Analyze Network Traffic Characteristics of Network Proxies Based on Socks5 Protocol

Hongsen Wang, Peidong Jiang, Rong Xia and Shuo Shen

Sichuan University

Abstract: How to monitor the behavior of applications is a challenging issue and the basic of realizing this monitor is identifying the traffic generated by the application accurately. We complete traffic characteristic analysis and identification for Shadowsocks(SS) and ShadowsocksR(SSR), two over-the-top applications, based on Socks5 protocol. In the traffic characteristic analysis phase, the traffic generated by the application program is captured to analyze the characteristics that can uniquely represent the traffic produced by the application program. In the traffic identification phase, the network data packets generated on the computer are collected first, and the corresponding application is identified using feature matching program.

Ⅰ. Introduction

Since China established the Great Wall of Firewall in 1998[1][2], we have been able to effectively identify some of the network agents, such as PPTP, L2TP, IPSec, and OpenVPN. In recent years, some socks5-based agents, such as Shadowsocks and SSR (a modified version of Shadowsocks), have emerged[3]. They use the principle of end-to-end encryption, which has brought great difficulties for the identification of traffic characteristics. Because the

firewall Great Wall system and the requirements for traversing the network blockade are unique to China, and these projects have certain confidentiality, we rarely see related technical documents in the public domain.

Proxy server is kind of server that acts as an intermediary for requests from clients seeking resources from other servers. With the rapid advance of Internet, proxy software is widely used for many reasons such as Improving access speed, Hiding real IP[4]. Though it brings us lots of convenience, it also brings some risks in Privacy and security. To ensure the safe use of proxies, we should monitor the behavior of proxy software and analyze network traffic characteristics of it so that we can control the application behavior.

The foundation of network traffic characteristics analysis is to precisely identify the traffic generated by the application. Because of the widespread use of technologies such as Secure Sockets Layer (SSL), Secure Shell (SSH) and Internet Service Providers' (ISP's) blockade of P2P applications, as well as the limitation of some companies for Instant Messaging (IM) and streaming media (such as YouTube), it is not easy for us to achieve this analysis. We choose to focus on the protocols and use the Wireshark software to accomplish a large number of packet captures and manual analysis during the connection between the applications that need to be identified and the proxies.

Traffic characteristic experiment was conducted by Deng, who applied for a patent that

identifies the characteristics of Shadowsocks[5] . This document describes the use of a random forest algorithm to learn and detect Shadowsocks' traffic characteristics. However, learning requires a large sample size (generally more than 1GB), which requires a lot of computing resources. The actual shadowsocks will not produce such a huge amount of data in the use of ordinary users, which brings difficulties to machine learning[5]. 6oreover, this model does not have the ability to identify improved SSR, which requires us to use another line of thought to study the traffic characteristics of Shadowsocks and SSR.

Applications based on SOCKS take advantage of Secure Shell (SSH) as a cryptographic network protocol for operating network services securely over an un secured network[7][8]. In the dominant web architecture, client-services architecture, SSH set up a secure network connecting a client with proxy servers. The main authentication mechanism that confirm remote host is public-key cryptography, especially Rivest–Shamir–Adleman algorithm (RSA)[9][10]. RSA is one the first public-key cryptosystem and widely used in secure data transmission. In RSA cryptosystem, two keys are required. Encryption key is public key and decryption is private key that only hold in the sender. The difficulty of factorization of the product of large prime numbers is the mathematic fundamental[11]. Experts and the practice have proved that 1024 bits and more can considered as computation security that it can not be uncover the key via the powerful supercomputer during the period of key validity.[12]

III .Methods

(1) Shadowsocks

Shadowsocks (SS) divides the Socks5 proxy into SS local and SS server. The workflow is as Fig.1. First, the client sends a request to the SS local in socks5. SS local are usually local hosts or routers and any other machine, they do not go through the Great Firewall (GFW), so they will not be disturbed by the GFW. Between the SS local and the SS server, they can communicate through various encryption methods[13][14]. Therefore, packets that pass through the GFW are displayed as public TCP packets. These data packets have no obvious function, and the GFW cannot decrypt the data, causing the GFW to fail to detect and interfere with the data. Finally, the SS server decrypts the received encrypted data, sends the real request to the actual server, and sends the response data to the SS local.

