free(&$x); return;
buffer_list_free(&$x); }' ./
```

Figure D.2.1: Weggli queries that can help find issues in the OpenVPN codebase

E. Fuzzing OpenVPN2

As part of the audit, Trail of Bits used fuzzing, a process of automatically testing given code paths by executing them against random data to find bugs resulting from incorrect handling of unexpected data. For this, we developed fuzzing harnesses against OpenVPN functions and extended the existing harnesses from the oss-fuzz project.

We ran the harnesses for a limited period of time, but we recommend running them even longer—for example, until the fuzzer does not find inputs that generate new coverage for a long time. In such a case, we recommend investigating the code coverage of the program after running it against the corpus generated by the fuzzer. This allows for identification of code paths that were not executed and helps developers to manually craft or modify the corpus in order to expand coverage and find new bugs.

The subsequent sections of this appendix detail the problems we encountered with existing fuzzing harnesses and describe the process for running fuzzing locally (out of oss-fuzz).

Problems with the existing fuzzing

We identified the following problematic areas that reduce effectiveness of the fuzzing process and increase the likelihood of false positives.

Build issues

Currently, the OpenVPN fuzzing harnesses live in the oss-fuzz repository. In order to fuzz certain things properly (for example, to provide fuzzer-generated data from read/recv functions), the harness build system patches the existing code using `sed`. Additionally, all the dependencies are installed in the Dockerfile, which is not tested by the OpenVPN repository CI/CD system.

We fixed those issues and added a script that can be used to run fuzzing locally. Those changes can be found in the OpenVPN/openvpn#208 pull request. We recommend integrating those changes back into the oss-fuzz repository.

Root cause for the fuzzer branch choice biases

As described in TOB-OVPN-1, we discovered bias in the fuzzer branch choice. The fuzzing setup includes a small utility library, `fuzz_randomizer.cpp`, that builds on top of the libFuzzer API (figure E.1). The function `ConsumeIntegralInRange` from libFuzzer returns the lower bound value when there are no more bytes in the fuzzer input (figure E.2). This introduces a bias that causes `fuzz_randomizer_get_int` (and other functions that use `ConsumeIntegralInRange`) to return the `min` value most of the time.

```
extern "C" int fuzz_randomizer_get_int(int min, int max) {
   assert(prov != NULL);
   return prov->ConsumeIntegralInRange<int>(min, max);
}
```

Figure E.1: github.com/google/oss-fuzz/projects/openvpn/fuzz_randomizer.cpp#L39-L42

```
// Returns a number in the range [min, max] by consuming bytes from the
// input data. The value might not be uniformly distributed in the given
// range. If there's no input data left, always returns |min|. |min| must
// be less than or equal to |max|.
template <typename T>
T FuzzedDataProvider::ConsumeIntegralInRange(T min, T max) {
```

Figure E.2: github.com/llvm/llvm-project/compiler-rt/…/FuzzedDataProvider.h#L199-L204

Multiple harnesses use the `fuzz_randomizer_get_int` function to distribute fuzzing across different functions. Unfortunately, the bias heavily skews this distribution towards the first case, as shown in figure 1.1 in TOB-OVPN-1. This makes the fuzzing inefficient and slower to reach more code paths.

We fixed this issue by introducing a `fuzz_randomizer_get_byte` function that 1) tries to consume a single byte and 2) returns -1 if there are no more remaining bytes to generate the values that are used for branching. We then use this function in switch statements, since all of them have fewer than 256 cases. Those changes resulted in eliminating the bias where "case 0" of switches was covered much more often than other cases (figure E.2).

