Understanding STARTTLS Stripping

Rose C. Howell
University of Michigan
rchowell@umich.edu

Abstract

E-mail is one of the most widely used methods of communication today, both for business and personal messages. Using modern encryption technology, it should be possible to encrypt every e-mail on the Internet. Unfortunately, research has shown that many e-mails are not being encrypted [2]. A process known as 'STARTTLS stripping' removes the banner that would normally signal the servers to begin an encrypted connection. This project seeks to discover more information about STARTTLS stripping, especially the technical causes, and contribute to a clearer picture of how and why these connections are affected.

1 Introduction

This work presents a proof-of-concept design to identify where the encryption is stripped from basic SMTP connections. Prior research found that up to 41,405 servers on the Internet had corrupted STARTTLS messages [2] [3] [1]. Based off these findings, I have designed a reconnaissance program (ReconMap) that allows us to connect to those servers, establish a TCP connection, and then send the STARTTLS request with varying TTLs. When the TTL (Time-To-Live) expires, the server at that hop must send an ICMP packet back to us that contains the contents of our sent message. This allows us to trace a connection, test each hop along the way, and determine exactly which servers are corrupting the STARTTLS message.

Contributions:

- I develop a tool for automatically connecting to mail servers, and testing if they are able to start an encrypted TLS connection.
- I test the tool on various friendly mail servers known to have this issue with STARTTLS stripping.

```
rose@mail:~$ telnet mail.rosehowell.com 25
Trying 107.4.83.232...
Connected to mail.rosehowell.com.
Escape character is '^]'.
220 mail.rosehowell.com ESMTP Postfix
ehlo me
250-mail.rosehowell.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
mail from: rose@rchowell.net
250 2.1.0 Ok
rcpt to: rose@rosehowell.com
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
Subject: Test from rchowell.net to rosehowell.com

This is a test.

.
250 2.0.0 Ok: queued as BFD2C806E7
```

Figure 1: Basic SMTP connection, as seen from a telnet session.

2 Background

In this section, we describe how the SMTP protocol works, how the STARTTLS command works, and what exactly happens at the packet-level.

2.1 The SMTP protocol

A basic SMTP connection is pictured in Figure 1. We can see that a sending mail server connects to a recipient mail server on port 25. The recipient mail server acknowledges the connection with a 220 banner that identifies itself (mail.rosehowell.com in this example). Then, the sending

```
rose@mail:~$ telnet mail.rosehowell.com 25
Trying 107.4.83.232...
Connected to mail.rosehowell.com.
Escape character is '^]'.
220 mail.rosehowell.com ESMTP Postfix
ehlo me
250-mail.rosehowell.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
starttls
220 2.0.0 Ready to start TLS
^]q

telnet> q
Connection closed.
```

Figure 2: Starting a TLS session over SMTP.

mail server sends either 'helo' or 'ehlo', followed by its own identifiers. In this example, our sender's message 'EHLO ME' asks the recipient server for its available email extensions, and identifies itself only as 'me'. The recipient provides its available extensions as a list of 250-* options. Then the sender enters the metadata ('mail from', 'rcpt to', etc.) followed by a newline character, and then sends the body of the message. The messages is closed by the '.' character, on its own line, and the recipient responds with a '250 2.0.0. Ok' message, which indicates that the message has been received successfully.

For this project, we are only concerned with the results of attempting to start an encrypted TLS session. Therefore, the scope of this project is limited to the following three steps:

- Establish the TCP Connection
- Send the EHLO, Receive the Extensions
- Attempt to start TLS

2.2 How STARTTLS works

Transport Layer Security (TLS) was added later, as an extension to the basic SMTP protocol. As seen in Figure 2, the connection is identical to Figure 1, up until the extensions are received. Instead of proceeding with a plain-text email at this point, we send the 'STARTTLS' command instead. (Note: these commands are not case-sensitive on most servers.)

The remote server then responds to us with the affirmative "220 2.0.0 Ready to start TLS" message. This

```
rose@mail:~$ telnet mx1.ischool.berkeley.edu 25
Trying 128.32.78.14...
Connected to mx1.ischool.berkeley.edu.
Escape character is '^]'.
220 ************************************************************************
************************************************************************
***********************************************
ehlo me
250-mx1.ischool.berkeley.edu Hello [41.231.120.150], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE 26214400
250-DSN
250-ETRN
250-XXXXXXXXA
250 XXXB
starttls
500 5.5.1 Command unrecognized: "XXXXXXXX"
STARTTLS
500 5.5.1 Command unrecognized: "XXXXXXXX"
mail from: rose@rchowell.net
250 2.1.0 rose@rchowell.net... Sender ok
^]q

telnet> q
Connection closed.
rose@mail:~$
```

Figure 3: While trying to start TLS, we receive a 500-level STARTTLS error message.

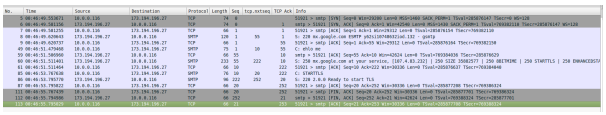

Figure 4: SMTP Connection at the Packet Level.

indicates that the server has an SSL certificate, is properly configured, and is available to begin an encrypted TLS session.

