Dissertation on

"Pseudo-System Protocol for Information Transfer"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE18CS390B – Capstone Project Phase - 2

Submitted by:

Akshay Vasudeva Rao	**PES1201800310**
Amogh R K	**PES1201801039**
Prajwal K Naik	**PES1201801455**
Rohan Iyengar	**PES1201800547**

Under the guidance of

Prof. Pushpa G
Assistant Professor
PES University

June - December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Pseudo-System Protocol for Information Transfer'

is a bonafide work carried out by

Akshay Vasudeva Rao	**PES1201800310**
Amogh R K	**PES1201801039**
Prajwal K Naik	**PES1201801455**
Rohan Iyengar	**PES1201800547**

in partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE18CS390B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June - December 2021. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature Prof. Pushpa G Assistant Professor	Signature Dr. Shylaja S S Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **"Pseudo-System Protocol for Information Transfer"** has been carried out by us under the guidance of Prof. Pushpa G, Assistant Professor and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June - December 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1201800310	**Akshay Vasudeva Rao**	
PES1201801039	**Amogh R K**	
PES1201801455	**Prajwal K Naik**	
PES1201800547	**Rohan Iyengar**	

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Pushpa G, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE18CS390B - Capstone Project Phase – 2.

I am grateful to the project coordinator, Prof. Silviya Nancy J, for organizing, managing, and helping with the entire process.

I take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me, various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

If one draws a parallelism between a computer and a building, one might say the ports behave as doors that facilitate the movement of people, who serve as metaphors for packets of data. Networking protocols would be managers who would decide what kind of people are allowed entry (packet formats), whom to expect when (packet exchange sequences) and when to open the doors and more importantly, for whom (host authentication). Firewalls would then serve as guards while anti-virus systems would be metaphoric bouncers, identifying and ejecting suspectedly troublesome people (malware).

This set-up is far from fool-proof and has an inherent set of flows. The safety of the building (the host) rests on the efficacy of the managers, guards and bouncers in performing their respective duties. Even if they performed their jobs diligently, there's always the risk of malicious attackers finding new and creative ways to enter the establishment either by masquerading their intentions (trojans) or by simply finding alternate doors with possibly less protection.

Now, imagine the building's address was secret. In this scenario, all potential entrants (packets) were received at a holding area disconnected from the main building where they would be thoroughly inspected and any suspicious persons would be ejected. The people who pass the inspection would then be securely transported to the main building (host) in a way that prevents them ever knowing its address. In this way, the building is much more inherently protected as it remains virtually nonexistent to attackers.

The Pseudo-System Protocol for Information Transfer uses pseudo-clients and pseudo-servers to enable hosts to communicate through the internet while being completely isolated (offline) from the internet. It enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Communication protocols have always looked at data security and host security as unrelated goals. Protocols like TLS have been successful in ensuring data security. The goal of PSPIT is to take a unified approach to data and host security by implementing pseudo-systems that help hosts isolate.

PSPIT aims to ensure host and data security while limiting latency to a minimum with the additional constraint of making the implemented pseudo-systems as economical as possible (by limiting buffer size and computational ability to the bare minimum).

The study began with an in-depth review of existing literature on TCP, SSL, and TLS which ultimately form the bedrock upon which PSPIT is built. It then moves on to a description of the practicalities of the implementation of the first proposed prototype - a socket simulation of the protocol.

Most commercially used communication protocols enable only data security. PSPIT goes a step further and implements host security by essentially allowing communication via the internet by never connecting to it.

This prototype enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds.

This protocol secures the data being transferred and masks the host so that a third party cannot intervene by stealing data or gaining access to the hosts. The protocol sequences the transfer of data such that the hosts always remain disconnected from the internet while using the internet as the primary medium for the transfer, thus making full use of the power of the internet but without any of the risks of intrusion that come with it.

Today’s protocols do not contain an inclusion for end system isolation, and PSPIT allows for information to be sent and received without the end systems ever being directly connected to the internet, which significantly reduces the risk of attack.

2. PROBLEM STATEMENT

Almost all of the existing communication protocols in the protocol stack used by the public internet today are more oriented towards data security. Multiple encryption techniques that have come to be used in recent days ensure that intercepting and deciphering messages across the internet is not an easy task.

However, these protocols fail to take into consideration the host security aspect of any communication session. Using the present protocols, any attacker on the internet could learn the identity of the hosts taking part in the communication and inject malicious scripts into the systems. None of the encryption methods in use presently can stop this disaster from happening.

This study takes this requirement of the modern internet seriously. The main goal of this study is to develop a protocol such that the systems taking part in a connection are never exposed to the public internet. This way, the probability of the attacker learning about the presence of the device is reduced by a massive amount.

3. LITERATURE SURVEY

3.1 Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes

Majority of today's distributed computing uses the client-server model. Servers handle multiple requests from clients. Since the modern day networks work on SSL, a considerable amount of server capacity goes on performing computationally expensive RSA Public Key decryptions as part of each SSL handshake. This step can be very detrimental and time consuming to the performance of the server.

This paper explores the existence of a technique for re-configuring RSA-based handshakes. As per the results obtained by the authors, the new method speeds up the processing of RSA operations 11-19 times depending on the size of the key.

There have been fast advances in trying to ensure services degrade gracefully and many of these methods have been implemented in real world servicing as well. Additionally, many studies in networking have yielded low latency results speeding up the communication between the client and the server. However none of these results thwart DoS attacks as their primary goal is to create a server overload and deny the servicing requests from legitimate clients.

In the SSL connections used these days, RSA decryption occurs on the server side when it decrypts the secret to receive the base message. This technique is called Server-Aided RSA (SA-RSA). This paper investigates this technique and tries to change the way the client and server sides handle encryption and decryption. The proposed method simply reassigns the roles of the client and the server. The SSL client becomes the "server" and the SSL server suffering from overload becomes the "weak client". This results in a Client-Aided RSA which has proven to speed up the decryption process.
The paper begins with a description of the SSL handshake.

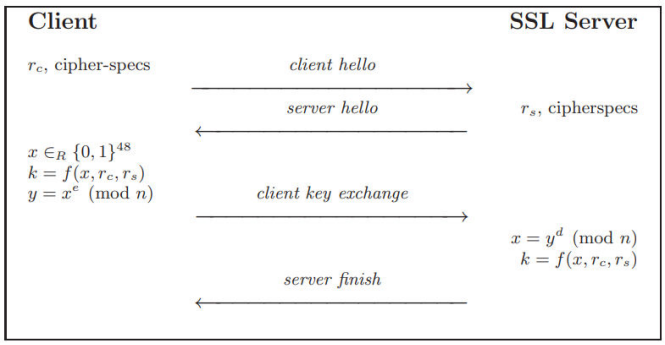


Figure 1. Normal SSL Handshake

The handshake starts with the delivery of a “hello” message from the client to the server. This is an indication of the willingness of the client to start a TLS/SSL session. This message contains the cypher suits and a random, one time number rc.

The server response includes a “hello” message from the server-side and includes a public-key certificate and another random one time number rs. The server additionally mentions it’s choice of cipher suite.

A pre-master secret x is chosen and the shared master secret k is obtained using f(x, rc, rs) where f is the hash function. x is then encrypted with the RSA public key of the server. Additionally, the ciphertext is attached to a “client key exchange” message. Finally, this is sent to the server.

Finally, the server uses its private key to decrypt the pre-master secret and in turn uses it to obtain the master secret as f(x, rc, rs). The handshake is concluded by sending a “server finished” message.

The most important step of the entire handshake process is the encryption of the key under the RSA public key (of the server) by the client. The computationally extensive problem is the RSA decryption.

Although RSA is very well-studied and tested, many of the RSA implementations are computationally imbalanced. When public exponents like 3, 17 and 216 + 1 are used, the RSA decryption is expensive while the corresponding encryption is comparatively cheap, even though the exponents are small. A possible solution to solve this imbalance is to select a small d (private exponent) which in turn speeds up computation. However, this compromise leads to an insecure RSA.

The authors of this paper have tried to flip around the SA-RSA technique and obtained the Client-Aided RSA. The main objective was to transfer a portion of computation from the server to the client. Through this, the clients perform the majority of tasks in decryption, letting the server dedicate more resources to handling the request rather than contributing to cryptography.

The algorithm starts of by the representation of the exponent, private to the server, as d = f1d1 + f2d2 + ... +fndn (mod ϕ(n)). Here, the di's and the fi's are vector elements which are random and of length |n| and c bits, respectively. Since the RSA decryption process involves computing xd (mod n) on the server side, the following steps are performed by it.

The server sends the vector D = (d1, d2, …, dk) to the client.

The vector Z = (z1, z2, …, zk), where zi = xdi (mod n), and sends it to the server.

As a final step, the server computes xd using the vector Z received from the server.

The algorithm assumes that breaking RSA is difficult. Using these assumptions, selecting the parameters of the algorithm such as k, c, and f does not introduce any compromises in security. To guess the values of c and k, any attacker would have to minimally use brute force which would have a complexity of 2c*k. This is the same as the time complexity required to crack RSA. Additionally, this basic description of the algorithm can be improved performance-wise by taking advantage of the Chinese Remainder Theorem.

The next section of the study includes a method to incorporate the CA-RSA method into the existing SSL handshake. The initial "hello" messages by the client and server remain mostly unchanged except the certificate of the server includes the D vector (D = (d1, d2, ...dk)). After the client encrypts using the server's public exponent, it constructs Z by computing the zi = xdi (mod n). This is then included in the client key exchange message. After receiving this, the server performs its CRT computations. The modified protocol is as follows:

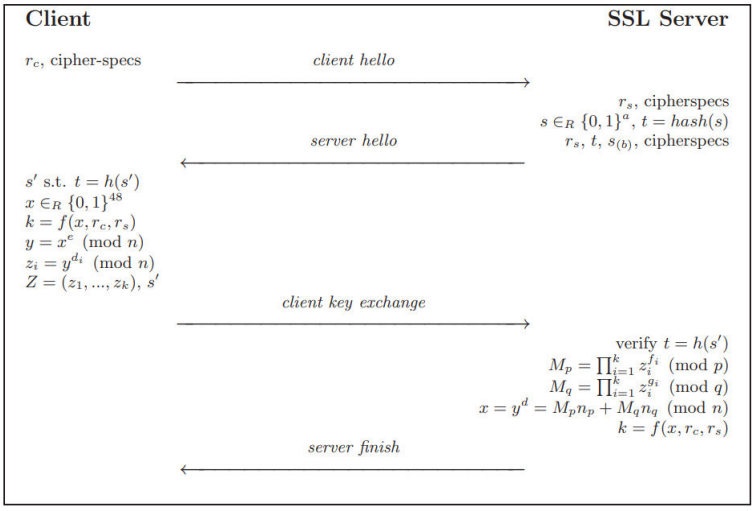


Figure 2. Modified SSL Handshake

The major concerns regarding the integrity of the systems are on choosing the parameters, c and k. The requirements needed to be met are such that the time complexity of exhausting the search space to guess the values of c and k are at least equivalent to breaking the original RSA cryptosystem.

The authors have measured the improvement in the execution times when the protocol was changed from SA-RSA to CA-RSA. The key attribute used for computation was the number of RSA decryptions performed by the test system in a specified time frame. The test machine used had the following specifications:

Processor: Intel Celeron 1.7 GHz

RAM Capacity: 256 MB

OS: Red Hat 9.0 Linux

The results of the experiment were as follows:

Key size	RSA	CA-RSA	$c \times k$	Improvement
1024	7.05	0.62	72	11.33
1536	19.79	1.25	80	15.76
2048	44.22	2.31	88	19.12

Figure 3. Results of New System

These results imply that 11 times as many decryptions can be performed by a server with a 1024 bit RSA key. The smallest value of k is 2, as specified by the guidelines for selecting parameters. Therefore, the values of c and k have been chosen as 2-36, 2-40 and 2-44.

CA-RSA introduces certain additional bandwidth and computational costs at the clients, despite reducing computational burden on the servers to a very large extent. As computed by the authors, this protocol adds an additional computation time of 21.9, 66 and 150.9 seconds while computing Z for 1024, 1536 and 2048 bits on the client-side. However, these computational costs seem to be negligible since the entire protocol is aimed at balancing the degree of computation between the server and the client. Additionally, the client key exchange message also includes certain bandwidth costs incurred while using the protocol since it now has to carry a vector Z. However, with the value of k set to 2, the over is calculated as |n| where n is the length of the RSA modulus. However, this overhead takes less than one ethernet frame to accommodate. However, in case the system being used is very sensitive to bandwidth and computational changes, the protocol can be used only in case of overload of servers. When this technique is incorporated, the overheads occur occasionally.

So far, the authors have focussed on rebalancing the computations between the servers and the client. The protocol assists in enabling the server to increase the quantity of SSL connections that are handled. This alone makes the task of a DoS attacker much harder. However, any attacker could simply overload a CA-RSA server by increasing their computational resources. The authors have included a mechanism to make this task more difficult. An attacker willing to carry out a DoS attack on a server only needs to bombard the server with as many requests as the number of RSA decryptions the server can perform. A possible fix to this would be to direct the client to solve a set of “client puzzles” before sending the encrypted message. This when combined with the lesser decryption time provided by CA-RSA would prove effective in thwarting DoS attacks on SSL servers.

The client puzzle aspect of the proposed solution for DoS attack protection can be implemented by using the method proposed by Juels and Brainard. This method requires a client to solve a computational puzzle before requiring the server to carry out the decryption. This ensures sufficient time between consequent service requests. Following the client's hello message, the server selects a random a-bit value s and passes it via a cryptographic hash function. It then sends the server greeting message to the client with the hash digest t = hash(s) and the b first bits of s (where b a). The client uses these b bits to brute-force to solve the "client puzzle" and discover a value s' that hashes to the desired t. After that, the client adds s' to its client key exchange message. The server will only proceed with the SSL handshake and decrypt the encrypted session key given by the client if s' verifies - that is, if it is the correct length and its hash output is t.

The protocol can be modified to incorporate this by adding the puzzle solution to the frame that contains the Z vector.

When compared to an RSA decryption, the computing cost of a hash computation is almost small, therefore the addition of a puzzle verification step adds just a minor server burden. The amount of labor required by the client to answer the problem is determined by its computer resources and, more significantly, the number of unknown bits in the server's preimage value.

The modified protocol is as follows:

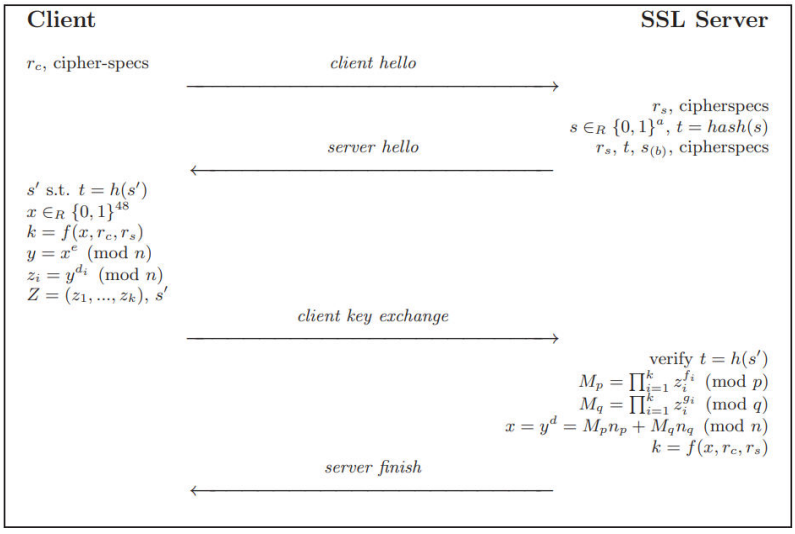


Figure 4. Further Modified Protocol

Even though there is a possibility that a malicious system may solve the puzzle and send fake Z vectors to the server, the possibility of this happening is 11 times less than when using SA-RSA.

In conclusion, the system proposed by the authors helps distribute the server load to the client and thus helps the server authenticate more requests with a very low possibility of overloading.

3.2 CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers

In this paper, they present CRLite(Certificate Revocation List Lite), an efficient and easily-deployable system for proactively pushing all TLS certificate revocations to browsers.

CRLite servers aggregate revocation information for all known, valid TLS certificates on the web, and store them in a space-efficient filter cascade data structure. Browsers periodically download and use this data to check for revocations of observed certificates in real time. CRLite does not require any additional trust beyond the existing PKI(Public Key Infrastructure), and it allows clients to adopt a fail-closed security posture even in the face of network errors or attacks that make revocation information temporarily unavailable.

Comparing CRLite to an idealized browser that performs the correct CRL(Certificate Revocation List)/OCSP(Online Certificate Status Protocol) checking, they show that CRLite reduces latency and eliminates privacy concerns.

Chrome and Firefox only make CRL and check via OCSP and requests for EV(Extra Validation), and not all mobile browsers check for revocation. And this leads to several problems while secure browsing.

CRLite is implemented in two parts: a server-side system that aggregates revocation information for all known, valid TLS certificates on the web and places them in a filter,and a client-side component that downloads filters and uses them to check for revocations of observed certificates. And it deals with majorly six challenges, which are efficiency, timeliness, failure, privacy, auditability, deployability.

Background:

A. The TLS Ecosystem

Certificate Validation, Certification Transparency, Measuring TLS Ecosystem

B. TLS Certificate Revocation

CRL Certificate Revocation List, OCSP Online Certificate Status Protocol

C. Revocation Checking

Revocation Checking in Practice, Fail-open vs. Fail-closed, CRLSet and OneCRL

D. Other Revocation Distribution Schemes

Micali's Certificate Revocation System, multi-certificate revocation, revocation trees, or combinations of these techniques.

E. Proposals to Replace PKI

AKI, PoliCert, ARPKI and PKISN aim to replace the existing PKI with a new hierarchy that avoids centralizing trust, is transparent, and supports seamless revocation.

At a high level, this method aggregates all revocation information for every known certificate, compactly represents them in a filter cascade, and provides a means by which clients can publicly audit them.

The server side of the application works in this order:

Obtaining Raw Certificates ⟶ Validating Certificates ⟶ Obtaining All Revocations ⟶ Filter Cascade Construction ⟶ Delta Updates ⟶ Audit Logs ⟶ Hosting

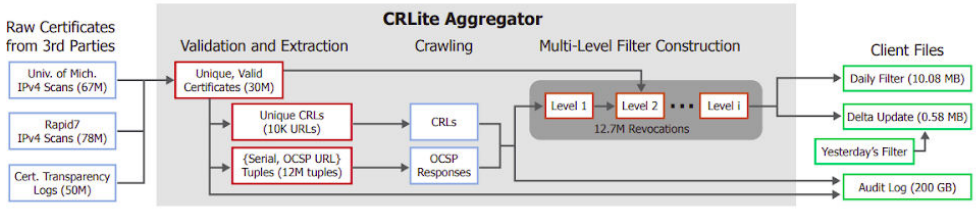


Figure 5. CRLite Aggregator

1.Obtaining Raw Certificates- To create the filter of revoked certificates, CRLite first needs a list of all valid certificates.

2.Validating Certificates- CRLite validates all certificates by looking for non-expired, well-formed leaf and intermediate certificates that cryptographically chain to a trusted root.

3.Obtaining All Revocations- To collect revocations, CRLite extracts CRL and OCSP responders from all valid certificates.

4.Filter Cascade Construction- CRLite uses Bloom Filters for storing and retrieving information as the rate/probability of finding the required certificates is more efficient, has higher probability and is protected.

5.Delta Updates- CRLite produces delta updates that allow clients to incrementally update their copy of the filter(a small percentage of certificates change on a daily basis).

6.Audit Logs-The audit log contains (1) copies of all CRLs and OCSP responses that were used to construct the corresponding filter cascade, and (2) copies of all certificates included in the whitelist

7.Hosting- This is the display of all delta updates and audit logs through web servers hosted on cloud storage.

Security Analysis:

Man-in-the-middle, forcing fail open, DoS, BackDated Certificate and Rogue Aggregator are taken care of by CRLite.

This paper helps one to keep track of the revocation of the TLS certificates, by adding an extension to the browser and they're really helpful for private organisation, who use certificates, as the list generated keeps updating every 24 hours and let's one know if it's safe to use/continue with the current in use certificate and the list is of reasonable size due to the usage of bloom filters and cascading filters.

3.3 Safe Configuration of TLS Connections Beyond Default Settings

This paper, authored by Michael Atigetchi, Nathaniel Soule, Partha Pal and Joseph Loyall, focuses on TLS (Transport Layer Security), its precursor SSL (Secure Sockets Layer), drawbacks of "out-of-the-box TLS configurations" and optimal settings to minimize the risk of TLS-compromise.

Since the deployment of SSL in 1995 and the subsequent release of TLS in 1999, the TLS-SSL standard has been the frontline for communication encryption, by countering attackers who try to sniff live traffic or even man-in-the-middle attacks, that try to compromise the confidentiality of information being transmitted. Despite the wide-scale deployment of TLS 1.3 (usage: 2014-current), it remains vulnerable to a number of threats at the protocol layer and does not provide robust security when used "out-of-the-box", hence requiring tweaks to its configuration in

order to attain the expected security benefits. The paper goes over these tweaks and elucidates the subsequent security enhancements.

To begin with, TLS is a security standard that makes use of certificates that have been issued by standardized bodies, called certificate authorities (CAs). These certificates serve the purpose of identity cards, that allow a page or a server to verify their identity with someone trying to communicate with them. The client, upon receiving the certificate from the server, checks the certificate for authenticity, and upon verification of the same, initiates the second step of the handshake by sending a copy of its own certificate. In this way, both parties have full awareness and full confidence in who they are communicating with.

But the role of the public key does not end there, as the client then encrypts the session using public key cryptography i.e., it uses the server's public key to encrypt the data. This encrypted data, upon reaching the server, can only be decrypted using the server's private key, which is stored confidentially on the server's end. This way, using public-key cryptography (aka symmetric cryptography), the data is encrypted and can only be decrypted by the server.

Beyond the working of TLS, the paper discusses additional means of protecting data encrypted by TLS/SSL, first of which is called the *Crumple Zone.*

Crumple zone is a safety technique used by cars to minimize the damage to core components and passengers inside the automobile in the event of a crash. It consists of reinforcing empty space inside a car with roll cages and strengthened steel bars in order to absorb and distribute the impact of a crash, diminishing its intensity before it is able to reach the passengers and cause any harm. This same idea is what inspired a "Crumple Zone" (CZ) for data encrypted by TLS-SSL.

The crumple zone is intended to reduce and absorb the effects of malicious attacks before it is able to cause any damage to the confidential data.

This necessity of creating a CZ for data arose out of the popularity of Service Oriented Architectures (SOAs). SOAs feature loose coupling, high dynamism and composition-oriented system constructions which make this architecture

highly appealing for application creation, but equally complicated to create viable security for, as they create more points of entry, as compared to self-contained architectures or monolithic systems.

In the context of SOAs, the CZ consists of a layer of intelligent filter proxies which present a higher barrier of entry for attackers, which minimize the damage made to critical data, and increases the chances of detection in the event of such a malicious attack.

These intelligent proxies behave as endpoints for any inbound TLS connections attempting to reach the service, and no data can reach the service before passing through the CZ. The proxies comprising the CZ are equipped to scan, filter, and even partially execute the data to verify its non-maliciousness before it can pass through and reach the service. Since the components present in the CZ also sometimes execute the information, they are expected to fail occasionally, and are hence monitored by a watchdog process which restarts them whenever necessary.