```
54  4.45k  #define NUM_TARGETS 32
55  4.45k        generic_ssizet = fuzz_randomizer_get_byte(0, NUM_TARGETS);
56  4.45k        if (generic_ssizet == -1) goto cleanup;
57  4.19k        switch (generic_ssizet) {
58    199        case 0:
59    199          buf_clear(bufp);
60    199          break;
61     43        case 1:
62     43          buf2 = clone_buf(bufp);
63     43          free_buf(&buf2);
64     43          break;
65     25        case 2:
66     25          buf_defined(bufp);
67     25          break;
68      9        case 3:
69      9          buf_valid(bufp);
70      9          break;
71     51        case 4:
72     51          buf_bptr(bufp);
73     51          break;
74     40        case 5:
75     40          buf_len(bufp);
76     40          break;
```

Figure E.2: A screenshot of code coverage report for the updated `fuzz_buffer` harness. The columns represent: line number, number of hits by the fuzzer's corpus inputs, and the code lines. The fuzzing is now distributed across branches without an obvious bias.

Bug discovered in fuzz_proxy harness

We found a buffer overflow bug in the fuzz_proxy harness after we improved and unbiased the harness to have better coverage. The harness reported the bug in a very distant place in the OpenVPN code; however, the bug was caused by a buffer overflow in the harness code itself that did not surface, as Address Sanitizer does not detect buffer overflows that happen on a buffer that exists in a structure and is not its last field.

The root cause of this bug appears to be a misunderstanding of the semantics of the `USER_PASS_LEN` constant from the OpenVPN code that is used in the fuzzing harness (figure E.3). The harness code first generated the string of length `USER_PASS_LEN` (excluding the null byte). This string was then copied to the `username` field in the `user_pass` structure whose size is set to `USER_PASS_LEN`, which includes a null byte (figure E.4). As a result, if the fuzzer generated a string of maximum length, the `strcpy` call caused a buffer overflow on the `username` field writing a null byte just after it (likely writing

it to the first byte of the `password` field). The same situation later occurred in the `password` field.

This issue can be reproduced by running the original harness which includes the bug from figure E.3 (that we fixed in our changes) with the following command:

```
      echo -ne
'\xf6\x00\x00\x00\x00\xff\xff\x00mmmmmmmmmmmmm\x01\x007003\x01\x00\x00\xf8\x01\x00\x01
/====\x17\x00\x00\x00;================;=,,,,\xb7,,=====algor\xff\xff\xff\xff\xff\xff=\
n\xff\xff=\xff=\xf1=\xf1\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\x
ff\xff\xff\xff====\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff
\xff\xff\xff\xff===========*==========[=,,real=""||\xff\xff\xff#\xf6\n\r#\xf6\x00\\\x
00\\\xff\\\xd9||\r\r\r4\x98007\r\r\r\n\r\r\r\n\xd8o\x9a\xff7alm===9==Ho\x9a\xff00==Ho\
x9a\xff70==Ho\x9a\xff7Ho\x9a\xff7=Ho\x9a\xff7el==uuuuuuuuu\xff\xff0\xff7=H\x9a\xff7Ho\
x9a\xff7==uuuuuuuuu\xff\xff00==Ho\x9a\xff700==Ho\x9a\xff70==Ho\x9a\xff7Ho\x9a\xff7=Ho\
x9a\xff7el==uuuuuuuuu\xff\xff00==Ho\x9a\xff7' > input && ./fuzz_proxy ./input
```

We fixed this issue by passing USER_PASS_LEN-1 to `fuzz_random_get_string_max_length`. Additionally, we recommend renaming this constant to USER_PASS_BUF_LEN to better reflect its semantics or adding a comment with a warning that this length includes a null byte.

```
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  ...
  char *fuzz_usrnm = fuzz_random_get_string_max_length(USER_PASS_LEN);
  strcpy(pi.up.username, fuzz_usrnm);
  ...
  char *pswd = fuzz_random_get_string_max_length(USER_PASS_LEN);
  strcpy(pi.up.password, pswd);
}
```

Figure E.3: `strcpy` overflows the `pi.up.username` buffer due to misuse of USER_PASS_LEN constant

```
struct user_pass {
    ...
/* max length of username/password */
#ifdef ENABLE_PKCS11
#define USER_PASS_LEN 4096
#else
#define USER_PASS_LEN 128
#endif
    char username[USER_PASS_LEN];
    char password[USER_PASS_LEN];
};
```

Figure E.4: The definition and usage of USER_PASS_LEN constant indicate that it describes buffer sizes rather than string length

Fuzzing harnesses

The table below summarizes the fuzzing harnesses we worked with during the assessment.

Fuzzing harness exercised	
Harness	**Description**
fuzz_base64	Fuzzing base64 encoding and decoding.
fuzz_buffer	Fuzzing `buffer.c` module with a different combination of buffer operations. We de-biased branches.
fuzz_dhcp	Fuzzing the `dhcp.c` module.
fuzz_forward	Fuzzing the `forward.c` module. We de-biased branches and updated the code so it works with the current version of OpenVPN.
fuzz_list	Fuzzing `list.c` module with a different combination of list operations. We de-biased branches.
fuzz_misc	Fuzzing functions from the `misc.c` module. We de-biased branches and added a few new cases to increase the coverage.
fuzz_mroute	Fuzzing the `mroute.c` module.
fuzz_mss	New harness, created during this assessment. Fuzzing the functions from the `mss.c` module.
fuzz_packet_id	Fuzzing the `packet_id.c` module.
fuzz_parse_argv	New harness, created during this assessment. Fuzzing the `parse_argv` function that serves as an entry point to the `options.c` module.
fuzz_proxy	Fuzzing the `proxy.c` module. We de-biased branches and added OpenVPN code changes to increase the coverage.
fuzz_route	Fuzzing functions from the `route.c` module. We de-biased branches.

Fuzzing script

In order to simplify local fuzzing, we prepared an `openvpn-fuzz.py` Python script along with a Nix shell environment (based on the Nix package manager). These could be used to enter a shell with all dependencies installed using a single command: `nix-shell fuzz/shell.nix`.

After entering the Nix shell environment, a user has to configure the build system with:

