Replay Attack Defense of V2Ray

Junchen Bao, Mingjie Zhu, Sicheng Wang, Yinliang Wang, Yuxiao Ran
University of California, San Diego
{jubao, miz155, siw029, yinliang, yuran}@ucsd.edu

1 Introduction

1.1 The Great Firewall

The Great Firewall is the combination of legislative actions and technologies enforced by the People's Republic of China to regulate the Internet domestically.[1] It's used to block access to selected foreign websites and to slow down cross-border internet traffic. Great Firewall operates by checking transmission control protocol packets for keywords or sensitive words. If there are keywords or sensitive words in the TCP packets, access will be closed. If one link is closed, more links from the same machine will be blocked by the Great Firewall.[2] The effect includes: limiting access to foreign information sources, blocking foreign internet tools. So, if you want to search something in China, you will find you can't even get to the Google website.

1.2 Virtual Private Network

In order to access blocked websites, people in China will use Virtual Private Network. It's also called VPN. It enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network. The benefits of a VPN include increases in functionality, security, and management of the private network. It provides access to resources that are inaccessible on the public network and is typically used for remote workers. A VPN is created by establishing a virtual point-to-point connection through the use of dedicated circuits or with tunneling protocols over existing networks. A VPN available from the public Internet can

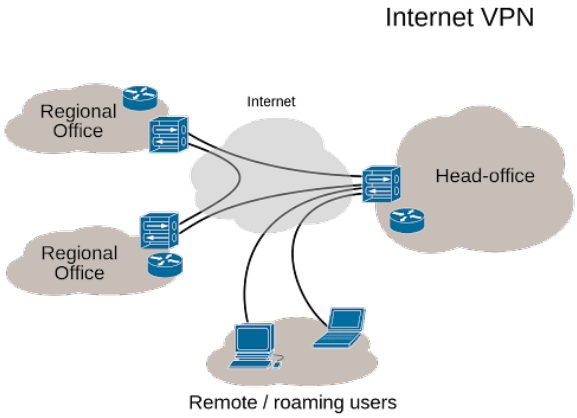

Figure 1: Virtual Private Network

provide some of the benefits of a wide area network (WAN). From a user perspective, the resources available within the private network can be accessed remotely.[3] Users will choose network nodes other than China, such as Japan, America, Singapore, to bypass the firewall.

1.3 Shadowsocks

Shadowsocks(SS) is a free and open-source encryption protocol project, widely used in China to bypass the GFW. It was created in 2012 by a Chinese programmer. But in 2015, programmers announced in a GitHub thread that they had been contacted by the police and could no longer maintain the project. Now, the data flow sent by ShadowSocks will be easily intercepted by the GFW. When you try to connect to the server, the GFW will send a replay attack package at the same time. Although the server can avoid replay attacks, the GFW knows it's a ShadowSocks server. After that, the GFW will cut the connection between client and server. So, no one chooses the Shadowsocks now.

1.4 V2Ray

V2Ray is the best choice to bypass the Great Firewall in recent years. It's a tool to provide users with the software to deploy specific network environments privately. Though V2Ray can be used for various purposes, its primary mission is to combat internet censorship. Through the use of V2Ray and its protocols, users in China can access content that would otherwise be blocked. When deploying V2Ray, you can dynamically change port, access advanced routing features, and more. The advancement of V2Ray is that data is encrypted to avoid GFW interception. The chief protocol in V2Ray is VMess.

1.5 V2Ray Features

- Multiple inbound/outbound proxies: one V2Ray instance supports in parallel multiple inbound and outbound protocols. Each protocol works independently.
- Customizable routing: incoming traffic can be sent to different outbounds based on routing configuration. It is easy to route traffic by target region or domain.
- Multiple protocols: V2Ray supports multiple protocols, including Socks, HTTP, Shadowsocks, VMess etc. Each protocol may have its own transport, such as TCP, mKCP, WebSocket etc.
- Obfuscation: V2Ray has built in obfuscation to hide traffic in TLS, and can run in parallel with web servers.
- Reverse proxy: General support of reverse proxy. Can be used to build tunnels to localhost.
- Multiple platforms: V2Ray runs natively on Windows, Mac OS, Linux, etc. There is also third party support on mobile.

2 VMess Protocol

VMess protocol is an original protocol of V2Ray, designed for encrypted communication. It's a TCP-based protocol, using TCP to transmit all data.

