Performing a MITM attack on the .NETGuard desktop application

766F6964
766F6964@protonmail.com

Washi
washiwashi1337@gmail.com

October 2, 2018

Abstract

Code obfuscation is a method of preventing third parties from reverse engineering the inner workings of software. One cloud-based service that provides this kind of protection for .NET applications is .NETGuard. .NETGuard distributes a desktop application that interacts with .NETGuard's API. In this paper, we show that the protocol used by the desktop application has several security flaws. The most serious flaws include the possibility of leaking account credentials and/or the original binary being restored from the generated network traffic. Additionally, the protocol performs no verification on the network traffic, which allows a **Man In the Middle (MITM)** attack to modify packets and send malicious content back to the client.

1 Introduction

In software development, obfuscation is the deliberate act of creating source code or machine code that is difficult for humans to understand [1]. By this definition, obfuscators are the tools used to apply obfuscation. There are multiple obfuscators available; some free while others are commercial. This write-up focuses on the commercial obfuscation tool .NETGuard [2]. The intent of this article is not to determine or conclude whether .NETGuard is a good or a bad obfuscator, nor as a means of attacking the developer personally. Rather, the content of this paper focuses on critical security issues found within the .NETGuard desktop client, and the communication between the desktop client and the .NETGuard servers. The main purpose of this paper is to raise awareness about the security concerns identified and inform .NETGuard's users that their data may be at risk.

2 Background

Unlike many obfuscators, .NETGuard is a cloud-based service. In other words, the obfuscation process occurs on a remote server instead of the client's machine. This makes understanding the obfuscation much more difficult. Consequently, the file to be obfuscated must be uploaded to the remote server before the obfuscation can be applied to it. The content of this paper focuses on the communication to and from the remote server. Typically, systems are not directly connected to other systems. Instead, communication is done via an Access Point (AP). The packets are then forwarded by the AP through one or more additional APs until it reaches the destination system. Any responses made by the destination are then routed the same way to the source system. A simplified scenario with a single AP is depicted in Figure 1.

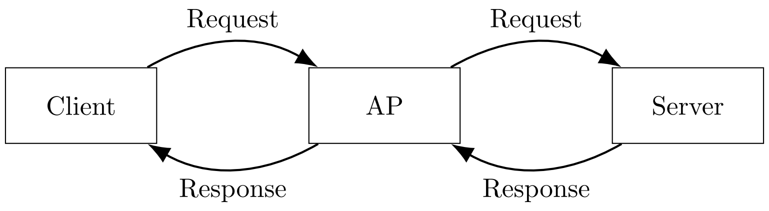


Figure 1: Network topology of a client-server model.

One type of attack on this kind of communication system is a Man-in-the-Middle (MITM) attack. A MITM attack inserts a malicious AP that acts as an eavesdropper on the communication between the client and the server. Doing so allows unencrypted packets to be modified before forwarding to the packet on [3]. This is depicted in Figure 2.

If an attacker were to intercept the network traffic when the unprotected file was being uploaded to the server, the original unobfuscated file could be obtained from the packets at the AP.

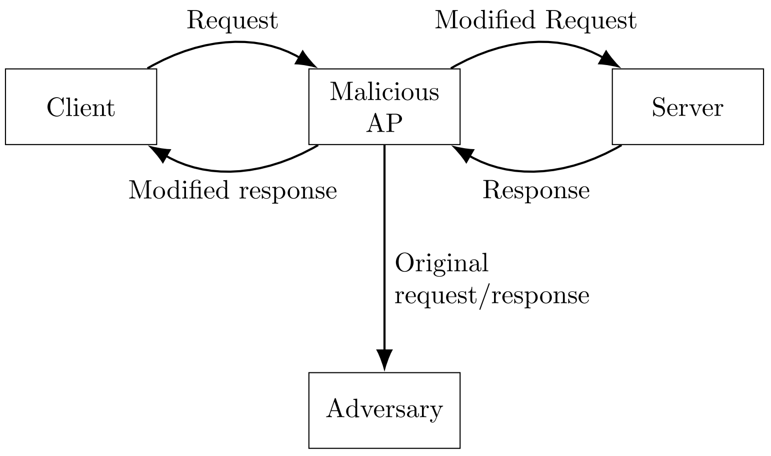


Figure 2: Network topology of a client-server model with a malicious access point, modifying the incoming and outgoing packets.

3 Methodology

To test whether the model used by .NETGuard is resistant to MITM attacks, we conduct a simple experiment with a small sample program. The sample program contains a secret phrase which will be determined if proprietary code is secure during transit. The sample source code can be found in listing 1.