```
autoreconf -i -v -f
./configure --disable-lz4
```

Next, a user can compile a given fuzzing harness with AddressSanitizer support and run it by invoking the script in the `fuzz` directory:

```
./openvpn-fuzz.py fuzz <harness>
```

Additionally, a user may pass in arguments to LibFuzzer by passing them after "--", e.g.:

```
./openvpn-fuzz.py fuzz parse_argv -- -fork=4 -ignore_crashes=1
```

Note that the `-ignore_crashes=1` flag causes the fuzzer to continue fuzzing when it finds a crash, which allows it to collect more crashes from one fuzzing campaign rather than stopping at the first crash. This is useful for code that uses `exit` to handle errors, as `exit` is reported as a crash.

The script can also be used to generate code coverage report with the "coverage" option as follows:

```
./openvpn-fuzz.py coverage <targets or "all">
```

Such an invocation will recompile the harnesses with code coverage instrumentation. Then, it will run the following (or all) target against all corpus inputs that were generated during the previous fuzzing run, and it will generate an HTML coverage report that can be opened from the `fuzz/build/coverage/report/index.html` path.

F. Compiler Mitigations

As mentioned in finding TOB-OVPN-13, this appendix details the lists of compiler security hardening flags available in modern compilers for Linux binaries. Note that some of them can be enabled by default in a given compiler and they may also influence the program's performance. We recommend reviewing those settings in order to harden production builds as much as possible.

GCC or Clang Flag	What It Enables or Does
`-z noexecstack`	This flag marks the program's data sections (including the stack and heap) as non-executable (NX). This makes it more difficult for an attacker to execute shellcode. Attackers who wish to bypass NX must resort to return-oriented programming (ROP), an exploitation method that is more difficult as well as less reliable across different builds of a program. This mitigation is enabled by default.
`-Wl,-z,relro,-z,now`	This flag enables full RELRO (relocations read-only). Segments are read-only after relocation, and lazy bindings are disabled. RELRO is a mitigation technique used to harden the data sections of an ELF process. It has three modes of operation: disabled, partial, and full. When a program uses a function from a dynamically loaded library, the function address is stored in the GOT.PLT (Global Offset Table for Procedure Linkage Table) section. When RELRO is disabled, each function address entry in the GOT.PLT table points to a dynamic resolver that resolves the entry to the actual address of the intended function when it is first called. In such a case, the memory location of the address is both readable and writable. As a result, an attacker who has control over the process control flow could change the entry of a given function in GOT.PLT to point to any other executable address. For example, the attacker could change the `puts` function's GOT.PLT entry to point to a system function. Then, if the program called `puts("bin/sh")`, `system("/bin/sh")` would be

	called instead. When RELRO is fully enabled, the dynamic resolver resolves all of the addresses upon a program's startup and changes the permissions of data sections (and therefore GOT.PLT) to read-only.

`-fstack-protector-all`
 	
Or (less secure):
 	
`-fstack-protector-strong --param ssp-buffer-size=4`
	This flag adds stack canaries for all functions. Note that this flag may affect the program's performance.
	
	Stack canaries (stack cookies) make it more difficult to exploit stack buffer overflow vulnerabilities. A stack canary is a global randomly generated value that is copied to the stack between the stack variables and stack metadata in a function's prologue. When a function returns, the canary on the stack is checked against the global value. The program exits if there is a mismatch, making it more difficult for an attacker to overwrite the return address on the stack. In certain circumstances, attackers may be able to bypass this mitigation by leaking the cookie through a separate information leak vulnerability or by brute-forcing the cookie byte by byte.
	
	To protect only functions that have buffers, use the alternative version indicated.

`-fPIE -pie`	This flag compiles the program as a position independent executable, which ASLR (detailed below in the "System" rows) depends on.

Only in GCC >=12.x:
 	
`-D_FORTIFY_SOURCE=3 -O2`
 	