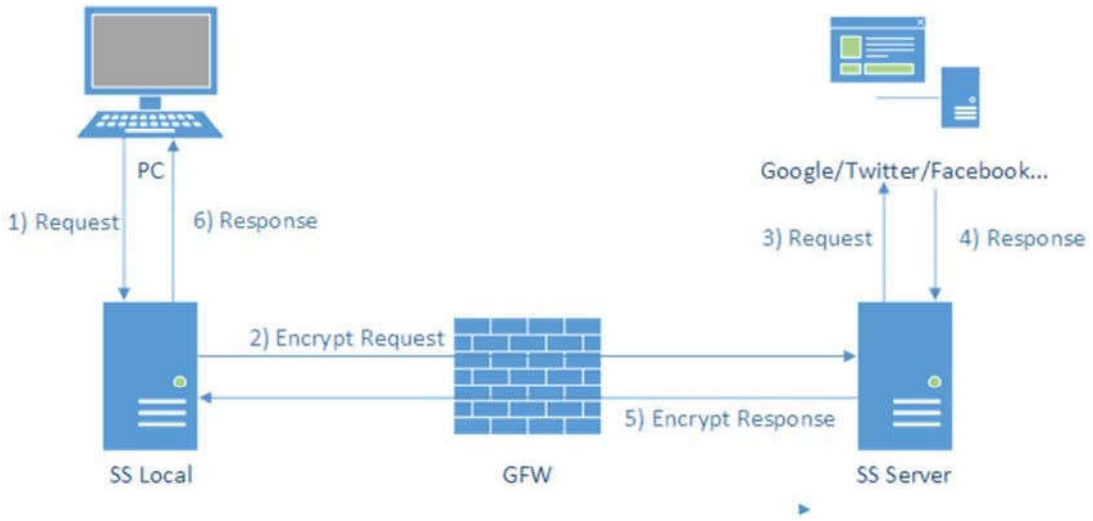


Fig.1. The workflow.

Fig.2 is the local peer's code, and the sever peer's code is the same as the local's but only it distinguishes between local and server when handling events (Fig.3).

```
try:
    logging.info("starting local at %s:%d" %
                 (config['local_address'], config['local_port']))

    dns_resolver = asyncdns.DNSResolver()   #asynchronous dns server
    tcp_server = tcprelay.TCPRelay(config, dns_resolver, True)  #depen
    udp_server = udprelay.UDPRelay(config, dns_resolver, True)  #resem
    loop = eventloop.EventLoop()
    dns_resolver.add_to_loop(loop)
    tcp_server.add_to_loop(loop)
    udp_server.add_to_loop(loop)

    def handler(signum, _):
        logging.warn('received SIGQUIT, doing graceful shutting down..')
        tcp_server.close(next_tick=True)
        udp_server.close(next_tick=True)
    signal.signal(getattr(signal, 'SIGQUIT', signal.SIGTERM), handler)

    def int_handler(signum, _):
        sys.exit(1)
    signal.signal(signal.SIGINT, int_handler)

    daemon.set_user(config.get('user', None))
    loop.run()
```

Fig.2. The source code of local.

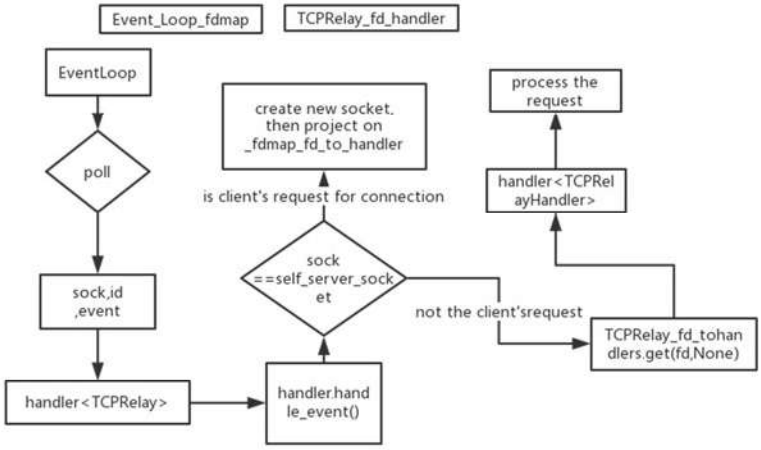
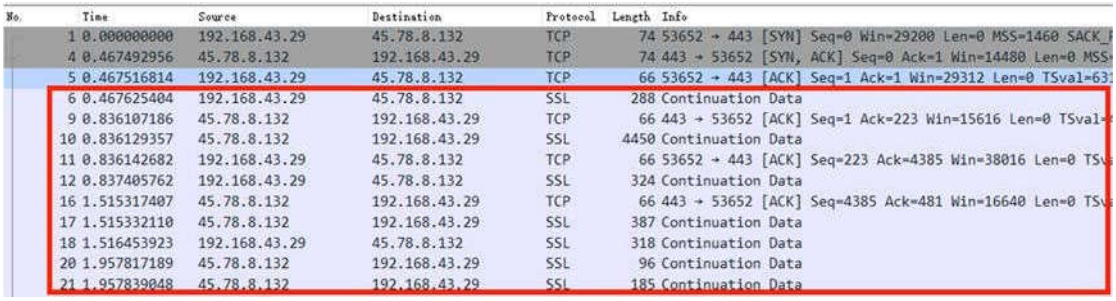


Fig.3. The workflow of handling events.