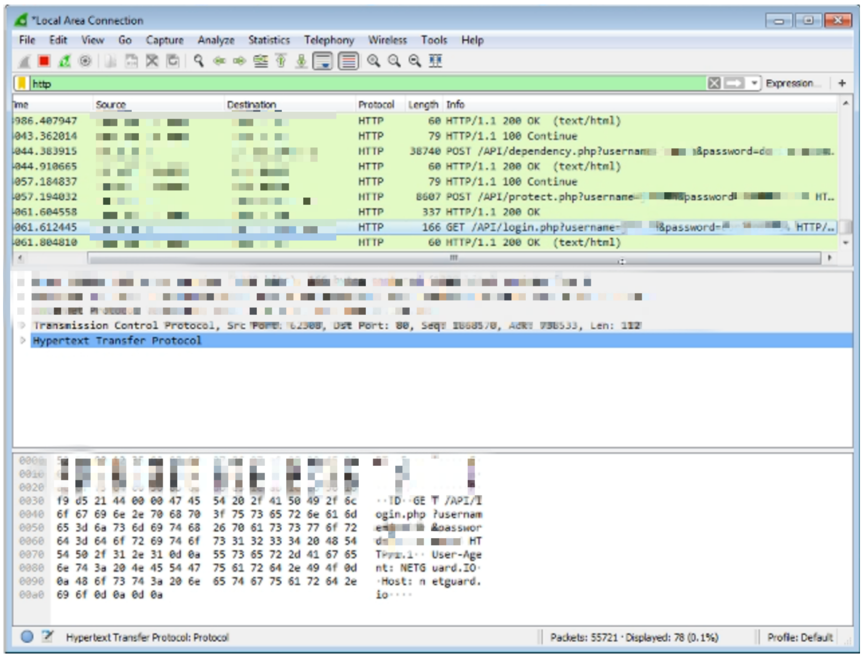
Listing 1: Sample code used in the experiment

```
public static void Main()
{
  Console.WriteLine("Secret");
  Console.ReadKey();
}
```

Before the obfuscator client is started, a new capture is begun in Wireshark [4] to capture all the incoming and outgoing traffic between the client and the server. The obfuscator is then tasked to load the sample program and make a request to protect it with at least one of the settings enabled. The capture is stopped when the obfuscator has received the obfuscated binary and written it to the output directory. The network data captured is then analyzed to assess if the communication is secure.

4 Results

In Figure 3 we can observe that the .NETGuard client application makes various HTTP requests to the server. What is important to note here is that .NETGuard deviates from standard authentication protocols. Both the username and the password are included as plain-text in the request URLs. Note, during the writing of this report, the desktop client received an update which resulted in a slight modification to the protocol used. Instead of the password being sent in plain-text; it now appears in a hashed form. This is illustrated in Figure 4. The communication between client and server is not encrypted. This can be seen clearly by looking at the packet's data when the unprotected binary is submitted to the .NETGuard server. Figure 5 shows the first few bytes of the MZ header of the test binary. In Figure 6 we can observe that the secret string of the test binary is also exposed in plain-text. This essentially means that the entire binary can be obtained by looking at the packet's raw data when the file is submitted to the server.
The traffic also shows that no verification signature (such as a checksum) is used to ensure that the submitted file was not being altered or replaced.


*Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
Expression...