Or (less secure):
 	
`-D_FORTIFY_SOURCE=2 -O2`
 	
Or (even less secure):
 	
`-D_FORTIFY_SOURCE=1 -O2`
	This flag enables source fortification protections. These protections require an optimization flag (`-O1`, `-O2` or `-O3`).
	
	The protection is a libc-specific feature that enables a series of mitigations primarily aimed at preventing buffer overflows. It is supported by both glibc and Apple Libc, but not by musl or uclibc.
	
	With a `_FORTIFY_SOURCE` level of 1, compile-time warnings are added for potentially unsafe calls to common libc functions (e.g., `memcpy` and `strcpy`). With a `_FORTIFY_SOURCE` level of 2, more stringent runtime checks are added to these functions and enable a number of lesser-known mitigations. For example, it will

	disallow the use of the %n format specifier in format strings that are not located in read-only memory pages. This will prevent overwriting data (and gaining code execution) with format string vulnerabilities. The latter version is less secure, as it enables only compile-time measures; the former adds additional runtime checks, which may affect the program's performance. The source fortification level 3 was added in GCC 12.x and further improves this feature's detection capabilities and coverage.
`-fstack-clash-protection`	This flag adds checks to functions that may allocate a large amount of memory on the stack to ensure that the new stack pointer and stack frame will not overlap with another memory region, such as the heap. It mitigates a "stack clash vulnerability" in which a program's stack memory region grows so much that it overlaps with another memory region. This bug makes the program confuse the stack memory address with another memory address (e.g., that of the heap); as a result, the regions' data will overlap, which could lead to a denial of service or to control flow hijacking. The stack clash protection mitigation adds explicit memory probing to any function that allocates a large amount of stack memory; when explicit memory probing is used, the function's stack allocation will never make the stack pointer jump over the stack memory guard page, which is located before the stack.

`-fsanitize=cfi`
`-fvisibility=hidden`
`-flto`
 	
(Clang/LLVM only)
	This flag enables CFI checks that help prevent control flow hijacking.

`-fsanitize=safe-stack` *(Clang/LLVM only)*	This flag enables `SafeStack`, which splits the stack frames of certain functions into a safe stack and an unsafe stack, making hijacking of the program's control flow more difficult (Clang/LLVM only).

`-Wall -Wextra -Wpedantic -Wshadow -Wconversion -Wformat-security -Wshorten-64-to-32`	These flags enable compile-time checks and warnings to detect potential problems in the code.

System	What It Enables or Does
`ASLR (Address Space Layout Randomization)`	This feature randomizes the memory location of each section of the program. This makes it more difficult for an attacker to write reliable exploits, primarily by impeding jumps to ROP gadgets. ASLR requires cooperation from both the system and the compiler. To fully support ASLR, a program must be compiled as a position-independent executable (PIE). Most of the Linux distributions have ASLR enabled. This can be checked by reading the value stored in the `/proc/sys/kernel/randomize_va_space` file: 0 means that ASLR is disabled, 1 means it is partially enabled (only some bits of the addresses are randomized), and 2 means it is fully enabled. This file is writable, and an admin can disable or enable the mitigation. An information leak in the program may enable an attacker to bypass ASLR.

Some of these security hardening options can be checked in a binary with the use of tools such as checksec.rs or BinSkim Binary Analyzer. We recommend running them on CI/CD systems to ensure the build production binaries use the necessary security options.

It is also worth noting that some compilers or packaging solutions are configured with some of these options enabled. For example:

- Some GCC builds enable the `--enable-default-pie` flag by default. If optimizations are enabled, the source fortification macro can also be defined by the compiler. The latter can be checked with the following command:

```
$ gcc -O3 -dM -E - < /dev/null | grep FORTIFY
#define _FORTIFY_SOURCE 2
```

- Debian packaging uses the `dpkg-buildflags --export=configure` tool to obtain C/C++ compiler flags, which enable some of the listed hardening options. Figure F.1 shows these flags.

