Commercial Solutions for Classified Layering Costs

Christopher Farrington

Deanna Hlavacek

Eric Rodine

Nicholas Spear

Iowa State University

Abstract

The focus of this project was on analyzing the performance metrics of security protocols in an attempt to discover new, efficient, ways of protecting data by layering protocols. We collected baseline measurements of protocols, and measurements of those protocols while layered together in order to determine the viability of different combinations of security protocols. We found that in some cases, such as executing typical user commands with low throughput, layering protocols together has no significant effect on performance metrics, while in other cases, such as downloading or uploading large files, certain protocols have a significant effect on performance metrics and are therefore a less optimal option.

Introduction

On any given day, at any given time, there is a large amount of data traveling across networks all over the world. Some of that data is as simple and "non-sensitive" as emails between friends coordinating weekend plans, while other data may be highly sensitive, such as a company's trade secrets or data pertaining to national security. Back in the early days of the internet, only known and trusted people could have access to it, so there was no reason to protect the data traveling across the network. As it expanded and became more public, you could not trust everyone who had access to it anymore. This brought about the need for standard security protocols that parties could implement to protect their data while in transit over the internet. One of the most common security protocol used today is SSL/TLS. Most people do not even realize they are using it while they browse some webpages. Other protocols include SSH, SFTP, IPSec, MACsec, WEBRTC, DTLS and others.

While having one layer of security is a big step in the right direction, and deters many attempts at accessing the data, we started to look at layering these different protocols together in an attempt to try to make it even more secure. However, the security of layering the protocols is beyond the scope of this project. Before we even start to look at the security of layering, we need to look at the efficiency of layering. It does not make sense to layer two protocols together for increased security if doing so decreases the efficiency so much that it is unfeasible to use. Therefore, in this project we looked at the performance metrics while layering protocols to try to determine whether it is a good idea to look further into layering certain protocols together.

Methods

Environment

The environment that we conducted our project in is the Internet-Scale Event and Attack Generation Environment (ISEAGE). It is an environment, housed at Iowa State University, to simulate the real internet in a controlled space where attacks can be run without the chance of them being released to the real world.

Using this environment, we set up a Client-Server Architecture with the Client and Server being on two different networks within ISEAGE. We then had a TAP set up, which has all of the traffic traveling through ISEAGE passing through it. On this TAP, we used Wireshark to monitor the network traffic and capture PCAPs that we could later analyze for the metrics we were measuring.

Tests

We started by deciding which protocols we were going to use for the project, and then we had to determine which protocols were actually interactive, meaning that they would be generating the initial traffic to be secured. The protocols we selected were SSH, SFTP, SSL, IPSEC, and HTTPS. We determined that SSH, SFTP, and HTTPS were the interactive protocols or "inner" protocols as we referred to them. We then implemented each protocol on both the server and the client.

For SSH and SFTP we used OpenSSH, and for HTTPS we used curl. For IPSEC we used ipsec-tools, along with racoon for Internet Key Exchange (IKE). IKE securely, and automatically, exchanges secret keys between the client and server for use with IPSEC. For SSL, we used stunnel4. For each inner protocol, we ran a minimum of one baseline test, which consisted of the protocols by themselves. We then ran tests of each protocol with each of the outer protocols layered on top. We finally ran tests of the inner protocols with both outer protocols layered on top making it a triple layered test. We scripted the test and each was run 100 times in order to get averages of the metrics.

SSH

For the SSH protocol, we created a script to be run that would connect to the server, execute typical user commands, and then end the connection to the server. Typical user commands used were mkdir, cd, touch, rm, rmdir, ls, echo, pwd, and exit. We used sshpass to pass the password with the ssh command, which allowed the script to run without asking for the password when trying to connect. The script used are located in Appendix A.