2.1 VMess Features

It's a stateless protocol, which means that the clients and server will directly send and receive data without handshake first. The work process is: First client want to get some data out of the GFW, so client will encrypt this data request and add some information according to VMess protocol. After server receives the client request, it will decrypt and check legitimation and decide how to response only depending on data from this request. (This will be changed after add features for replay attack defend. The server can record the sessionID, which will influence the server behavior) In the end, the client will decrypt server response and get the requested data.

To add some variance so that it will be harder for the GFW to detect, VMess is designed to be asymmetric, which means client request and server response will have different format. Here we will introduce the client request format in details. Because later the man-in-the-middle attack will get this data and modify some bits to perform the replay attack.

2.2 Client Request Details

16 bytes	X bytes	The rest
Certification Information	Instruction part	Data part

Figure 2: Client Request Structure

A client request contains 3 parts: Certification Information, Instruction Part, Data Part. Certification part is for user and time validation. The first part of Instruction Part influences how the server will deal with the second part of Instruction Part and Data Part.

2.2.1 Certification Information

Certification is the out put of Hash-based message authentication code(HMAC)[4]. MD5[5] used to be default hash function setting. After 2022/01/01 it's deprecated, if you don't change the code, you cannot use MD5 version even if you configure it. And since 2020 AEAD version of V2Ray is recommended. We will research on MD5 version of V2Ray. There are

still people using old versions of V2Ray.(not pure VMess, but VMess plus TLS/wsweb to confuse the GFW. VMess can work with many other technologies together, that's also the reason why it's so famous now) The key and message of HMAC are "Client User ID" and UTC time. Client user ID is random 16 bytes number, works as a token. And the timestamp is for the server to check request time are within 30 seconds, this design we think is for reducing the risk of man-in-the-middle utilizing requests long time ago to do replay attack.

2.2.2 Instruction Part

The second part, Instruction Part, contains many elements. We'll introduce some important ones. Firstly, this part contains the information about encryption method, server utilizes these information to know how to process the third part - Data Part. What's more, Margin P is also in this part, which will influence the bytes received by server. V2Ray server receives instruction part in 2 steps: First read 38 bytes, process and set the corresponding data structures(including P). And then read another N+4+P bytes, process and do the checksum verification. The N is bytes of address, because IPv4 and IPv6 are in different lengths. The P value is for adding variance on the client request length, making it not easy detected by the GFW. But in the end, the GFW can perform replay attack using this feature. We will introduce this in details later.

3 Weaknesses of V2Ray

We have taken a close look at V2Ray-core source code and researched on the weaknesses V2Ray in order to see what possible vulnerabilities attackers may exploit to identify its traffic, and what defense techniques we may develop.

3.1 Unique TLS ClientHello Fingerprints

In early versions ($\leq$ 4.23.2), V2Ray would send TLS ClientHello messages with very unique fingerprints. This was caused by a special set of cipher suites that the developers hardcoded in the configuration file of its TLS module[1], where it imported the Golang *crypto/tls* package. More specifically, when V2Ray set up its TLS Config struct, it used a fixed set of TLS cipher constants instead of the default settings (Figure 4).

```
config.CipherSuites = []uint16{
        tls.TLS_AES_128_GCM_SHA256,
        tls.TLS_AES_256_GCM_SHA384,
        tls.TLS_CHACHA20_POLY1305_SHA256,

        tls.TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
        tls.TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,
        tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
        tls.TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
        tls.TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
        tls.TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
        tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
        tls.TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
}
```

Figure 4: Hardcoded cipher suites in the V2Ray TLS module

Such a fixed set of cipher suites would result in an easily identifiable series of packet bytes. The GFW would be able to deploy a machine learning model, or simply match these special fingerprints to identify TLS traffic from V2Ray with a high accuracy and block the traffic. In fact, according to the *crypto/tls* documentation[2], if *CipherSuites* is *nil*, a safe default list will be used instead, and the default cipher suites may change over time, which will make the TLS traffic less identifiable. This vulnerability was mitigated in later versions of V2Ray by reverting back to this default setting.