The CZ then finally establishes all outbound TLS connections to then pass on the approved data from the CZ to the protected service.
The three main principles that need to be maintained for proper CZ functionality are as follows:

- Consistency: All data flowing in and out of the CZ must be protected via TLS
- No bypass: No data flows exist such that they can reach the protected service without passing through the CZ
- Security: The TLS configuration used must be capable of defense against malicious attacks and must be cryptographically strong.

Beyond these ideas, the paper discusses the TLS threat model, and the best practice configuration recommendations for TLS to be utilized for information interchange.
The TLS Attack Surface:

Since the rollout of SSL/TLS in 1999, it has been used widely to secure information interchanges on a variety of communication channels, be it browser authentication or service verification. Due to the large number of browsers, browser versions, SSL/TLS protocol versions and implementations, the tuple space of <version, implementation, configuration> is so terribly large, which leads to an expansive attack surface. In short, the authors are trying to convey that the regular, out of the box TLS will never be fully equipped to handle the large number of possible vulnerabilities across all members of this tuple space. The authors also mention that in the past few years, even the most recent versions of TLS have been successfully breached and results have been published on various networking journals.

Furthermore, the authors also mention that due to the slow and piecewise upgrades of the different browsers to later versions, as well as the non-uniform upgrade of TLS versions used on different servers, there exist a huge number of TLS version incompatibilities even to this day. To overcome this, the client and server must negotiate on a common protocol so as to facilitate communication between the older and newer version of TLS. This down-negotiation means that even though many servers have upgraded to later versions of TLS, they may still be forced to use older versions of TLS, with all their vulnerabilities.

The author mentions the example of BEAST (Browser Exploit Against SSL/TLS) in which attackers exploit a vulnerability present in the CBC (Cipher Block Chaining) cryptography method used in TLS 1.0, which can allow an attacker to perform a plaintext extraction of data secured by CBC. This meant that even if servers upgraded to later versions of TLS, if they

were communicating with services using TLS 1.0, they would be exposing themselves to this vulnerability present in the CBC encryption format. The authors state that according to SSL-Pulse, a global monitoring service, 65.2% of sites surveyed by them were susceptible to this attack, even though many had upgraded to later versions of TLS.

Similarly, another attack method known as CRIME (Compression Ratio Info-leak Made Easy) which relies on information leakage from observations of compression behavior to allow attackers to break SSL/TLS encryption. According to SSL-Pulse, 24.1% of sites they surveyed were vulnerable to this attack, and these sites used compression techniques alongside TLS, which is what made them vulnerable to CRIME.
To counter above mentioned *attack surface* of TLS, the following recommendations are mentioned by the author:

- Configuration restriction
- Mutual authentication
- Hostname verification

Configuration Restriction: -
Due to the expansive number of configurable TLS parameters, cipher combinations, and multiple provider options, there are a large number of doors left open for attackers to exploit. The authors

recommend the below mentioned techniques to reduce the attack surface of TLS arising from configuration variation:

- Cipher restriction: Due to the large number of available cipher techniques (such as MD5, SHA, RC4, etc.), many of these do not provide the best possible security but are still used as they take lesser time for encryption/decryption hence making lesser overheads. These leave servers open to BEAST attacks as mentioned earlier, or collision attacks orchestrated by a malicious client. If TLS were to be configured to use only a small set of ciphers that are known to be resilient against current exploits, then this would considerably reduce the attack surface of TLS.
- Protocol Restriction: As mentioned earlier, incompatible versions of TLS can cause down-negotiation which can increase the vulnerability of even the latest versions of TLS. To prevent this, older versions of TLS must be phased out and newer versions must be enforced.
- Options Restriction: The CRIME attack was mainly a result of servers using compression techniques alongside TLS, which exposed them to the attack. Eliminating the option of TLS compression completely will help reduce the attack surface of TLS.

Mutual Authentication:

TLS provides confidentiality and message integrity, but is also used to verify the identity of servers that are being contacted by the use of digital certificates as shown earlier. However, TLS does not provide client certification, and there is no way to find out if a server is communicating with a legitimate client. TLS has mainly relied on company specific password schemes for credentials.

Military, Army, Defense and other high priority channels however, use client authentication methods such as smart cards or Common Access Cards (CACs) which allow the client as well as the server to establish crypto-strong trust. If this is enforced internet-wide, it could significantly mitigate attacks on TLS's confidentiality and integrity.

Hostname Verification

TLS performs certificate verification, and not endpoint verification. This means that if a party X where to get its hands on a compromised but valid/trusted certificate belonging to party Y, and party X presents this certificate to any client trying to communicate with party Y, then the clients would accept the certificate and begin trusted communication even though the endpoint is not party, but

party X. This endpoint verification must be explicitly enabled on TLS, and external code blocks can be run on received communications in order to verify endpoint authenticity, over certificate authenticity. This creates a layer of security over boilerplate TLS, and reduces the risk of attack due to compromised TLS certificates.

Overall, the paper illustrates the role of TLS in modern day communications, the vulnerabilities present in its use, the past attacks made on TLS based communications using cryptographic decryption, and the various tweaks that can be used over "out-of-the-box" TLS to make it more secure against today's exploits.

Our project PSPIT, which will be using TLS to secure its communication between the pseudo clients and servers can be significantly helped by these tweaks mentioned, and we would definitely want to explore these options, post completion of the final multi-system prototype.

3.4 A Study on Large Scale Network Simulators

The paper presents an overview of network simulation. It talks about the applications of network simulation, the various methods of network simulation and the most popular tools available for this purpose.

The authors begin their discussion with a presentation of the advantages of simulating networks. The stated advantages include:

- The ability to gain knowledge about system/protocol improvements
- The ability to understand how a given system/protocol/network works
- The ability to evaluate new protocols

The papers states that there are primarily 3 categories of network simulators:

- Discrete Event Simulation: In this category, the operation of a network is modeled as a sequence of discrete events associated with a point in time.

- Parallel Discrete Event Simulation: This method is employed to effectively simulate very large networks by splitting the nodes of the network into partitions and then mapping the partitions to different processors.

- USSF: This method is based on Time Warp and built on top of the WARPED parallel discrete event simulator which claims to simulate over 100,000 nodes. This is the most complex form of simulation and more details elude the scope of this paper.

The authors go on to state that most simulators follow the Discrete Event Simulation technique. Some examples of popular tool include:

- OPNET (Optimized Network Engineering Tools)
- NS2 (Network Simulator 2)
- NS3 (Network Simulator 3)
- GNS (Graphical Network Simulator)
- OMNet++ (Optical Micro-Networks Plus Plus)
- REAL
- SSFNet
- J-Sim
- NetSim (Network Based Environment for Modeling and Simulation)
- QualNet

Following is a summary of the various simulators presented:

OPNET

- e OPNET is considered as one of the most famous and commercial network simulators because of its wide uses in the industry/
- Uses an OOPs method to implement the graphical representation of networks.

NS2

- NS2 is an open source network simulator. Mostly used in academia.
- NS2 is an object-oriented, discrete event driven network simulator NS2 uses C++ and OTcl programming language.

OMNet++

- OMNet++ is an open source, component-based network simulator with GUI support and widely acknowledged in academia.
- OMNeT++ provides GUI support, and due to its modular architecture, the simulation kernel can be embedded into all kinds of different users' applications.

NS2

- The core is written with C++ and Python.
- The simulator focuses on establishing a sense of realism between simulation entities and real network nodes.
- Allows the integration of more open source tools.

NetSim

- Supports 42 routers and 6 switches.
- Simulates network traffic with virtual packet technology
- Provides two different viewing styles: Telnet mode or Console mode

GNS3

- GNS3 is the graphical extended version of NS.
- It offers two architectures:
    - The GNS3-all-in-one software (GUI)
    - The GNS3 virtual machine (VM)
- The server is also available in three options:
    - Local GNS3 server
    - Local GNS3 VM
    - Remote GNS3 VM
- GNS3 offers both emulated and simulated devices.

3.5 Survey on Covert Storage Channel in Computer Network Protocols

Covert channel in a network is a communication medium that allows the exchange of data between two running processes in such a way that breaks the system's security policies. These mediums are exploited to pass malicious notes, trojans, viruses etc. The ways used to do these can't be easily detected by anti malware systems and firewalls.

These channels are broadly categorized into two major types: storage and timing channels. The former involves encoding secret information into protocol header fields and receiving back the information. Timing channels involve manipulation of message timings, frames and packets.

The survey includes the following studies:

Using the Markov model to propose detection algorithms that discover covert message passing in TCP flows. The issue with this proposal is the inability to detect such stealth communications using tunnelling technology rather than TCP ports such as ICMP.

A hybrid detection system that includes both anomaly detection and stealth channel profiling.

Construction of covert channels based on packet segregation. This method is too hard to be detected by systems such as IDS/IPS. However, these channels can't be eliminated.

Training neural networks for detection of ISN covert channels. The false negative rate varies from 5% to 10% and the false positive rate is below 0.5%.

A traffic padding approach which randomly estimates and limits packet-size. This introduces a dummy packet.

Majority of the proposed methods rely on recognition of abnormal behavior. In reality however, they would fail to detect such traffic with considerable variations. Reducing the capacity of covert channels is seen as one of the most effective solutions in controlling covert channel threats. Like any other penetration loopholes, complete elimination of these channels is impossible. However, the capacity of these channels could be reduced in order to make communication mediums more secure. Future research should pay more attention in developing such methods.

3.6 Network Security and Types of Attacks in Networks

This study provides an overview of all the computer network loopholes and their exploitation in the present scenario.

This paper is relevant to this project because it updates us on the types of attacks that developers of any new protocol need to be aware of to make their protocols immune to such attacks

Development of a secure network needs to handle the following:

Confidentiality: Assurance that a non-authorized system cannot and will not examine the data.

Integrity: Assurance that data received by the system, through the network, is unchanged and unmodified.

Attacks in this study have been classified into passive and active. Passive attacks are those when an intruder in a network intercepts the data. Active attacks are those in which an intruder commands disruption with the networks.

Active attacks include spoofing,modification, wormhole, fabrication, DoS, sinkhole and sybil.

Passive attacks include, traffic analysis, eavesdropping, monitoring.
Other advanced attacks include blackholes, byzantine, rushing, replay, location disclosure.

3.7 Modern Network Security: Issues and Challenges

This paper provides an overview of all the security issues a company faces when transferring data and maintaining secured data, in the present world.

This paper also discussed how to encounter it with respective methods, which are useful for the implementation of our protocol, such as cryptography and firewalls.

Cryptography: this method encrypts and decrypts data by the application of mathematical functions. This is related to the application of SSL and TLS to configure the data security.

Firewalls are of three types which are Application gateway, Packet filtering and Hybrid systems each forming a wall between networks to make them independently secure and inaccessible by anonymous parties.

The application of these methods are achieved by ASIC based appliances, SSL-VPN and Intrusion Detection prevention Systems.

Configuration and update of softwares are also a necessity for protecting data and company networks and mandating employee data risk awareness programs.

This paper mentions the minimum set of parameters that are required for establishment of a secure networking environment using a software development firm as a case study. Security policies should be flexible and not fixed to ensure that the needs of the organization are fulfilled and to tackle future threats while being easy to manage and adopt.

3.8 Design and Implementation Issues of Flexible Network Architecture

This study emphasizes the issues in design and implementation of flexible network architectures with the SOA(Service Oriented Architecture). The aim is to address current network application problems as well the ones that might appear in the future.

Accordingly the changes to be made are of three types and each part of the internet are considered a constructive step, and the three changes identified are

Decoupling of protocols, Flexible Protocol Composition and Service Oriented API.

The protocol family assumes accessibility of at least two routes between the users with end to end communication, considering lag, latency and variance in the network, which keeps up in a changing network environment.

So by using the pseudo hosts the path of data transmission need not be the same every time and masking of the user's identity can prevent the above discussed attacks. The encryption and division of data is also going to make it difficult to trace the message and reassemble it.

3.9 CryptoNET Security Management Protocols

Here many network security protocols have been embodied by the CryptoNET system architecture. The concept of managing protocols are formed on highly established security and privacy technologies and standards.

Individual attributes of the protocols are:

-Effortless incorporation with any applications,

-FIPS 201 (PIV) smart cards are used to handle all security/privacy related issues in the background

These protocols are: remote user authentication protocol, single-sign-on protocol, SAML authorization protocol, and secure sessions protocol.

Single-sign on protocol is utilized for constructing a secure session between two users. The receiver receives a KeyExchange certificate from the sender. Between the two users the KeyExchange certificate helps transport, id and key of the session created between the two.

Upon reception of the certificate request, the client fetches the KeyExchange certificate from a smart card and sends it back to the Secure Application Server.

After receiving the request for the certification, the user gets the KeyExchange certificate from the smart card and is sent to the server.

The protocol makes use of a dedicated container in order to distinguish attackers with fake authentication certificates to differentiate spam and real requests

The key takeaways that can be implemented on PSPIT, are that we can use the various protocols used in cryptoNET in order to implement packet level encryption between pseudo senders and pseudo receivers.

Using dedicated containers for packets which can only be “opened” upon successful single sign on and keyExchange, we create a secure interface between the pseudo machines. This adds a layer of additional security along with packet division and scrambling.

3.10 A Security Architecture for Internet Protocol

IPSEC is an Internet standard protocol from the Internet Engineering Task Force (IETF).

IPSEC protocol defines the semantic and syntax for placing the packet to be transferred inside another packet.IPSEC protocol-specific functionalities are performed on the IP basic packet to protect its integrity and privacy, by using encryption algorithms, then the output is appended to a IPSEC packet header in order to create IPSEC packet; at the end the IPSEC packet is added to IP packet in order to transport it through the internet.

IPSEC protocol works on an unidirectional packet as the packet is made secure in a way that the packet is sent to the specified destination with a key, which can be used to decrypt the packet and nothing other than that can be used.

IPSEC packet has the following details inside in order to securely transfer to destination:

Destination IP address: retriever of packets

Security Parameter index (SPI): the designator of the security association. The SPI should be distinctive to each destination address of the security association in such a way that (destination address, SPI) distinctively identifies a security association. Any IPSEC packet built on the authority of a security association transports the SPI of the security association such that the destination can go through it's packet.

Security protocol: privacy and integrity or both together are provided by security association on the IP packets. Below this, a combination of encryption/decryption algorithms and its criterias are key's size and lifetime.

Encapsulation mode: fragments of the IP packet will be made secure by the security association

Secret keys: key used by the encryption/decryption algorithm.

The primary intention of a key control scheme is to offer two speaking events with a common, recently shared cryptographic key that is forty six CHENG ET AL. In general, a typical key control scheme will attain that in phases: one in which a grasp key is shared among the events,

and the alternative in which the already shared grasp key is used for the derivation, sharing, and refreshment of additional consultation keys.

requirements supported by session key protocol : Security handshake, Secrecy and authenticity, Efficiency, Forward secrecy, Simplicity.

3.11 A Comprehensive Survey On SSL/TLS and their Vulnerabilities

SSL has two major layers on which it works and they are session and connection. One of the major layers is Connection, which connects the client and server on the transport layer. P2P conglomerates allow sessions to be built up which is transient. Each SSL session is conglomerated with one Connection.

SSL/ TLS handshaking protocol is used to establish sessions by changing the parameters.

The header section of SSL protocol consists of 4 parts and they are compressed length, content type, major and minor versions. The handshake protocol consists of three parts which are Type ,Length and Content which vary in the size of 0-3 bytes each.

The Alert part of the protocol which is compressed into 2 bytes like the above parts. First byte indicates the level of issue/alert and the second byte indicates the degree of severity of the issue caused during transmission of the packet across the network.

TLS is just an updated version of SSL, as the basic structure is the same but with minor changes with parameters.

The handshake protocol takes place in three steps, as follows:

Key exchanging and encryption/decryption algorithms are applied.

Responses to key exchange by certificate types and it’s methods.

And the client certificate is verified and made sure that it is not overlapped with the master certificate.

And in the final step padding takes place inside the packet so that there is no concatenation between the bytes present inside the packet.

4. PROJECT REQUIREMENT SPECIFICATION

4.1 Project Perspective

Most commercially used communication protocols enable only data security. PSPIT goes a step further and implements host security by essentially allowing communication via the internet by never connecting to it.

4.1.1. Project Features

1. Data security
2. Host security
3. Low latency delivery

4.1.2. Operating Environment

The prototype will run primarily on the application layer and will use sockets to simulate inter-host communication.

4.1.3 General Constraints

1. Need to follow IEEE standards for internet communications

2. Need to limit pseudo-system memory to minimum

3. Need to simulate real-time latencies

4. Need to test extensively for security

4.1.4 Risks

1. Security Leaks
2. Latency over the limit

4.2 External Interface Requirements

4.2.1 User Interfaces

The prototype will be operated via a terminal, enabling the user to specify configuration details as well as the information to be transmitted.

4.2.2 Hardware Requirements

Standard PC configuration of 2020

4.2.3. Software Requirements

Windows 10, Python 3.6+, “Sockets” module.

4.2.4. Communication Interfaces

Sockets and ports intra to the system.

4.3 Functional Requirements

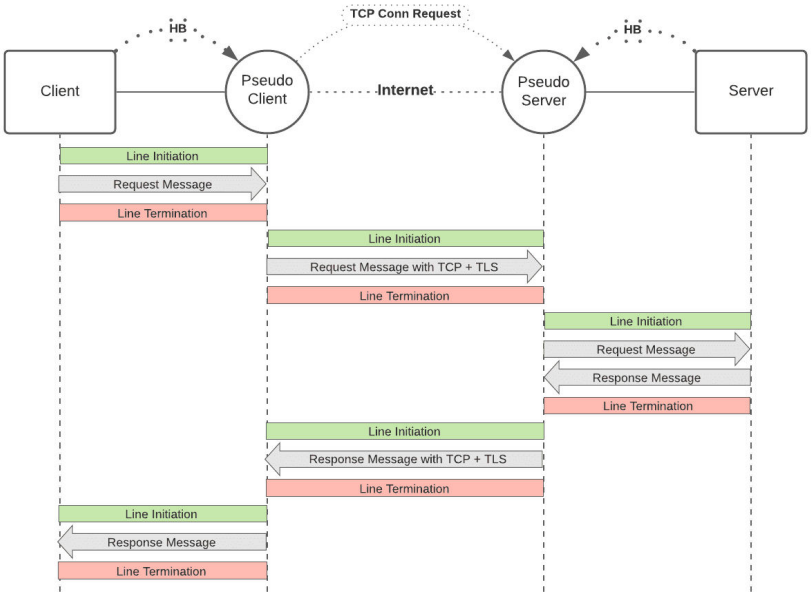


Figure 6. Prototype Skeleton

4.4 Non-Functional Requirements

4.4.1. Performance Requirement

The prototype must ensure reliability, robustness and minimal latency (as comparable with other commercially used protocols)

4.4.2. Security Requirements

The prototype must provide:

1. Message security
2. Host security

5. High Level Design Document

5.1 Current System

At the time of proposal of this protocol, the entire internet operated on the well known, tried and tested OSI protocol stack. This is a 7 layer stack comprising application, presentation, session, transport, network, data link and the physical layers.

The application layer is responsible for making requests to the server (in case of a client-server communication). Application layer protocols like HTTPS are highly data security focussed. By using them, it is ensured that the data being received by a network node is being sent by a certified host. Additionally, this data can only be accessed by the individual system it was meant for by using encryption techniques like SHA, RSA etc.

Transport layer protocols like TCP and UDP provide best effort delivery services to the hosts. In most cases this is adequate. TCP is more appropriate in cases where the data has to be delivered intact and in order while UDP is appropriate for cases where performance is preferred over integrity. The transport layer, however, does not provide any security enforcements.

In the data link layer, network nodes on the local network are connected using ethernet, the LAN technology which has been in use for the longest time. Ethernet makes use of MAC addresses to deliver link layer frames and uses ARP to ensure accurate mapping of IP and MAC addresses.

5.2 Novelty

The above mentioned systems work satisfactorily when it comes to the everyday uses of the internet. The only downside to these existing protocols is that they do not pay attention to the host security aspect of network communications. Data security and host security are the two faces of the same coin. The existing protocols are more oriented towards data security. While travelling across the internet, there is a strong possibility of the data packets being sniffed. Since the addresses of the nodes taking part in a communication session are not encrypted in any way, it becomes extremely easy for an attacker on the internet to identify the presence and location of the host systems and send malicious scripts to the host systems thereby rendering the system insecure.

The system proposed in this study is extremely host security focussed and this is the novelty in the proposed system.

At no point in time during the transmission or receipt of data, are the host systems simultaneously connected to the public internet thereby strengthening each of their integrities.

5.3 Interoperability

The pseudo-system protocol for information transfer is built to operate over the existing transport layer protocols. This system is designed to be an innovation in the application layer and hence deals with data directly. This system does not handle the packet flow/acknowledgements of the transport layer or does not provide assurance of data delivery since these aspects are already looked into by the lower layer protocols. Due to the slight modification of incorporating two additional nodes at either ends of the communication, the interoperability of the system with existing machines is guaranteed.

5.4 Performance

The performance of the system is directly dependent on the underlying OSI protocol stack layers. The proposed system does not add any performance penalties by itself to the communication session between any two systems as the new system is equivalent in performance hits obtained by adding two extra nodes in the communication process. The availability of the system however is marginally deteriorated because of the addition of the uncertainty of availability of the pseudo-systems.

5.5 Security

This system is developed mainly for the purpose of host security. The system achieves the same by ensuring that the host systems are never connected to the public internet at the time of transmission of sensitive data. By doing so, even if an attacker finds out the identity of the pseudo-systems, PSPIT makes it impossible for the attacker to gain direct access to the host systems.

Data security can be enhanced by incorporating existing encryption methodologies like RSA, SHA etc. However, enhancing the data security of the existing transmission methodologies is not the intention of this protocol.

5.6 Reliability

The reliability of the protocol is dependent on the availability and integrity of the pseudo-systems that are present at either end of the communication session.

To ensure that the requests are not continued being sent despite failures of any of the systems, a heartbeat/probe methodology is implemented.

An additional failure detection method is to implement timers in the client side systems to keep track of the delay in the receipt of messages and to interpret node failure.

5.7 Maintainability

Since the protocol has four major systems which are to be managed other than the connectors in between, it depends on if the four are in proper working condition or not.

To handle the system failures, while trying to send or receive data packets, we have come up with a node failure handling technique. The node failure handling technique has two methods first, system check nodes and second dedicated timer in each system.

All pseudo systems are to be looked after as equally as the client and server systems, so they do not falter when in use.

There are two reasons for the QoS of the protocol to be affected. One is when there is a latency in the delivery or receipt of the packets and the other is when the network nodes are offline. The node failure handling mechanisms are made for each of these conditions separately.

5.8 Portability

Portability isn't an issue until the pseudo-server or pseudo-client can be transported with their respective machines. The major reason for development of this protocol is that the user's identity isn't revealed to the outside world, and to make a secure data packet transaction.

After transporting the systems, it is to be made sure that all the connections are to be established as before, so that there is no node failure occurring while transporting data packets.

This implementation is similar to adding another layer to the existing OSI network layer model, and that additional layer is a pseudo-system to mask the user's identity.

5.9 Legacy to Modernization

The protocol builds upon the foundations of TCP + SSL (TLS). So the extent of modernization is limitless, like our implementation of PSPIT protocol.

If the performance measures are favourable, PSPIT could replace TLS in certain use-cases, provided the costs of pseudo-servers and pseudo-clients are justified.