When we receive the affirmative 220-message from the recipient, we know that the "STARTTLS" command has not been blocked.

A note about error messages Per the SMTP protocol, mail servers will generally send error messages that fall into one of three categories, 200-level, 400-level, or 500-level.

- 200-level messages are affirmative. They indicate success, and continuation of the conversation - sort of a green light.
- 400-level messages indicate a delay, deferral, downtime, or other 'try again later' type of errors - sort of a yellow light.
- 500-level messages are hard-fails. A 500-level message indicates a total failure - a red light.

When we try to connect to a server that is having STARTTLS issues, we receive a 500-level error, as seen in Figure 3. This hard-fail is interesting. We can see that the STARTTLS message has been converted into an unrecognizable XXXXXXXX message.

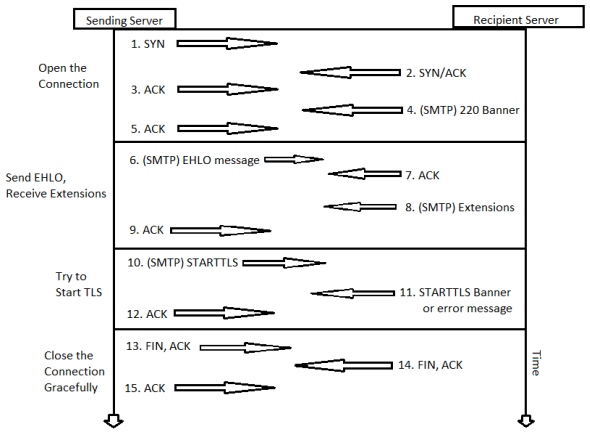

Figure 5: The packets necessary for an SMTP connection.

2.3 At the Packet Level

As seen in Figure 3, the 500-level error informs us that the command 'XXXXXXXX' is unrecognized. We can also see that some of the 250-extensions from the server have been blocked out with X's. This is indicative of a firewall or other middle-box filtering the connections. In order to discover exactly where the issue is occurring, we need to craft the packets necessary to establish the TCP handshake, then send the EHLO message, and then send the STARTTLS command.

For the reconnaissance, I used the Python package Scapy to craft the necessary packets, and found that other packets are also necessary in this SMTP connection.

As seen in Figure 4, there are at least 15 packets sent and received for a complete connection. In this case, a complete connection (1) establishes the connection, (2) sends EHLO and receives the extensions, (3) attempts to start TLS, and (4) closes the connection. Note: Closing the connection is helpful for testing, because some servers will stop responding (and possibly blacklist or graylist us) if we leave too many open connections to them.

3 Crafting Packets

Using the Python plug-in Scapy, we craft the packets necessary to mimic an actual SMTP connection. By crafting each packet, we will then be able to control the TTL of the STARTTLS packet, which will let us see exactly which hop is stripping the TLS. We will then write the results to a file.

Establish the TCP Connection As shown in Figure 5, there are five packets involved in establishing an SMTP connection. Our mail server:

- Sends the initiating SYN
- Receives the SYN/ACK
- Sends an ACK with the correct sequence number
- Receives the recipient's 220-banner
- Sends an ACK that we received the banner

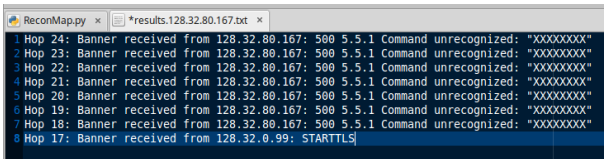

Figure 6: Sample results printed to a file using ReconMap.

Send EHLO, Receive Extensions There are four packets involved in sending the EHLO and receiving the extensions. Our mail server needs to:

- Send EHLO
- Receive the server's ACK to the EHLO
- Receive the 250-extensions from the server
- Send an ACK that we received the extensions

Attempt to Start TLS

- Send 'STARTTLS'
- Receive banner - potentially either '220 2.0.0 Ready to start TLS' or '500 5.5.1 Command unrecognized: XXXXXXXX'.
- Send an ACK that we received the banner

Close the Connection Gracefully

- Send a Fin/Ack
- Receive a Fin/Ack
- Send a final ACK

The Reconnaissance program sends each packet, receives the results, and then sends the next packet based on the previous one. This is especially important for the ACK (acknowledgement) packets, which must accurately reflect the size of the previous packet's payload. Using Scapy, the Reconnaissance program is able to choose a target, craft each packet on the spot, and print the results to a file for further analysis. At the time of writing, the results print to a text file, but this could be altered to print to a CSV, or other type of file for easier data parsing.

ReconMap will print the hop (the TTL number), the IP Address that sent the packet, and the message that the

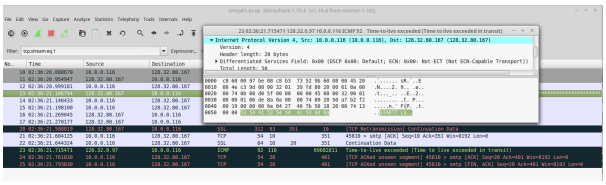

Figure 7: ICMP reply indicating that the TTL has been exceeded.