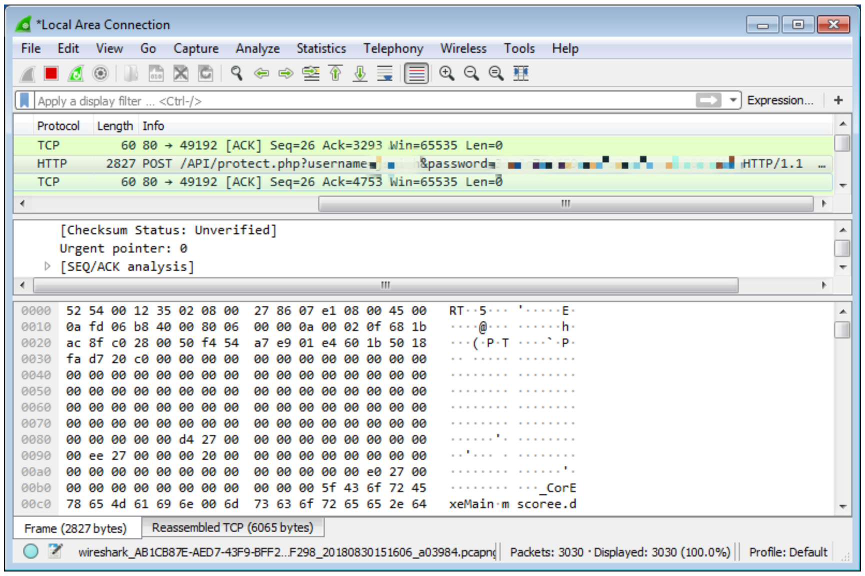
```
Time          Source      Destination  Protocol  Length  Info
986.407947    [redacted]  [redacted]   HTTP      60      HTTP/1.1 200 OK  (text/html)
043.362014    [redacted]  [redacted]   HTTP      79      HTTP/1.1 100 Continue
044.383915    [redacted]  [redacted]   HTTP      38740   POST /API/dependency.php?usernam[redacted]&password=d[redacted]
044.910665    [redacted]  [redacted]   HTTP      60      HTTP/1.1 200 OK  (text/html)
057.184837    [redacted]  [redacted]   HTTP      79      HTTP/1.1 100 Continue
057.194032    [redacted]  [redacted]   HTTP      8607    POST /API/protect.php?username=[redacted]&password[redacted] HT...
061.604558    [redacted]  [redacted]   HTTP      337     HTTP/1.1 200 OK
061.612445    [redacted]  [redacted]   HTTP      166     GET /API/login.php?username=[redacted]&password=[redacted] HTTP/...
061.804810    [redacted]  [redacted]   HTTP      60      HTTP/1.1 200 OK  (text/html)
```

```
[redacted]
Transmission Control Protocol, Src Port: 62308, Dst Port: 80, Seq: 1868570, Ack: 738533, Len: 112
Hypertext Transfer Protocol
```

```
0000  [redacted]
0010
0020
0030  f9 d5 21 44 00 00 47 45  54 20 2f 41 50 49 2f 6c   ··!D··GE T /API/l
0040  6f 67 69 6e 2e 70 68 70  3f 75 73 65 72 6e 61 6d   ogin.php ?usernam
0050  65 3d 6a 73 6d 69 74 68  26 70 61 73 73 77 6f 72   e=[redacted] &passwor
0060  64 3d 64 6f 72 69 74 6f  73 31 32 33 34 20 48 54   d=[redacted] HT
0070  54 50 2f 31 2e 31 0d 0a  55 73 65 72 2d 41 67 65   TP/1.1·· User-Age
0080  6e 74 3a 20 4e 45 54 47  75 61 72 64 2e 49 4f 0d   nt: NETG uard.IO·
0090  0a 48 6f 73 74 3a 20 6e  65 74 67 75 61 72 64 2e   ·Host: n etguard.
00a0  69 6f 0d 0a 0d 0a                                  io····
```

Hypertext Transfer Protocol: Protocol
Packets: 55721 · Displayed: 78 (0.1%)
Profile: Default

Figure 3: The protect POST request. The credentials in the URL, as well as the IP addresses are blurred out for obvious reasons.


*Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
Frame (2827 bytes)
Reassembled TCP (6065 bytes)