Modernisation of the protocol could also include more robust methods to handle delay of transmission and failure of nodes. Presently, the method proposed to handle such drawbacks are using system check nodes and by using two dedicated timers each on the client and the pseudo-client.

5.10 Reusability

The system is entirely open to reuse and can be used as a base for multiple authentication and latency reduction techniques that might be developed later on. Since the underlying protocols used are TCP and other well known LAN technologies, any developments made on these fast changing fields will be applicable to PSPIT as well.

5.11 Application Compatibility

This protocol is designed to operate in the application layer of the OSI model. Hence, it can be used by any network system that uses the request-response model. However, the protocol is designed for host security sensitive applications like bank transactions and military uses.

5.12 Resource Utilization

The resources used by the protocol are the same as that used by any other existing protocol. An additional requirement is the usage of one extra LAN connection on either side of the communication tunnel. This would require an ethernet connection connecting the client and server to the pseudo-client and the pseudo-server respectively. Additionally, in case of implementation of the node failure handling mechanisms, additional connections would be required by the server check, server and pseudo-server and the client, pseudo-client and the client check. An additional LAN connection would be required between the client and the server check nodes. The bandwidth utilization of these connections would be bare minimum because all these connections would be local.

6. Low Level Design and Implementation

6.1 Proposed Methodology and System Design

6.1.1. Client

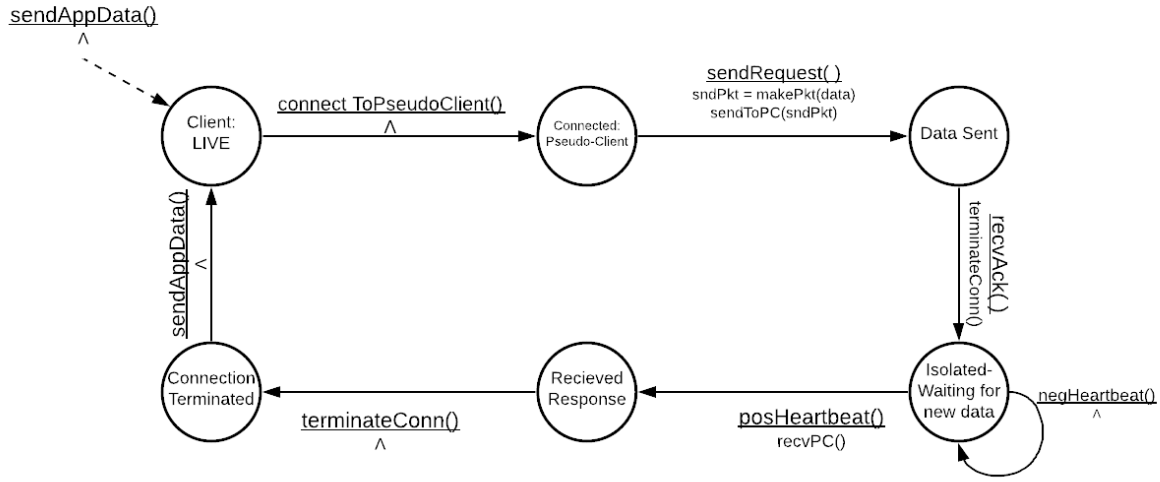


Figure 7. Client

This is the state diagram for the client/host system. There are 6 states including that of the terminal isolated state. The client system triggers the entire protocol when it receives a send command from the upper layer protocol. The first step in this cycle is the establishment of a connection with the pseudo-client. This is done using standard TCP procedures. Once the connection is established(and secured) the client sends the request packet through the data flow link to the pseudo-client. This is again handled by the underlying TCP which takes care of the delayed or unsuccessful transmission. Upon receipt of acknowledgment of successful transmission of the request packet, the client terminates its connection with the pseudo-client. At this point, the client is isolated from the pseudo-client and hence from the internet.

Immediately after isolation of the client, the heartbeat phase of the client takes over. The client sends repetitive beacons in order to check for the availability of response to its request at the pseudo-client. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-client is closed, the client is unable to send the beacons(this connection is later established by the pseudo-client).

Once the heartbeat beacon returns, indicating that the pseudo-client has a new response to send to the client, the connection is re-established with the pseudo-client and the client receives the pending response. After this is done, the client terminates its connection with the pseudo-client and is thus isolated from the network.

6.1.2. Pseudo-Client

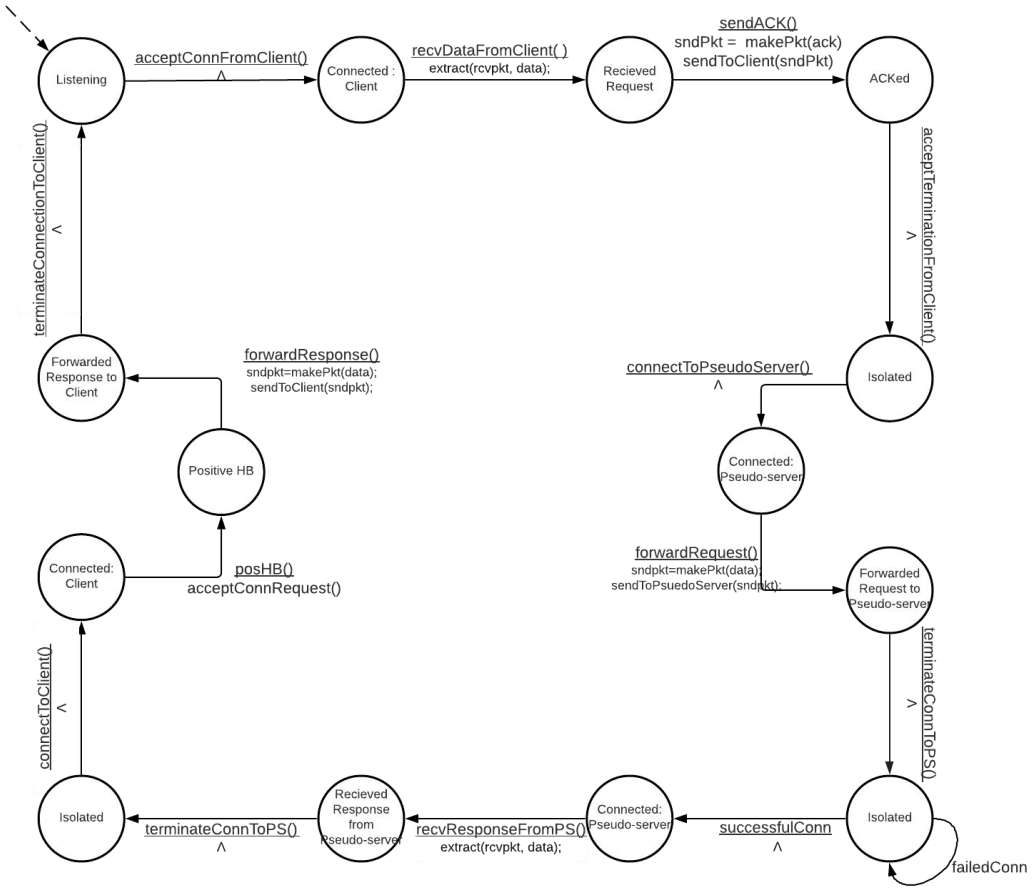


Figure 8. Pseudo-Client

This is the finite state diagram for the pseudo-client system of PSPIT. This system can be in 7 states in 2 directions throughout a request-response cycle. The initial state is achieved when the client system(depicted in the previous FSM) tries to establish a connection with the pseudo-client. The pseudo-client accepts this connection request. Once a connection is established using standard TCP procedures, the pseudo-client receives the request packet from the client. Upon successful receipt of this request, the system sends an ACK message as an

acknowledgment. Once the client receives the ACK, the connection is terminated between itself and the pseudo-client. At this point in the protocol, the pseudo-client is isolated from the other systems as there are no active connections to or from itself.

As its next phase, the pseudo-client connects to the pseudo-server and forwards the request packet to the pseudo-server. Upon receiving an acknowledgment of successful transmission of the same, the pseudo-client terminates its connection to the pseudo-server. Immediately after this step, the pseudo-client starts attempting to establish another connection with the pseudo-server. Since the latter is not accepting connections at the moment, all these connection requests fail. When the pseudo-server has new response packets to send, the connection requests succeed and a new connection for receiving the response packet is established. After receiving the response packet from the pseudo-server, the connection to the pseudo-server is terminated. Note that the transmission of the packet is done using normal TCP procedures.

The next phase is to relay the newly received packet to the client. All this while, the client would have been sending heartbeats probing for the existence of a new response message. Once the pseudo-client is prepared to forward the response to the client, it accepts a connection request from the client and responds to the heartbeats sent by the client. After this, it transmits the response message hence completing its part in the protocol.

6.1.3. Pseudo-Server

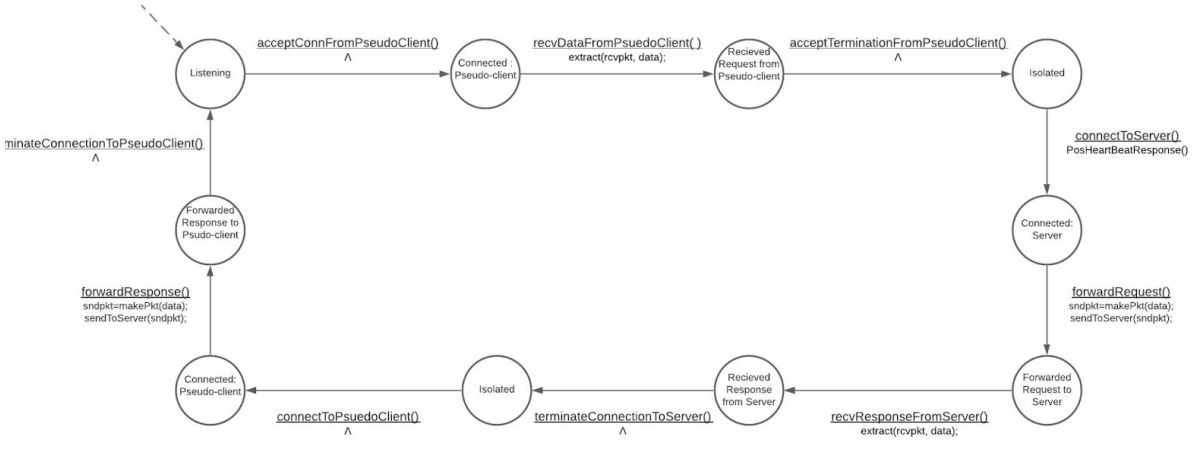


Figure 9. Pseudo-Server

This is the finite state diagram for the pseudo-server system of PSPIT. The initial state for this machine is that it is in a listening and isolated state waiting for a connection request from a pseudo-client. From the previous FSM given for pseudo-client, we can see that once the

connection is established between pseudo-client and pseudo-server, the data transmission takes place via TCP protocol procedure. Once the data packet is received, the pseudo-server terminates the connection and goes into an isolated state(no internet connection).

In the next phase, the pseudo-server keeps receiving heartbeats from the server, but only sends a positive response when it has data packets to be transmitted to the server and a connection with the server is established. The request is forwarded to the server using normal TCP procedures and the server responds back to the request. After receiving a response from the server, it terminates the connection with the server leading it to an isolated state.

And in this phase, the isolated pseudo-server acknowledges connection requests from the pseudo-client, as the response has been received from the server, via a normal TCP three-way handshake connection. The response is forwarded to the pseudo-client and the connection with the pseudo-client is terminated. Leading it back to the first state where it is in an isolated state waiting for the pseudo-client to send the request to establish the connection.

6.1.4. Server

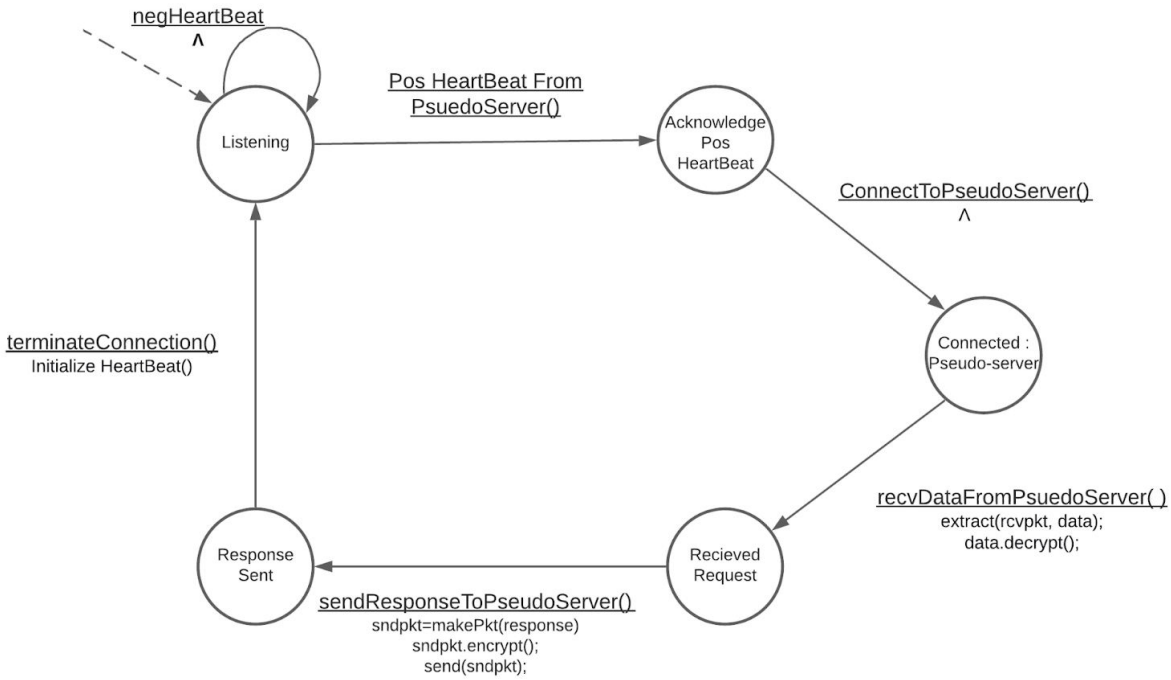


Figure 10. Server

This is the finite state diagram for the server system of PSPIT. This machine has 5 states including the listening/isolated state. This is the final machine to which all the data packets are

being sent. In the initial state of this machine, the server is an isolated state, sending heartbeats to the pseudo-server. The server sends repetitive beacons in order to check for the availability of requests at the pseudo-server. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-server is closed, the server is unable to send the beacons(this connection is later established by the pseudo-server).

In the next phase, the pseudo-server sends a positive response to those heartbeats, which are then acknowledged by the server and a connection is established. Request/Data packets are sent from the pseudo-server to the server and the server responds back with the necessary data packets/response requested by the client to the pseudo-server. Then it goes back to the isolated state, wherein it restarts sending heartbeats to the pseudo-server, and the cycle repeats itself.

6.2 Algorithm and Pseudo-code

6.2.1 Client

Sending Request to Pseudo-Client	`clientSocket = createSocket()`
	`clientSocket.connect(pseudoClient)`
	`pkt = createReqPacket()`
	`clientSocket.sendReq(pkt)`
	`ack = recvAck()`
	`clientSocket.close()`

Receive Response from Pseudo-Client	`//use socket returned from heartbeat module as clientSocket`
	`serverResponse = clientSocket.recieve()`
	`clientSocket.close()`

Heartbeats	`while(true):` ` try:`

	clientSocket = createSocket(pseudoClient).connect( )
	hb = createPacket(heartBeat)
	clientSocket.send(hb)
	hb_reponse = clientSocket.receive()
	if(hb = positive):
	return clientSocket
	except:
	if(clientSocket):
	clientSocket.close()

6.2.2 Pseudo-Client

Receiving Request from Client	fromClientSocket = createSocket(client) fromClientSocket.acceptConnecti on() request = fromClientSocket.receive() ack = makePacket("ACK") fromClientSocket.send(ack) fromClientSocket.close()
Forwarding to Pseudo-Server	pseudoServerSocket = createSocket(pseudoServer) //data received from client

	request = makePacket(data)
	pseudoServerSocket.send(request)
	pseudoServerSocket.close()
	while(true):
	  try:
	  pseudoServerSocket = createSocket(pseudoServer)
	  break
	  except:
	  pass
	  sleep(1) //This can be a moving average value

Receiving Response from Pseudo-Server	response = pseudoServerSocket.receive() pseudoServerSocket.close()

Forwarding Response to Client	respondHeartbeat(client)
	//toClient socket obtained from respondHeartbeat module
	toClientSocket.send(response)
	toClientSocket.close()

respondHeartbeat	toClientSocket = createSocket(client) toClientSocket.connect() hb = recieveHB()

	`toClientSocket.send("NEW")` `return toClientSocket`

6.2.3 Pseudo-Server

Receiving Request from Pseudo-Client	`pseudoClientSocket = createSocket(pseudoClient)`
	`request = pseudoClientSocket.receive()`
	`ack = makePacket("ACK")`
	`pseudoClientSocket.send(ack)`
	`pseudoClientSocket.close()`

Forwarding Request to Server	`respondHeartBeat()`
	`//serverSocket returned by respondHeartBeat module`
	`request = makePacket(request)`
	`serverSocket.send(request)`

Receive Response to Server	`response = serverSocket.receive()` `serverSocket.close()`

Forward Response to Pseudo-Client	`pseudoClientSocket = makeSocket(pseudoClient)`
	`pseudoClientSocket.acceptConnReq()`
	`response = makePacket(response)`
	`pseudoClientSocket.send(respons`

	e) pseudoClientSocket.close()
Respond Heart Beat	serverSocket = createSocket(server) hb = serverSocket.receive() hb_res = makePacket(“NEW”) serverSocket.send(hb_res) return serverSocket

6.2.4 Server

Receive Request from Pseudo-Server	//pseudoServerSocket received from heartbeat module request = pseudoServerSocket.receive()
Respond to Request	response = makePacket(reponse) pseudoServer.send(response) pseudoServer.close()

Heartbeats	while(true):
	try:
	pseudoServerSocket =
	createSocket(pseudoServer)
	hb = makePacket(hb)
	pseudoServerSocket.send(hb)
	hb_response =

```
pseudoServer.receive()
  if(hb_reponse == "NEW"):
   return pseudoServerSocket
 except:
  if(pseudoServerSocket):
   pseudoServerSocket.close()
```

6.3 Node Failure Handling

The pseudo-system protocol for information transfer, despite enforcing stringent host security measures, is prone to failures or nodes or delays of request-response messages like any other protocol. After all, the nodes in the system are physical machines running on sources of power with the possibility of being miles apart from each other which can contribute towards packet delays and losses. While delays are already handled by lower layer protocols like TCP, there are a few methods to implement delay detection/handling. Additionally, there are multiple methods to handle system downtimes/failures.

6.3.1 System Check Nodes

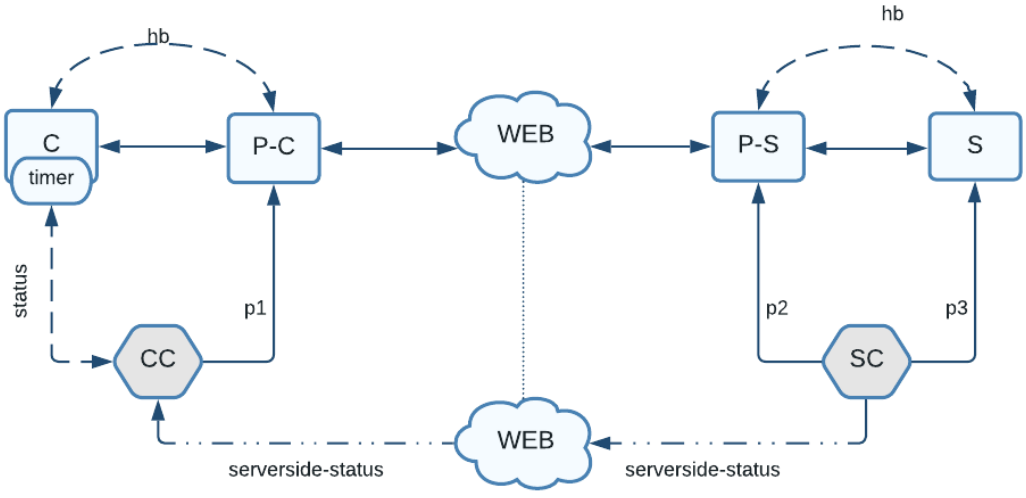


Figure 11. System Check Nodes

Here, two extra nodes(routers) are added to the entire topology. These are the Client Check(CC) on the client side and Server Check(SC) on the server side. These nodes maintain the availability of the entities in the protocol. Server Check handles the availability of the two server side machines,

namely the pseudo-server and the server machine and Client Check handles the availability of the client side node pseudo-client. Additionally, Client Check maintains the status of the server side machines in the form of bit strings. Eg, 101 means that the pseudo-client and the server are active but the pseudo-server system is down.

Client Check sends probe p1 to pseudo-client while Server Check sends probe p2 and p3 to pseudo-server and server, respectively. These probes are sent through a TCP channel which is initially set up when the systems come online for the first time. So, these channels are up and running as long as both sides of these systems are up and running. This means that at any point of time, Server Check is able to send a probe to check the availability of the systems under its supervision. The same also applies to Client Check.

The moment Server Check is unable to send a probe to it's child systems, it implicitly assumes that the system it was trying to probe is down. In the occurrence of such an event, the Server Check sends a SYSTEM_DOWN status message to the Client Check through the world wide web using conventional TCP protocols. The SYSTEM_DOWN message is simply a bitstring showing the availability of the systems under the Server Check node supervision. The next time(given that a SYSTEM_DOWN message has been sent) it is able to send the probes successfully to both the server side systems, it sends a SYSTEM_UP message to the Client Check machine.

Similarly, the Client Check continually probes the liveness of the pseudo-client system by probing it at regular intervals. Upon encountering the failure of the pseudo-client, it internally sets the status bit of the pseudo-client machine to 0. The next time it is able to probe the pseudo-client successfully, it resets the bit for that particular system.

Upon receiving the SYSTEM_DOWN message from Server Check, it changes the status of the respective machines to 0.

These are the internal workings of Server and Client Check machines.

Every request sent by the client starts a timer exclusive to that request. The expiration of the timer could imply the occurrence of two events.

- Delay in receiving the response: In this case, all systems taking part in the protocol are live. However, the excess traffic on the external internet is causing a delay in the transmission of packets in the lower layers. To detect this, the client sends a STATUS_CHECK message to the Client Check. This results in a response with a code of 111 which implies that all the machines

in the protocol are alive and functioning. The client then restarts the timer for that particular request. Upon three such cycles of timer + live discovery, the client immediately retransmits the request message.

- Failure of the systems: In this case, one of the systems in the protocol has failed. This is almost always preceded by the receipt of the SYSTEM_DOWN message by the Client Check. Upon expiry of the timer in the client for a particular request, it sends a STATUS_CHECK message to the Client Check. This results in a response with an indication that one of the systems is down. Eg, 101, 001, 100 etc. This is followed by the immediate re-transmission of the request by the client.

Drawbacks: The extra nodes added to the protocol, Client Check and Server Check are also additional power dependent machines implemented to keep track of the status of the machines taking part in the protocol. These are also susceptible to the same probability of failure as that of the protocol systems. If there is a failure of the check machines, the client would have to assume that one of the protocol systems is dysfunctional and wait for it to come online. However, this system does solve the task of identifying the faulty systems.

Additionally, these systems could be compromised and false data could enter the client machine leading to security lapses. A possible solution to this problem is to ensure that the check machines are isolated before the flow of data to the client.