3.2 Vulnerable to Replay Attacks

As mentioned in the previous section, VMess protocol relies on the first 16 bytes (certification information) in a client request to authenticate the connection. It records the timestamp and user information of a request and store them in a hash map, so that additional

[1] V2Ray TLS module: https://github.com/v2ray/v2ray-core/blob/edb4fed3/transport/internet/tls/config.go

[2] Golang *crypto/tls* type *Config* documentation: https://pkg.go.dev/crypto/tls#Config

1 byte	16 bytes	16 bytes	1 byte	1 byte	4 bit	4 bit	1 byte	1 byte	2 bytes	1 byte	N bytes	P byte	4 bytes
Version number Ver	Data encryption IV	Data encryption Key	Response authentication V	Option Opt	Margin P	Encryption method Sec	Keep	Command Cmd	Port Port	Address type T	Address A	Random value	Check F

Figure 3: Instruction Part Details

```
user, timestamp, valid := s.userValidator.Get(buffer.Bytes())
if !valid {
        return nil, newError("invalid user")
}

iv := hashTimestamp(md5.New(), timestamp)
vmessAccount := user.Account.(*vmess.MemoryAccount)

aesStream := crypto.NewAesDecryptionStream(vmessAccount.ID.CmdKey(), iv[:])
decryptor := crypto.NewCryptionReader(aesStream, reader)

buffer.Clear()
if _, err := buffer.ReadFullFrom(decryptor, 38); err != nil {
        return nil, newError("failed to read request header").Base(err)
}
```

Figure 5: V2Ray server blindly reads more bytes after authentication that attackers may bypass

requests from valid users may not need to be re-authenticated before its certification information expires in around 30 seconds. However, this will still allow possible of replay attacks, if the attacker records previously used valid certification information, reuses it before its expiration and sends out more malicious requests. In fact, upon receiving a new request, it looks up these information in the hash map, and without further validation, the server starts to read more bytes from the stream (Figure 5).

As an attacker can send out multiple requests by replaying the same credential before it expires, and V2Ray server blindly reads the bytes in the instruction part, due to the malleability of the request, the attacker may forge different legitimate requests each time with a different P value, which is the 4-bit length of the padding. V2Ray server shuts down the connection when it detects an errored checksum value in the last 4 bytes of the request. At that point, the attacker may be able to measure the length of Data part (M). This is similar to a "padding oracle", and we will explain more in the next section.

4 Replay Attacks

Long story short, the replay attack is that the GFW maliciously amend the command part and send this to the server. Then the server will have certain reactions to the changed request, like reading a few more bytes then closing the connection. The GFW is using these behaviors to determine if the server is providing a V2Ray service.

To explain the replay attack thoroughly, understanding V2Ray's code becomes necessary[3]. The main vulnerability is in server.go, function DecodeRequestHeader(line 124) in specific. On line 132, it validates the user's identity by the 16 bytes hashing of userID and timestamp. As this information is valid in a certain period of time, the attacker can simply reuse the same hashing to pass validation. After that, the server initializes the AES decryption stream and reads in the first 38 bytes of command without any further validation.

On line 159, there is a function call, s.sessionHistory.addIfNotExits that can return an error indicating that the server may be under replay attack. However, this line of code does not protect the server at all. The addIfNotExits

[3]Source code for V2ray server: server.go

function on line 53 is checking if the session ID exists and has not expired, if so it closes the connection due to suspicion of replay attack. The problem is that the session ID(line 23), is a self-defined structure made of three 16 byte fields, user, key and nonce. If any one of these three fields changed then that's a new session ID, thus can pass the addIfNotExits checking.

The detailed steps of the replay attack is as following:

Step 1: Send replay with the first 16+38 bytes of a valid connection, everytime change the last bit of Encryption key, and Margin P.

Step 2: Send M bytes of zero, wait until server closes the connection

Step 3: Do step 1 and 2 for 16 times, record M values, if those values are X+0, X+1, X+2 ... X+15, let X be any number, it is VMess connection

The reason that this attack works is that:

The First 16 bytes are based on timestamp and userID and can be reused to pass validation. Then, Changing the last bit of Encryption key can pass the check of session ID in addIfNotExits. Because of instruction is AES-CFB encrypted, a cipher that encrypt each 16 byte block separately, and this changed bit is in same block with Margin P, thus there will be no error propagation. Finally, the server reads in the address, padding and checksum. Then it will close the connection after finding out that the checksum is wrong. Knowing that M is the number of byte that the server read after the first 38 bytes, and M = address + padding + checksum, only padding length can change.

5 Potential Solutions for Replay Attacks

To bypass replay attack, we need some modifications to VMess protocol. In VMess protocol, the server closes the connection immediately after it fails the verification of the checksum. And this property is exploited by the attacker to conduct the replay attack.

A possible solution is, for the server, after it fails the verification, it doesn't close the connection immediately, but receive several more bytes, which is a random number between 0 and 5, and then close the connection. With this small tricks, we can successfully confuse the attacker. Previously, with the original VMess protocol, after the attacker receives a request from the client, it will send 16 separate malicious requests to the server, and if the values of M are distinct and are $X + 0$, $X + 1$,..., $X + 15$, the attacker can almost guarantee that the server is running some VMess service. But now, with our optimized version of VMess, the server doesn't close the connection immediately after it fails the verification. So the attacker can't manage the man-in-the-middle attack. The pseudocode of original VMess protocol and our optimized version are shown in algorithm 1 and algorithm 2, respectively.