After a TCP three-way handshake with the server, the SS client and server do not perform authentication and key exchange because the key and the encryption method have been negotiated through the secure channel. Instead, they directly receive and send SSL data. As shown in the Fig.4, the 192.168.43.29 host sends an

SSL packet directly after three handshakes with the server. Without identification and key exchange, it became the first feature of SS.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.43.29	45.78.8.132	TCP	74	53652 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_
4	0.467492956	45.78.8.132	192.168.43.29	TCP	74	443 → 53652 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS
5	0.467516814	192.168.43.29	45.78.8.132	TCP	66	53652 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=63
6	0.467625404	192.168.43.29	45.78.8.132	SSL	288	Continuation Data
9	0.836107186	45.78.8.132	192.168.43.29	TCP	66	443 → 53652 [ACK] Seq=1 Ack=223 Win=15616 Len=0 TSval=
10	0.836129357	45.78.8.132	192.168.43.29	SSL	4450	Continuation Data
11	0.836142682	192.168.43.29	45.78.8.132	TCP	66	53652 → 443 [ACK] Seq=223 Ack=4385 Win=38016 Len=0 TSv
12	0.837405762	192.168.43.29	45.78.8.132	SSL	324	Continuation Data
16	1.515317407	45.78.8.132	192.168.43.29	TCP	66	443 → 53652 [ACK] Seq=4385 Ack=481 Win=16640 Len=0 TSv
17	1.515332110	45.78.8.132	192.168.43.29	SSL	387	Continuation Data
18	1.516453923	192.168.43.29	45.78.8.132	SSL	318	Continuation Data
20	1.957817189	45.78.8.132	192.168.43.29	SSL	96	Continuation Data
21	1.957839048	45.78.8.132	192.168.43.29	SSL	185	Continuation Data

Fig.4. The packages of SS.

In the source code, we found that both the server and the client set the TCP_NODELAY option when performing TCP communication. This option disables Nagle's algorithm, and its data packet presents the following characteristics: When a party to the communication receives a packet After that, it immediately sends a TCP ACK packet to the other party, and the length of the data packet is 66 bytes. The code in Fig.5 comes from the tcprelay.py module, which is used by both the client and server. As shown in Fig.6, packets 114 and 115 indicate that as soon as a packet is received, the receiver will immediately send a TCP ACK packet to respond with a very small interval; packets 121 and 122 indicate that even if there is a need to send For the data packet, the sender will not adopt piggy-back confirmation. It will send a TCP ACK packet before the packet is sent to confirm the reply to the previously received data packet.

```
#prohibit Nagle algorithm
local_sock.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
```

Fig.5. The code in tcprelay.py.

When accessing a specific website, there are multiple ports that establish a TCP connection with the server's port 443. SSL data packets that each port

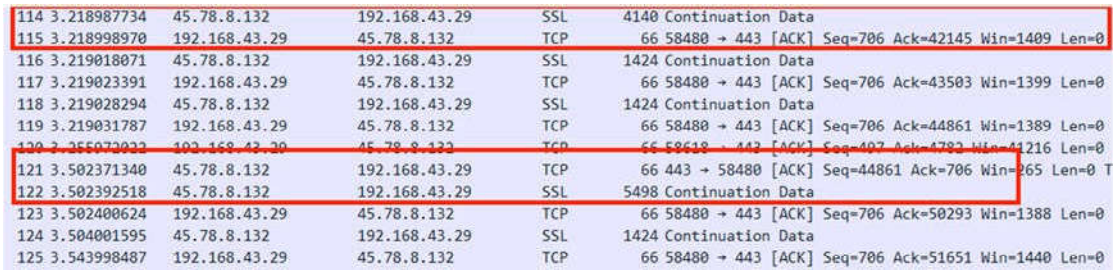


Fig.6. The packages of SS.

communicates with the server show highly consistent sequence characteristics. We use developer.github.com as an example. Sequences +623, -242, +117, +326, -3691, +192, -117, +306, -413, +192, -117, the length of packages are very common, showed in Fig.7-9. With highlighting in Fig.10, we can also see some characteristics of SS sending and receiving data through such a sequence. And we also analyzed the sequence characteristics when visiting google.com (Fig.11-13).

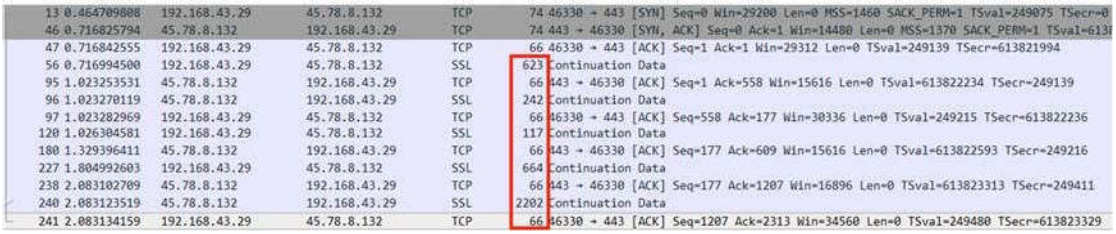


Fig.7. The packages of SS.