6.3.2 Dedicated Timers In Each System

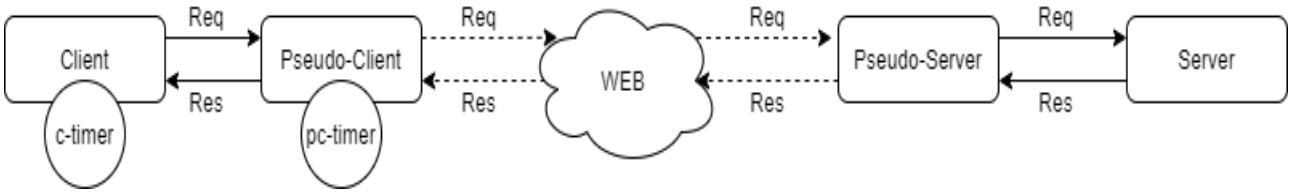


Figure 12. Dedicated Timers

In this proposed solution, each of the client-side machines have a dedicated timer built into each of them. These timers are asynchronous with respect to each other. Additionally, these timers are always running which means that, the moment each of those systems start up, the timer associated with that system is activated. The timers are depicted in the above diagram as follows:

- c-timer: This is the client timer exclusive to the client machine.
- pc-timer: This is the pseudo-client timer exclusive to the pseudo-client machine.

These timers are responsible for only the requests leaving their respective machines and do not keep track of the delay/latency of the requests handled by the other machines.

When the client system sends a request to the pseudo-client, its timer is started. This timer ends only when the response for that particular request is received back at the client. When the timer expires, the client resends the request message and restarts the timer. In case the client receives a delay indicator message from the pseudo-client before the expiry of its own timer, it ends the timer immediately and resends the request to the pseudo-client. In case the client is unable to establish a connection with the pseudo-client initially, it implies that the latter is dysfunctional and is unable to receive requests at the moment. However, the problem with using this methodology is that the client attempts to send a heartbeat immediately after sending a request message as well, and the inability to send the heartbeat could be mistaken for the dysfunctionality of the pseudo-client. In this case, the timer of the client is used as a fall back. The moment the timer expires, the client assumes that there is a delay or one of the systems has failed.

When the pseudo-client system sends a request to the pseudo-server, its timer is started; this timer ends only when the response for that particular request is received back at the pseudo-client. When the timer expires, the pseudo-client relays a delay indication message to the client. The client can then resend the request. In case the pseudo-client receives a delayed indicator message from the pseudo-server before its timer expires, it immediately ends its timer and relays this delay message to the client. Again, the client can attempt to resend the message. If the pseudo-client is unable to send the request in the first place, it can be implied that the pseudo-server or the server is disabled and a delay indicator message is sent to the pseudo-client. However, since the pseudo-client utilizes the inability to connect to the pseudo-server as a means to identify if there are new responses at the pseudo-server, this could cause havoc. The solution to this is that, if the request is sent and the pseudo-client is unable to connect to the pseudo-server, it is assumed that the server side machines are performing correctly. In case this assumption turns out to be incorrect, the already running timer for that request eventually expires and the pseudo-client can then send a delay indicator to the client.

The pseudo-server also has a fall back mechanism in case of errors. In case the pseudo-server is unable to send the request to the server, it can send a delay indicator to the pseudo-client which is relayed upstream to the client which can then resend the request once the server is online.

Drawbacks: The major drawback of this solution is that it is not easy to decide if the timeout is due to the delay in transmission of the messages or due to the failure of the machines.

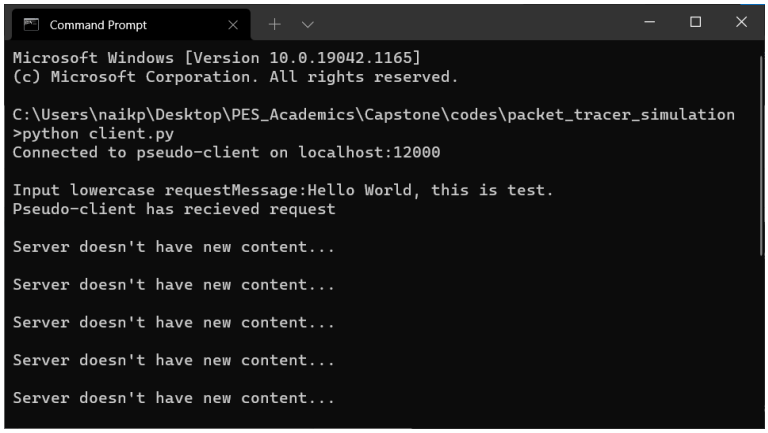
7. RESULTS AND DISCUSSIONS

7.1 Prototype in Action

The functioning of the protocol is illustrated using 4 terminals to simulate the 4 systems in the protocol: client, pseudo-client, pseudo-server and server in the local system prototype.

The prototype is initiated by running in the following order.

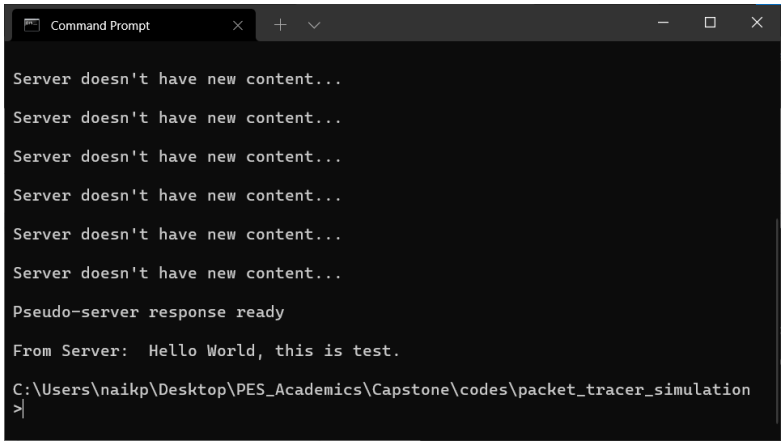
- Server
- Pseudo-server
- Pseudo-client
- Client



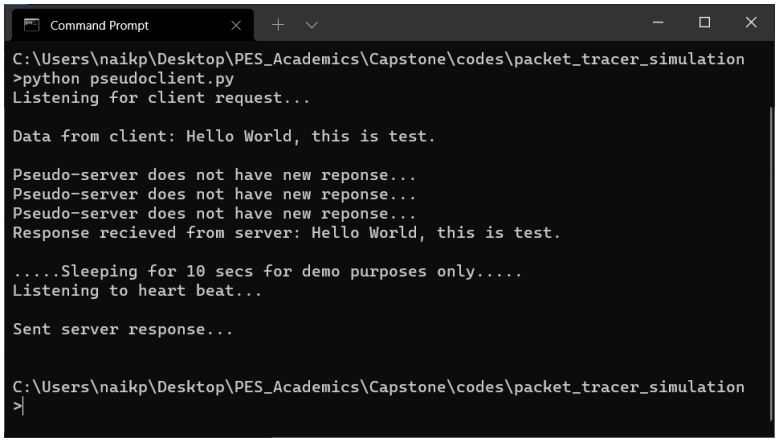
The above screenshot belongs to the simulated client terminal.

This is the result obtained after executing the client protocol script. In the present prototype, the client requests for input from the user. Immediately after this phase is done, the client starts probing the pseudo-client for the presence of a new response from the server.

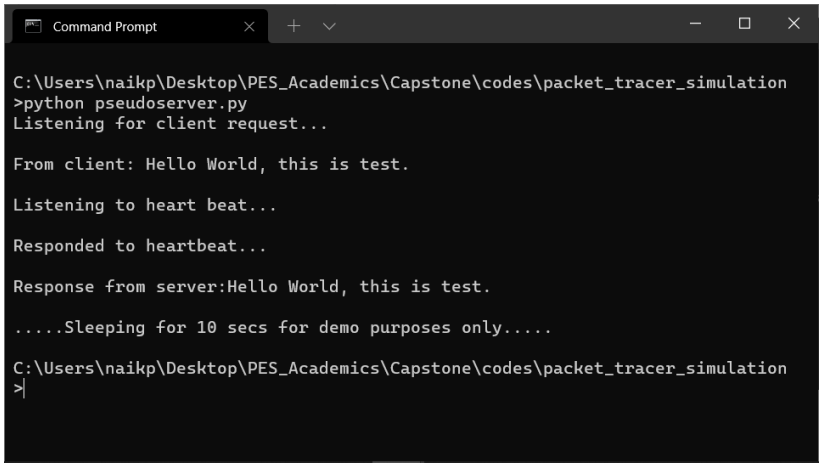
The consecutive “Server doesn’t have new content” messages are the outputs that indicate that the pseudo-client does not have new responses. This is done by attempting to send a packet through an open socket on the client.



As long as the pseudo-client has no response to provide to the client, it continues to probe the pseudo-client. The moment the pseudo-client has a new response, it responds to the probe. This step is indicated by the "Pseudo-server response ready" message being displayed on the client terminal. Once this is done, the pseudo-client sends the server response to the client. This is indicated by the message "From Server: Hello, World, this is test".



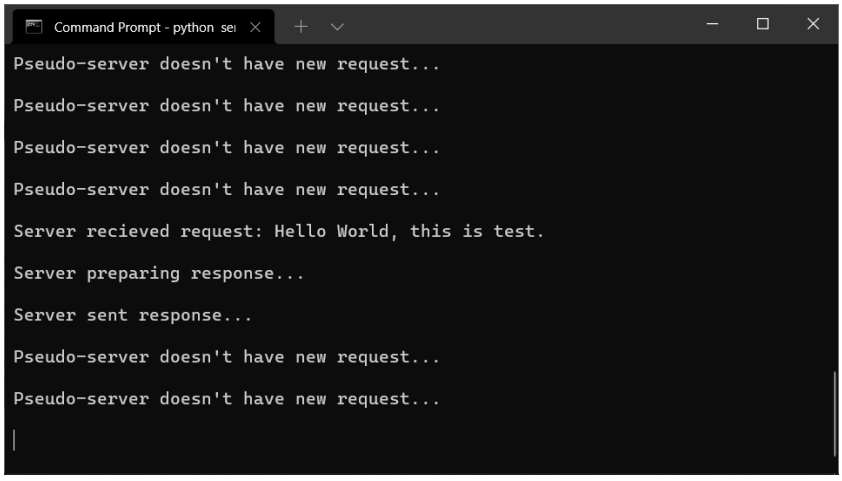
This is the pseudo-client screenshot. Initially, it receives the request from the client. This is indicated by "Data from client: Hello World, this is test". Immediately after that, it closes the client port, forwards the request to the pseudo-server, closes the pseudo-server port and tries to re-establish the connection with the pseudo-server. The re-establishment is indicated by "Pseudo-server does not have new response". Once the connection has been re-established, the pseudo-client receives the response and forwards it to the client. This is indicated by "Response received from server: Hello World, this is test" and "Sent server response"



This terminal initiates the pseudo-server python file. After the initiation, it starts listening for requests from pseudo-client. “From client: Hello World, this is test.” This is the input provided by the user at the client end.

“Listening to heartbeat. . .” is the step wherein the server keeps sending heartbeats to the pseudo-server, to check if there are any requests for itself. “Responded to heartbeat. . .” is the pseudo-server responding to the server with a positive message.

“Response from server: Hello World , this is test.” is the final response from the server back to the client.



This terminal corresponds to the server wherein the server keeps sending heartbeats to the pseudo-server, as a result, we can see the following message “Pseudo-server doesn’t have new request. . .”. After this step, the pseudo-server connects to the server when a message is received by it.

The next step is when the message is received, it prepares a response and sends it back to the client, as we can see the following steps are also displayed in the terminal. The next step is for the server to disconnect itself from the pseudo-server and start transmitting heartbeats through the UDP protocol.

In the multi-system implementation, four individual systems have been used, instead of one single simulation system, and packets will flow through the use of real-world physical media such as the internet (gateway routers and transmission links) and local area connections (MAC-Address communication).

The systems and their respective pseudo-systems will be connected to a Local Area Network (LAN), through the use of ethernet cables and RJ-45 connectors, to create a secure closed circuit communication medium between the two. Once packets have been sent from the pseudo-machine or vice versa, the local area connection is closed, and hence the main system is never connected to the internet.

The pseudo-system then establishes a connection to the internet and sends the packet to the destination pseudo-system via the internet and the ISP's router network. Similarly, on the receiving end, the pseudo-system receives the packets, shuts itself off from the internet, and connects to the destination system via LAN, in the same way as the sender and the pseudo-sender communicated.

The protocol also sees the use of "heartbeats", which is a term used for when a machine, unprompted, checks for a status change. If no status change is found, then the machine stays in the periodic check state, but if a change is detected, then a series of actions is carried out.

In order to ensure complete host isolation, we could not store the host's communication details on the pseudo-machine, as it connects to the internet, and if any malicious attacker is able to get access to the pseudo-machine, they could easily start communications with the main system, posing as a pseudo-system.

To prevent this, we made the use of heartbeats in our protocol. Since the communication details are not stored on the pseudo-system, the main system must initiate the communication all on its own. Since the pseudo-system is connected to the internet to check for incoming messages, it cannot be constantly connected to the main system, and hence periodically opens the LAN port to check for any queued messages from the main system, therefore the term "heartbeats".

The multi system implementation of the protocol utilizes a VPN service for demonstration purposes. Sans VPN, the traffic would be required to be routed through the public internet to the nodes that are concealed behind a NAT router. This made it inconvenient to demonstrate the proper working of the protocol as the NAT router configurations of the commercial routers were difficult to edit. The routers being used by the developers were made by Huawei, D-Link, TP-Link and Dragon. The router firmware had sophisticated configurations which could not be changed easily. Using a VPN service eased the process of communications while simulating the behaviour of transport of packets across the public internet.

In the multi-system implementation two physical systems were used to simulate the client and the server sides respectively. The first system ran the client and the pseudo-client scripts, while the second system ran the pseudo-server and the server scripts. The two processes on each of the machines are communicated using local connections as the protocol requires them to be in a local network. The VPN tunnel between the two systems simulated the public internet through which the protocol requires the pseudo-client and the pseudo-server systems to be connected. The results of this set up were similar to the local system implementation of the protocol which indicated a successful request and response cycle.

8. CONCLUSION AND FUTURE WORK

In a world where the value of data rises exponentially with every passing moment, its security becomes of paramount importance. Hackers and data breaches are taking new faces and finding ingenious ways to breach even the most secure protocols. In such a situation, it feels as though systems are better off not even connecting themselves to the internet. Although infeasible, PSPIT finds the best of both worlds, the power of the internet, as well as the safety of staying off the grid. This is what we call Host Isolation. The system with sensitive data sits off the grid, whereas a pseudo-system does the heavy lifting of data download/upload, and establishment of encrypted channels over the internet. This way, PSPIT has a lot of real-world applications where sensitive data is to be secured over a primary system, such as defense, finance, medicine, etc.

Pushing PSPIT over the well-accepted standards of TCP and TLS shows a strong trust factor on the protocol, and can even be made an optional security recommendation for high-security interconnected systems. Collaboration with international certification organizations such as IETF is critical in the future work of the protocol, and we intend on getting the protocol certified as a trusted security service framework.

The future work for the protocol involves working on further safeguards localized in the pseudo-systems which includes but is not limited to

- Partial execution of code
- Anti-malware screening of payload
- Stability testing of payload
- Encrypted sandbox testing

These additional features ensure that, alongside preventing malicious attackers from being able to discover the primary system, the data itself does not pose a risk to the primary system. This allows PSPIT to be an all-in-one toolkit which can be used to effectively create a blanket solution for all system security needs

This would require extensive research, further to the research we have done to develop the protocols, as this forays into the domain of the operating system. Implementing OS safeguards over

the network level safeguards would be our final stage of completion, post publication of this research paper, and post IETF certification of PSPIT as a viable security standard.

Furthermore, these multi-system communications will be encrypted with the use of TLS, which is Transport Layer Security. TLS is a secure communication encryption standard used worldwide for encrypting internet communications from unintended and/or malicious attackers. TLS uses keys in the form of certificates, cipher and decipher messages, and the certificates are confidentially maintained by each pseudo-system. This TLS security layer acts on top of TCP, with the systems sending their certificates to each other via the protocol, following which messages are encrypted with the destination machine's public key, and decrypted with its private key.

We will be researching the best configurations of TCP-TLS that can be used as part of our protocol, which was a part of our literature survey as well, to find the most secure and convenient settings package for our use, as well as which configuration works most efficiently on the largest number of systems.

The protocol shows great promise for real-world applications, due to its closeness to the issues that real-world systems face, and the simple solution of enforcing host isolation can mitigate a large number of security volatilities. We picture PSPIT to be one among the industry standards in host security - a domain yet unexplored.

REFERENCES AND BIBLIOGRAPHY

[1] Elsadig, Muawia & Fadlalla, Yahia. (2016). Survey on Covert Storage Channel in Computer Network Protocols: Detection and Mitigation Techniques. International Journal of Advances in Computer Networks and Its Security. 6.

[2] Pawar, Mohandas & Anuradha, J.. (2015). Network Security and Types of Attacks in Network. Procedia Computer Science. 48. 10.1016/j.procs.2015.04.126.

[3] Pandey, Shailja. (2011). MODERN NETWORK SECURITY: ISSUES AND CHALLENGES. International Journal of Engineering Science and Technology. 3.

[4] Manu, A. & Rudra, Bhawana & Reuther, Bernd & Vyas, O.. (2011). Design and Implementation Issues of Flexible Network Architecture. 10.1109/CICN.2011.59.

[5] Abbasi, Abdul & Muftic, Sead. (2010). CryptoNET: security management protocols. 15-20.

[6] Satapathy, Ashutosh & Livingston, Jenila. (2016). A Comprehensive Survey on SSL/ TLS and their Vulnerabilities. International Journal of Computer Applications. 153. 31-38. 10.5120/ijca2016912063.

[7] Cheng, Pau--chen & Garay, Juan & Herzberg, Amir & Krawczyk, H.. (1998). A security architecture for the Internet Protocol. IBM Systems Journal. 37. 42 - 60. 10.1147/sj.371.0042.

[8] Sirohi, Preeti & Agarwal, Amit & Tyagi, Sapna. (2016). A comprehensive study on security attacks on SSL/TLS protocol. 893-898. 10.1109/NGCT.2016.7877537.

[9] Chakravarty, Sambuddho & Portokalidis, Georgios & Polychronakis, Michalis & Keromytis, Angelos. (2015). Detection and analysis of eavesdropping in anonymous communication networks. International Journal of Information Security. 14. 10.1007/s10207-014-0256-7.

[10] James F. Kurose and Keith W. Ross. 2012. Computer Networking: A Top-Down Approach (6th Edition) (6th. ed.). Pearson.

[11] Behrouz A. Forouzan and Sophia Chung Fegan. 2002. TCP/IP Protocol Suite (2nd. ed.). McGraw-Hill Higher Education.

[12] Atighetchi, Michael & Soule, Nate & Pal, Partha & Loyall, Joseph & Sinclair, Asher & Grant, Robert. (2013). Safe Configuration of TLS Connections - Beyond Default Settings. 6th IEEE Symposium on Security Analytics and Automation (SafeConfig 2013).

[13] Castelluccia, Claude & Mykletun, Einar & Tsudik, Gene. (2006). Improving secure server performance by re-balancing SSL/TLS Handshakes. 2006. 26-34. 10.1145/1128817.1128826.

[14] Larisch, James & Choffnes, David & Levin, Dave & Maggs, Bruce & Mislove, Alan & Wilson, Christo. (2017). CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. 539-556. 10.1109/SP.2017.17.

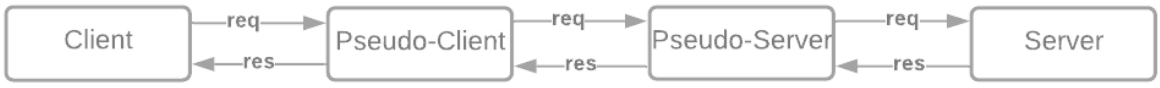
[15] Borboruah, Gayatry & Nandi, Gypsy. (2014). A Study on Large Scale Network Simulators. International Journal of Computer Science and Information Technologies. 5. 7318-7322.

APPENDIX A: DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1. TCP: Transmission Control Protocol
2. UDP: User Datagram Protocol
3. IP: Internet Protocol
4. TLS: Transport Layer Security
5. SSL: Secure Socket Layer
6. PSPIT: Pseudo-System Protocol for Information Transfer
7. Client: The node in the network initiating a PSPIT connection.
8. Server: The node in the network responding to the PSPIT client request on the opposite end of the client.
9. Pseudo-systems: The intermediate nodes in the path from the client to the server acting which is practical, router/switches.
10. Pseudo-client: The node which immediately receives the request from the client and forwards the same to the next node, called the pseudo-server. This also relays the response from the pseudo-server to the client.
11. Pseudo-server: The node that receives the request from the Pseudo-client and forwards the same to the server. On receipt of a response from the server, it relays it to the pseudo-client.
12. Heartbeat(s): Repetitive beacons/probes transmitted by the client and the server to check for availability of new data in pseudo-client and pseudo-server respectively.

APPENDIX B: USER MANUAL

1. Download the prototypal codes for client, pseudo-client, pseudo-server and server.
2. Start 4 terminals / command prompts, one to mimic each system.
3. Start the prototype by first instantiating the server system(in this case, the command prompt) by running the server.py file
4. Start the pseudo-server by executing the pseudoServer.py file.
5. Start the pseudo-client by executing the pseudoClient.py file.
6. Start the client by executing the stdclient.py file.
7. This order of executing the files is important due to the required direction of data flow, which is:



8. The prototype can be verified by the success messages displayed on the server and the client which indicate that data flows between server and client and vice-versa have been successful.

Pseudo-System Protocol for Information Transfer

by Prajwal Naik

Submission date: 20-Oct-2021 08:31AM (UTC+0530)
Submission ID: 1678695055
File name: report.pdf (1.75M)
Word count: 13098
Character count: 69929

Pseudo-System Protocol for Information Transfer

ABSTRACT

If one draws a parallelism between a computer and a building, one might say the ports behave as doors that facilitate the movement of people, who serve as metaphors for packets of data. Networking protocols would be managers who would decide what kind of people are allowed entry (packet formats), whom to expect when (packet exchange sequences) and when to open the doors and more importantly, for whom (host authentication). Firewalls would then serve as guards while anti-virus systems would be metaphoric bouncers, identifying and ejecting suspectedly troublesome people (malware).

This set-up is far from fool-proof and has an inherent set of flows. The safety of the building (the host) rests on the efficacy of the managers, guards and bouncers in performing their respective duties. Even if they performed their jobs diligently, there's always the risk of malicious attackers finding new and creative ways to enter the establishment either by masquerading their intentions (trojans) or by simply finding alternate doors with possibly less protection.

Now, imagine the building's address was secret. In this scenario, all potential entrants (packets) were received at a holding area disconnected from the main building where they would be thoroughly inspected and any suspicious persons would be ejected. The people who pass the inspection would then be securely transported to the main building (host) in a way that prevents them ever knowing its address. In this way, the building is much more inherently protected as it remains virtually nonexistent to attackers.

The Pseudo-System Protocol for Information Transfer uses pseudo-clients and pseudo-servers to enable hosts to communicate through the internet while being completely isolated (offline) from the internet. It enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds

1. INTRODUCTION

Communication protocols have always looked at data security and host security as unrelated goals. Protocols like TLS have been successful in ensuring data security. The goal of PSPIT is to take a unified approach to data and host security by implementing pseudo-systems that help hosts isolate.

