Chasing Shadows: A security analysis of the ShadowTLS proxy

Gaukas Wang
University of Colorado Boulder

Anonymous
Anonymous Institution

Jackson Sippe
University of Colorado Boulder

Hai Chi
Anonymous Institution

Eric Wustrow
University of Colorado Boulder

Abstract

ShadowTLS is a new type of circumvention tool where the relay forwards traffic to a legitimate (unblocked) TLS server until the end of the handshake, and then connects the client to a hidden proxy server (e.g. Shadowsocks). In contrast to previous probe-resistant proxies, this design can evade SNI-based blocking, since to the censor it appears as a legitimate TLS connection to an unblocked domain.

In this paper, we describe several attacks against ShadowTLS which would allow a censor to identify if a suspected IP is hosting a ShadowTLS relay or not (and block it accordingly), distinguishing it from the legitimate TLS servers it mimics. Our attacks require only a few TCP connections to the suspected IP, a capability that censors including China have already demonstrated in order to block previous proxies.

We evaluate these vulnerabilities by performing Internet-wide scans to discover potential ShadowTLS relays, and find over 15K of them. We also describe mitigations against this attack that ShadowTLS (and proxies like it) can implement, and work with the ShadowTLS developers to deploy these fixes.

1 Introduction

Internet censors often employ protocol-specific allowlists in an attempt to block circumvention proxies without negatively impacting legitimate uncensored traffic. One example is a TLS allowlist, where a censor will only allow a TLS connection through if the plaintext SNI or server certificate is an allowed domain (such as a popular website). Connections to domains that are not in the allowed list are blocked, making it difficult for users to connect to proxies using TLS.

ShadowTLS [13, 14] is a TLS-based circumvention proxy gaining popularity in China that aims to circumvent this type of TLS allowlist blocking by relaying the TLS handshake to existing unblocked websites, and then switching to a ***proxy server*** afterward for the proxy requests and responses. In a typical setup of ShadowTLS, shown in Figure 1, a ShadowTLS client performs a TLS handshake with a ShadowTLS ***relay***, that forwards the handshake messages to a legitimate HTTPS server (***mask site***) that is on the censors allowed list. After the handshake completes, the relay forwards subsequent data to a proxy server, such as a Shadowsocks [5] or V2Ray [4]. Since the initial handshake will appear to have been performed with a legitimate (and unblocked) TLS website, the idea is that the censor will not be able to distinguish these proxies from the popular websites they mimic, making them harder to block.

In this paper, we identify and evaluate several attacks against ShadowTLS' mimicry technique. In particular, despite its design, we find ways a censor could actively probe a ShadowTLS relay or passively analyze client traffic in order to distinguish ShadowTLS relays from the legitimate TLS servers they mimic. To evaluate our attacks, we perform an Internet-wide scan of TLS servers behaving similar to ShadowTLS, and identify over 15K deployments of the proxy.

We suggest changes to the ShadowTLS design that could mitigate these problems, and work with the developers to get them implemented and deployed. We also discuss other types of attacks that may pose a threat to ShadowTLS and similar TLS-based proxies. To our knowledge, our work is the first to investigate the emerging technique of censorship circumvention used by ShadowTLS.

2 Background

2.1 Motivation

Prior to ShadowTLS, fully-encrypted proxies like Shadowsocks were popular in censoring countries like China [15]. Fully-encrypted proxies work by encrypting every byte, including headers, to avoid matching a specific protocol fingerprint. However, these protocols can still stand out and be blocked by censors due to specific patterns/behaviors. Recently, China has performed active probing attacks to discover and block Shadowsocks servers [1]. Although countermeasures have been researched and deployed [11], there are more

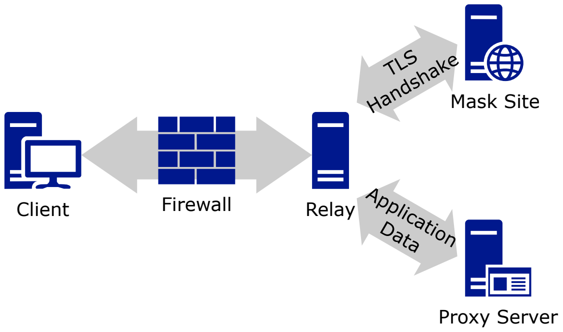

Figure 1: In a typical ShadowTLS setup, the ***client*** within a firewall will perform the TLS handshake with a ***relay*** across the firewall. The ***relay*** will forward these handshake packets to a *Mask Site* which is a legitimate (unblocked) HTTPS server. Once the handshake is done, all following packets are forwarded to a ***proxy server***.

recent reports showing that Shadowsocks and other fully-encrypted proxies being detected and blocked passively [2].