SFTP

For the SFTP protocol, we had three different tests, the first executed typical user commands, the second downloaded a large file, and the third uploaded a large file. The typical commands used in SFTP were ls, mkdir, chdir, put, get, rm, and rmdir. The typical command test uploaded and downloaded a small file which was created by using the touch command and manually entering a small amount of text into the file. The same large file was used in the upload and download test for consistency. That file was created using the following command: "dd if=/dev/urandom of=gigfile bs=64M count=16". We used public key authentication, between the client and server, which allowed the script to run without asking for the password when trying to connect. The scripts used for these test are located in Appendix B.

HTTPS

For the HTTPS protocol, we only performed a baseline test and a test with IPSEC. We did this because HTTPS goes over SSL, so it would not make sense to layer SSL on top of that. In order to make these test easily scriptable, we used curl to get a webpage over HTTPS.

Data Collection and Analysis

In order to collect the network traffic for analysis, we connected a laptop to the physical server hosting the TAP interface. We ran Wireshark on the laptop to monitor the traffic as it was being sent between the client and the server. We then saved the network traffic in a PCAP file for further analysis. It is worth noting that for the tests where we uploaded and downloaded the large file, we did not collect the network traffic, as we did not have the resources, on our capturing laptop, to save over 100Gbs of network traffic per test. For these tests, we instead used the "time" command in Linux in order to obtain an average transmission time. In order to get an average number of packets sent for those tests, we ran the test a second time with only 10 iterations instead of the full 100 and captured the traffic on that. In

order to account for removing files on the client side, which did not require network traffic, we subtracted the runtime that it took the command to remove that file from the total time.

After collecting the data, we performed a few different analyses on it. First, we found the average transmission time per iteration for each of the different tests. For the large file tests, we took the output from the "time" command, subtracted the file removal time, and divided by the number of iterations (100). For the other tests, we examined the PCAP files, individually calculated, and then added the time taken for each iteration, and then divided by 100 to get the average. Second, we found the average number of packets sent per iteration.

The other analysis we performed was where we actually compared the baseline tests' measurements to the layered tests' measurements. We took the average time it took each of the layered tests and compared it to the average time of its respective baseline test to get the percentage increase or decrease in average transmission time. This proved to be one of the more interesting measurements that we took.

Results

Protocol	Avg. time for single iteration	Avg. # Packets		Avg Tme in Sec.
SFTP (large down)	59.8804 sec	1050835		59.8804
SFTP -IPSEC (large down)	1 min. 51.31824 sec	1008111		111.31824
SFTP-SSL (large down)	1 min. 5.297 sec	1068229		65.297
SFTP-SSL-IPSEC (large down)	1 min. 47.253 sec	1009588		107.253
SFTP (large up)	1 min. 10.62735 sec	1040111		70.62735
SFTP-IPSEC (large up)	1 min. 49.92137 sec	853787		109.92137
SFTP-SSL (large up)	1 min. 10.814 sec	1032107		70.814
SFTP-SSL-IPSEC (large up)	1 min. 47.703 sec	876240		107.702
HTTPS	0.01316 sec	27		0.01316
HTTPS-IPSEC	0.013295 sec	27		0.013295
SSH	10.28578 sec	50		10.28578
SSH-IPSEC	10.30473 sec	50		10.30473
SSH-SSL	10.3175 sec	101		10.3175
SSH-SSL-IPSEC (typical)	10.33287 sec	50.19		10.33287
SFTP (typical)	10.36877 sec	103		10.36877
SFTP-IPSEC (typical)	10.36873 sec	103		10.36873
SFTP -SSL (typical)	10.38566568 sec	106.5		10.38566568
SFTP-SSL-IPSEC (typical)	10.382366 sec	106.05		10.382366

Table 1. Analysis Results

Our results show that in most cases, the addition of a second layer does increase the transmission time, although only by a negligible amount. The tests where this was not the case, were where we uploaded or downloaded a large file through SFTP layered with IPSEC. In these cases, the transmission time increased significantly (55.6% and 85.9%, respectively). This appears to hold true

when adding the third layer to this test, however, it is slightly faster with all three layers, than it is with only SFTP and IPSEC (Upload – 3.65% faster; Download – 2.02% faster). The significant increase in time appears to be due to the increased throughput, which in turn leads to increased overhead from IPSEC needing to encrypt the packets.