PSPIT aims to ensure host and data security while limiting latency to a minimum with the additional constraint of making the implemented pseudo-systems as economic as possible (by limiting buffer size and computational ability to the bare minimum).

The study began with an in-depth review of existing literature on TCP, SSL, and TLS which ultimately form the bedrock upon which PSPIT is built. It then moves on to a description of the practicalities of the implementation of the first proposed prototype - a socket simulation of the protocol.

Most commercially used communication protocols enable only data security. PSPIT goes a step further and implements host security by essentially allowing communication via the internet by never connecting to it.

This prototype enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds.

This protocol secures the data being transferred and masks the host so that a third party cannot intervene for stealing data or gaining access to the hosts. The protocol sequences the transfer of data such that the hosts always remain disconnected from the internet while using the internet as the primary medium for the transfer, thus making full use of the power of the internet but without any of the risks of intrusion that come with it.

Today's protocols do not contain an inclusion for end system isolation, and PSPIT allows for information to be sent and received without the end systems ever being directly connected to the internet, which significantly reduces the risk of attack.

2. PROBLEM STATEMENT

Almost all of the existing communication protocols in the protocol stack used by the public internet today are more oriented towards data security. Multiple encryption techniques that have come to be used in recent days ensure that intercepting and deciphering messages across the internet is not an easy task.

However, these protocols fail to take into consideration the host security aspect of any communication session. Using the present protocols, any attacker on the internet could learn the identity of the hosts taking part in the communication and inject malicious scripts into the systems. None of the encryption methods in use presently can stop this disaster from happening.

This study takes this requirement of the modern internet seriously. The main goal of this study is to develop a protocol such that the systems taking part in a connection are never exposed to the public internet. This way, the probability of the attacker learning about the presence of the device is reduced by a massive amount.

3. LITERATURE SURVEY

3.1 Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes

Majority of today's distributed computing uses the client-server model. Servers handle multiple requests from clients. Since the modern day networks work on SSL, a considerable amount of server capacity goes on performing computationally expensive RSA Public Key decryptions as part of each SSL handshake. This step can be very detrimental and time consuming to the performance of the server.

This paper explores the existence of a technique for re-configuring RSA-based handshakes. As per the results obtained by the authors, the new method speeds up the processing of RSA operations 11-19 times depending on the size of the key.

There have been fast advances in trying to ensure services degrade gracefully and many of these methods have been implemented in real world servicing as well. Additionally, many studies in networking have yielded low latency results speeding up the communication between the client and the server. However none of these results thwart DoS attacks as their primary goal is to create a server overload and deny the servicing requests from legitimate clients.

In the SSL connections used these days, RSA decryption occurs on the server side when it decrypts the secret to receive the base message. This technique is called Server-Aided RSA (SA-RSA). This paper investigates this technique and tries to change the way the client and server sides handle encryption and decryption. The proposed method simply reassigns the roles of the client and the server. The SSL client becomes the "server" and the SSL server suffering from overload becomes the "weak client". This results in a Client-Aided RSA which has proven to speed up the decryption process.

The paper begins with a description of the SSL handshake.

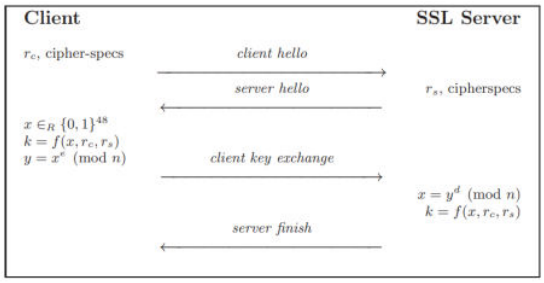


Figure 1. Normal SSL Handshake

The handshake starts with the delivery of a "hello" message from the client to the server. This is an indication of the willingness of the client to start a TLS/SSL session. This message contains the cypher suits and a random, one time number rc.

The server response includes a "hello" message from the server-side and includes a public-key certificate and another random one time number rs. The server additionally mentions it's choice of cipher suite.

A pre-master secret x is chosen and the shared master secret k is obtained using f(x, rc, rs) where f is the hash function. x is then encrypted with the RSA public key of the server. Additionally, the ciphertext is attached to a "client key exchange" message. Finally, this is sent to the server.

Finally, the server uses its private key to decrypt the pre-master secret and in turn uses it to obtain the master secret as f(x, rc, rs). The handshake is concluded by sending a "server finished" message.

The most important step of the entire handshake process is the encryption of the key under the RSA public key (of the server) by the client. The computationally extensive problem is the RSA decryption.

Although RSA is very well-studied and tested, many of the RSA implementations are computationally imbalanced. When public exponents like 3, 17 and 216 + 1 are used, the RSA decryption is expensive while the corresponding encryption is comparatively cheap, even though the exponents are small. A possible solution to solve this imbalance is to select a small d (private exponent) which in turn speeds up computation. However, this compromise leads to an insecure RSA.

The authors of this paper have tried to flip around the SA-RSA technique and obtained the Client-Aided RSA. The main objective was to transfer a portion of computation from the server to the client. Through this, the clients perform the majority of tasks in decryption, letting the server dedicate more resources to handling the request rather than contributing to cryptography.

The algorithm starts of by the representation of the exponent, private to the server, as d = f1d1 + f2d2 + ... +fndn (mod ϕ(n)). Here, the di's and the fi's are vector elements which are random and of length |n| and c bits, respectively. Since the RSA decryption process involves computing xd (mod n) on the server side, the following steps are performed by it.

The server sends the vector D = (d1, d2, …, dk) to the client.

The vector Z = (z1, z2, …, zk), where zi = xdi (mod n), and sends it to the server.

As a final step, the server computes xd using the vector Z received from the server.

The algorithm assumes that breaking RSA is difficult. Using these assumptions, selecting the parameters of the algorithm such as k, c, and f does not introduce any compromises in security. To guess the values of c and k, any attacker would have to minimally use brute force which would have a complexity of 2c*k. This is the same as the time complexity required to crack RSA. Additionally, this basic description of the algorithm can be improved performance-wise by taking advantage of the Chinese Remainder Theorem.

The next section of the study includes a method to incorporate the CA-RSA method into the existing SSL handshake. The initial "hello" messages by the client and server remain mostly unchanged except the certificate of the server includes the D vector (D = (d1, d2, ...dk)). After the client encrypts using the server's public exponent, it constructs Z by computing the zi = xdi (mod n). This is then included in the client key exchange message. After receiving this, the server performs its CRT computations. The modified protocol is as follows:

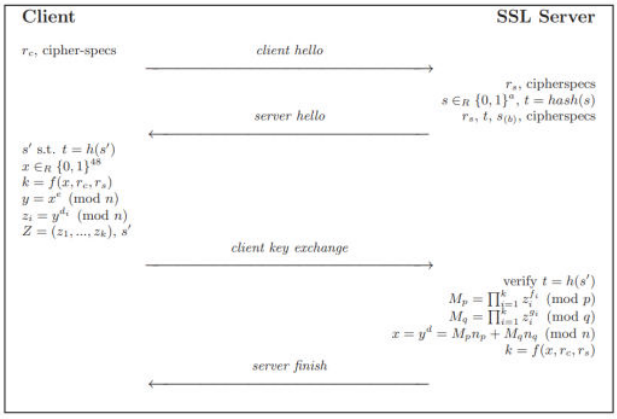


Figure 2. Modified SSL Handshake

The major concerns regarding the integrity of the systems are on choosing the parameters, c and k. The requirements needed to be met are such that the time complexity of exhausting the search space to guess the values of c and k are at least equivalent to breaking the original RSA cryptosystem.

The authors have measured the improvement in the execution times when the protocol was changed from SA-RSA to CA-RSA. The key attribute used for computation was the number of RSA decryptions performed by the test system in a specified time frame. The test machine used had the following specifications:

Processor: Intel Celeron 1.7 GHz

RAM Capacity: 256 MB

OS: Red Hat 9.0 Linux

The results of the experiment were as follows:

Key size	RSA	CA-RSA	$c \times k$	Improvement
1024	7.05	0.62	72	11.33
1536	19.79	1.25	80	15.76
2048	44.22	2.31	88	19.12

Figure 3. Results of New System

These results imply that 11 times as many decryptions can be performed by a server with a 1024 bit RSA key. The smallest value of k is 2, as specified by the guidelines for selecting parameters. Therefore, the values of c and k have been chosen as 2-36, 2-40 and 2-44.

CA-RSA introduces certain additional bandwidth and computational costs at the clients, despite reducing computational burden on the servers to a very large extent. As computed by the authors, this protocol adds an additional computation time of 21.9, 66 and 150.9 seconds while computing Z for 1024, 1536 and 2048 bits on the client-side. However, these computational costs seem to be negligible since the entire protocol is aimed at balancing the degree of computation between the server and the client. Additionally, the client key exchange message also includes certain bandwidth costs incurred while using the protocol since it now has to carry a vector Z. However, with the value of k set to 2, the over is calculated as |n| where n is the length of the RSA modulus. However, this overhead takes less than one ethernet frame to accommodate. However, in case the system being used is very sensitive to bandwidth and computational changes, the protocol can be used only in case of overload of servers. When this technique is incorporated, the overheads occur occasionally.

So far, the authors have focussed on rebalancing the computations between the servers and the client. The protocol assists in enabling the server to increase the quantity of SSL connections that are handled. This alone makes the task of a DoS attacker much harder. However, any attacker could simply overload a CA-RSA server by increasing their computational resources. The authors have included a mechanism to make this task more difficult. An attacker willing to carry out a DoS attack on a server only needs to bombard the server with as many requests as the number of RSA decryptions the server can perform. A possible fix to this would be to direct the client to solve a set of "client puzzles" before sending the encrypted message. This when combined with the lesser decryption time provided by CA-RSA would prove effective in thwarting DoS attacks on SSL servers.

The client puzzle aspect of the proposed solution for DoS attack protection can be implemented by using the method proposed by Juels and Brainard. This method requires a client to solve a

computational puzzle before requiring the server to carry out the decryption. This ensures sufficient time between consequent service requests. Following the client's hello message, the server selects a random a-bit value s and passes it via a cryptographic hash function. It then sends the server greeting message to the client with the hash digest t = hash(s) and the b first bits of s (where b a). The client uses these b bits to brute-force to solve the "client puzzle" and discover a value s' that hashes to the desired t. After that, the client adds s' to its client key exchange message. The server will only proceed with the SSL handshake and decrypt the encrypted session key given by the client if s' verifies - that is, if it is the correct length and its hash output is t.

The protocol can be modified to incorporate this by adding the puzzle solution to the frame that contains the Z vector.

When compared to an RSA decryption, the computing cost of a hash computation is almost small, therefore the addition of a puzzle verification step adds just a minor server burden. The amount of labor required by the client to answer the problem is determined by its computer resources and, more significantly, the number of unknown bits in the server's preimage value.

The modified protocol is as follows:

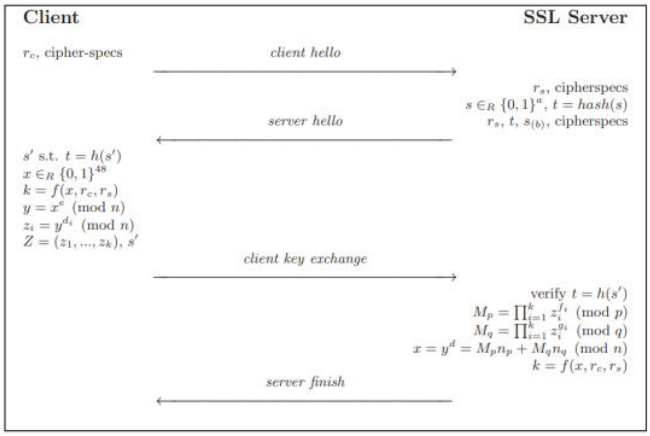


Figure 4. Further Modified Protocol

Even though there is a possibility that a malicious system may solve the puzzle and send fake Z vectors to the server, the possibility of this happening is 11 times less than when using SA-RSA.

In conclusion, the system proposed by the authors helps distribute the server load to the client and thus helps the server authenticate more requests with a very low possibility of overloading.

3.3 CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers

In this paper, they present CRLite(Certificate Revocation List Lite), an efficient and easily-deployable system for proactively pushing all TLS certificate revocations to browsers.

CRLite servers aggregate revocation information for all known, valid TLS certificates on the web, and store them in a space-efficient filter cascade data structure. Browsers periodically download and use this data to check for revocations of observed certificates in real time. CRLite does not require any additional trust beyond the existing PKI(Public Key Infrastructure), and it allows clients to adopt a fail-closed security posture even in the face of network errors or attacks that make revocation information temporarily unavailable.

Comparing CRLite to an idealized browser that performs the correct CRL(Certificate Revocation List)/OCSP(Online Certificate Status Protocol) checking, they show that CRLite reduces latency and eliminates privacy concerns.

Chrome and Firefox only make CRL and check via OCSP and requests for EV(Extra Validation), and not all mobile browsers check for revocation. And this leads to several problems while secure browsing.

CRLite is implemented in two parts: a server-side system that aggregates revocation information for all known, valid TLS certificates on the web and places them in a filter,and a client-side component that downloads filters and uses them to check for revocations of observed certificates. And it deals with majorly six challenges, which are efficiency, timeliness, failure, privacy, auditability, deployability.

Background:

A. The TLS Ecosystem

Certificate Validation, Certification Transparency, Measuring TLS Ecosystem

B. TLS Certificate Revocation

CRL Certificate Revocation List, OCSP Online Certificate Status Protocol

C. Revocation Checking

Revocation Checking in Practice, Fail-open vs. Fail-closed, CRLSet and OneCRL

D. Other Revocation Distribution Schemes

Micali's Certificate Revocation System, multi-certificate revocation, revocation trees, or combinations of these techniques.

E. Proposals to Replace PKI

AKI, PoliCert, ARPKI and PKISN aim to replace the existing PKI with a new hierarchy that avoids centralizing trust, is transparent, and supports seamless revocation.

At a high level, this method aggregates all revocation information for every known certificate, compactly represents them in a filter cascade, and provides a means by which clients can publicly audit them.

The server side of the application works in this order:

Obtaining Raw Certificates ⟶ Validating Certificates ⟶ Obtaining All Revocations ⟶ Filter Cascade Construction ⟶ Delta Updates ⟶ Audit Logs ⟶ Hosting

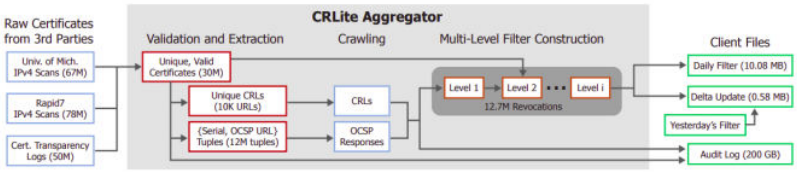


Figure 5. CRLite Aggregator

1. *Obtaining Raw Certificates*- To create the filter of revoked certificates, CRLite first needs a list of all valid certificates.

2. *Validating Certificates*- CRLite validates all certificates by looking for non-expired, well-formed leaf and intermediate certificates that cryptographically chain to a trusted root.

3. *Obtaining All Revocations*- To collect revocations, CRLite extracts CRL and OCSP responders from all valid certificates.

4. *Filter Cascade Construction*- CRLite uses Bloom Filters for storing and retrieving information as the rate/probability of finding the required certificates is more efficient, has higher probability and is protected.

5. *Delta Updates*- CRLite produces delta updates that allow clients to incrementally update their copy of the filter(a small percentage of certificates change on a daily basis).

6. *Audit Logs*-The audit log contains (1) copies of all CRLs and OCSP responses that were used to construct the corresponding filter cascade, and (2) copies of all certificates included in the whitelist

7. *Hosting*- This is the display of all delta updates and audit logs through web servers hosted on cloud storage.

Security Analysis:

Man-in-the-middle, forcing fail open, DoS, BackDated Certificate and Rogue Aggregator are taken care of by CRLite.

This paper helps one to keep track of the revocation of the TLS certificates, by adding an extension to the browser and they're really helpful for private organisation, who use certificates, as the list generated keeps updating every 24 hours and let's one know if it's safe to use/continue with the current in use certificate and the list is of reasonable size due to the usage of bloom filters and cascading filters.

3.4 Safe Configuration of TLS Connections Beyond Default Settings

This paper, authored by Michael Atigetchi, Nathaniel Soule, Partha Pal and Joseph Loyall, focuses on TLS (Transport Layer Security), its precursor SSL (Secure Sockets Layer), drawbacks of "out-of-the-box TLS configurations" and optimal settings to minimize the risk of TLS-compromise.

Since the deployment of SSL in 1995 and the subsequent release of TLS in 1999, the TLS-SSL standard has been the frontline for communication encryption, by countering attackers who try to sniff live traffic or even man-in-the-middle attacks, that try to compromise the confidentiality of information being transmitted. Despite the wide-scale deployment of TLS 1.3 (usage: 2014-current), it remains vulnerable to a number of threats at the protocol layer and does not provide robust security when used "out-of-the-box", hence requiring tweaks to its configuration in order to attain the expected security benefits. The paper goes over these tweaks and elucidates the subsequent security enhancements.

To begin with, TLS is a security standard that makes use of certificates that have been issued by standardized bodies, called certificate authorities (CAs). These certificates serve the purpose of identity cards, that allow a page or a server to verify their identity with someone trying to communicate with them. The client, upon receiving the certificate from the server, checks the certificate for authenticity, and upon verification of the same, initiates the second step of the handshake by sending a copy of its own certificate. In this way, both parties have full awareness and full confidence in who they are communicating with.

But the role of the public key does not end there, as the client then encrypts the session using public key cryptography i.e., it uses the server's public key to encrypt the data. This encrypted data, upon reaching the server, can only be decrypted using the server's private key, which is stored confidentially on the server's end. This way, using public-key cryptography (aka symmetric cryptography), the data is encrypted and can only be decrypted by the server.

Beyond the working of TLS, the paper discusses additional means of protecting data encrypted by TLS/SSL, first of which is called the *Crumple Zone*.

Crumple zone is a safety technique used by cars to minimize the damage to core components and passengers inside the automobile in the event of a crash. It consists of reinforcing empty space inside a car with roll cages and strengthened steel bars in order to absorb and distribute the impact of a crash, diminishing its intensity before it is able to reach the passengers and cause any harm. This same idea is what inspired a "Crumple Zone" (CZ) for data encrypted by TLS-SSL.

The crumple zone is intended to reduce and absorb the effects of malicious attacks before it is able to cause any damage to the confidential data.

This necessity of creating a CZ for data arose out of the popularity of Service Oriented Architectures (SOAs). SOAs feature loose coupling, high dynamism and composition-oriented system constructions which make this architecture

highly appealing for application creation, but equally complicated to create viable security for, as they create more points of entry, as compared to self-contained architectures or monolithic systems.

In the context of SOAs, the CZ consists of a layer of intelligent filter proxies which present a higher barrier of entry for attackers, which minimize the damage made to critical data, and increases the chances of detection in the event of such a malicious attack.

These intelligent proxies behave as endpoints for any inbound TLS connections attempting to reach the service, and no data can reach the service before passing through the CZ. The proxies comprising the CZ are equipped to scan, filter, and even partially execute the data to verify its non-maliciousness before it can pass through and reach the service. Since the components present in the CZ also sometimes execute the information, they are expected to fail occasionally, and are hence monitored by a watchdog process which restarts them whenever necessary.

The CZ then finally establishes all outbound TLS connections to then pass on the approved data from the CZ to the protected service.

The three main principles that need to be maintained for proper CZ functionality are as follows:

- Consistency: All data flowing in and out of the CZ must be protected via TLS
- No bypass: No data flows exist such that they can reach the protected service without passing through the CZ
- Security: The TLS configuration used must be capable of defense against malicious attacks and must be cryptographically strong.

Beyond these ideas, the paper discusses the TLS threat model, and the best practice configuration recommendations for TLS to be utilized for information interchange.

The TLS Attack Surface:

Since the rollout of SSL/TLS in 1999, it has been used widely to secure information interchanges on a variety of communication channels, be it browser authentication or service verification. Due to the large number of browsers, browser versions, SSL/TLS protocol versions and implementations, the tuple space of <version, implementation, configuration> is so terribly large, which leads to an expansive attack surface. In short, the authors are trying to convey that the regular, out of the box TLS will never be fully equipped to handle the large number of possible vulnerabilities across all members of this tuple space. The authors also mention that in the past few years, even the most recent versions of TLS have been successfully breached and results have been published on various networking journals.

Furthermore, the authors also mention that due to the slow and piecewise upgrades of the different browsers to later versions, as well as the non-uniform upgrade of TLS versions used on different servers, there exist a huge number of TLS version incompatibilities even to this day. To overcome this, the client and server must negotiate on a common protocol so as to facilitate communication between the older and newer version of TLS. This down-negotiation means that even though many servers have upgraded to later versions of TLS, they may still be forced to use older versions of TLS, with all their vulnerabilities.

The author mentions the example of BEAST (Browser Exploit Against SSL/TLS) in which attackers exploit a vulnerability present in the CBC (Cipher Block Chaining) cryptography method used in TLS 1.0, which can allow an attacker to perform a plaintext extraction of data secured by CBC. This meant that even if servers upgraded to later versions of TLS, if they

were communicating with services using TLS 1.0, they would be exposing themselves to this vulnerability present in the CBC encryption format. The authors state that according to SSL-Pulse, a global monitoring service, 65.2% of sites surveyed by them were susceptible to this attack, even though many had upgraded to later versions of TLS.

Similarly, another attack method known as CRIME (Compression Ratio Info-leak Made Easy) which relies on information leakage from observations of compression behavior to allow attackers to break SSL/TLS encryption. According to SSL-Pulse, 24.1% of sites they surveyed were vulnerable to this attack, and these sites used compression techniques alongside TLS, which is what made them vulnerable to CRIME.

To counter above mentioned *attack surface* of TLS, the following recommendations are mentioned by the author:

- Configuration restriction
- Mutual authentication
- Hostname verification

Configuration Restriction: -

Due to the expansive number of configurable TLS parameters, cipher combinations, and multiple provider options, there are a large number of doors left open for attackers to exploit. The authors recommend the below mentioned techniques to reduce the attack surface of TLS arising from configuration variation:

- Cipher restriction: Due to the large number of available cipher techniques (such as MD5, SHA, RC4, etc.), many of these do not provide the best possible security but are still used as they take lesser time for encryption/decryption hence making lesser overheads. These leave servers open to BEAST attacks as mentioned earlier, or collision attacks orchestrated by a malicious client. If TLS were to be configured to use only a small set of ciphers that are known to be resilient against current exploits, then this would considerably reduce the attack surface of TLS.
- Protocol Restriction: As mentioned earlier, incompatible versions of TLS can cause down-negotiation which can increase the vulnerability of even the latest versions of TLS. To prevent this, older versions of TLS must be phased out and newer versions must be enforced.
- Options Restriction: The CRIME attack was mainly a result of servers using compression techniques alongside TLS, which exposed them to the attack. Eliminating the option of TLS compression completely will help reduce the attack surface of TLS.

Mutual Authentication:

TLS provides confidentiality and message integrity, but is also used to verify the identity of servers that are being contacted by the use of digital certificates as shown earlier. However, TLS does not provide client certification, and there is no way to find out if a server is communicating with a legitimate client. TLS has mainly relied on company specific password schemes for credentials.