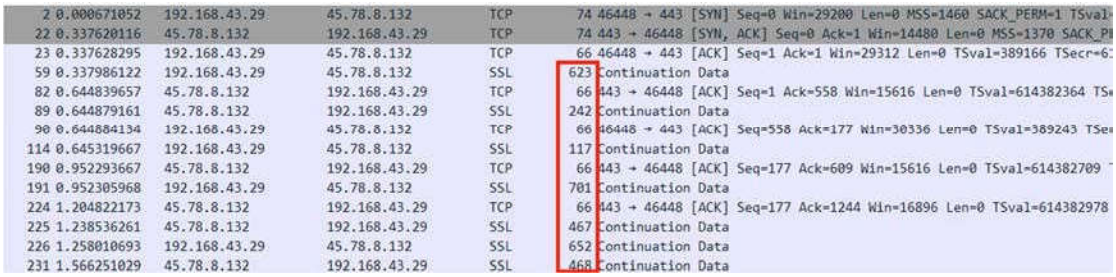


Fig.8. The packages of SS.

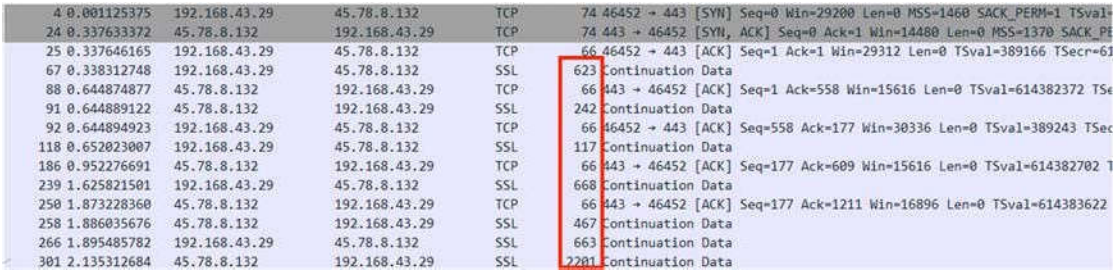


Fig.9. The packages of SS.