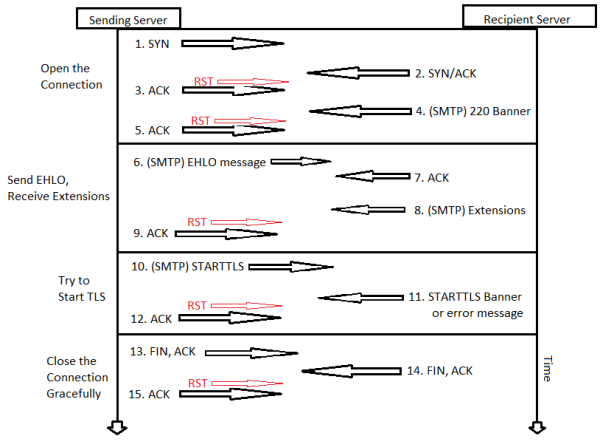

Figure 8: The Kernel tries to Reset the Connection.

packet contained. As shown in Figure 6, we decrement the hop each time, until we receive a result that is not stripping the 'STARTTLS'. We can see that Hop 17 returned the 'STARTTLS' message to us, and came from a different server. At the packet level, this type of bounce message is an ICMP reply, that contains the payload of the original packet that we sent. Shown in Figure 7, we can use Wireshark to inspect the packets. We see the "Time-to-live exceeded" message in packet No. 23, and on closer inspection, we can see the 'STARTTLS' payload contained in the ICMP message's Secure Sockets Layer. Using ReconMap, we can parse this packet and print the payload to a file, as shown in Hop 17 of Figure 6.

4 Testing

4.1 Bypassing the kernel's reset packets

When running these tests from my home Linux server, I needed to adjust the Firewall rules to block outgoing reset (RST) packets. This is necessary because when ReconMap sends the initial SYN packet, the remote server immediately responds with a SYN/ACK, but our kernel did not see the initial request. The Linux kernel doesn't recognize the incoming SYN/ACK as belonging to any connection that it is aware of, so it automatically sends a RST packet to the recipient server, as shown in Figure 8.

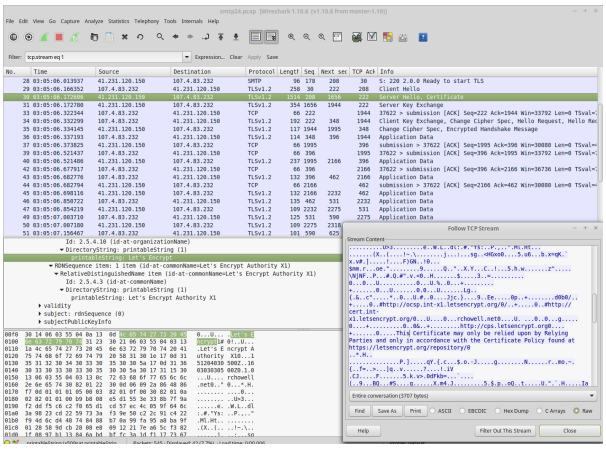

Figure 9: An e-mail ecrypted with a certificate from Let's Encrypt.

Blocking the outbound RST packets at the firewall allows us to continue the connection with ReconMap.

4.2 SSL Certificates

In order to communicate over TLS, both of the servers involved must have SSL certificates. Up until recently, this was a relatively tedious, expensive process. Since the launch of Let's Encrypt, this process has been made much easier, and free [4]. I was able to acquire and configure two mail servers, one in Tunisia and one in Michigan for testing. I was able to install SSL Certificates from Let's Encrypt onto both mail servers, and then send e-mails back and forth while capturing all the packets. Shown in Figure 9, we can see the certificate from Let's Encrypt, and then the rest of the email message is encrypted and transmitted over the wire using TLSv1.2.

5 Future Work

The program presented here can help map STARTTLS failures. Future development on ReconMap seeks to adjust the input of IP Addresses to the program, and to adjust the output to better map the results visually. Based on results from ZMap [3] and Censys [1], we can find filtered mail servers, and use ReconMap to identify the points where STARTTLS stripping occurs. These reconnaissance maps may prove useful for further research.

6 Conclusion

I have inspected the SMTP protocol thoroughly, and explored how it works with the STARTTLS extension to provide encrypted e-mail. I have developed a tool that will craft the packets necessary to probe the routes leading

to mail servers. This will help us discover exactly where the encryption is being blocked.

Acknowledgments

The author thanks J. Alex Halderman, Zakir Durumeric, Ariana Mirian, David Adrian, Jessica Wilson, and Michael Zandi for their guidance, help and feedback.

References

[1] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by internet-wide scanning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 542–553, New York, NY, USA, 2015. ACM.

[2] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J. Alex Halderman. Neither snow nor rain nor mitm...: An empirical analysis of email delivery security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, pages 27–39, New York, NY, USA, 2015. ACM.

[3] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, D.C., 2013. USENIX.

[4] Internet Security Research Group (ISRG). Let's Encrypt, 2015. https://letsencrypt.org.