Military, Army, Defense and other high priority channels however, use client authentication methods such as smart cards or Common Access Cards (CACs) which allow the client as well as the server to establish crypto-strong trust. If this is enforced internet-wide, it could significantly mitigate attacks on TLS's confidentiality and integrity.

Hostname Verification

TLS performs certificate verification, and not endpoint verification. This means that if a party X where to get its hands on a compromised but valid/trusted certificate belonging to party Y, and party X presents this certificate to any client trying to communicate with party Y, then the clients would accept the certificate and begin trusted communication even though the endpoint is not party, but party X. This endpoint verification must be explicitly enabled on TLS, and external code blocks can be run on received communications in order to verify endpoint authenticity, over certificate authenticity. This creates a layer of security over boilerplate TLS, and reduces the risk of attack due to compromised TLS certificates.

Overall, the paper illustrates the role of TLS in modern day communications, the vulnerabilities present in its use, the past attacks made on TLS based communications using cryptographic decryption, and the various tweaks that can be used over "out-of-the-box" TLS to make it more secure against today's exploits.

Our project PSPIT, which will be using TLS to secure its communication between the pseudo clients and servers can be significantly helped by these tweaks mentioned, and we would definitely want to explore these options, post completion of the final multi-system prototype.

3.5 Survey on Covert Storage Channel in Computer Network Protocols

Covert channel in a network is a communication medium that allows the exchange of data between two running processes in such a way that breaks the system's security policies. These mediums are exploited to pass malicious notes, trojans, viruses etc. The ways used to do these can't be easily detected by anti malware systems and firewalls.

These channels are broadly categorized into two major types: storage and timing channels. The former involves encoding secret information into protocol header fields and receiving back the information. Timing channels involve manipulation of message timings, frames and packets.

The survey includes the following studies:

Using the Markov model to propose detection algorithms that discover covert message passing in TCP flows. The issue with this proposal is the inability to detect such stealth communications using tunnelling technology rather than TCP ports such as ICMP.

A hybrid detection system that includes both anomaly detection and stealth channel profiling.

Construction of covert channels based on packet segregation. This method is too hard to be detected by systems such as IDS/IPS. However, these channels can't be eliminated.

Training neural networks for detection of ISN covert channels. The false negative rate varies from 5% to 10% and the false positive rate is below 0.5%.

A traffic padding approach which randomly estimates and limits packet-size. This introduces a dummy packet.

Majority of the proposed methods rely on recognition of abnormal behavior. In reality however, they would fail to detect such traffic with considerable variations. Reducing the capacity of covert channels is seen as one of the most effective solutions in controlling covert channel threats. Like any other penetration loopholes, complete elimination of these channels is impossible. However, the capacity of these channels could be reduced in order to make communication mediums more secure. Future research should pay more attention in developing such methods.

3.6 Network Security and Types of Attacks in Networks

This study provides an overview of all the computer network loopholes and their exploitation in the present scenario.

This paper is relevant to this project because it updates us on the types of attacks that developers of any new protocol need to be aware of to make their protocols immune to such attacks

Development of a secure network needs to handle the following:

Confidentiality: Assurance that a non-authorized system cannot and will not examine the data.

Integrity: Assurance that data received by the system, through the network, is unchanged and unmodified.

Attacks in this study have been classified into passive and active. Passive attacks are those when an intruder in a network intercepts the data. Active attacks are those in which an intruder commands disruption with the networks.

Active attacks include spoofing,modification, wormhole, fabrication, DoS, sinkhole and sybil.

Passive attacks include, traffic analysis, eavesdropping, monitoring.
Other advanced attacks include blackholes, byzantine, rushing, replay, location disclosure.

3.7 Modern Network Security: Issues and Challenges

This paper provides an overview of all the security issues a company faces when transferring data and maintaining secured data, in the present world.

This paper also discussed how to encounter it with respective methods, which are useful for the implementation of our protocol, such as cryptography and firewalls.

Cryptography: this method encrypts and decrypts data by the application of mathematical functions. This is related to the application of SSL and TLS to configure the data security.

Firewalls are of three types which are Application gateway, Packet filtering and Hybrid systems each forming a wall between networks to make them independently secure and inaccessible by anonymous parties.

The application of these methods are achieved by ASIC based appliances, SSL-VPN and Intrusion Detection prevention Systems.

Configuration and update of softwares are also a necessity for protecting data and company networks and mandating employee data risk awareness programs.

This paper mentions the minimum set of parameters that are required for establishment of a secure networking environment using a software development firm as a case study. Security policies should be flexible and not fixed to ensure that the needs of the organization are fulfilled and to tackle future threats while being easy to manage and adopt.

3.8. Design and Implementation Issues of Flexible Network Architecture

This study emphasizes the issues in design and implementation of flexible network architectures with the SOA(Service Oriented Architecture). The aim is to address current network application problems as well the ones that might appear in the future.

Accordingly the changes to be made are of three types and each part of the internet are considered a constructive step, and the three changes identified are

Decoupling of protocols, Flexible Protocol Composition and Service Oriented API.

The protocol family assumes accessibility of at least two routes between the users with end to end communication, considering lag, latency and variance in the network, which keeps up in a changing network environment.

So by using the pseudo hosts the path of data transmission need not be the same every time and masking of the user's identity can prevent the above discussed attacks. The encryption and division of data is also going to make it difficult to trace the message and reassemble it.

3.9 CryptoNET Security Management Protocols

Here many network security protocols have been embodied by the CryptoNET system architecture. The concept of managing protocols are formed on highly established security and privacy technologies and standards.

Individual attributes of the protocols are:

-Effortless incorporation with any applications,

-FIPS 201 (PIV) smart cards are used to handle all security/privacy related issues in the background

These protocols are: remote user authentication protocol, single-sign-on protocol, SAML authorization protocol, and secure sessions protocol.

Single-sign on protocol is utilized for constructing a secure session between two users. The receiver receives a KeyExchange certificate from the sender. Between the two users the KeyExchange certificate helps transport, id and key of the session created between the two.

Upon reception of the certificate request, the client fetches the KeyExchange certificate from a smart card and sends it back to the Secure Application Server.

After receiving the request for the certification, the user gets the KeyExchange certificate from the smart card and is sent to the server.

The protocol makes use of a dedicated container in order to distinguish attackers with fake authentication certificates to differentiate spam and real requests

The key takeaways that can be implemented on PSPIT, are that we can use the various protocols used in cryptoNET in order to implement packet level encryption between pseudo senders and pseudo receivers.

Using dedicated containers for packets which can only be “opened” upon successful single sign on and keyExchange, we create a secure interface between the pseudo machines. This adds a layer of additional security along with packet division and scrambling.

3.10 A Security Architecture for Internet Protocol

IPSEC is an Internet standard protocol from the Internet Engineering Task Force (IETF).

IPSEC protocol defines the semantic and syntax for placing the packet to be transferred inside another packet.IPSEC protocol-specific functionalities are performed on the IP basic packet to protect its integrity and privacy, by using encryption algorithms, then the output is appended to a IPSEC packet header in order to create IPSEC packet; at the end the IPSEC packet is added to IP packet in order to transport it through the internet.

IPSEC protocol works on an unidirectional packet as the packet is made secure in a way that the packet is sent to the specified destination with a key, which can be used to decrypt the packet and nothing other than that can be used.

IPSEC packet has the following details inside in order to securely transfer to destination:

Destination IP address: retriever of packets

Security Parameter index (SPI): the designator of the security association. The SPI should be distinctive to each destination address of the security association in such a way that (destination address, SPI) distinctively identifies a security association. Any IPSEC packet built on the authority of a security association transports the SPI of the security association such that the destination can go through it's packet.

Security protocol: privacy and integrity or both together are provided by security association on the IP packets. Below this, a combination of encryption/decryption algorithms and its criterias are key's size and lifetime.

Encapsulation mode: fragments of the IP packet will be made secure by the security association

Secret keys: key used by the encryption/decryption algorithm.

The primary intention of a key control scheme is to offer two speaking events with a common, recently shared cryptographic key that is forty six CHENG ET AL. In general, a typical key control scheme will attain that in phases: one in which a grasp key is shared among the events, and the alternative in which the already shared grasp key is used for the derivation, sharing, and refreshment of additional consultation keys.

requirements supported by session key protocol : Security handshake, Secrecy and authenticity, Efficiency, Forward secrecy, Simplicity.

3.10 A Comprehensive Survey On SSL/TLS and their Vulnerabilities

SSL has two major layers on which it works and they are session and connection. One of the major layers is Connection, which connects the client and server on the transport layer. P2P conglomerates allow sessions to be built up which is transient. Each SSL session is conglomerated with one Connection.

SSL/ TLS handshaking protocol is used to establish sessions by changing the parameters.

The header section of SSL protocol consists of 4 parts and they are compressed length, content type, major and minor versions. The handshake protocol consists of three parts which are Type ,Length and Content which vary in the size of 0-3 bytes each.

The Alert part of the protocol which is compressed into 2 bytes like the above parts. First byte indicates the level of issue/alert and the second byte indicates the degree of severity of the issue caused during transmission of the packet across the network.

TLS is just an updated version of SSL, as the basic structure is the same but with minor changes with parameters.

The handshake protocol takes place in three steps, as follows:

Key exchanging and encryption/decryption algorithms are applied.

Responses to key exchange by certificate types and it's methods.

And the client certificate is verified and made sure that it is not overlapped with the master certificate.

And in the final step padding takes place inside the packet so that there is no concatenation between the bytes present inside the packet.

4. PROJECT REQUIREMENT SPECIFICATION

4.1 Project Perspective

Most commercially used communication protocols enable only data security. PSPIT goes a step further and implements host security by essentially allowing communication via the internet by never connecting to it.

4.1.1. Project Features

1. Data security
2. Host security
3. Low latency delivery

4.1.2. Operating Environment

The prototype will run primarily on the application layer and will use sockets to simulate inter-host communication.

4.1.3 General Constraints

1. Need to follow IEEE standards for internet communications
2. Need to limit pseudo-system memory to minimum
3. Need to simulate real-time latencies
4. Need to test extensively for security

4.1.4 Risks

1. Security Leaks
2. Latency over the limit

21

4.2 External Interface Requirements

4.2.1 User Interfaces

The prototype will be operated via a terminal, enabling the user to specify configuration details as well as the information to be transmitted.

4.2.2 Hardware Requirements

Standard PC configuration of 2020

4.2.3. Software Requirements

Windows 10, Python 3.6+, “Sockets” module.

4.2.4. Communication Interfaces

Sockets and ports intra to the system.

4.3 Functional Requirements

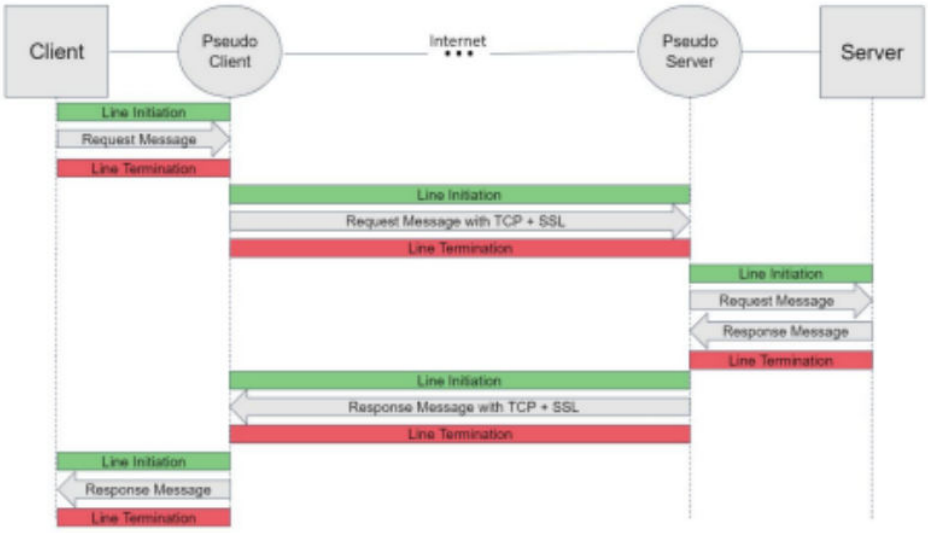


Figure 6. Prototype Skeleton

4.4 Non-Functional Requirements

4.4.1. Performance Requirement

The prototype must ensure reliability, robustness and minimal latency (as comparable with other commercially used protocols)

4.4.2. Security Requirements

The prototype must provide:

1. Message security
2. Host security

5. High Level Design Document

6.1 Current System

At the time of proposal of this protocol, the entire internet operates on the well known, tried and tested OSI protocol stack. This is a 7 layer stack comprising application, presentation, session, transport, network, data link and the physical layers.

The application layer is responsible for making requests to the server (in case of a client-server communication). Application layer protocols like HTTPS are highly data security focussed. By using them, it is ensured that the data being received by a network node is being sent by a certified host. Additionally, this data can only be accessed by the individual system it was meant for by using encryption techniques like SHA, RSA etc.

Transport layer protocols like TCP and UDP provide best effort delivery services to the hosts. In most cases this is adequate. TCP is more appropriate in cases where the data has to be delivered intact and in order while UDP is appropriate for cases where performance is preferred over integrity. The transport layer, however, does not provide any security enforcements.

In the data link layer, network nodes on the local network are connected using ethernet, the LAN technology which has been in use for the longest time. Ethernet makes use of MAC addresses to deliver link layer frames and uses ARP to ensure accurate mapping of IP and MAC addresses.

6.2 Novelty

The above mentioned systems work satisfactorily when it comes to the everyday uses of the internet. The only downside to these existing protocols is that they do not pay attention to the host security aspect of network communications. Data security and host security are the two faces of the same coin. The existing protocols are more oriented towards data security. While travelling across the internet, there is a strong possibility of the data packets being sniffed. Since the addresses of the nodes taking part in a communication session are not encrypted in any way, it becomes extremely easy for an attacker on the internet to identify the presence and location of the host systems and send malicious scripts to the host systems thereby rendering the system insecure.

The system proposed in this study is extremely host security focussed and this is the novelty in the proposed system.

At no point in time during the transmission or receipt of data, are the host systems simultaneously connected to the public internet thereby strengthening each of their integrities.

6.3 Interoperability

The pseudo-system protocol for information transfer is built to operate over the existing transport layer protocols. This system is designed to be an innovation in the application layer and hence deals with data directly. This system does not handle the packet flow/acknowledgements of the transport layer or does not provide assurance of data delivery since these aspects are already looked into by the lower layer protocols. Due to the slight modification of incorporating two additional nodes at either ends of the communication, the interoperability of the system with existing machines is guaranteed.

6.4 Performance

The performance of the system is directly dependent on the underlying OSI protocol stack layers. The proposed system does not add any performance penalties by itself to the communication session between any two systems as the new system is equivalent in performance hits obtained by adding two extra nodes in the communication process. The availability of the system however is marginally deteriorated because of the addition of the uncertainty of availability of the pseudo-systems.

6.5 Security

This system is developed mainly for the purpose of host security. The system achieves the same by ensuring that the host systems are never connected to the public internet at the time of transmission of sensitive data. By doing so, even if an attacker finds out the identity of the pseudo-systems, PSPIT makes it impossible for the attacker to gain direct access to the host systems.

Data security can be enhanced by incorporating existing encryption methodologies like RSA, SHA etc. However, enhancing the data security of the existing transmission methodologies is not the intention of this protocol.

6.6 Reliability

The reliability of the protocol is dependent on the availability and integrity of the pseudo-systems that are present at either end of the communication session.

To ensure that the requests are not continued being sent despite failures of any of the systems, a heartbeat/probe methodology is implemented.

An additional failure detection method is to implement timers in the client side systems to keep track of the delay in the receipt of messages and to interpret node failure.

6.7 Maintainability

Since the protocol has four major systems which are to be managed other than the connectors in between, it depends on if the four are in proper working condition or not.

To handle the system failures, while trying to send or receive data packets, we have come up with a node failure handling technique. The node failure handling technique has two methods first, system check nodes and second dedicated timer in each system.

All pseudo systems are to be looked after as equally as the client and server systems, so they do not falter when in use.

There are two reasons for the QoS of the protocol to be affected. One is when there is a latency in the delivery or receipt of the packets and the other is when the network nodes are offline. The node failure handling mechanisms are made for each of these conditions separately.

6.8 Portability

Portability isn't an issue until the pseudo-server or pseudo-client can be transported with their respective machines. The major reason for development of this protocol is that the user's identity isn't revealed to the outside world, and to make a secure data packet transaction.

After transporting the systems, it is to be made sure that all the connections are to be established as before, so that there is no node failure occurring while transporting data packets.

This implementation is similar to adding another layer to the existing OSI network layer model, and that additional layer is a pseudo-system to mask the user's identity.

6.9 Legacy to Modernization

The protocol builds upon the foundations of TCP + SSL (TLS). So the extent of modernization is limitless, like our implementation of PSPIT protocol.

If the performance measures are favourable, PSPIT could replace TLS in certain use-cases, provided the costs of pseudo-servers and pseudo-clients are justified.

Modernisation of the protocol could also include more robust methods to handle delay of transmission and failure of nodes. Presently, the method proposed to handle such drawbacks are using system check nodes and by using two dedicated timers each on the client and the pseudo-client. As mentioned further on in the report, these methods have their own drawbacks in their inability to differentiate between failure and delay.

6.10 Reusability

The system is entirely open to reuse and can be used as a base for multiple authentication and latency reduction techniques that might be developed later on. Since the underlying protocols used are TCP and other well known LAN technologies, any developments made on these fast changing fields will be applicable to PSPIT as well.

6.11 Application Compatibility

This protocol is designed to operate in the application layer of the OSI model. Hence, it can be used by any network system that uses the request-response model. However, the protocol is designed for host security sensitive applications like bank transactions and military uses. Even though the protocol can be used for everyday networking needs, it would be impractical to require such a high degree of host security for these uses.

6.12 Resource Utilization

The resources used by the protocol are the same as that used by any other existing protocol. An additional requirement is the usage of one extra LAN connection on either side of the communication tunnel. This would require an ethernet connection connecting the client and server to the pseudo-client and the pseudo-server respectively. Additionally, in case of implementation of the node failure handling mechanisms, additional connections would be required by the server check, server and pseudo-server and the client, pseudo-client and the client check. An additional LAN connection would be required between the client and the server check nodes. The bandwidth utilization of these connections would be bare minimum because all these connections would be local.

6. Low Level Design and Implementation

5.1 Proposed Methodology and System Design

5.1.1. Client

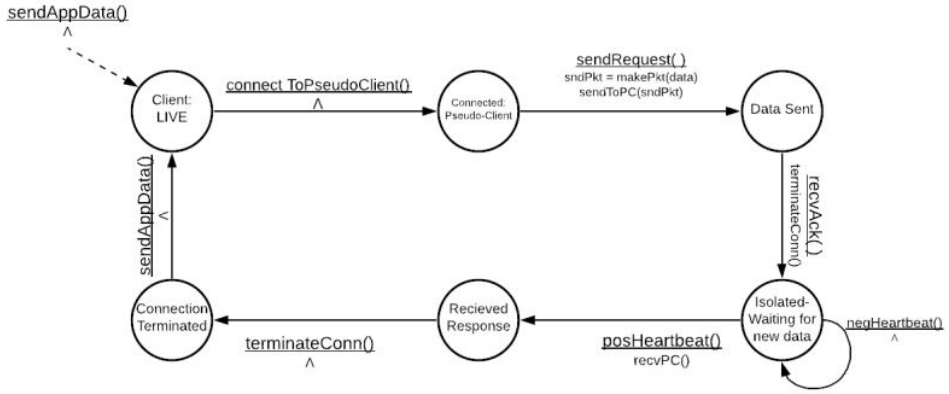


Figure 7. Client

This is the state diagram for the client/host system. There are 6 states including that of the terminal isolated state. The client system triggers the entire protocol when it receives a send command from the upper layer protocol. The first step in this cycle is the establishment of a connection with the pseudo-client. This is done using standard TCP procedures. Once the connection is established(and secured) the client sends the request packet through the data flow link to the pseudo-client. This is again handled by the underlying TCP which takes care of the delayed or unsuccessful transmission. Upon receipt of acknowledgment of successful transmission of the request packet, the client terminates its connection with the pseudo-client. At this point, the client is isolated from the pseudo-client and hence from the internet.

Immediately after isolation of the client, the heartbeat phase of the client takes over. The client sends repetitive beacons in order to check for the availability of response to its request at the pseudo-client. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-client is closed, the client is unable to send the beacons(this connection is later established by the pseudo-client).

Once the heartbeat beacon returns, indicating that the pseudo-client has a new response to send to the client, the connection is re-established with the pseudo-client and the client receives the pending response. After this is done, the client terminates its connection with the pseudo-client and is thus isolated from the network.

5.1.2. Pseudo-Client

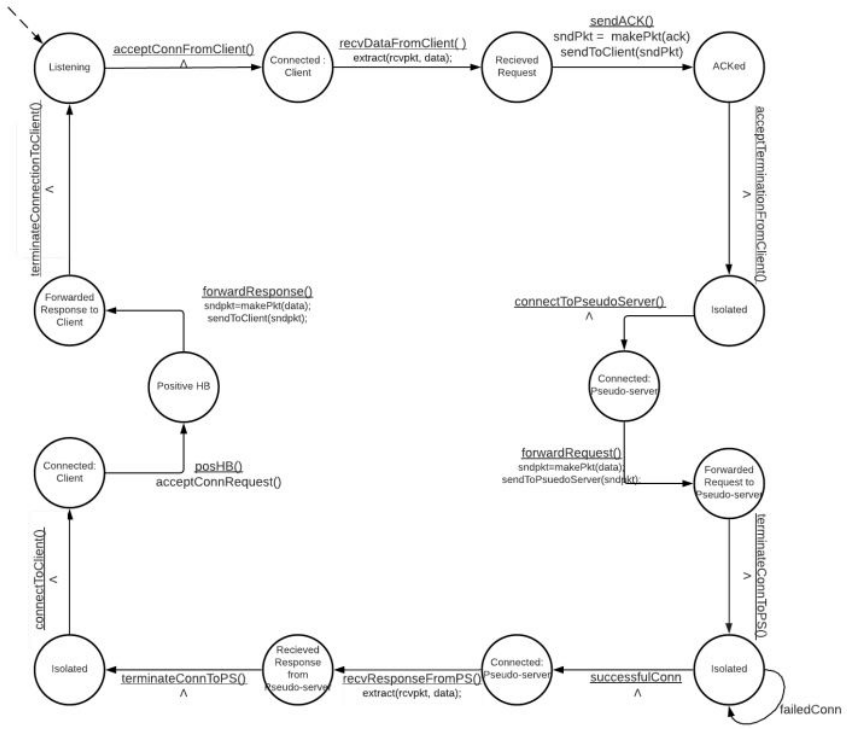


Figure 8. Pseudo-Client

This is the finite state diagram for the pseudo-client system of PSPIT. This system can be in 7 states in 2 directions throughout a request-response cycle. The initial state is achieved when the client system(depicted in the previous FSM) tries to establish a connection with the pseudo-client. The pseudo-client accepts this connection request. Once a connection is established using standard TCP procedures, the pseudo-client receives the request packet from the client. Upon successful receipt of this request, the system sends an ACK message as an acknowledgment. Once the client receives the ACK, the connection is terminated between itself and the pseudo-client. At this point in the protocol, the pseudo-client is isolated from the other systems as there are no active connections to or from itself.