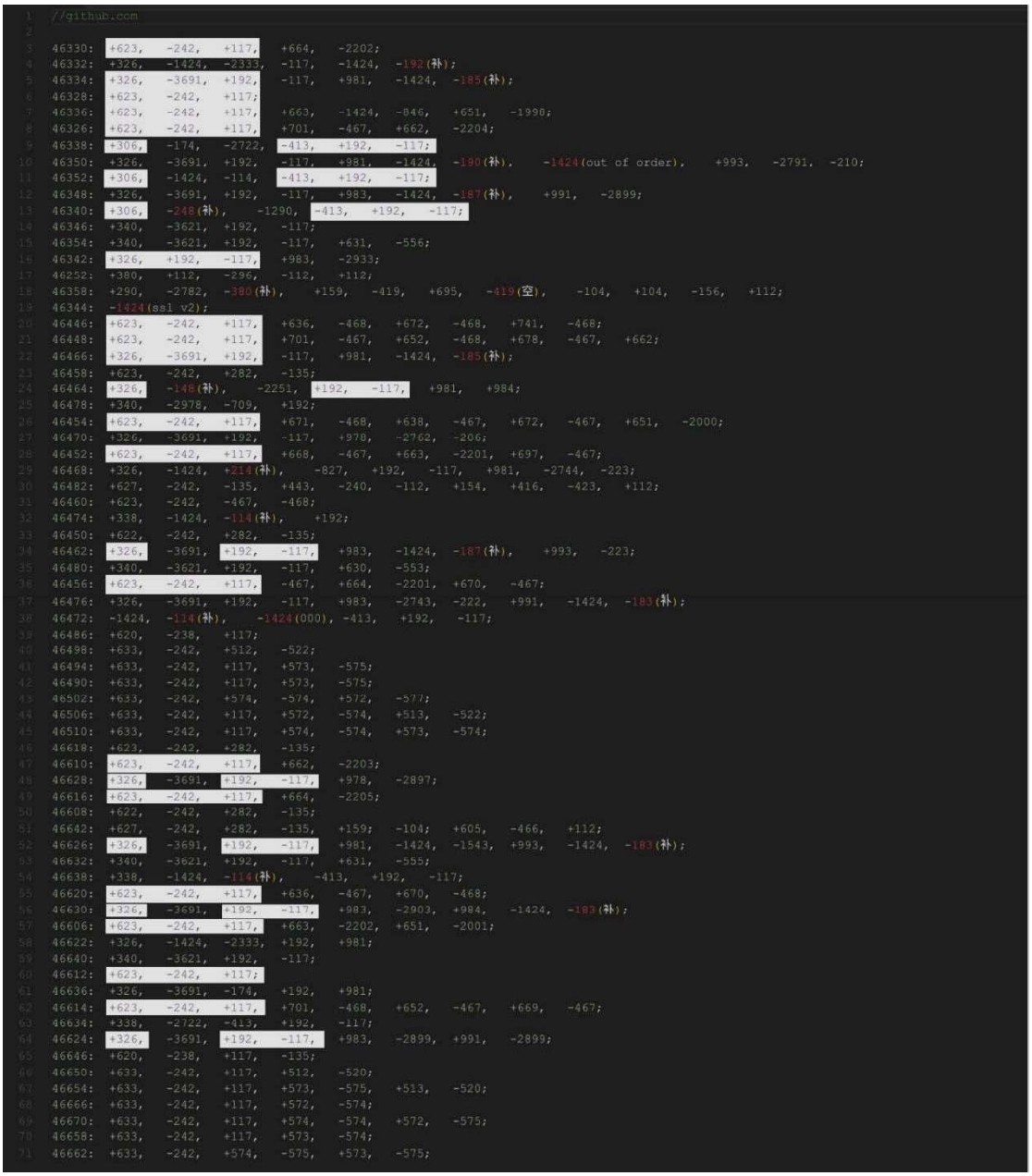


Fig.10. IO statistics, '+' for sending, '-' for receiving, same as Fig.13.

7	0.447092886	192.168.43.29	45.78.8.132	TCP	74 58616 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
15	0.841330472	45.78.8.132	192.168.43.29	TCP	74 443 → 58616 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=137
16	0.841357635	192.168.43.29	45.78.8.132	TCP	66 58616 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=268106
22	0.841889986	192.168.43.29	45.78.8.132	SSL	324 Continuation Data
42	1.262088211	45.78.8.132	192.168.43.29	TCP	66 443 → 58616 [ACK] Seq=1 Ack=259 Win=15616 Len=0 TSval=71967
53	1.262618378	45.78.8.132	192.168.43.29	SSL	2782 Continuation Data
54	1.262625253	192.168.43.29	45.78.8.132	TCP	66 58616 → 443 [ACK] Seq=259 Ack=2717 Win=34688 Len=0 TSval=26
58	1.268791703	45.78.8.132	192.168.43.29	SSL	1637 Continuation Data
59	1.268801882	192.168.43.29	45.78.8.132	TCP	66 58616 → 443 [ACK] Seq=259 Ack=4288 Win=37888 Len=0 TSval=26
64	1.269965075	192.168.43.29	45.78.8.132	SSL	324 Continuation Data
77	1.570136039	45.78.8.132	192.168.43.29	SSL	456 Continuation Data
82	1.608050614	192.168.43.29	45.78.8.132	TCP	66 58616 → 443 [ACK] Seq=517 Ack=4678 Win=40576 Len=0 TSval=26

Fig.11. The packages of SS.

6	0.606057128	192.168.43.29	45.78.8.132	TCP	74 58796 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SA
25	0.940379843	45.78.8.132	192.168.43.29	TCP	74 443 → 58796 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0
26	0.940405458	192.168.43.29	45.78.8.132	TCP	66 58796 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval
42	0.940581407	192.168.43.29	45.78.8.132	SSL	314 Continuation Data
69	1.250495182	45.78.8.132	192.168.43.29	SSL	2782 Continuation Data
70	1.250510233	192.168.43.29	45.78.8.132	TCP	66 58796 → 443 [ACK] Seq=249 Ack=2717 Win=34688 Len=0
71	1.250517947	45.78.8.132	192.168.43.29	SSL	384 [TCP Previous segment not captured] , Continuation
72	1.250523968	192.168.43.29	45.78.8.132	TCP	78 [TCP Window Update] 58796 → 443 [ACK] Seq=249 Ack=2
75	1.250539748	45.78.8.132	192.168.43.29	TCP	1424 [TCP Out-Of-Order] 443 → 58796 [ACK] Seq=2717 Ack=2
76	1.250546057	192.168.43.29	45.78.8.132	TCP	66 58796 → 443 [ACK] Seq=249 Ack=4393 Win=40064 Len=0
77	1.251786687	192.168.43.29	45.78.8.132	SSL	324 Continuation Data
114	1.758675188	45.78.8.132	192.168.43.29	SSL	456 Continuation Data
132	1.796020871	192.168.43.29	45.78.8.132	TCP	66 58796 → 443 [ACK] Seq=507 Ack=4783 Win=42880 Len=0

Fig.12. The packages of SS.

Fig.13. IO statistics.

From the above analysis, we can see that SS does not show particularly obvious detection characteristics compared to other overthe-top software or protocols, but the delay-free characteristics and sequence characteristics of the send and receive packets are enough to expose themselves. In particular, we can use the random forest algorithm for machine learning to identify SS.

(2) ShadowsocksR

ShadowsocksR (SSR) is an improved version of Shadowsocks. ShadowsocksR is an improved version of Shadowsocks. Based on the original SS protocol, it adds tamper-resistant protocol plug-ins and obfuscated plug-ins to mask message length characteristics. The tamperresistant protocols are shown in Fig. 14 and the obfuscated plug-ins
are shown in Fig. 15.

```
"protocols": [
    "origin",
    "verify_deflate",
    "verify_sha1",
    "auth_sha1_v2",
    "auth_sha1_v4",
    "auth_aes128_md5",
    "auth_aes128_sha1"
],
```

Fig.14. Code of protocol plug-ins.

```
"obfses": [
    "plain",
    "http_simple",
    "http_post",
    "ramdom_head",
    "tls1.2_ticket_auth"
]
```

Fig.15. Code of obfuscated plug-ins.