TLS-based proxies avoid many of the shortcomings of fully-encrypted ones. Since TLS traffic is ubiquitous online [8], censors are unlikely to block the protocol outright.

However, TLS-based circumvention tools face several challenges. One major challenge on the server side is that the proxy needs to appear to host a realistic TLS service or website without any identifiable features unique to the proxy. Otherwise, censors could actively probe suspected proxies and block them based on the presence of such unique features, such as discrepancies in content or TLS protocol fingerprints.

Proxies must also commit to a real domain, since the domain used is visible to the censor in the Client Hello Server Name Indication (SNI) extension and the censor can actively probe the server to obtain its certificate. The domain that the proxy uses must be popular, or censors could easily block proxies by maintaining a domain allowlist, and blocking TLS connections to domains not on that list. This technique has already been observed in Quanzhou, China [3].

2.2 ShadowTLS

ShadowTLS attempts to solve these server mimicry issues by "putting on a play" in front of the censor. ShadowTLS operates using four components, as shown in Figure 1:

- A ***client*** starts a TCP connection with the ***relay***, and performs a TLS handshake with the relay.
- The ***relay*** forwards the client's TLS handshake messages to a relay-chosen ***mask site***, which is a popular website not blocked by the censor. Effectively, the ***client*** is performing a TLS handshake with the ***mask site***, with the ***relay*** acting as an intermediary. The ***relay*** does not learn the negotiated secret.

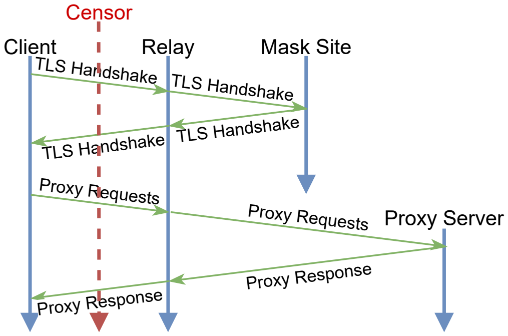

Figure 2: Data flow diagram of ShadowTLS

- After the TLS handshake between the ***client*** and ***mask site*** is complete, the ***client*** begins to send data intended for the ***proxy server*** (e.g. Shadowsocks traffic) through the same TCP connection.
- Following the handshake, the ***relay*** forwards data to the ***proxy server***.

In this way, as shown in Figure 2, the client performs a TLS handshake with the mask site (via the relay's IP), which circumvents the censor's TLS certificate or domain allowlist. Once the handshake is complete, the client uses the connection to communicate with the proxy server (e.g. Shadowsocks). A similar idea levering mask sites was introduced (but not deployed) by Conjure [10].

For ShadowTLS to work, the relay needs to know if the TLS handshake has completed in order to relay subsequent (non-TLS) data to the proxy server. In ShadowTLS, the relay uses the `ChangeCipherSpec` message followed by an encrypted `Handshake` message (the `Finished` message) as a signal that the TLS handshake is finished. However, in TLS 1.3, these messages are encrypted and sent as TLS Application Data (`0x17`) records, making it difficult for the relay to know when the handshake has completed. For this reason, ShadowTLS only supports TLS 1.2 and earlier, where the handshake messages are not completely encrypted.

3 Security Analysis of ShadowTLS

3.1 Threat Model

We assume a censor with similar demonstrated capabilities to that of the Great Firewall of China. In particular, the censor is able to passively observe traffic between the client and relay, and can block, inject, or spoof fake requests or responses. The censor can also actively probe suspected proxies and observe responses. We assume the censor is unwilling to block all TLS traffic, but may maintain a list of allowed domains that can appear in a TLS certificate or Client Hello SNI field, and block any other connections, as seen in Quanzhou [3]. We assume the client and relay have a shared secret that the censor is not privy to, such as the secret used in Shadowsocks.

3.2 Attacks

In this section, we present attacks we have identified based on program analysis and experiments with our own ShadowTLS v0.1.4 instances. These attacks exploit behavior differences between a ShadowTLS ***relay*** and other well-known TLS server implementations, such as those deployed at the mask sites ShadowTLS uses.

TLS fingerprinting Prior work has shown that the TLS fingerprint of the Client Hello message can be used to distinguish TLS implementations [12], and this attack has been used by censors previously to block circumvention tools [9].