Protocol	% of base	% of base/SSL	% of base/IPSEC
SFTP (large down)	-----		
SFTP -IPSEC (large down)	185.90%		
SFTP-SSL (large down)	109.05%		
SFTP-SSL-IPSEC (large down)	179.11%	164.25%	96.35%
SFTP (large up)	----------		
SFTP-IPSEC (large up)	155.64%		
SFTP-SSL (large up)	100.26%		
SFTP-SSL-IPSEC (large up)	152.49%	152.09%	97.98%
HTTPS	----------		
HTTPS-IPSEC	101.03%		
SSH	----------		
SSH-IPSEC	100.18%		
SSH-SSL	100.31%		
SSH-SSL-IPSEC (typical)	100.46%	100.15%	100.27%
SFTP (typical)	----------		
SFTP-IPSEC (typical)	100.00%		
SFTP -SSL (typical)	100.16%		
SFTP-SSL-IPSEC (typical)	100.13%	99.97%	100.13%

Table 2. Transmission Time Relations

Future Work

One avenue for future work consists of testing more protocols, such as the others listed in the introduction: DTLS, MACsec, and WebRTC. The more protocols that are tested, the more likely it is that an optimal solution will be found. Another avenue for future work would be to conduct our tests again, both within ISEAGE as well as in a real world environment. While ISEAGE simulates the internet, it is also a controlled environment; this means that our tests did not suffer common transmission factors like: packet loss, jitter, or other variables that could affect the efficiency. With this in mind analyzing other performance metrics, such as overhead, jitter, and delay would provide a better overall picture of the efficiency in layering protocols.

Conclusions

In conclusion, based upon our results so far, layering protocols may be feasible in cases where it does not add a significant amount of transmission time, or the time spent in transmission is not a

priority. However, as was stated in the beginning, efficiency is only one part of the problem. The problem as a whole requires a look at not only the efficiency, but also the security of layering protocols. Whereas one solution may be very efficient, the implementation of the solution may prove to be insecure. With our results in place, the focus of future studies can be on the security of the implementation and not its efficiency.

Appendix A: SSH Script

```
#!/bin/bash
USERNAME=nsa
HOST="64.39.3.4"
SCRIPT="ls -la; mkdir testingdirectory; cd testingdirectory; touch testing.txt; echo 'hell
world!'>testing.txt; pwd; ls; rm testing.txt; cd ..; rmdir testingdirectory; ls; exit"
for i in {1..100};
do
        sshpass -p 'password' ssh -l ${USERNAME} ${HOST} "${SCRIPT}"
done
```

Appendix B: SFTP Scripts

SFTP Typical Script

```
#!/bin/bash
USERNAME=nsa
HOST="64.39.3.4"
for i in {1..100};
do
        sftp -b sftpCommands/sftpTypicalCommands ${USERNAME}@${HOST}
        rm smallfileDw
done
```

SFTP Typical Commands File (sftpTypicalCommand)

```
Ls
mkdir testingdir
chdir testingdir
put smallfileUp
get ../smallfileDw
rm smallfileUp
chdir ..
rmdir testingdir
ls
```

SFTP Upload Script

```
#!/bin/bash
USERNAME=nsa
HOST="64.39.3.4"
for i in {1..100};
do
        sftp -b sftpCommands/sftpUpCommand ${USERNAME}@${HOST}
done
```

SFTP Upload Commands File (sftpUpCommand)

```
put gigfile
rm gigfile
```

SFTP Download Script

```
#!/bin/bash
USERNAME=nsa
HOST="64.39.3.4"
for i in {1..100};
do
        sftp -b sftpCommands/sftpDownCommand nsa@64.39.3.4
        rm old_gigfile
done
```

SFTP Download Commands File (sftpDownCommand)

```
get old_gigfile
```