Algorithm 1 VMess DecodeRequestHeader

Input Request header
1: Read first 16 bytes
2: **if** !valid(16 bytes) **then**
3: Close the connection
4: **end if**
5: Read 38 bytes and decode
6: ReadAddressPort()
7: Read P+4 bytes ▷ P bytes for random value, and 4 bytes for Check F
8: **if** !valid(decrypted content) **then**
9: Close the connection
10: **else**
11: Read the following bytes (data part)
12: **end if**

6 Experiment

We then conduct the experiment to verify our ideas. Both of the codes for V2ray and the man-in-the-middle attacker are open-source codes[4][5].

For the details of the setting, we set up two v2ray instances, one on the local machine, one on the server. For the client, we use VMess protocol to send to 4444 port, and for server, we use VMess to listen on 4445 ports. Also, on the client, 1080 port accepts socks traffic. So

[4]Source code for V2ray: https://github.com/jarvisgally/v2simple

[5]Source code for man-in-the-middle-attack: https://github.com/v2ray/v2ray-core/issues/2523#issue-628032465

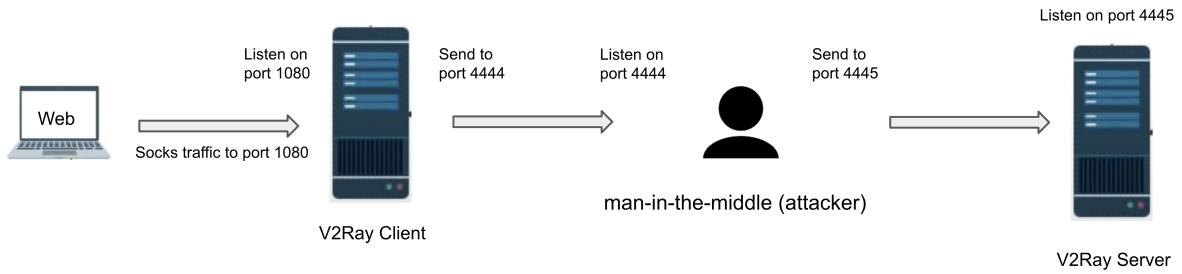

Figure 6: Illustration for the topology of our experiment

Algorithm 2 Ours

Input Request header

Read first 16 bytes
if !valid(16 bytes) **then**
    Close the connection
end if
Read 38 bytes and decode
ReadAddressPort()
Read P+4 bytes ▷ P bytes for random value, and 4 bytes for Check F
if !valid(decrypted content) **then**
    for $j \in range(0, random\_value)$ **do**
        Read one byte
    end for
    Close the connection
else
    Read the following bytes (data part)
end if

the browser first sends socks request to 1080 port, the client send VMess request to 4444 port. Attacker starts man-in-the-middle attack, gets the first 16 bytes to pass validation, then send the malicious incomplete command to the 4445 ports of the server, and conduct the replay attack. The illustration for the topology of our setting is shown in Figure 6.

Note that in a real life setting, the client should directly send to the server, but here to simulate the man-in-the-middle attack, the client sends data to the attacker, and then attacker sends 16 separate requests to the server to conduct the replay attack.

The result shows that, for the original VMess protocol, the attacker can successfully verify that if the server is running some VMess protocol. But for our optimized version of VMess protocol, the attacker fails the replay attack, which shows that our solution can successfully defend the replay attack.

7 Conclusion

In this project we learned about the Great Fire Wall, Virtual Private Network and Shadowsocks. We examine two vulnerabilities of V2Ray: unique TLS ClientHello fingerprints and replay attack resulting from Margin P mechanism. Then we dive into the replay attack and learned its specific logic. Finally, we developed a method to protect the V2Ray server from the replay attack, and the experiment result shows the feasibility of our method.

References

[1] Clayton, Richard, Steven J. Murdoch, and Robert NM Watson. Ignoring the great firewall of china.

[2] Richard Clayton, Steven J. Murdoch, , and Robert N. M. Watson. Ignoring the great firewall of china. 2006.

[3] Davies. Ipv6 traffic over vpn connections. 2021.

[4] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication, 1997.

[5] Ronald Rivest. The md5 message-digest algorithm. Technical report, 1992.