We measured the TLS fingerprint of the ShadowTLS client against the TLSFingerprint.io database, which collects TLS fingerprints from a university network tap. We found that the fingerprint produced by ShadowTLS (`ebaa863800590426`) was not observed in the tap dataset collected by TLSFingerprint.io, meaning the client fingerprint is likely unique to the ShadowTLS tool and could be used by censors to block it. To address this issue, we recommend that ShadowTLS use a library such as uTLS [16] to mimic more popular TLS fingerprints and avoid this attack.

Alternative protocols TLS servers often return errors if they receive a malformed request. For instance, sending a non-TLS record to a TLS server could result in a TLS Alert response, indicating the request was not understood. Alternatively, some TLS server implementations will respond to other protocols they can parse, such as HTTP, to help out users that have mistakenly connected using the wrong protocol.

While not being defined in any standard/specification, we find many TLS implementations will respond when they receive a plaintext HTTP request instead of a TLS Client Hello as the first message in a connection. We performed an Internet wide scan of servers running on TCP port 443, and found that over 75% of them responded with either a non-TLS response (e.g. an HTTP error response), or reset the TCP connection. Only 17% of them behaved like ShadowTLS, which closed the connection with a `FIN-ACK`. This means that over 80% of servers running on TLS port 443 can be trivially distinguished from ShadowTLS by seeing how they respond to a plaintext HTTP request.

TLS Application Data records Normal TLS connections encapsulate all data in TLS records, which contain a short header specifying record type, TLS version, and length of the data, followed by the data itself. Sending data without this header is undefined behavior, and could trigger a response that distinguishes normal TLS servers from ShadowTLS relays.

By analyzing the source code of ShadowTLS and our own network traces while using it, we observe that after the handshake, ShadowTLS does not encapsulate data in TLS records. This presents two potential attacks: first, a censor could passively observe that data to a ShadowTLS server is not encapsulated in TLS records after the TLS handshake. Second, the censor could actively probe the ShadowTLS server, complete the TLS handshake to the mask site, and then check to see how the server responds to being sent random data.

A normal TLS server should error or close the connection, since it has received an invalid TLS record. However, ShadowTLS will forward the data to the proxy (e.g. Shadowsocks), and the server would ignore the data (since the probing censor does not have the shared secret required to use the proxy). Thus, a real TLS server would close the connection, while ShadowTLS would remain silent by ignoring the invalid TLS record.

Response	Non-record	Ratio	Bad MAC	Ratio
Fatal Alert	26.9 M	87.3%	28.4 M	88.9%
Reset	2.5 M	8.2%	2.4 M	7.5%
Closed	1.2 M	3.8%	821 K	2.6%
Alert	167 K	0.5%	288 K	0.9%
No Response	**44 K**	**0.14%**	**40 K**	**0.12%**
Non-TLS	2 K	0.01%	5 K	0.02%
Total	*30.8 M*		*31.9 M*	

Table 1: Response from TLS Servers after we send Non-TLS Record Data (Non-record) or a well-formed TLS record but with an incorrect MAC tag (Bad MAC) following a successful TLS handshake. ShadowTLS' relay behavior is in **Bold**.

We used ZMap [18] and our own custom TLS tool written in Go to perform an Internet-wide scan and determine what fraction of tcp/443 servers would complete a TLS handshake, but then silently ignore non-record data. To each server, we sent 35 bytes of ASCII, with an invalid record type, TLS version, but an accurate length field to avoid potentially triggering buggy implementations. We found that over 99% of the 30 million TLS hosts we completed handshakes with responded to our invalid record, most frequently with a fatal TLS alert. Only 0.14% of hosts behaved like ShadowTLS by not responding and keeping the connection open.

Corrupted TLS Application Data TLS protects against tampering by using a MAC or authenticated cipher on Application Data records. If a TLS host receives an Application Data record that does not properly decrypt or has an invalid MAC, the TLS RFCs [6, 7, 17] specifies that a fatal TLS alert (`bad_record_mac`) must be sent, and the connection must be closed.

In ShadowTLS, the relay does not have the shared secret negotiated between the client and mask site, meaning the relay cannot validate Application Data records to determine if the MAC is valid or it decrypts properly. Instead, after the handshake ShadowTLS indifferently redirects data received to the proxy server.