```
$ dpkg-buildflags --export=configure
CFLAGS="-g -O2 -fdebug-prefix-map=/home/dc/ovpn=. -fstack-protector-strong -Wformat
-Werror=format-security" CPPFLAGS="-Wdate-time -D_FORTIFY_SOURCE=2" CXXFLAGS="-g -O2
-fdebug-prefix-map=/home/dc/ovpn=. -fstack-protector-strong -Wformat
-Werror=format-security" FCFLAGS="-g -O2 -fdebug-prefix-map=/home/dc/ovpn=.
-fstack-protector-strong" FFLAGS="-g -O2 -fdebug-prefix-map=/home/dc/ovpn=.
-fstack-protector-strong" GCJFLAGS="-g -O2 -fdebug-prefix-map=/home/dc/ovpn=.
-fstack-protector-strong" LDFLAGS="-Wl,-Bsymbolic-functions -Wl,-z,relro"
OBJCFLAGS="-g -O2 -fdebug-prefix-map=/home/dc/ovpn=. -fstack-protector-strong
-Wformat -Werror=format-security" OBJCXXFLAGS="-g -O2
-fdebug-prefix-map=/home/dc/ovpn=. -fstack-protector-strong -Wformat
-Werror=format-security"
```

Figure F.1: The `dpkg-buildflags` on Ubuntu 20.04.3, which enable some (but not all) of the available security mitigations

However, we do not recommend relying on those defaults and, instead, explicitly passing all the options during compilation to ensure that the flags are actually enabled.

G. Additional Considerations

Required version of OpenSSL is no longer supported by the OpenSSL project

The OpenVPN `configure` script checks for the OpenSSL library version (figure 9.1). However, the minimum version required is 1.0.2, which is no longer supported by the OpenSSL project as of January 1, 2020 (figure 9.2). However, most Linux distributions that are supported by their maintainers have newer OpenSSL versions, and there may be systems that receive patches for 1.0.2 through the extended paid support.

```
$ ./configure
# (...)
checking for OPENSSL... no
checking additionally if OpenSSL is available and version >= 1.0.2... configure:
error: OpenSSL version too old
```

Figure G.1: Running the `configure` script when the OpenSSL development package is not installed in the system returns the error about required version.

```
With regards to current and future releases the OpenSSL project has adopted the
following policy:

Version 3.0 will be supported until 2026-09-07 (LTS).
Version 1.1.1 will be supported until 2023-09-11 (LTS).
Version 1.0.2 is no longer supported. Extended support for 1.0.2 to gain access to
security fixes for that version is
[available](https://www.openssl.org/support/contracts.html).
Versions 1.1.0, 1.0.1, 1.0.0 and 0.9.8 are no longer supported.
```

Figure G.2: `https://www.openssl.org/policies/releasestrat.html`

Autoconf-generated configure script silently fails if diff command is missing and continues

The `autoconf` utility generates a `configure` script which doesn't verify the existence of the `diff` program. This causes the `diff` program to fail, but the `configure` script continues to run (figure G.3). It is very unlikely that a system building OpenVPN wouldn't have `diff` installed; however, in such cases, this can lead to the incorrect determination of certain configuration options such as the compiler features. Furthermore, the user running the configure script will be able to see the errors on the screen as a result of the system missing the `diff` utility available.

Some of the uses of the `diff` tool are shown in figure G.4. The lack of `diff` tool verification was also reported to the autoconf project in 2008, but it hasn't been fixed since then.

```
$ ./configure
# (...)
checking if gcc supports -fno-rtti -fno-exceptions... ./configure: line 11257: diff:
command not found
# (...) - configure keeps going
```

Figure G.3: The `configure` command silently fails if the `diff` program is not present.

```
if test ! -s conftest.er2 || diff conftest.exp conftest.er2 >/dev/null; then
  lt_cv_prog_compiler_rtti_exceptions=yes
fi
# (...)
if test ! -s conftest.er2 || diff conftest.exp conftest.er2 >/dev/null; then
  lt_cv_prog_compiler_pic_works=yes
fi
# (...)
if diff conftest.exp conftest.er2 >/dev/null; then
  lt_cv_prog_compiler_static_works=yes
fi
# (...)
if test ! -s out/conftest.er2 || diff out/conftest.exp out/conftest.er2 >/dev/null;
then
  lt_cv_prog_compiler_c_o=yes
fi
# (...)