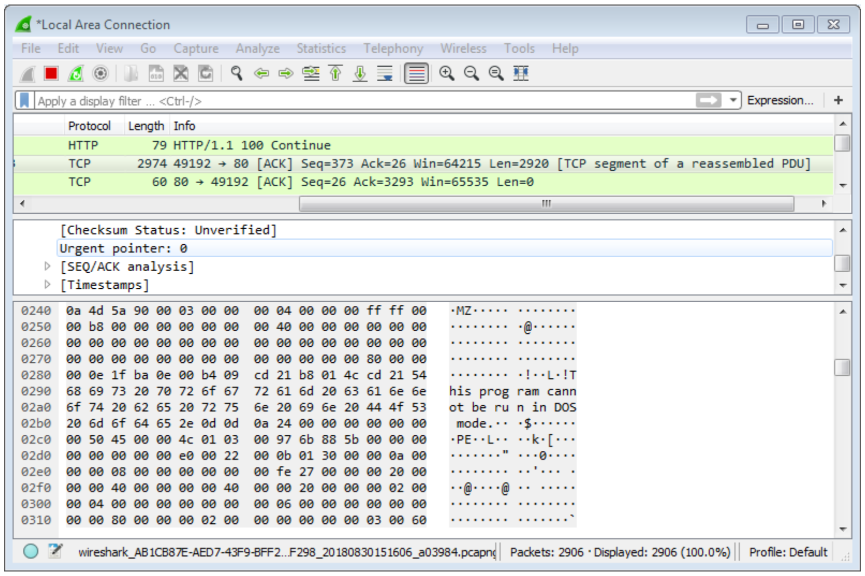
```
Protocol  Length  Info
TCP         60 80 → 49192 [ACK] Seq=26 Ack=3293 Win=65535 Len=0
HTTP      2827 POST /API/protect.php?username=[redacted]&password=[redacted] HTTP/1.1 …
TCP         60 80 → 49192 [ACK] Seq=26 Ack=4753 Win=65535 Len=0
```

```
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▷ [SEQ/ACK analysis]
```

```
0000  52 54 00 12 35 02 08 00  27 86 07 e1 08 00 45 00   RT··5··· '·····E·
0010  0a fd 06 b8 40 00 80 06  00 00 0a 00 02 0f 68 1b   ····@··· ······h·
0020  ac 8f c0 28 00 50 f4 54  a7 e9 01 e4 60 1b 50 18   ···(·P·T ····`·P·
0030  fa d7 20 c0 00 00 00 00  00 00 00 00 00 00 00 00   ·· ····· ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0080  00 00 00 00 00 d4 27 00  00 00 00 00 00 00 00 00   ······'· ········
0090  00 ee 27 00 00 00 20 00  00 00 00 00 00 00 00 00   ··'··· · ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 e0 27 00   ········ ······'·
00b0  00 00 00 00 00 00 00 00  00 00 00 5f 43 6f 72 45   ········ ···_CorE
00c0  78 65 4d 61 69 6e 00 6d  73 63 6f 72 65 65 2e 64   xeMain·m scoree.d
```

```
wireshark_AB1CB87E-AED7-43F9-BFF2...F298_20180830151606_a03984.pcapng | Packets: 3030 · Displayed: 3030 (100.0%) | Profile: Default
```

Figure 4: The protect POST request after the application update of August 20th 2018.


*Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

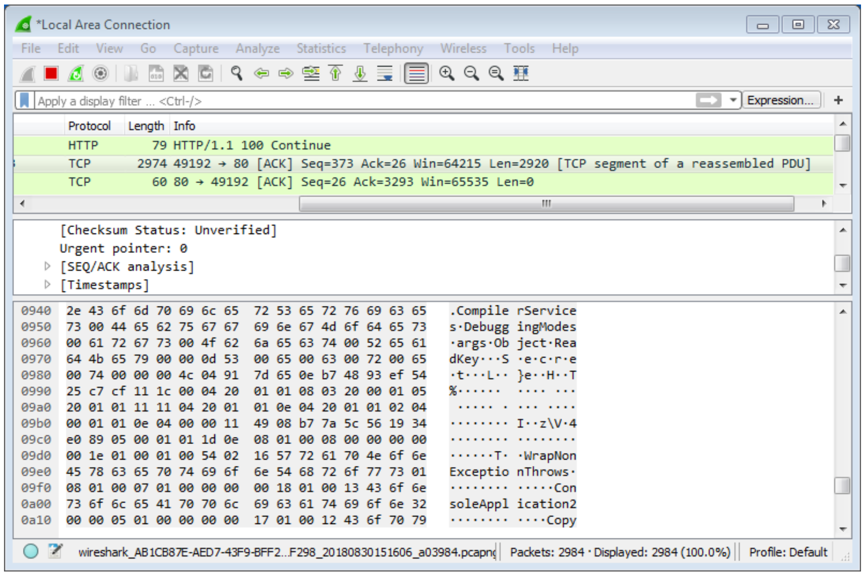
```
Protocol  Length  Info
HTTP        79 HTTP/1.1 100 Continue
TCP       2974 49192 → 80 [ACK] Seq=373 Ack=26 Win=64215 Len=2920 [TCP segment of a reassembled PDU]
TCP         60 80 → 49192 [ACK] Seq=26 Ack=3293 Win=65535 Len=0
```

```
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▷ [SEQ/ACK analysis]
  ▷ [Timestamps]
```

```
0240  0a 4d 5a 90 00 03 00 00  00 04 00 00 00 ff ff 00   ·MZ····· ········
0250  00 b8 00 00 00 00 00 00  00 40 00 00 00 00 00 00   ········ ·@······
0260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0270  00 00 00 00 00 00 00 00  00 00 00 00 00 80 00 00   ········ ········
0280  00 0e 1f ba 0e 00 b4 09  cd 21 b8 01 4c cd 21 54   ········ ·!··L·!T
0290  68 69 73 20 70 72 6f 67  72 61 6d 20 63 61 6e 6e   his prog ram cann
02a0  6f 74 20 62 65 20 72 75  6e 20 69 6e 20 44 4f 53   ot be ru n in DOS
02b0  20 6d 6f 64 65 2e 0d 0d  0a 24 00 00 00 00 00 00    mode.·· ·$······
02c0  00 50 45 00 00 4c 01 03  00 97 6b 88 5b 00 00 00   ·PE··L·· ··k·[···
02d0  00 00 00 00 00 e0 00 22  00 0b 01 30 00 00 0a 00   ·······" ···0····
02e0  00 00 08 00 00 00 00 00  00 fe 27 00 00 00 20 00   ········ ··'··· ·
02f0  00 00 40 00 00 00 00 40  00 00 20 00 00 00 02 00   ··@····@ ·· ·····
0300  00 04 00 00 00 00 00 00  00 06 00 00 00 00 00 00   ········ ········
0310  00 00 80 00 00 00 02 00  00 00 00 00 00 03 00 60   ········ ·······`
```

```
wireshark_AB1CB87E-AED7-43F9-BFF2...F298_20180830151606_a03984.pcapng | Packets: 2906 · Displayed: 2906 (100.0%) | Profile: Default
```