1. Protocol plug-in

(1) origin (Original protocol): There is no data integrity check, it cannot be confirmed whether the data has been tampered with, it is vulnerable to attack, and it has not been confused about the length of the data packet. It can be detected through packet length statistical analysis[15][16][17].

(2) auth_sha1 and auth_sha1_v2 protocols: These two protocols are almost the same. Both protocols have bidirectional integrity checks. The length of data packets is obfuscated, and they have the ability to resist replay attacks. However, the disadvantage is that they cannot completely resist CCA (Chosen). Ciphertext attack) can be detected using the packet length statistical analysis method[18].

(3) auth_sha1_v4 protocol: Compared to auth_sha1, despite increasing the length check to avoid server-side behavior attacks, because the check uses CRC32, CRC32 can pass the correct known data packet and modify the packet length at the same time. Fields and CRC32 to fake the correct value.

(4)auth_aes128_md5 and auth_aes128_sha1 protocols: There is no effective attack method at this time.

2. Confusion plugin

(1)"plain": indicates that the data packet is directly transmitted using protocol-encrypted results without confusion.

(2)"http_simple": It is not implemented in full compliance with the http1.1 standard. It just made a header GET request and a simple response, and it is still the original protocol stream. This provides a strong feature that detects and filters the standard HTTP request.

(3)"http_post": Basically similar to "http_simple". The difference is that the data is sent using the POST method and complies with the HTTP specification. But only POST requests, this behavior is an easily statistical analysis of abnormalities. When only POST requests are detected, they can be filtered. [19]

(4)"ramdom_head": Sends an almost random packet of data before starting communication, followed by the original protocol stream. The goal is for the first packet to have no valid information at all and to invalidate the statistical learning mechanism. However, the last 4 bytes are CRC32, which will become a feature. However, monitoring using this feature will consume a lot of computing resources.

(5)"tls1.2_ticket_auth": Simulates the handshake connection of TLS 1.2 when the client has a session ticket. It is currently implemented as a complete simulation. Perfectly disguised as TLS1.2 by packet capture software testing. There are no obvious features.

(6)In addition, the obfuscator plug-in also sets optional obfuscation parameters. This obfuscation parameter can be disguised as an access to any host according to the setting. By default, the client will directly use the node's domain name as a parameter

You can deploy a DNS authentication system (which may be deliberately targeted at the SSR): If the list of IP addresses in the HTTP/tls request that you send out is inconsistent with the actual IP address that is connected, you can use this IP after multiple requests. Join the temporary blacklist.[19][20]

Based on the above two improvements, it is difficult to capture traffic characteristics with a small number of packets. However, when the sample size is large enough, we can use the statistical analysis method to obtain the SSR flow characteristics. Compared with the normal situation (take Tsinghua Open Source Mirror as an example, Fig.16 up), SSR presents a single protocol type (Fig.16 up), or purely transport layer TCP protocol, or all application layer HTTPS protocol; normal

