Fine penetration tests for fine websites

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

Pentest-Report ExpressVPN Linux Clients 07.-08.2022

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, MSc. F. Fäßler

Index

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

Introduction

"A VPN (virtual private network) is the easiest and most effective way for people to protect their internet traffic and keep their identities private online. As you connect to a secure VPN server, your internet traffic goes through an encrypted tunnel that nobody can see into, including hackers, governments, and your internet service provider."

From https://www.expressvpn.com/what-is-vpn

This report describes the results of a penetration test and source code audit against the ExpressVPN Linux Clients, codebase and associated OS services. Carried out by Cure53 in the frames of an established, long-term cooperation, the project was registered as *EXP-09.*

The work delineated within *EXP-09* was requested by ExpressVPN in June 2022 and initiated by Cure53 in late July and early August 2022. The testing team, consisting of three senior testers, worked on the scope in CW29, CW30 and CW31. A total of fifteen days were invested to reach the coverage expected for this project. The work was split into two separate work packages (WPs). These read as follows:

- **WP1:** Source code-assisted penetration tests against ExpressVPN Linux client binaries
- **WP2:** Source code audits and reviews of the ExpressVPN Linux client codebase

Note that some parts of the scope have already been covered in one of the previous evaluations conducted by Cure53, namely in *EXP-04*, which was completed in March 2021. In this project, the Cure53 team analyzed the Lightway VPN protocol (known as Helium internally). While Lightway was audited again as part of *EXP-09,* it was treated with a lower priority. Furthermore, other components were assessed as part of *EXP-08* (macOS) and this rendered the tests of the ExpressVPN Linux clients a bit more compact in this instance. Cure53 was given access to the codebase which was shared during the pentest preparatory phase. Detailed, test-supporting information and scope documentation were also provided. The methodology chosen here was white-box.

All preparations were done in July 2022, namely in CW28, making it possible for the Cure53 team to have a smooth start into the actual testing phase. Communications during the project were done using a shared Slack channel into which all involved personnel from ExpressVPN and Cure53 were invited. Test-related information could be exchanged on Slack, especially in regard to progress being made. The discussions throughout the test were very good and productive and not many questions had to be asked. The scope was well-prepared and clear, greatly contributing to the fact that no noteworthy roadblocks were encountered during the test.

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

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was not specifically requested and, given the rather low severity ratings of all findings, would not have been necessary. The Cure53 team managed to get very good coverage over the WP1-WP2 scope items. Among five security-relevant discoveries, two were classified to be security vulnerabilities and three to be general weaknesses with lower exploitation potential. It needs to be stated clearly that this list of issues is very short, pointing to the overall good outcome of this testing round. In addition, in *EXP-09* the highest severity score reached by a single issue stood at *Medium,* which is rather impressive.

While Cure53 obtained good coverage of all the in-scope items and source code, it is worth noting that most native implementations of the RPC methods were employed in a library where no sources could be shared. Due to coverage of most of the shared components as part of the *EXP-08* (macOS) assessment, this audit was quite narrowly scoped and focused on the interactions with the Linux OS through the Golang interfaces. Nevertheless, findings such as EXP-09-003 - which describes a buffer overflow vulnerability in the Helium CLI - are quite noteworthy. Still, they are not easily exploitable without sufficient access to the backend systems of ExpressVPN. Next to this, EXP-09-001 is a rate-limiting issue that most likely affects one of the backend API systems but can be exploited from the Linux clients as well. Finally, EXP-09-002 is a finding that was originally spotted during the macOS audit and has been tracked as *EXP-08-004.* This problem was found to affect the Linux deployment as well. The remaining issues can be regarded as recommendations pertinent to further improving the code quality. Remediation of these minor flaws can raise the overall robustness of the app further.

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs. A dedicated section then offers a glossary to explain the system behind the categorization of the security problems spotted. This is followed by a chapter on test methodology and coverage, which specifies what the Cure53 team did in terms of attack-attempts and other test-relevant tasks.

Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions pertinent to this summer 2022 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the ExpressVPN complex, specifically the Linux clients and the corresponding codebase, are also incorporated into the final section.

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

Scope

- **Code audits & Security assessments of ExpressVPN's Linux clients & codebase**
  - **WP1**: Source-code assisted penetration tests against ExpressVPN Linux client binaries
    - **Primary audit focus:**
      - ExpressVPN client applications for Linux
      - Version: v3.28.0.6-1
    - **Secondary audit focus:**
      - Integrated *lightway* client
    - **In-scope items:**
      - All communications from the CLI binary and browser extensions to the ExpressVPN daemon, listening on the UNIX socket, including the respective implementations for CLI and browser extensions.
      - The ExpressVPN daemon, which is responsible for handling all functionality and logic of the application.
      - The ExpressVPN Linux application installers for the respective Linux distributions.
    - **Out-of-scope items:**
      - Denial-of-Service on the *lightway* protocol included in the Linux client application and caused by out-of-memory errors
      - Dependencies for "*xv_engine", "xv_linux*", OpenVPN and Lightway
      - Development or testing tools provided in the source code
      - VPN servers themselves with the exception of all traffic to/from the VPN servers, which is in scope.
      - Any split tunneling-related files that are not supported by the Linux application.
      - AWS APIs used by the client application with the exception of MitM between the APIs and the client application, which are included in scope
  - **WP2**: Source code audits & Reviews of the ExpressVPN Linux client codebase
    - All in-scope sources for ExpressVPN client for Linux were shared
- **Test-user accounts were created and activated for the auditing team**
- **All binaries in scope were shared with Cure53**
- **Test-supporting material was shared with Cure53**
- **All relevant sources were made available for Cure53**

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

Severity Glossary

The following section details the varying severity levels assigned to the issues discovered in the ExpressVPN complex.

Critical: The highest possible severity level. Indicates issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data or other pertinent components in scope.

High: This marker is used for issues that allow attackers to achieve significant yet somewhat limited access to sensitive areas in scope. It also includes issues with limited exploitability that can nevertheless facilitate a significant impact upon the target in scope.

Medium: This level is ascribed to issues that do not cause major implications for the areas in scope. Additionally, the issues requiring a more limited exploitation are graded as *Medium*.

Low: This level characterizes issues that have a highly limited impact on the areas in scope. Mostly they do not point to the level of exploitation but rather to the minor consequence of obtainable information or lower grade damage on the targeted components or areas in scope.

Info: This category covers issues considered merely informational in nature. They should mostly be viewed as hardening recommendations or improvements that can generally enhance the security posture of the areas in scope.

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

Table of Findings

Identified Vulnerabilities

ID	Title	Severity
EXP-09-002	WP1: Lack of application firewall rules for VPN gateway	*Medium*
EXP-09-003	WP2: Buffer overflow through config-entries on endpoints	*Medium*

Miscellaneous Issues

ID	Title	Severity
EXP-09-001	FP: MFA code verification throttled incorrectly	*Medium*
EXP-09-004	WP2: JSON helpers should check for 0-length strings	*Low*
EXP-09-005	WP2: Inconsistent use of *he_cli_calloc()*	*Low*

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

Testing Methodology

The following section summarizes Cure53's testing process in a more detailed manner. The goal of this section is to make the overall coverage more transparent by including all steps that were taken during the assessment. A section of this type is usually included when the number of findings is quite low or when the findings seem to cover only one aspect of the targeted scope.

With the following bullet points, Cure53 highlights the pentest-relevant tasks and findings from different angles. Additionally, the list can inform the maintainers about the attempted attacks which did not work or were mitigated more generally.