Figure 5: One of the packets sent to the .NETGuard server. The MZ header of the PE executable file can clearly be seen in the raw packet data.


*Local Area Connection
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
Protocol  Length Info
HTTP          79 HTTP/1.1 100 Continue
TCP         2974 49192 → 80 [ACK] Seq=373 Ack=26 Win=64215 Len=2920 [TCP segment of a reassembled PDU]
TCP           60 80 → 49192 [ACK] Seq=26 Ack=3293 Win=65535 Len=0
```

```
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▷ [SEQ/ACK analysis]
  ▷ [Timestamps]
```

```
0940  2e 43 6f 6d 70 69 6c 65  72 53 65 72 76 69 63 65   .Compile rService
0950  73 00 44 65 62 75 67 67  69 6e 67 4d 6f 64 65 73   s·Debugg ingModes
0960  00 61 72 67 73 00 4f 62  6a 65 63 74 00 52 65 61   ·args·Ob ject·Rea
0970  64 4b 65 79 00 00 0d 53  00 65 00 63 00 72 00 65   dKey···S ·e·c·r·e
0980  00 74 00 00 00 4c 04 91  7d 65 0e b7 48 93 ef 54   ·t···L·· }e··H··T
0990  25 c7 cf 11 1c 00 04 20  01 01 08 03 20 00 01 05   %······  ···· ···
09a0  20 01 01 11 11 04 20 01  01 0e 04 20 01 01 02 04    ····· · ··· ····
09b0  00 01 01 0e 04 00 00 11  49 08 b7 7a 5c 56 19 34   ········ I··z\V·4
09c0  e0 89 05 00 01 01 1d 0e  08 01 00 08 00 00 00 00   ········ ········
09d0  00 1e 01 00 01 00 54 02  16 57 72 61 70 4e 6f 6e   ······T· ·WrapNon
09e0  45 78 63 65 70 74 69 6f  6e 54 68 72 6f 77 73 01   Exceptio nThrows·
09f0  08 01 00 07 01 00 00 00  00 18 01 00 13 43 6f 6e   ········ ·····Con
0a00  73 6f 6c 65 41 70 70 6c  69 63 61 74 69 6f 6e 32   soleAppl ication2
0a10  00 00 05 01 00 00 00 00  17 01 00 12 43 6f 70 79   ········ ····Copy
```