Technique	Ratio
Plain HTTP Request	17.0%
Non-TLS Record Data	0.14%
Corrupted TLS Application Data	0.12%
Combined	**0.05%**

Table 2: The ratio of TLS servers on the Internet that respond like a ShadowTLS relay to each active probing technique. When the results of these attacks were combined, only a very small fraction of hosts (0.05%) behaved like ShadowTLS.

Therefore, a censor could actively probe a suspicious ShadowTLS relay, first by completing the handshake, and then sending a TLS Application Data record with correct version and length but random (and therefore invalid) encrypted data in its payload. According to the TLS specifications, a normal TLS server should close the connection with a fatal TLS alert, but ShadowTLS might not send a response, due to the fact that the payload does not come with the correct proxy (e.g. Shadowsocks) secret and many proxy servers such as Shadowsocks silently ignore any invalid payload.

We again performed an Internet-wide scan to see what fraction of hosts behaved like ShadowTLS and silently ignored corrupt TLS Application Data records. We found only 0.12% of hosts behaved like ShadowTLS, making it possible for censors to distinguish it from real TLS servers.

Fixing this issue will require a design change to how ShadowTLS relays operate, which we discuss in Section 4.

Combining active attacks While each of the active probing attacks can distinguish ShadowTLS relays from a large majority of TLS servers, we find that these attacks work even better in concert. Table 2 shows that by combining these three attacks, our Internet-wide scan reveals approximately 15K servers (0.05%) on the Internet that behave similarly to ShadowTLS. Of these, only 6K of them provided TLS certificates for domains of the Alexa Top 1000 domains, with the most frequent being subdomains of *webex.com* (5969 servers) and *zoom.us* (149 servers). While many of these servers may be non-ShadowTLS, and we cannot confirm what fraction of them truly are, this low number suggests these attacks would be feasible and effective in practice.

While we cannot determine our actual false negative rate, we can get a sense of our true positive rate. We set up four public ShadowTLS relays prior to our scans, and confirmed that our scans discovered all four of our relays, and labelled them as ShadowTLS based on their responses.

Less Efficient: Redirecting Connections A strong censor could also try to *prevent* use of ShadowTLS by redirecting connections to their alternative "true" destination resolved from the SNI in the *ClientHello* message. However, our experiment results in Appendix A.1 indicate that this attack is neither efficient nor accurate enough.

4 Defenses

An underlying issue in all of the attacks we discovered is that a censor is able to observe a behavioral discrepancy between the ShadowTLS relay and the mask site it is mimicking. We note that if the ShadowTLS relay forwarded *all* packets to the mask site (and relayed responses), our attacks would not work. Of course, we need some way for packets to still reach the proxy server, otherwise ShadowTLS would only function as a transparent TCP relay.

To defend against these attacks, we suggest a subtle change to the ShadowTLS design. Rather than having the relay switch over to forwarding to the proxy after the TLS handshake, we instead have the ShadowTLS relay to forward only TLS Application Data records that are encrypted and authenticated under a secret known only to the client and relay. The client will complete the TLS handshake as it currently does, and then will change the secret that it uses to encrypt/authenticate the TLS Application Data records that follow to one derived from the server random and client-relay shared secret. When the relay receives an Application Data record, it validates it using the same information. If validation fails, the record is forwarded to the mask site. Otherwise, the relay removes the TLS record and forwards the payload to the proxy.

This prevents our attacks, since the censor will not have the client-relay shared secret. Therefore, any packets that they attempt to send will end up being forwarded to the mask site, and the ShadowTLS relay will not have any application-layer distinguishing features: all of its responses to the censor will come from the true mask site that it is mimicking.

Responsible disclosure We disclosed our attacks to the ShadowTLS developers, and they incorporated our suggested defense. As of *ShadowTLS v0.2.3*, relays are no longer vulnerable to the active probing attacks we identified.

5 Conclusion

We presented several techniques to identify ShadowTLS *relays* on the Internet that would be possible for censors to implement and deploy. We evaluate our attacks with Internet-wide scans, and find only 15K servers behave like a ShadowTLS relay among over 30 million HTTPS servers on the Internet. This is concerning, as it suggests censors could use these attacks to actively probe and block ShadowTLS relays. To address this, we identify a small design change to ShadowTLS that defends against these attacks generally, and we worked with the ShadowTLS developers to implement and deploy this fix, protecting future ShadowTLS users from the active probing attacks we discovered.

References

[1] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How china detects and blocks shadowsocks. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 111–124, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Anonymous, Vinicius Fortuna, David Fifield, Xiaokang Wang, Mygod, moranno, et al. Properly configured shadowsocks servers reportedly blocked in china, November 2021. https://github.com/net4people/bbs/issues/69#issuecomment-962666385.