As its next phase, the pseudo-client connects to the pseudo-server and forwards the request packet to the pseudo-server. Upon receiving an acknowledgment of successful transmission of the same, the pseudo-client terminates its connection to the pseudo-server. Immediately after this step, the pseudo-client starts attempting to establish another connection with the pseudo-server. Since the latter is not accepting connections at the moment, all these connection requests fail. When the pseudo-server has new response packets to send, the connection requests

succeed and a new connection for receiving the response packet is established. After receiving the response packet from the pseudo-server, the connection to the pseudo-server is terminated. Note that the transmission of the packet is done using normal TCP procedures.

The next phase is to relay the newly received packet to the client. All this while, the client would have been sending heartbeats probing for the existence of a new response message. Once the pseudo-client is prepared to forward the response to the client, it accepts a connection request from the client and responds to the heartbeats sent by the client. After this, it transmits the response message hence completing its part in the protocol.

5.1.3. Pseudo-Server

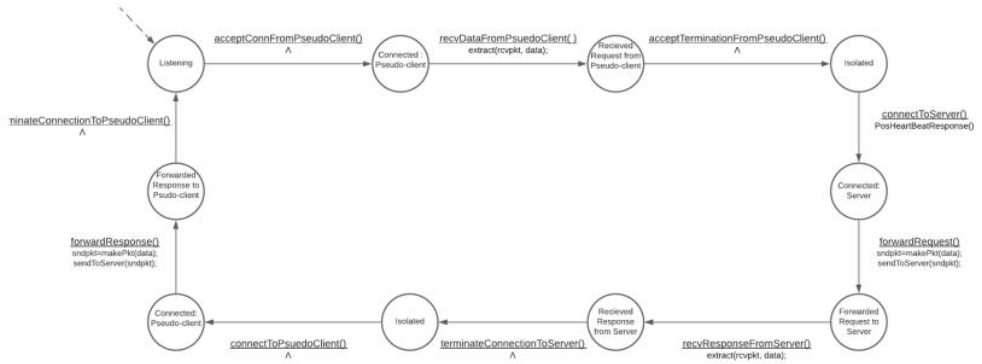


Figure 9. Pseudo-Server

This is the finite state diagram for the pseudo-server system of PSPIT. The initial state for this machine is that it is in a listening and isolated state waiting for a connection request from a pseudo-client. From the previous FSM given for pseudo-client, we can see that once the connection is established between pseudo-client and pseudo-server, the data transmission takes place via TCP protocol procedure. Once the data packet is received, the pseudo-server terminates the connection and goes into an isolated state(no internet connection).

In the next phase, the pseudo-server keeps receiving heartbeats from the server, but only sends a positive response when it has data packets to be transmitted to the server and a connection with the server is established. The request is forwarded to the server using normal TCP procedures and the server responds back to the request. After receiving a response from the server, it terminates the connection with the server leading it to an isolated state.

And in this phase, the isolated pseudo-server acknowledges connection requests from the pseudo-client, as the response has been received from the server, via a normal TCP three-way handshake connection. The response is forwarded to the pseudo-client and the connection with

the pseudo-client is terminated. Leading it back to the first state where it is in an isolated state waiting for the pseudo-client to send the request to establish the connection.

5.1.4. Server

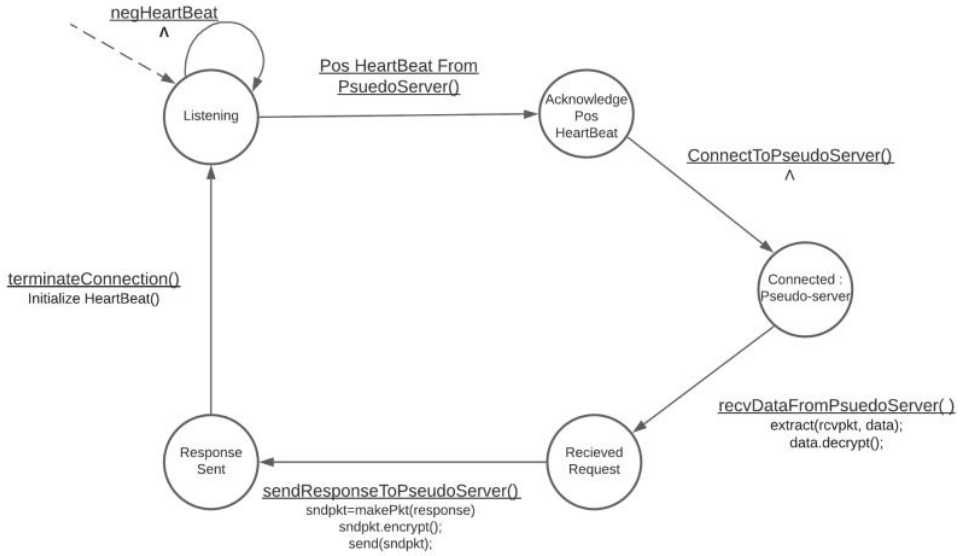


Figure 10. Server

This is the finite state diagram for the server system of PSPIT. This machine has 5 states including the listening/isolated state. This is the final machine to which all the data packets are being sent. In the initial state of this machine, the server is an isolated state, sending heartbeats to the pseudo-server. The server sends repetitive beacons in order to check for the availability of requests at the pseudo-server. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-server is closed, the server is unable to send the beacons(this connection is later established by the pseudo-server).

In the next phase, the pseudo-server sends a positive response to those heartbeats, which are then acknowledged by the server and a connection is established. Request/Data packets are sent from the pseudo-server to the server and the server responds back with the necessary data packets/response requested by the client to the pseudo-server. Then it goes back to the isolated state, wherein it restarts sending heartbeats to the pseudo-server, and the cycle repeats itself.

5.2 Algorithm and Pseudo-code

5.2.1 Client

Sending Request to Pseudo-Client	clientSocket = createSocket() clientSocket.connect(pseudoClient) pkt = createReqPacket() clientSocket.sendReq(pkt) ack = recvAck() clientSocket.close()

Receive Response from Pseudo-Client	//use socket returned from heartbeat module as clientSocket
	serverResponse = clientSocket.recieve()
	clientSocket.close()

Heartbeats	while(true):
	try:
	clientSocket = createSocket(pseudoClient).connect()
	hb = createPacket(heartBeat)
	clientSocket.send(hb)
	hb_reponse = clientSocket.receive()
	if(hb = positive):
	return clientSocket
	except:
	if(clientSocket):
	clientSocket.close()

5.2.2 Pseudo-Client

Receiving Request from Client	fromClientSocket = createSocket(client) fromClientSocket.acceptConnection() request = fromClientSocket.receive() ack = makePacket("ACK") fromClientSocket.send(ack) fromClientSocket.close()

Forwarding to Pseudo-Server	pseudoServerSocket = createSocket(pseudoServer)
	//data received from client
	request = makePacket(data)
	pseudoServerSocket.send(request)
	pseudoServerSocket.close()
	while(true):
	  try:
	  pseudoServerSocket = createSocket(pseudoServer)
	  break
	  except:
	  pass
	  sleep(1) //This can be a moving average value

Receiving Response from Pseudo-Server	response = pseudoServerSocket.receive() pseudoServerSocket.close()

Forwarding Response to Client	respondHeartbeat(client)
	//toClient socket obtained from respondHeartbeat module
	toClientSocket.send(response)
	toClientSocket.close()

respondHeartbeat	toClientSocket = createSocket(client) toClientSocket.connect() hb = recieveHB() toClientSocket.send("NEW") return toClientSocket

5.2.3 Pseudo-Server

Receiving Request from Pseudo-Client	pseudoClientSocket = createSocket(pseudoClient) request = pseudoClientSocket.receive() ack = makePacket("ACK") pseudoClientSocket.send(ack) pseudoClientSocket.close()

Forwarding Request to Server	respondHeartBeat()
	//serverSocket returned by respondHeartBeat module
	request = makePacket(request)
	serverSocket.send(request)

Receive Response to Server	response = serverSocket.receive() serverSocket.close()

Forward Response to Pseudo-Client	pseudoClientSocket = makeSocket(pseudoClient) pseudoClientSocket.acceptConnReq() response = makePacket(response) pseudoClientSocket.send(response) pseudoClientSocket.close()
Respond Heart Beat	serverSocket = createSocket(server) hb = serverSocket.receive() hb_res = makePacket("NEW") serverSocket.send(hb_res) return serverSocket

5.2.4 Server

Receive Request from Pseudo-Server	//pseudoServerSocket received from heartbeat module
	request = pseudoServerSocket.receive()

Respond to Request	response = makePacket(reponse) pseudoServer.send(response) pseudoServer.close()

	while(true):
	try:
	pseudoServerSocket = createSocket(pseudoServer)

Heartbeats	hb = makePacket(hb)
	pseudoServerSocket.send(hb)
	hb_response = pseudoServer.receive()
	if(hb_reponse == "NEW"):
	return pseudoServerSocket
	except:
	if(pseudoServerSocket):
	pseudoServerSocket.close()

5.3 Node Failure Handling

The pseudo-system protocol for information transfer, despite enforcing stringent host security measures, is prone to failures or nodes or delays of request-response messages like any other protocol. After all, the nodes in the system are physical machines running on sources of power with the possibility of being miles apart from each other which can contribute towards packet delays and losses. While delays are already handled by lower layer protocols like TCP, there are a few methods to implement delay detection/handling. Additionally, there are multiple methods to handle system downtimes/failures.

5.3.1 System Check Nodes

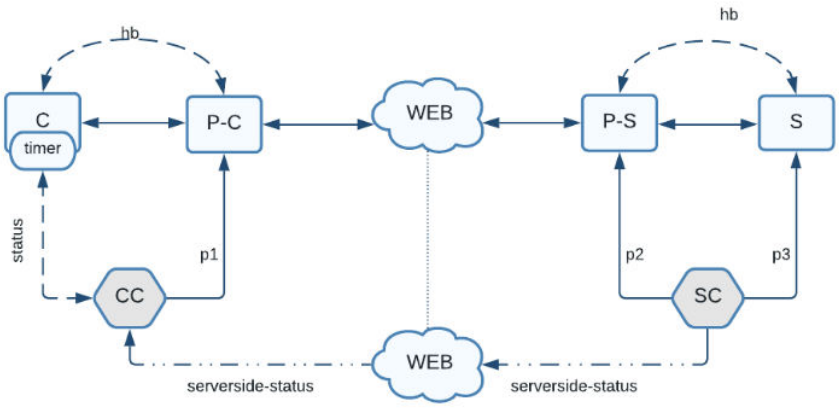


Figure 11. System Check Nodes

Here, two extra nodes(routers) are added to the entire topology. These are the Client Check(CC) on the client side and Server Check(SC) on the server side. These nodes maintain the availability of the entities in the protocol. Server Check handles the availability of the two server side machines,

namely the pseudo-server and the server machine and Client Check handles the availability of the client side node pseudo-client. Additionally, Client Check maintains the status of the server side machines in the form of bit strings. Eg, 101 means that the pseudo-client and the server are active but the pseudo-server system is down.

Client Check sends probe p1 to pseudo-client while Server Check sends probe p2 and p3 to pseudo-server and server, respectively. These probes are sent through a TCP channel which is initially set up when the systems come online for the first time. So, these channels are up and running as long as both sides of these systems are up and running. This means that at any point of time, Server Check is able to send a probe to check the availability of the systems under its supervision. The same also applies to Client Check.

The moment Server Check is unable to send a probe to it's child systems, it implicitly assumes that the system it was trying to probe is down. In the occurrence of such an event, the Server Check sends a SYSTEM_DOWN status message to the Client Check through the world wide web using conventional TCP protocols. The SYSTEM_DOWN message is simply a bitstring showing the availability of the systems under the Server Check node supervision. The next time(given that a SYSTEM_DOWN message has been sent) it is able to send the probes successfully to both the server side systems, it sends a SYSTEM_UP message to the Client Check machine.

Similarly, the Client Check continually probes the liveness of the pseudo-client system by probing it at regular intervals. Upon encountering the failure of the pseudo-client, it internally sets the status bit of the pseudo-client machine to 0. The next time it is able to probe the pseudo-client successfully, it resets the bit for that particular system.

Upon receiving the SYSTEM_DOWN message from Server Check, it changes the status of the respective machines to 0.

These are the internal workings of Server and Client Check machines.

Every request sent by the client starts a timer exclusive to that request. The expiration of the timer could imply the occurrence of two events.

- Delay in receiving the response: In this case, all systems taking part in the protocol are live. However, the excess traffic on the external internet is causing a delay in the transmission of packets in the lower layers. To detect this, the client sends a STATUS_CHECK message to the Client Check. This results in a response with a code of 111 which implies that all the machines in the protocol are alive and functioning. The client then restarts the timer for that particular request. Upon three such cycles of timer + live discovery, the client immediately retransmits the request message.

- Failure of the systems: In this case, one of the systems in the protocol has failed. This is almost always preceded by the receipt of the SYSTEM_DOWN message by the Client Check. Upon expiry of the timer in the client for a particular request, it sends a STATUS_CHECK message to the Client Check. This results in a response with an indication that one of the systems is down. Eg, 101, 001, 100 etc. This is followed by the immediate re-transmission of the request by the client.

Drawbacks: The extra nodes added to the protocol, Client Check and Server Check are also additional power dependent machines implemented to keep track of the status of the machines taking part in the protocol. These are also susceptible to the same probability of failure as that of the protocol systems. If there is a failure of the check machines, the client would have to assume that one of the protocol systems is dysfunctional and wait for it to come online. However, this system does solve the task of identifying the faulty systems.

Additionally, these systems could be compromised and false data could enter the client machine leading to security lapses. A possible solution to this problem is to ensure that the check machines are isolated before the flow of data to the client.

5.3.2 Dedicated Timers In Each System

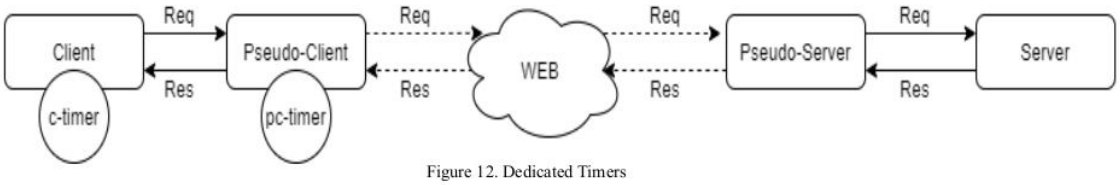


Figure 12. Dedicated Timers

In this proposed solution, each of the client-side machines have a dedicated timer built into each of them. These timers are asynchronous with respect to each other. Additionally, these timers are always running which means that, the moment each of those systems start up, the timer associated with that system is activated. The timers are depicted in the above diagram as follows:

- c-timer: This is the client timer exclusive to the client machine.
- pc-timer: This is the pseudo-client timer exclusive to the pseudo-client machine.

These timers are responsible for only the requests leaving their respective machines and do not keep track of the delay/latency of the requests handled by the other machines.

When the client system sends a request to the pseudo-client, its timer is started. This timer ends only when the response for that particular request is received back at the client. When the timer expires, the client resends the request message and restarts the timer. In case the client receives a delay indicator message from the pseudo-client before the expiry of its own timer, it ends the timer

immediately and resends the request to the pseudo-client. In case, the client is unable to establish a connection with the pseudo-client initially, it implies that the latter is dysfunctional and is unable to receive requests at the moment. However, the problem with using this methodology is that the client attempts to send a heartbeat immediately after sending a request message as well, and the inability to send the heartbeat could be mistaken for the dysfunctionality of the pseudo-client. In this case, the timer of the client is used as a fall back. The moment the timer expires, the client assumes that there is a delay or one of the systems has failed.

When the pseudo-client system sends a request to the pseudo-server, its timer is started; this timer ends only when the response for that particular request is received back at the pseudo-client. When the timer expires, the pseudo-client relays a delay indication message to the client. The client can then resend the request. In case the pseudo-client receives a delayed indicator message from the pseudo-server before its timer expires, it immediately ends its timer and relays this delay message to the client. Again, the client can attempt to resend the message. If the pseudo-client is unable to send the request in the first place, it can be implied that the pseudo-server or the server is disabled and a delay indicator message is sent to the pseudo-client. However, since the pseudo-client utilizes the inability to connect to the pseudo-server as a means to identify if there are new responses at the pseudo-server, this could cause havoc. The solution to this is that, if the request is sent and the pseudo-client is unable to connect to the pseudo-server, it is assumed that the server side machines are performing correctly. In case this assumption turns out to be incorrect, the already running timer for that request eventually expires and the pseudo-client can then send a delay indicator to the client.

The pseudo-server also has a fall back mechanism in case of errors. In case the pseudo-server is unable to send the request to the server, it can send a delay indicator to the pseudo-client which is relayed upstream to the client which can then resend the request once the server is online.

Drawbacks: The major drawback of this solution is that it is not easy to decide if the timeout is due to the delay in transmission of the messages or due to the failure of the machines.

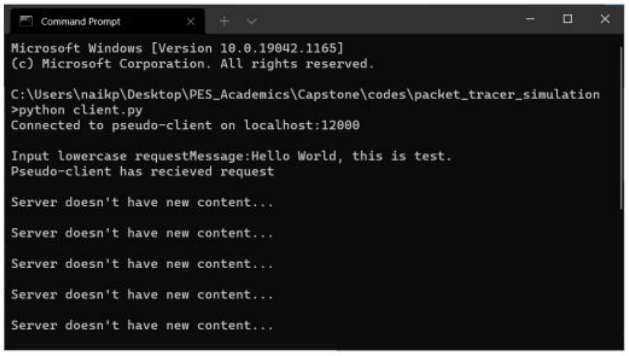
7. RESULTS AND DISCUSSIONS

7.1 Prototype in Action

The functioning of the protocol is illustrated using 4 terminals to simulate the 4 systems in the protocol: client, pseudo-client, pseudo-server and server in the local system prototype.

The prototype is initiated by running in the following order.

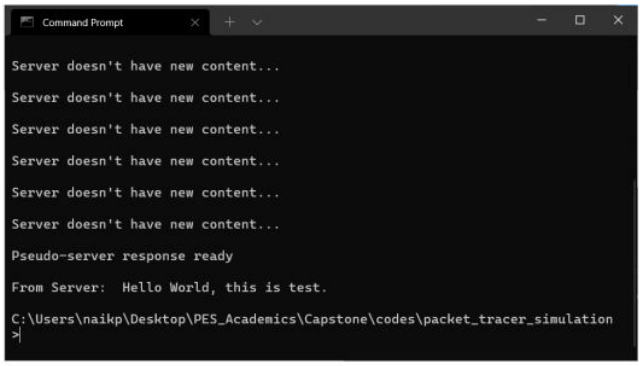
- Server
- Pseudo-server
- Pseudo-client
- Client



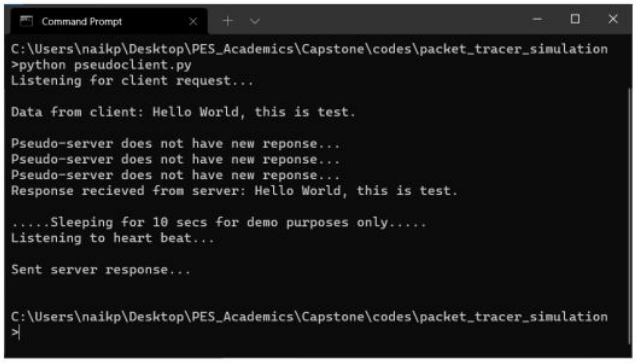
The above screenshot belongs to the simulated client terminal.

This is the result obtained after executing the client protocol script. In the present prototype, the client requests for input from the user. Immediately after this phase is done, the client starts probing the pseudo-client for the presence of a new response from the server.

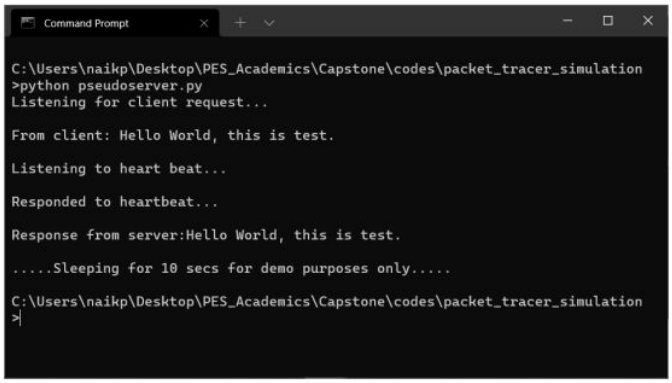
The consecutive “Server doesn’t have new content” messages are the outputs that indicate that the pseudo-client does not have new responses. This is done by attempting to send a packet through an open socket on the client.



As long as the pseudo-client has no response to provide to the client, it continues to probe the pseudo-client. The moment the pseudo-client has a new response, it responds to the probe. This step is indicated by the "Pseudo-server response ready" message being displayed on the client terminal. Once this is done, the pseudo-client sends the server response to the client. This is indicated by the message "From Server: Hello, World, this is test".



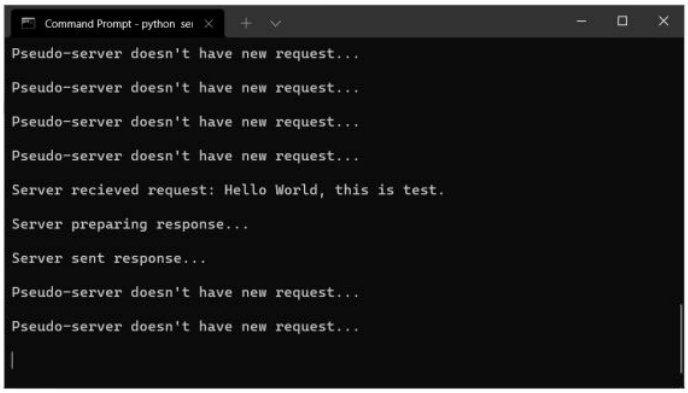
This is the pseudo-client screenshot. Initially, it receives the request from the client. This is indicated by "Data from client: Hello World, this is test". Immediately after that, it closes the client port, forwards the request to the pseudo-server, closes the pseudo-server port and tries to re-establish the connection with the pseudo-server. The re-establishment is indicated by "Pseudo-server does not have new response". Once the connection has been re-established, the pseudo-client receives the response and forwards it to the client. This is indicated by "Response received from server: Hello World, this is test" and "Sent server response"



This terminal initiates the pseudo-server python file. After the initiation, it starts listening for requests from pseudo-client. "From client: Hello World, this is test." This is the input provided by the user at the client end.

“Listening to heartbeat. . .” is the step wherein the server keeps sending heartbeats to the pseudo-server, to check if there are any requests for itself. “Responded to heartbeat. . .” is the pseudo-server responding to the server with a positive message.

“Response from server: Hello World , this is test.” is the final response from the server back to the client.



This terminal corresponds to the server wherein the server keeps sending heartbeats to the pseudo-server, as a result, we can see the following message “Pseudo-server doesn’t have new request. . .”. After this step, the pseudo-server connects to the server when a message is received by it.

The next step is when the message is received, it prepares a response and sends it back to the client, as we can see the following steps are also displayed in the terminal. The next step is for the server to disconnect itself from the pseudo-server and start transmitting heartbeats through the UDP protocol.

In the multi-system implementation, four individual systems have been used, instead of one single simulation system, and packets will flow through the use of real-world physical media such as the internet (gateway routers and transmission links) and local area connections (MAC-Address communication).