printf %s "checking if $CC understands -b... " >&6; }
# (...)
  if diff conftest.exp conftest.er2 >/dev/null; then
    lt_cv_prog_compiler__b=yes
  fi
# (...)
   if diff "$ac_file" "$ac_tmp/config.h" >/dev/null 2>&1; then
      { printf "%s\n" "$as_me:${as_lineno-$LINENO}: $ac_file is unchanged" >&5
printf "%s\n" "$as_me: $ac_file is unchanged" >&6;}

# Check for GNU ac_path_SED and select it if it is found.
# (...)
  while :
  do
  # (...)
    diff "conftest.out" "conftest.nl" >/dev/null 2>&1 || break
```

Figure G.4: Parts of the generated `configure` script that use the `diff` tool

H. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From August 15 to August 17, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the OpenVPN team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 10 issues described in this report, OpenVPN has resolved two issues, has partially resolved two issues, and has not resolved the remaining six issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Broken fuzzing harnesses	Unresolved
2	Certain error paths do not free allocated memory leading to memory leaks	Partially Resolved
3*	<retracted>	
4	Stack buffer out-of-bounds read in command line options parsing	Resolved
5*	<retracted>	
6*	<retracted>	
7	Support of weak proxy authentication algorithms	Resolved
8	Decoding username can silently cause truncated or empty username	Unresolved
9*	<retracted>	

10*	<retracted>	
11	Lack of TLS support by the HTTP and SOCKS proxy may lead to compromise of a user's proxy authentication credentials	**Unresolved**
12	Implicit conversions that lose integer precision	**Unresolved**
13	The OpenVPN build system does not enable compiler security mitigations	**Unresolved**
14	The ntlm_phase_3 function does not verify if it received the correct length of the challenge data	**Partially Resolved**
15	The establish_http_proxy_passthru function proxy-authenticate header check is insufficient	**Unresolved**

* These findings had been previously reported in a provisional state. Further investigation determined that findings 5 and 6 were invalid; these have since been retracted. Findings 3, 9, and 10 were determined to be non-security-related and they have been moved either to Appendix C or Appendix G. These entries remain as placeholders in order to preserve the previously reported finding IDs.

Detailed Fix Review Results

TOB-OVPN-1: Broken fuzzing harnesses
Unresolved; risk accepted. The client stated the following:

> *Unresolved; something to be handled later on as it will require lots of adjustments to be acceptable upstream.*

TOB-OVPN-2: Certain error paths do not free allocated memory leading to memory leaks
Partially resolved in commit `0567da5`. The `verify_user_pass_script` function now frees the allocated memory in the event of an error. However, the problem remains in the `get_console_input_systemd` function and the `set_lladdr` functions.

TOB-OVPN-4: Stack buffer out-of-bounds read in command line options parsing
Resolved in commit `749beb6`. The `parse_argv` function now uses an array large enough for an ending NULL element. In addition, a similar pattern was fixed in the `remove_iroutes_from_push_route_list` function.

TOB-OVPN-7: Support of weak proxy authentication algorithm
Resolved in commit `e005b8d1`. Users are informed that NTLM v1 authentication is deprecated and will be removed in OpenVPN 2.7.

TOB-OVPN-8: Decoding username can silently cause truncated or empty username
Unresolved; risk accepted. The client stated the following:

> *We accept the current situation for now - not considered a security risk. This issue may happen on the client side which will on parsing failures end up sending an invalid username to the server. Parsing errors would also imply the server most likely sent incorrect data to the client.*

TOB-OVPN-11: Lack of TLS support by the HTTP and SOCKS proxy may lead to compromise of a user's proxy authentication credentials
Unresolved; risk accepted. The client stated the following:

> *Documentation improvements will come later.*

TOB-OVPN-12: Implicit conversions that lose integer precision, and TOB-OVPN-13: The OpenVPN build system does not enable compiler security mitigations
Unresolved; risk accepted. The client stated the following:

> *Compiler flags related. Documentation with recommended compiler flags will be looked into later.*

TOB-OVPN-14: The ntlm_phase_3 function does not verify if it received the correct length of the challenge data

Partially resolved in commit `f193911`. The code declaring the `buf2` variable and clearing it has been moved, and a comment has been added explaining that if the challenge is shorter than expected, the missing bytes will be NULL. However, the code still does not verify that the length of the decoded buffer matches the expected length of the challenge data.

TOB-OVPN-15: The establish_http_proxy_passthru function proxy-authenticate header check is insufficient

Unresolved; risk accepted. The client stated the following:

> *We've started evaluating customer demands for NTLM authentication support in general.*

I. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.