```
wireshark_AB1CB87E-AED7-43F9-BFF2...F298_20180830151606_a03984.pcapng | Packets: 2984 · Displayed: 2984 (100.0%) | Profile: Default
```

Figure 6: The same packet as referenced in Figure 5. This time the secret phrase can be seen in the hex dump.

5 Discussion

5.1 Credential leakage

As mentioned in section 2, any information that is being sent over the internet that is not encrypted, can be obtained by an adversary that is capturing the network traffic. Sending account credentials in plain-text therefore allows the eavesdropper to take control of the .NETGuard account. Furthermore, most internet users tend to use the same passwords for other services (such as e-mail or business accounts) [5]. Transferring credentials in plain-text puts them at risk without them even being aware of it.

After the update on August 20th, the password was not found in plain-text anymore but appears as a hash code. After a few attempts, the hash code was determined to be the result of applying the MD5 hashing algorithm twice on the original password. MD5 is considered an insecure hash function, and collisions or the original password can be found in a relatively short time span. Projects like HashCat are able to test billions of candidates per second on a single GPU, rendering a brute-force attack feasible [6]. Furthermore, more systematic methods have also been published such as the one suggested by Mark Stevens in 2012 [7].

5.2 Intellectual property theft

In addition to the possibility of compromised accounts, there is also the possibility of proprietary code being 'leaked' from being sent over the network unencrypted. As discussed in section 1, the goal of obfuscation tools is to hide intellectual property to prevent it from being stolen. Therefore, it is of most importance that the original source code is not shared with anyone. However, with no encryption scheme in place, any person on the same network could intercept the traffic and reassemble the original unprotected binary from the packets, and use reverse engineering tools such as a disassembler or a decompiler to extract the original source code.

5.3 Absence of verification

Also important to note, is that there does not seem to be any form of endpoint verification. This could mean that an adversary would be able to replace the entire payload with a malicious executable, unbeknownst to the client and/or the server. This is a critical problem, as the malicious executable will be obfuscated, and eventually be sent back to the user. Since its very common for a user to test their application after the obfuscation is applied - since extreme obfuscation can result in an executable that does not run - they could easily become the victim of a malware attack.

5.4 Proof of Concept

Using Python 3 and the libraries NetfilterQueue [8] and Scapy [9], a Proof of Concept (PoC) was built to demo such a malicious AP, which replaces the string message with a new one upon receiving the obfuscated binary. The implementation is publicly available [10].

6 Conclusion

.NETGuard is a cloud code obfuscation tool that distributes a desktop application for communicating with the remote server. We have shown that, at the time of writing this report, the communication between the application and the server is highly vulnerable to MITM attacks. We demonstrated that with some very rudimentary traffic analysis, an attacker is able to steal account credentials, as well as the intellectual property of the programmer. We also provided a proof of concept that shows that the protocols used lack any form of authentication and/or verification of the traffic to prevent tampering, allowing an adversary to modify the packets in such a way that the final obfuscated file is replaced with malicious content. [11] This means not only the software's source code is at risk but also the user's system can potentially become the victim of a malware attack. Based on these observations we strongly advise all .NETGuard customers to change their passwords. It is not recommended to use .NETGuard before these vulnerabilities are addressed. We encourage the developers to properly implement an encryption and verification scheme, such as SSL, to ensure the file was not altered before it has been sent to or from the server.

We have communicated these security issues to the .NETGuard team, but they have fallen on deaf ears. Other than changing the plain-text passwords to a double MD5 hash, no further action was taken. A month in, the vulnerabilities are still un-patched, so we took it upon ourselves to communicate our findings to the community. Now it is for the public to decide whether or not to continue using this software.

7 Acknowledgements

We would like to thank CodeBlue for taking his time to proofread this paper.

References

[1] *Wikipedia: Obfuscation (software)*. URL: https://en.wikipedia.org/wiki/Obfuscation_(software) (visited on 08/30/2018).

[2] *.NET Guard*. URL: https://www.netguard.io/ (visited on 08/30/2018).

[3] *What Is a Man-in-the-Middle Attack (MITM)?* URL: https://www.techopedia.com/definition/4018/man-in-the-middle-attack-mitm (visited on 08/30/2018).

[4] *Wireshark - Go Deep*. URL: https://www.wireshark.org/ (visited on 08/30/2018).

[5] *52% of users reuse their passwords*. URL: https://www.pandasecurity.com/mediacenter/security/password-reuse/ (visited on 08/30/2018).

[6] *hashcat - advanced password recovery*. URL: https://hashcat.net/hashcat/ (visited on 09/19/2018).

[7] Mark Stevens. *Single-block collision attack on MD5*. 2012.

[8] *python-netfilterqueue*. URL: https://github.com/kti/python-netfilterqueue (visited on 09/19/2018).

[9] *Scapy - Packet crafting for Python2 and Python3*. URL: https://scapy.net/ (visited on 09/19/2018).

[10] *.NetGuard MITM python script*. URL: https://gitlab.com/rtn-team.cc/public/netguard-mitm (visited on 09/23/2018).

[11] *Performing a MITM attack on the .NetGuard desktop client*. URL: https://www.youtube.com/watch?v=J6Qn9k7NMfg (visited on 10/02/2018).