[3] Chism. 前几天去泉州体验了一把白名单 - V2EX. https://web.archive.org/web/20220507153405/https://v2ex.com/t/851473, May 2022.

[4] V2Fly Community. Project v. https://www.v2fly.org/.

[5] Shadowsocks contributors. Shadowsocks | a fast tunnel proxy that helps you bypass firewalls. https://shadowsocks.org/.

[6] T. Dierks, Independent, E. Rescorla, and RTFM Inc. Rfc 4346 the transport layer security (tls) protocol version 1.1. https://www.rfc-editor.org/rfc/rfc4346, April 2006.

[7] T. Dierks, Independent, E. Rescorla, and RTFM Inc. Rfc 5246 the transport layer security (tls) protocol version 1.2. https://www.rfc-editor.org/rfc/rfc5246, August 2008.

[8] Let's Encrypt. Let's encrypt stats: Percentage of web pages loaded by firefox using https. https://letsencrypt.org/stats/#percent-pageloads, 2018.

[9] David Fifield. Fortiguard firewall blocks meek by TLS signature. https://groups.google.com/forum/#!topic/traffic-obf/fwAN-WWz2Bk, 2016.

[10] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Conjure: Summoning proxies from unused address space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2215–2229, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting probe-resistant proxies. In *NDSS*, 2020.

[12] Sergey Frolov and Eric Wustrow. The use of tls in censorship circumvention. In *2019 NDSS Symposium*. NDSS Symposium, 2019.

[13] ihciah. Shadow tls: 一个可以使用别人的受信证书的 tls 伪装代理. https://github.com/ihciah/shadow-tls/releases/tag/v0.1.4, 2022. v0.1.4.

[14] ihciah. Shadowtls: 更好用的 tls 伪装代理. https://web.archive.org/web/20220828114811/https://www.v2ex.com/t/875975, 2022. Archived on 2022-08-28.

[15] Zhen Lu, Zhenhua Li, Jian Yang, Tianyin Xu, Ennan Zhai, Yao Liu, and Christo Wilson. Accessing google scholar under extreme internet censorship: A legal avenue. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, Middleware '17, page 8–14, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Refraction Networking. utls. https://github.com/refraction-networking/utls.

[17] E. Rescorla and Mozilla. Rfc 8446 the transport layer security (tls) protocol version 1.3. https://www.rfc-editor.org/rfc/rfc8446, August 2018.

[18] Zmap. Zmap/zgrab2: Fast go application scanner. https://github.com/zmap/zgrab2.

A Less Efficient Attacks

A.1 Redirecting connections

A censor could also try to *prevent* use of ShadowTLS altogether, by redirecting connections to their "true" destination. For instance, when a client sends a Client Hello with a server name indication (SNI), the censor could perform a DNS lookup for the domain, and if the IP returned is different from the one the client is connected to, the censor could start a new TCP connection to the correct IP, and relay packets from the client to that IP. If the client were communicating to a ShadowTLS relay before, the censor would effectively redirect them to the mask site's actual IP address, bypassing the ShadowTLS relay.

However, such an attack might break legitimate TLS connections. For instance, if the censor used a different DNS resolver than the client, or the client is connecting to a private network address not reflected in public DNS. To evaluate this, we looked at TLS connections from our university's network, and collected the Client Hello SNI and destination server IP. We collected 27M `<SNI, IP>` pairs over 24 hours , and identified 472K unique pairs. We then queried DNS for each SNI domain to obtain the lookup IP(s), and compared it to the IP in the connection (the original IP). We found 6K SNIs that did not resolve, and 141K SNIs where the original IP was found in the A record. In other words, SNIs of about 325K pairs resolve to different IP addresses.

We perform a TLS connection using Zgrab2, to see if the first resolved IP of these 325K pairs will complete a TLS connection for the desired domain. If it does not, then we label the original `<SNI, IP>` pair as potentially being disrupted by this attack. We find that 4K failed with the original IP and 2K failed with the IP from DNS lookup. The intersection of these two sets is 1K SNIs, we determine that 1K unique SNIs would be affected.

However, those SNIs made up 315K of the 27M connections collected. Therefore, 1.1% of connections we observed on our tap could be "disrupted" if a censor implemented this attack. We find Amazon Web Services, Apple iCloud and Hulu SNIs among the connections we expect to fail. This likely makes this attack too costly for a censor to deploy to all traffic.