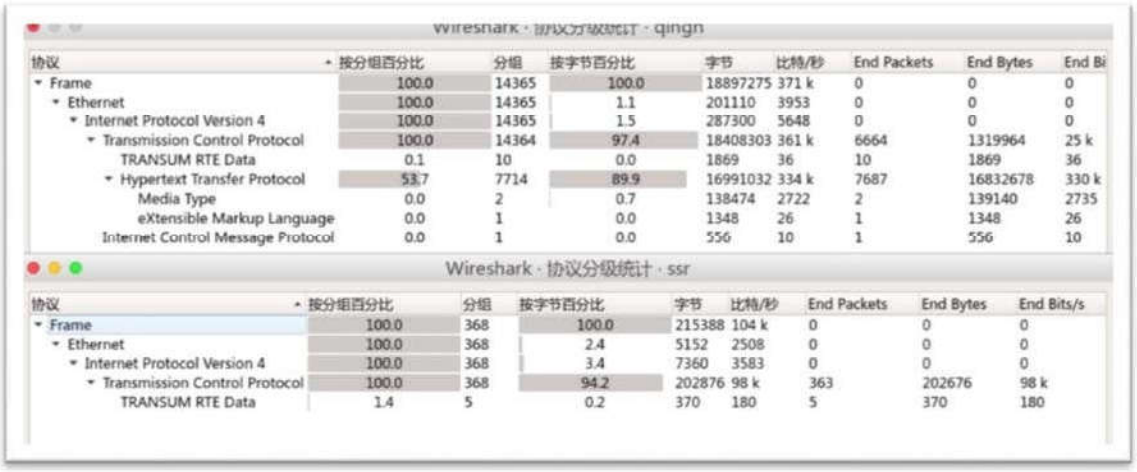
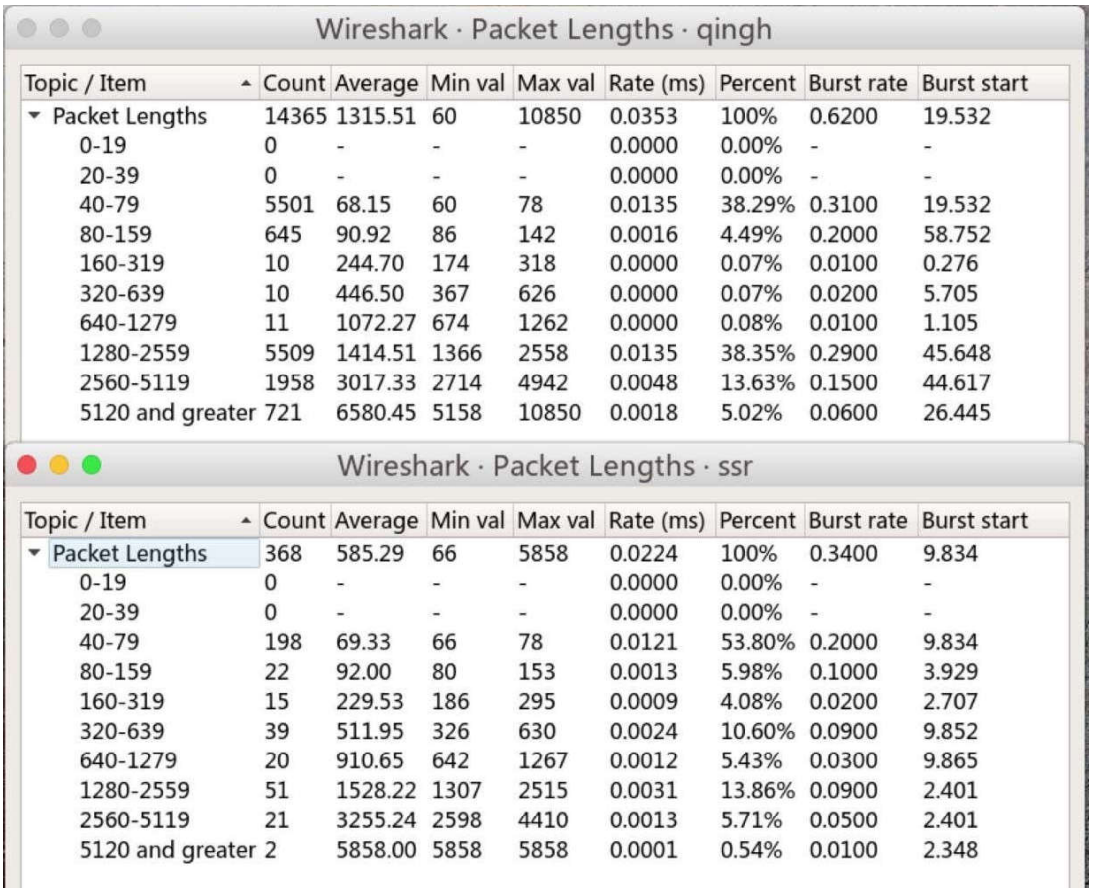


Fig.16. Packages of Tsinghua Open Source Mirror and SSR.

application layer HTTP, ICMP, and other protocols exist.

In addition to the characteristics of the packet protocol type, the packet length characteristics also have characteristics. The packet length feature here is the traffic characteristics of the SSR server because the SSR server reassembles

the packets sent by the real target server. Packets that are shorter than 80 packets are protocol control packets. Compared with normal conditions, the proportion of protocol control packets of SSR software is very high, even more than 50% (this data may be affected by GFW, but in the network environment in mainland China we cannot troubleshoot this factor). Both SSR software and normal data transmission packet lengths are mostly concentrated in 1280-2559. At the same time, because of the obfuscation function of SSR software, the dispersion of its packet size is relatively high, and the length of Tsinghua's update source is obviously concentrated at 1414 bytes.



Wireshark · Packet Lengths · qingh

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	14365	1315.51	60	10850	0.0353	100%	0.6200	19.532
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	5501	68.15	60	78	0.0135	38.29%	0.3100	19.532
80-159	645	90.92	86	142	0.0016	4.49%	0.2000	58.752
160-319	10	244.70	174	318	0.0000	0.07%	0.0100	0.276
320-639	10	446.50	367	626	0.0000	0.07%	0.0200	5.705
640-1279	11	1072.27	674	1262	0.0000	0.08%	0.0100	1.105
1280-2559	5509	1414.51	1366	2558	0.0135	38.35%	0.2900	45.648
2560-5119	1958	3017.33	2714	4942	0.0048	13.63%	0.1500	44.617
5120 and greater	721	6580.45	5158	10850	0.0018	5.02%	0.0600	26.445