- As mentioned in the scope document, the main goal of the pentest was to find vulnerabilities within the ExpressVPN Linux clients, as well as in the associated services that they register with. Some libraries were examined using the black-box methodology, as sources for them could not be provided for internal reasons.
- Cure53 started by enumerating which services the ExpressVPN installer registers with and determined which components of the whole software architecture provide the most interesting attack surface.
- Since a multi-user scenario on the host system was deemed as out-of-scope, cross-user attacks have been entirely omitted and attention was mostly on privilege escalation, data stealing and overall robustness of the clients.
- Among the targets, the *expressvpnd*, the VPN command and control service, and the *lightway* protocol service (actually called *xv_helium_cli*) run with highest permissions as *root.* Thus, they form the most sensitive layer that can be targeted for privilege escalation. While *expressvpnd* can be communicated with through JSON RPC over a UNIX domain socket in the default case, interaction with *lightway* mostly runs through *config* files that are generated on the fly at runtime when VPN connections are going to be established.
- As such, the first specific aim was to find how JSON RPC calls are handled by the privileged VPN service. The definition for all reachable RPC calls can be found in the *xvpnd/jsonrpc/server/service.go*. From here, all implementations for each RPC method can be reached, along with all necessary properties and types for the passed arguments.
- For each of the listed RPC methods, Cure53 went ahead and studied the implementation for potential pitfalls in the Go language. Calls that branch out to *os.exec* or sensitive file read operations were studied for potential side-effects that could result in CLI injection or information leaks through passed files.
- It was also checked how the arguments of each RPC call are handled and whether the type-definitions made sense. Especially when string types were used, it had to be made sure that additional validations are present, for example when generating values for *config* file entries.

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

- For example, the *SetEnginePreferences* method takes a significant portion of client configurations and later passes them to *lightway* in order to define how connections are established. In this process, every preference needs to be sufficiently validated and it must be ensured that potentially malicious input cannot escape the JSON config definition. In addition, it needs to allow injection of arbitrary *config* entries that result in arbitrary *up* and *down* scripts to be invoked.
- In this context, the relevant code of *xvpnd/jsonrpc/server/service_default.go* needed to be carefully studied and every *args* definition of type *string* had to be validated. Actual validation happens in *xvpnd/vpn/preferences.go* where the call to *SetUserPreferences()* guarantees that all types are correct.
- Cure53 attempted to spot potential flaws that originate from Go's interaction with the underlying operating system. Additionally, the more native functions in the socket delegations and inside the xvclient were studied as well. Many of the implementations there were found decorated with Go's unsafe keyword and, thus, had to be written very carefully, especially when natively handling C strings which get allocated dynamically.
- As to not create potential *use-after-free* scenario's, a check concerned deferring and freeing memory in a correct way. For this, careful auditing of *xv_engine/pkg/xvclient/client.go* was carried out. While that portion of the pentest took up a lot of the allocated testing time, Cure53 did not find any flaws in this area. The testers shared the feeling that the ExpressVPN developers knew what they were doing when implementing the native interfaces.
- However, it should be noted that Cure53 only got access to the Go part of the client, and no source code of the other internal libraries was shared. This means it was not possible to conduct code review beyond the Go layer. Especially in regard to the potentially unsafe data passed from Go to the internal libraries, it was not possible to verify whether their handling was safe in the actual source code.
- Next to the source code audit, additional dynamic testing of the RPC methods was performed. This was necessary since many native implementations of the method branch out to the native *libxvclient* library, which was tested using a black-box methodology as no source code could be provided. This is, for example, how ticket EXP-09-001 was discovered, since random invocations of different RPC calls already trigger unexpected behaviors.
- Further dynamic testing was done on how the VPN connection is set up in general. For example, a simple check with the *route* command to check how the routing through the network interfaces is set up showed one potential leak. This leak can be accomplished when an attacker has Man-in-the-Middle capabilities. It turned out that a very similar issue was already present and found during the test of the macOS previously, as mentioned in EXP-09-002. No further issues were

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

identified in that regard, mainly because the Linux client is much simpler from a design perspective.