Wireshark · Packet Lengths · ssr

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	368	585.29	66	5858	0.0224	100%	0.3400	9.834
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	198	69.33	66	78	0.0121	53.80%	0.2000	9.834
80-159	22	92.00	80	153	0.0013	5.98%	0.1000	3.929
160-319	15	229.53	186	295	0.0009	4.08%	0.0200	2.707
320-639	39	511.95	326	630	0.0024	10.60%	0.0900	9.852
640-1279	20	910.65	642	1267	0.0012	5.43%	0.0300	9.865
1280-2559	51	1528.22	1307	2515	0.0031	13.86%	0.0900	2.401
2560-5119	21	3255.24	2598	4410	0.0013	5.71%	0.0500	2.401
5120 and greater	2	5858.00	5858	5858	0.0001	0.54%	0.0100	2.348

Fig.17. Packet length statistics

IV. Results

After research and analysis, we get the following characteristics of SS traffic:

1. There is no authentication and key exchange process because the keys are shared over the secure channel.
2. The Nagle algorithm is disabled and the acknowledgment message is returned separately instead of the piggybacked acknowledgment method.
3. It can’ t hide the message length characteristics of the real visited websites. At the beginning of the establishment of a new TCP connection, regular lengths of messages are sent and received.

The SSR has the same first two characteristics as the SS, but the third feature is hidden. In addition, it has the following characteristics:

1. Non-standard HTTP requests (except tls1.2_ticket_auth). When taking ambiguity, SS will all use GET format or POST format.
2. There is no high-level communication protocol. In the SSR communication, the TCP protocol is only used at the application layer, and normal communication will adopt an open source protocol or a proprietary protocol at the application layer to ensure reliable delivery, such as HTTP, ICMP and other protocols.
3. The ratio of protocol control packets is high. It is found that over 50% of the protocol control packets whose packet length is less than 80 are in the SSR communication, and this proportion of normal communication is 30~40%.
4. The packet has a large degree of dispersion. Due to the application of the obfuscation protocol, packets forwarded by the agent will be reassembled, and the packet length will be more discrete than normal communication.

V. Discussion

Since the GFW has been updating, we cannot determine whether the GFW has the ability to detect the traffic characteristics of the above two software and take certain measures when we do experiments because many information about the GFW is confidential. Experimental data are not strictly original traffic characteristics of the software in the sense, because they cannot eliminate the influence of GFW in a network environment in mainland China and we cannot go to foreign countries to catch such packets. While we were experimenting, a new socks5 proxy, the brook, appeared, but we lacked an offshore server to do the experiment so the article lacks analysis of the brook. The experiment for brook has to wait until conditions are met.

Acknowledge

This work is supported by college of Electronics and information engineering, that supplies us experiment devices and network. We also gratefully acknowledge the support of Liu Lin, Zeng Yuetian, Jiang Peidong, conducting the early experiments.

Reference

1. Bu, R. (2015). The Great Firewall of China.
2. Normile, D. (2017). Science suffers as China plugs holes in Great Firewall.
3. Gervasiychuk, I. N. (2017). U.S. Patent Application No. 15/000,019.
4. Luciano Floridi, A Proxy culture, Philosophy & Technology, 2015, Vol.28 (4), pp.487-490
5. Deng, Z., Liu, Z., Chen, Z., & Guo, Y. (2017, August). The Random Forest Based Detection of Shadowsocj;s Traffic, Intelligent Human-machine Systems and Cybernetics(IHMSC), 2017 9TH.
6. Cusack, G., Michel, O., & Keller, E. Poster: Machine Learning-Based Fingerprinting of Network Traffic Using Programmable Forwarding Engines.
7. Network Working Group of the IETF, January 2006, RFC 4251, The Secure Shell (SSH) Protocol Architecture
8. Network Working Group of the IETF, January 2006, RFC 4252, The Secure Shell (SSH) Authentication Protocol
9. Network Working Group of the IETF, January 2006, RFC 4254, The Secure Shell (SSH) Connection Protocol
10. Network Working Group of the IETF, January 2006, RFC 4254, The Secure Shell (SSH) Connection Protocol
11. Rivest, R.; Shamir, A.; Adleman, L. (February 1978). *"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (PDF). Communications of the ACM. 21 (2): 120–126. doi:10.1145/359340.359342*
12. Steve Burnett and Stephen Paine. 2001. *The RSA Security's Official Guide to Cryptography*. McGraw-Hill, Inc., New York, NY, USA.
13. Steve Burnett and Stephen Paine. 2001. *The RSA Security's Official Guide to Cryptography*. McGraw-Hill, Inc., New York, NY, USA.
14. Network Working Group of the IETF, December 2005, IETF RFC 4324, Calendar Access Protocol (CAP)
15. Network Working Group of the IETF, MARCH 1996, IETF RFC 1928, SOCKS Protocol Version5
16. Network Working Group of the IETF, MARCH 1996, IETF 1929, Username/Password Authentication for SOCKS V5
17. Network Working Group of the IETF, April 2001, IETF RFC 3089, A SOCKS-based IPv6/IPv4 Gateway Mechanism
18. Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). Hypertext Transfer Protocol – HTTP/1.1. IETF. doi:10.17487/RFC2616. RFC 2616
19. Stephen A. Thomas (2000). SSL and TLS essentials securing the Web. New York: Wiley. ISBN 0-471-38354-6.