- As one tester already gained experience with several areas of the application during the OSX specific test, Cure53 could spend time on the secondary scope items as well. For example, the *lightway* client (*xv_helium_cli*) was the secondary focus for this assessment and as such was studied for potential issues. Since it is natively written in C, it is much more prone to potential memory corruption issues.
- As already mentioned, the main interaction between the user-facing ExpressVPN CLI and *lightway* happens through *config* files that are dynamically generated and stored in safe locations on the OS. Again, it was made sure that all of the relevant file permissions for the created directories and config files were correct and could not be read or stolen by unauthorized users.
- While the threat surface of the *lightway* client is greatly reduced due to it being only invoked by the daemon, it still makes sense from an audit point of view to treat the application as a standalone binary. In this case, the *config* file is the most obvious point of attack. During code review of the JSON parsing routine, it was found that bounds checking was missing from the endpoint list. This led to a classic buffer overflow situation EXP-09-003.
- Besides the logic of the JSON parsing helper functions, the code was also reviewed with regard to other classic C anti-patterns, such as format string vulnerabilities, use-after-frees and buffer/heap overflows. This uncovered a few C anti-patterns, as clarified in EXP-09-004 and EXP-09-005. However, nothing else of significance was found.
- After the initial review of the first layer, Cure53 wanted to go deeper into the code review by extending the test with fuzzing. Quite some time was spent on mocking missing functions and types in order to build and execute the config parse functions.
- While the source code for these binaries was out of scope, Cure53 was unable to draw meaningful conclusions from the mocked-up code. ExpressVPN is planning a more thorough review of the client at a later date.

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *EXP-09-001*) for the purpose of facilitating any future follow-up correspondence.

EXP-09-002 WP1: Lack of application firewall rules for VPN gateway *(Medium)*

Note from ExpressVPN: *There are multiple preconditions required for there to be any security impact on this finding, some of which include social engineering or getting the user to visit a malicious website. Furthermore, any attempts to remediate this finding is likely to worsen security as a result of the large complexity associated.*

The only impact to users exists as a result of social engineering, where a user is tricked into visiting a malicious website. In this case, there are significantly more damaging actions an attacker would likely try to take.

While checking whether previous vulnerabilities that have been reported in past pentest iterations might be applicable for the Linux scope of this testing round, it was noticed that the ticket "*EXP-08-004 WP1: Lack of application firewall rules for VPN gateway (Medium)*" was applicable. While this finding was originally spotted on macOS, the problematic route is present on Linux machines with the activated ExpressVPN as well. This is shown in the following excerpt of the routing table that is pushed by the client.

Affected route *(shell excerpt)*:

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.65.0.5       128.0.0.0       UG    0      0        0 tun0
default         10.0.0.1        0.0.0.0         UG    0      0        0 enp0s3
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
10.0.0.0        10.0.0.1        255.0.0.0       UG    0      0        0 enp0s3
10.65.0.1       10.65.0.5       255.255.255.255 UGH   0      0        0 tun0
10.65.0.5       0.0.0.0         255.255.255.255 UH    0      0        0 tun0
84.247.59.229   10.0.0.1        255.255.255.255 UGH   0      0        0 enp0s3
[..]
```

As such, the PoC which has already been documented in the original ticket works on Linux as well. It has to be noted that this issue can only be exploited by nation-state level attackers who are able to intercept connections to the highlighted IP address - either by directly being able to perform an MitM attack or by compromising the machine and accessing incoming connections to it. In this case, an attacker can trick the victim into

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

sending a clear text packet through the victim's default network interface (*enp0s3* in this case). From this it follows that the same recommendation applies here, too:

To prevent leaking the clear IP of VPN users to attackers with network sniffing capabilities, it is recommended to configure an application firewall rule that only allows the ExpressVPN daemon to use this route. Furthermore, one should consider disabling services that allow unencrypted connections on the VPN gateways.

EXP-09-003 WP2: Buffer overflow through config-entries on *endpoints* *(Medium)*

Fix Note: *The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

During a deep-dive into the *xv_helium_cli* client, a buffer overflow in the config-parsing code was found. The number of possible endpoints is a constant value defined by *MAX_ENDPOINTS* to be eight. By passing in a config with more than eight endpoints, arbitrary memory can be overwritten.

Affected file:
xv_helium_cli/src/he_config.c

Affected code:

```
// endpoints
json_object* jendpoints = json_object_object_get(jobj, "endpoints");
if (jendpoints) {
    // array
    if (json_object_is_type(jendpoints, json_type_array)) {
        size_t len = json_object_array_length(jendpoints);
        if (len == 0) {
            HE_LOG_ERROR("error parsing config: empty 'endpoints'");
            goto fail;
        }
        for (size_t i = 0; i < len; i++) {
            json_object* item = json_object_array_get_idx(jendpoints, i);
            if (item) {
                int rc = he_endpoint_from_json(item, &config->endpoints[i]);
```

Proof-of-Concept:

The following *poc.json* config file can overflow the *endpoints* array.

```
{
  "up": {
    "path": "/usr/sbin/expressvpnd",  "args": ["--update-dns-
config=static_resolv_conf"]
  },
```

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

```
  "down": {
    "path": "/usr/sbin/expressvpnd", "args": ["--update-dns-
config=static_resolv_conf"]
  },
  "endpoints": [
    {
      "server": "127.0.0.1", "server_dn": "AAAAAAAAAAAAAAAA",
      "protocol": "udp",  "port": 4919,
      "username": "xsi6vtrg3qjfiswsqy4yl64u",
      "password": "jb13noadqb357n5hl6stkf1e"
    },
    // [....] repeat endpoint objects dozens of times
    {
      "server": "127.0.0.1", "server_dn": "AAAAAAAAAAAAAAAA",
      "protocol": "udp", "port": 4919,
      "username": "xsi6vtrg3qjfiswsqy4yl64u",
      "password": "jb13noadqb357n5hl6stkf1e"
    }
  ],
  "keepalive": 10, "keepalive_timeout": 60
}
```

The following output shows various crashes from the overflowing config:

```
# /usr/lib/expressvpn/lightway -c poc.json
{"time":"2022-07-25T22:37:43.660+0000","log_level":"ERROR","message":"error
parsing config: invalid 'up'"}
Segmentation fault (core dumped)

# /usr/lib/expressvpn/lightway -c poc.json
{"time":"2022-07-25T22:43:46.525+0000","log_level":"ERROR","message":"error
parsing config: invalid 'up'"}
free(): invalid next size (fast)
Aborted (core dumped)
```

The reason that this issue exposes a lower-to-medium severity level is attributed to the fact that it cannot easily be exploited by a remote attacker. Being able to send the relevant config entries to the ExpressVPN client would require access to ExpressVPN backend systems. Nevertheless, this is an input validation issue where it is recommended to limit the *endpoints* array loop to eight or to dynamically allocate the array. This should be depending on the endpoints configured in the JSON file.

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

Miscellaneous Issues

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

EXP-09-001 FP: MFA code verification throttled incorrectly *(Medium)*

***Note**: After review of the issue, Cure53 and ExpressVPN agree that the the rate limits currently in place are sufficient, and that ExpressVPN has sufficient protections in place for the GenerateMfaCode endpoint. As a result of these protections, this issue is limited to a self-attack. Due to the protections in place, there is no security issue here and this issue is marked as a false positive.*

The *xvpnd* service under Linux implements two additional *JSON-RPC* commands that are used to verify MFA codes. While the original usage of both calls is not entirely clear, they are apparently employed to verify new client app installations through a second factor, in this case the client's email address.

While the *XVPN.RequestMfaCode* call is used to initiate the MFA verification flow and send out the six-digit value to the client's registered email address, *XVPN.ValidateMfaCode* simply validates it through the *libxvclient.so* library. It was noticed that the latter *RPC* call is highly throttled and allows only a few attempts at verifying the submitted code.

However, an alternative to submitting *RequestMfaCode* and *ValidateMfaCode* seems to reset the brute-force protection, so that a new MFA code is being resent for every iteration of the loop. As such, simply sending messages to *xvpnd* - like in the following example - would signify bruteforcing of the MFA code without interruption of throttling. In addition to that, this functionality will also generate an unlimited number of emails in the user's address box.

Example message:

```
{"method":"XVPN.RequestMfaCode","params":[{}],"id":3}
{"method":"XVPN.ValidateMfaCode","params":[{"code":"123456"}],"id":3}
{"method":"XVPN.RequestMfaCode","params":[{}],"id":3}
{"method":"XVPN.ValidateMfaCode","params":[{"code":"123456"}],"id":3}
{"method":"XVPN.RequestMfaCode","params":[{}],"id":3}
{"method":"XVPN.ValidateMfaCode","params":[{"code":"123456"}],"id":3}
```

It is recommended to correctly apply the throttling and brute force protections when both RPC messages are called.

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

EXP-09-004 WP2: JSON helpers should check for 0-length strings *(Low)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

During further source code audits of the JSON parsing functionalities that are used for *lightweight*, it was noticed that the string helper function might get misused. This is because the function itself expects buffer sizes that are always greater than zero, without actually checking whether the passed length-field is actually long enough. The following snippet depicts the affected code:

Affected file:
xv_helium_cli/src/json_helpers.c

Affected code:

```
he_json_object_retval_t json_object_object_get_string(json_object* jobj,
                                                      const char* key,
                                                      char* buf,
                                                      size_t buflen) {
[...]
    const char* str = json_object_get_string(jval);
    strncpy(buf, str, buflen);
    buf[buflen - 1] = 0;
```

The problem here is that the function expects the *buflen* parameter, which is the length of the supplied buffer to write the string value to, to be greater than zero. This is because it requires one byte of room for the 0-byte. However, in the case of an exactly 0-sized buffer and thus a 0-length, the subtraction of *buflen - 1* will actually wrap around and cause an out-of-bounds write. Developers who implement further usage of the *json_object_object_get_string* function might not be aware of this behavior and could introduce additional bugs into the codebase.

It is recommended to make sure that the function checks whether *buflen* is greater than zero and returns with an appropriate *retval_t* value if that is not the case.

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

EXP-09-005 WP2: Inconsistent use of *he_cli_calloc()* *(Low)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Another rather minor code quality issue was identified with the usage of the *he_cli_malloc()* function, which is called instead of *he_cli_calloc()*. When *libuv* is supposed to handle the UDP communication between clients and servers, the *uv_udp_recv_start()* function is used to register the callbacks for allocating temporary memory and receiving the actual data. The first callback is defined at *on_udp_alloc()* and can be seen in the following snippet.

Affected file:
xv_helium_cli/src/uv_callbacks.c

Affected code:

```
#define NUM_MMSG_CHUNK 16

void on_udp_alloc(uv_handle_t *handle, size_t suggested_size, uv_buf_t *buf) {
#if defined(__linux__) || defined(__APPLE__)
    // Must allocate multiples of `suggested_size` to use `UV_UDP_RECVMMSG`
    buf->base = he_cli_malloc(NUM_MMSG_CHUNK * suggested_size);
    buf->len = NUM_MMSG_CHUNK * suggested_size;
#else
    he_cli_buffer_alloc(handle, suggested_size, buf);
#endif
}
```

Since *he_cli_malloc()* is used to reserve heap memory, the arithmetic operation to determine the total size of the needed memory (*NUM_MMSG_CHUNK * suggested_size)* might overflow, depending on the value of *suggested_size*. Since no check for 0-sized allocation parameters is made, it is possible that only a very small memory region is reserved. This could be entirely prevented by using *he_cli_calloc()* instead, since the underlying *libc* call will additionally check for overflows and return an error code that can be caught.

It is recommended to simply replace the *he_cli_malloc()* call with an appropriate call to *he_cli_calloc()*.

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

Conclusions

As already suggested in the *Introduction,* this assessment of the ExpressVPN Linux client and codebase demonstrated that the components in scope have been developed and deployed with a lot of attention to security best practices. Therefore, three Cure53 testers responsible for this white-box examination only managed to spot five security-relevant issues with limited impact. The positive impression about the scope of *EXP-09* is further ensured by the fact that this summer 2022 examination benefited from a generous budget and a tight scope, thus making it less likely that some issues evaded detection.

Absence of findings beyond a *Medium* rank is yet another strong positive indicator of the condition of the security premise at the ExpressVPN Linux targets. One could argue that the scope was possibly too narrow to fully judge the codebase, but the code provided for review was consistently clean, correct and free from security pitfalls. In sum, the overall code of the main ExpressVPN CLI and VPN command and control services adhere to a very high standard. This is applicable to pretty much every area of the code that is written in the Go language. None of the common Go security flaws could be detected throughout the project. Sanitization routines of the sensitive RPC layer leading to the privileged services were complete and stopped all potential attacks that Cure53 attempted. Side-channel leaks through, for example, incorrectly set directory permissions, could also not be identified.

Only two non-serious, *Medium*-ranked issues, were spotted in EXP-09-002 and EXP-09-001 in the key area of the codebase. The first is a rediscovery of incorrectly pushed VPN interface routes. They might allow nation-state level attackers with Man-in-the-Middle capabilities to extract a user's original IP address. The latter is a faultily implemented rate-limiting mechanism that allows brute-forcing of the requested 2FA token.

Other than that, no issues of major significance were spotted during the Linux-specific audit of the Go codebase of ExpressVPN and its underlying architecture. This is also why this report is extended to include a substantial testing methodology section that highlights the overall process followed when covering the scope. Specifically, that section made it possible to present a little more detail regarding what Cure53 tried to accomplish when auditing specific parts of the application.

For the secondary in-scope items, findings such as EXP-09-003, together with the miscellaneous issues that were uncovered in this codebase, show that writing C code can be very tricky. As such, Cure53 recommends a full review of the *lightway* client is performed to ensure that such issues are identified and removed from the software stack written in C and C++.

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

It needs to be noted that the native *libxvclient* library that was only looked at using the black-box methodology during this pentest, even though it actually implements most functionalities of the underlying RPC mechanism of the VPN command and control service. Cure53 could only assess the client "frontend" or "wrapper code" written in Go which severely limited the testing depth.

Upon several requests from the testing team, some additional source files were shared, allowing Cure53 to get slightly more insights. However, the provided data turned out to reference even more source code that was missing. Generally, it can be said that this severely hindered the in-depth source code reviews and Cure53 had to rely mostly on the dynamic, closed-source testing of certain parts.

With applications written in unsafe languages like C, C++ or even *unsafe* Go code which is directly calling C functions, it is very important to understand all referenced types and functions. If the provision of additional materials was more extensive, Cure53 could have used the time to conduct additional fuzz tests and get a better overview and a much deeper understanding of the application complex as a whole.

In conclusion, it is important to note that there is a clear recommendation of performing further tests on the areas that were deemed out-of-scope or could not be audited fully. Nevertheless, the items that were in fact in scope and auditable received good coverage, as the comprehensive testing methodology chapter shows. The overall impression here is a rather good one. After a remediation of the mentioned findings, the general robustness of the code should be raised to an even better level.

Cure53 would like to thank Brian Schirmacher and Timothy Tan from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.