The systems and their respective pseudo-systems will be connected to a Local Area Network (LAN), through the use of ethernet cables and RJ-45 connectors, to create a secure closed circuit communication medium between the two. Once packets have been sent from the pseudo-machine or vice versa, the local area connection is closed, and hence the main system is never connected to the internet.

The pseudo-system then establishes a connection to the internet and sends the packet to the destination pseudo-system via the internet and the ISP’s router network. Similarly, on the receiving

end, the pseudo-system receives the packets, shuts itself off from the internet, and connects to the destination system via LAN, in the same way as the sender and the pseudo-sender communicated.

The protocol also sees the use of “heartbeats”, which is a term used for when a machine, unprompted, checks for a status change. If no status change is found, then the machine stays in the periodic check state, but if a change is detected, then a series of actions is carried out.

In order to ensure complete host isolation, we could not store the host’s communication details on the pseudo-machine, as it connects to the internet, and if any malicious attacker is able to get access to the pseudo-machine, they could easily start communications with the main system, posing as a pseudo-system.

To prevent this, we made the use of heartbeats in our protocol. Since the communication details are not stored on the pseudo-system, the main system must initiate the communication all on its own. Since the pseudo-system is connected to the internet to check for incoming messages, it cannot be constantly connected to the main system, and hence periodically opens the LAN port to check for any queued messages from the main system, therefore the term “heartbeats”.

The multi system implementation of the protocol utilizes a VPN service for demonstration purposes. Sans VPN, the traffic would be required to be routed through the public internet to the nodes that are concealed behind a NAT router. This made it inconvenient to demonstrate the accurate working of the protocol as the NAT router configurations of the commercial routers were difficult to edit. The routers being used by the developers were made by Huawei, D-Link, TP-Link and Dragon. The router firmwares had sophisticated configurations which could not be changed easily. Using a VPN service eased the process of communications while simulating the behaviour of transport of packets across the public internet.

In the multi-system implementation two physical systems were used to simulate the client and the server sides respectively. The first system ran the client and the pseudo-client scripts, while the second system ran the pseudo-server and the server scripts. The two processes on each of the machines communicated using local connections as the protocol requires them to be in a local network. The VPN tunnel between the two systems simulated the public internet through which the protocol requires the pseudo-client and the pseudo-server systems to be connected. The results of this set up were similar to the local system implementation of the protocol which indicated a successful request and response cycle.

8. CONCLUSION AND FUTURE WORK

In a world where the value of data rises exponentially with every passing moment, its security becomes of paramount importance. Hackers and data breaches are taking new faces and finding ingenious ways to breach even the most secure protocols. In such a situation, it feels as though systems are better off not even connecting themselves to the internet. Although infeasible, PSPIT finds the best of both worlds, the power of the internet, as well as the safety of staying off the grid. This is what we call Host Isolation. The system with sensitive data sits off the grid, whereas a pseudo-system does the heavy lifting of data download/upload, and establishment of encrypted channels over the internet. This way, PSPIT has a lot of real-world applications where sensitive data is to be secured over a primary system, such as defense, finance, medicine, etc.

Pushing PSPIT over the well-accepted standards of TCP and TLS shows a strong trust factor on the protocol, and can even be made an optional security recommendation for high-security interconnected systems. Collaboration with international certification organizations such as IETF is critical in the future work of the protocol, and we intend on getting the protocol certified as a trusted security service framework.

The future work for the protocol involves working on further safeguards localized in the pseudo-systems which includes, but is not limited to

- Partial execution of code
- Anti-malware screening of payload
- Stability testing of payload
- Encrypted sandbox testing

These additional features ensure that, alongside preventing malicious attackers from being able to discover the primary system, the data itself does not pose a risk to the primary system. This allows PSPIT to be an all-in-one toolkit which can be used to effectively create a blanket solution for all system security needs

This would require extensive research, further to the research we have done to develop the protocols, as this forays into the domain of the operating system. Implementing OS safeguards over the network level safeguards would be our final stage of completion, post publishment of this research paper, and post IETF certification of PSPIT as a viable security standard.

Furthermore, these multi-system communications will be encrypted with the use of TLS, which is Transport Layer Security. TLS is a secure communication encryption standard used worldwide for encrypting internet communications from unintended and/or malicious attackers. TLS uses keys in the form of certificates, cipher and decipher messages, and the certificates are confidentially maintained by each pseudo-system. This TLS security layer acts on top of TCP, with the systems sending their certificates to each other via the protocol, following which messages are encrypted with the destination machine's public key, and decrypted with its private key.

We will be researching the best configurations of TCP-TLS that can be used as part of our protocol, which was a part of our literature survey as well, to find the most secure and convenient settings package for our use, as well as which configuration works most efficiently on the largest number of systems.

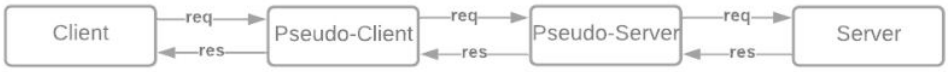
The protocol shows great promise for real-world applications, due to its closeness to the issues that real-world systems face, and the simple solution of enforcing host isolation can mitigate a large number of security volatilities. We picture PSPIT to be one among the industry standards in host security - a domain yet unexplored.

APPENDIX A: DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1. TCP: Transmission Control Protocol
2. UDP: User Datagram Protocol
3. IP: Internet Protocol
4. TLS: Transport Layer Security
5. SSL: Secure Socket Layer
6. PSPIT: Pseudo-System Protocol for Information Transfer
7. Client: The node in the network initiating a PSPIT connection.
8. Server: The node in the network responding to the PSPIT client request on the opposite end of the client.
9. Pseudo-systems: The intermediate nodes in the path from the client to the server acting which is practical, router/switches.
10. Pseudo-client: The node which immediately receives the request from the client and forwards the same to the next node, called the pseudo-server. This also relays the response from the pseudo-server to the client.
11. Pseudo-server: The node that receives the request from the Pseudo-client and forwards the same to the server. On receipt of a response from the server, it relays it to the pseudo-client.
12. Heartbeat(s): Repetitive beacons/probes transmitted by the client and the server to check for availability of new data in pseudo-client and pseudo-server respectively.

APPENDIX B: USER MANUAL

1. Download the prototypal codes for client, pseudo-client, pseudo-server and server.
2. Start 4 terminals / command prompts, one to mimic each system.
3. Start the prototype by first instantiating the server system(in this case, the command prompt) by running the server.py file
4. Start the pseudo-server by executing the pseudoServer.py file.
5. Start the pseudo-client by executing the pseudoClient.py file.
6. Start the client by executing the stdclient.py file.
7. This order of executing the files is important due to the required direction of data flow, which is:



8. The prototype can be verified by the success messages displayed on the server and the client which indicate that data flows between server and client and vice-versa have been successful.

Pseudo-System Protocol for Information Transfer

ORIGINALITY REPORT

8%	6%	7%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ccs.neu.edu Internet Source	2%
2	www.ics.uci.edu Internet Source	2%
3	Michael Atighetchi, Nathaniel Soule, Partha Pal, Joseph Loyall, Asher Sinclair, Robert Grant. "Safe configuration of TLS connections", 2013 IEEE Conference on Communications and Network Security (CNS), 2013 Publication	2%
4	docplayer.net Internet Source	1%
5	James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, Christo Wilson. "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers", 2017 IEEE Symposium on Security and Privacy (SP), 2017 Publication	<1%
6	mafiadoc.com Internet Source	<1%

7	Submitted to University of Bristol Student Paper	<1 %
8	tools.ietf.org Internet Source	<1 %
9	Gene Tsudik. "Improving secure server performance by re-balancing SSL/TLS handshakes", Proceedings of the 2006 ACM Symposium on Information computer and communications security - ASIACCS 06 ASIACCS 06, 2006 Publication	<1 %
10	Submitted to Colorado Technical University Online Student Paper	<1 %
11	Abba Garba, Arne Bochem, Benjamin Leiding. "Chapter 18 BlockVoke – Fast, Blockchain-Based Certificate Revocation for PKIs and the Web of Trust", Springer Science and Business Media LLC, 2020 Publication	<1 %
12	Hengbo Li, Guang Zhao. "Improving Secure Server Performance by EAMRSA SSL Handshakes", 2012 International Conference on Industrial Control and Electronics Engineering, 2012 Publication	<1 %
	kth.diva-portal.org	

13	Internet Source	<1%
14	www.govbids.com Internet Source	<1%
15	www.onlinenetworkssolution.com Internet Source	<1%
16	F. Rodriguez-Henriquez, C.E. Lopez-Peza, M.A. Leon-Chavez. "Comparative performance analysis of public-key cryptographic operations in the WTLS handshake protocol", (ICEEE). 1st International Conference on Electrical and Electronics Engineering, 2004., 2004 Publication	<1%
17	Ilsun You. "Chapter 100 A One-Time Password Authentication Scheme for Secure Remote Access in Intelligent Home Networks", Springer Science and Business Media LLC, 2006 Publication	<1%
18	dokumen.pub Internet Source	<1%
19	Fang Qi, Zhe Tang, Guojun Wang, Jie Wu. "User requirements-aware security ranking in SSL protocol", The Journal of Supercomputing, 2011 Publication	<1%

20	Claude Castelluccia, Einar Mykletun, Gene Tsudik. "Improving secure server performance by re-balancing SSL/TLS handshakes", Proceedings of the 2006 ACM Symposium on Information, computer and communications security - ASIACCS '06, 2006 Publication	<1%
21	www.ijariit.com Internet Source	<1%

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		

Pseudo-System Protocol for Information Transfer

Akshay Vasudeva Rao
Dept. of CSE
PES University
Bangalore, India
akshayvasudev007@gmail.com

Amogh R K
Dept. of CSE
PES University
Bangalore, India
amoghrameshk@gmail.com

Prajwal K Naik
Dept. of CSE
PES University
Bangalore, India
naikprajwal40@gmail.com

Rohan Iyengar
Dept. of CSE
PES University
Bangalore, India
rohaniyengar20@gmail.com

Prof. Pushpa G
Asst. Professor
Dept. of CSE
PES University
Bangalore, India
pushpag@pes.edu

Prof. Prasad H B
Professor
Dept. of CSE
PES University
Bangalore, India
prasadhb@pes.edu

***Abstract*—The Pseudo-System Protocol for Information Transfer uses pseudo-clients and pseudo-servers to enable hosts to communicate through the internet while being completely isolated (offline) from the internet. It enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds.**

Keywords—Pseudo-System, Host, Transmission Control Protocol, Host Security, Heartbeats, Sockets

I. Introduction

Communication protocols have always looked at data security and host security as unrelated goals. Protocols like TLS have been successful in ensuring data security. The goal of PSPIT is to take a unified approach to data and host security by implementing pseudo-systems that help hosts isolate.

PSPIT aims to ensure host and data security while limiting latency to a minimum with the additional constraint of making the implemented pseudo-systems as economic as possible (by limiting buffer size and computational ability to the bare minimum).

Most commercially used communication protocols enable only data security. PSPIT goes a step further and implements host security by essentially allowing communication via the internet by never connecting to it.

This prototype enables hosts to use the power of the internet while making the most of the safety of remaining offline and isolated, providing users with the best of both worlds.

This protocol secures the data being transferred and masks the host so that a third party cannot intervene for stealing data or gaining access to the hosts. The protocol sequences the transfer of data such that the hosts always remain disconnected from the internet while using the internet as the primary medium for the transfer, thus making full use of the power of the internet but without any of the risks of intrusion that come with it.

Today's protocols do not contain an inclusion for end system isolation, and PSPIT allows for information to be sent and received without the end systems ever being directly connected to the internet, which significantly reduces the risk of attack.

II. Novelty

At the time of proposal of this protocol, the entire internet operates on the well known, tried and tested OSI protocol stack. This is a 7 layer stack comprising application, presentation, session, transport, network, data link and the physical layers.

The application layer is responsible for making requests to the server (in case of a client-server communication). Application layer protocols like HTTPS are highly data security focussed. By using them, it is ensured that the data being received by a network node is being sent by a certified host. Additionally, this data can only be accessed by the individual system it was meant for by using encryption techniques like SHA, RSA etc.

Transport layer protocols like TCP and UDP provide best effort delivery services to the hosts. In most cases this is adequate. TCP is more appropriate in cases where the data has to be delivered intact and in order while UDP is appropriate for cases where performance is preferred over integrity. The transport layer, however, does not provide any security enforcements.

In the data link layer, network nodes on the local network are connected using ethernet, the LAN technology which has been in use for the longest time. Ethernet makes use of MAC addresses to deliver link layer frames and uses ARP to ensure accurate mapping of IP and MAC addresses.

The above mentioned systems work satisfactorily when it comes to the everyday uses of the internet. The only downside to these existing protocols is that they do not pay attention to the host security aspect of network communications. Data security and host security are the two faces of the same coin. The existing protocols are more oriented towards data security. While travelling across the internet, there is a strong possibility of the data packets being sniffed. Since the addresses of the nodes taking part in a communication session are not encrypted in any way, it becomes extremely easy for an attacker on the internet to identify the presence and location of the host systems and send malicious scripts to the host systems thereby rendering the system insecure.

The system proposed in this study is extremely host security focussed and this is the novelty in the proposed system. At no point in time during the transmission or receipt of data, are the host systems simultaneously connected to the public internet thereby strengthening each of their integrities.

III. Review of Literature

A. *Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes*

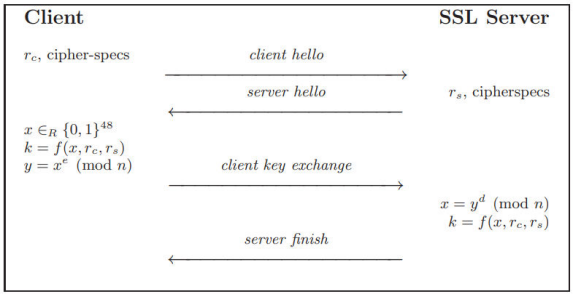
Majority of today's distributed computing uses the client-server model. Servers handle multiple requests from clients. Since the modern day networks work on SSL, a considerable amount of server capacity goes on performing computationally expensive RSA Public Key decryptions as part of each SSL handshake. This step can be very detrimental and time consuming to the performance of the server.

This paper explores the existence of a technique for re-configuring RSA-based handshakes. As per the results obtained by the authors, the new method speeds up the processing of RSA operations 11-19 times depending on the size of the key.

There have been fast advances in trying to ensure services degrade gracefully and many of these methods have been implemented in real world servicing as well. Additionally, many studies in networking have yielded low latency results speeding up the communication between the client and the server. However none of these results thwart DoS attacks as their primary goal is to create a server overload and deny the servicing requests from legitimate clients.

In the SSL connections used these days, RSA decryption occurs on the server side when it decrypts the secret to receive the base message. This technique is called Server-Aided RSA (SA-RSA). This paper investigates this technique and tries to change the way the client and server sides handle encryption and decryption. The proposed method simply reassigns the roles of the client and the server. The SSL client becomes the "server" and the SSL server suffering from overload becomes the "weak client". This results in a Client-Aided RSA which has proven to speed up the decryption process.

The paper begins with a description of the SSL handshake.



The handshake starts with the delivery of a "hello" message from the client to the server. This is an indication of the willingness of the client to start a TLS/SSL session. This message contains the cypher suits and a random, one time number rc.

The server response includes a "hello" message from the server-side and includes a public-key certificate and another random one time number rs. The server additionally mentions it's choice of cipher suite.

A pre-master secret x is chosen and the shared master secret k is obtained using f(x, rc, rs) where f is the hash function. x is then encrypted with the RSA public key of the server. Additionally, the ciphertext is attached to a "client key exchange" message. Finally, this is sent to the server.

Finally, the server uses its private key to decrypt the pre-master secret and in turn uses it to obtain the master secret as f(x, rc, rs). The handshake is concluded by sending a "server finished" message.

The most important step of the entire handshake process is the encryption of the key under the RSA public key (of the server) by the client. The computationally extensive problem is the RSA decryption.

Although RSA is very well-studied and tested, many of the RSA implementations are computationally imbalanced. When public exponents like 3, 17 and 216 + 1 are used, the RSA decryption is expensive while the corresponding encryption is comparatively cheap, even though the exponents are small. A possible solution to solve this imbalance is to select a small d (private exponent) which in turn speeds up computation. However, this compromise leads to an insecure RSA.

The authors of this paper have tried to flip around the SA-RSA technique and obtained the Client-Aided RSA. The main objective was to transfer a portion of computation from the server to the client. Through this, the clients perform the majority of tasks in decryption, letting the server dedicate more resources to handling the request rather than contributing to cryptography.

The algorithm starts of by the representation of the exponent, private to the server, as d = f1d1 + f2d2 + ... +fndn (mod ϕ(n)). Here, the di's and the fi's are vector elements which are random and of length |n| and c bits, respectively. Since the RSA decryption process involves computing xd (mod n) on the server side, the following steps are performed by it.

The server sends the vector D = (d1, d2, …, dk) to the client.

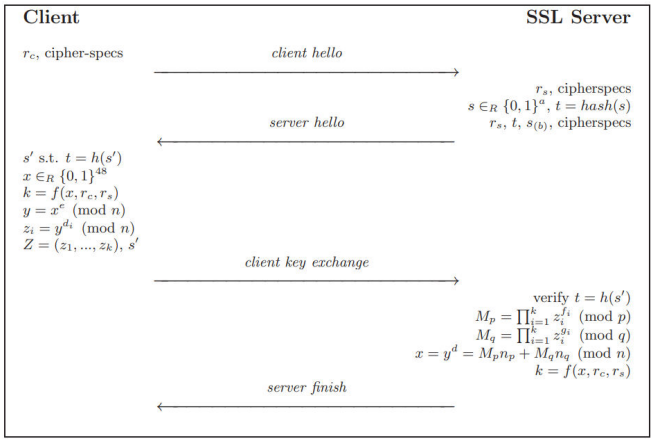
The vector Z = (z1, z2, …, zk), where zi = xdi (mod n), and sends it to the server.

As a final step, the server computes xd using the vector Z received from the server.

The algorithm assumes that breaking RSA is difficult. Using these assumptions, selecting the parameters of the algorithm such as k, c, and f does not introduce any compromises in security. To guess the values of c and k, any attacker would have to minimally use brute force which would have a complexity of 2c*k. This is the same as the time complexity required to crack RSA. Additionally, this

basic description of the algorithm can be improved performance-wise by taking advantage of the Chinese Remainder Theorem.

The next section of the study includes a method to incorporate the CA-RSA method into the existing SSL handshake. The initial "hello" messages by the client and server remain mostly unchanged except the certificate of the server includes the D vector (D = (d1, d2, ...dk)). After the client encrypts using the server's public exponent, it constructs Z by computing the zi = xdi (mod n). This is then included in the client key exchange message. After receiving this, the server performs its CRT computations. The modified protocol is as follows:



The major concerns regarding the integrity of the systems are on choosing the parameters, c and k. The requirements needed to be met are such that the time complexity of exhausting the search space to guess the values of c and k are at least equivalent to breaking the original RSA cryptosystem.

The authors have measured the improvement in the execution times when the protocol was changed from SA-RSA to CA-RSA. The key attribute used for computation was the number of RSA decryptions performed by the test system in a specified time frame. The test machine used had the following specifications:

Processor: Intel Celeron 1.7 GHz

RAM Capacity: 256 MB

OS: Red Hat 9.0 Linux

The results of the experiment were as follows:

Key size	RSA	CA-RSA	$c \times k$	Improvement
1024	7.05	0.62	72	11.33
1536	19.79	1.25	80	15.76
2048	44.22	2.31	88	19.12

These results imply that 11 times as many decryptions can be performed by a server with a 1024 bit RSA key. The smallest value of k is 2, as specified by the guidelines for selecting parameters. Therefore, the values of c and k have been chosen as 2-36, 2-40 and 2-44.

CA-RSA introduces certain additional bandwidth and computational costs at the clients, despite reducing computational burden on the servers to a very large extent. As computed by the authors, this protocol adds an additional computation time of 21.9, 66 and 150.9 seconds while computing Z for 1024, 1536 and 2048 bits on the client-side. However, these computational costs seem to be negligible since the entire protocol is aimed at balancing the degree of computation between the server and the client. Additionally, the client key exchange message also includes certain bandwidth costs incurred while using the protocol since it now has to carry a vector Z. However, with the value of k set to 2, the over is calculated as |n| where n is the length of the RSA modulus. However, this overhead takes less than one ethernet frame to accommodate. However, in case the system being used is very sensitive to bandwidth and computational changes, the protocol can be used only in case of overload of servers. When this technique is incorporated, the overheads occur occasionally.

So far, the authors have focussed on rebalancing the computations between the servers and the client. The protocol assists in enabling the server to increase the quantity of SSL connections that are handled. This alone makes the task of a DoS attacker much harder. However, any attacker could simply overload a CA-RSA server by increasing their computational resources. The authors have included a mechanism to make this task more difficult. An attacker willing to carry out a DoS attack on a server only needs to bombard the server with as many requests as the number of RSA decryptions the server can perform. A possible fix to this would be to direct the client to solve a set of "client puzzles" before sending the encrypted message. This when combined with the lesser decryption time provided by CA-RSA would prove effective in thwarting DoS attacks on SSL servers.

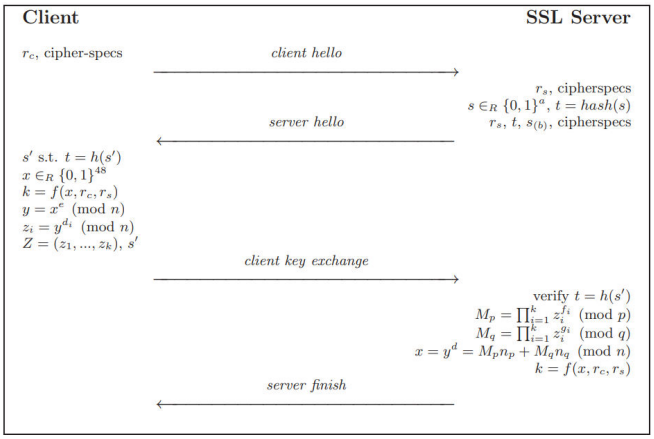
The client puzzle aspect of the proposed solution for DoS attack protection can be implemented by using the method proposed by Juels and Brainard. This method requires a client to solve a computational puzzle before requiring the server to carry out the decryption. This ensures sufficient time between consequent service requests. Following the client's hello message, the server selects a random a-bit value s and passes it via a cryptographic hash function. It then sends the server greeting message to the client with the hash digest t = hash(s) and the b first bits of s (where b a). The client uses these b bits to brute-force to solve the "client puzzle" and discover a value s' that hashes to the desired t. After that, the client adds s' to its client key exchange message. The server will only proceed with the SSL handshake and decrypt the encrypted session key given by the client if s' verifies - that is, if it is the correct length and its hash output is t.

The protocol can be modified to incorporate this by adding the puzzle solution to the frame that contains the Z vector.

When compared to an RSA decryption, the computing cost of a hash computation is almost small, therefore the addition of a puzzle verification step adds just a minor server burden. The amount of labor required by the client to answer the problem is determined by its computer resources

and, more significantly, the number of unknown bits in the server's preimage value.

The modified protocol is as follows:



Even though there is a possibility that a malicious system may solve the puzzle and send fake Z vectors to the server, the possibility of this happening is 11 times less than when using SA-RSA.

In conclusion, the system proposed by the authors helps distribute the server load to the client and thus helps the server authenticate more requests with a very low possibility of overloading.

B. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers

In this paper, they present CRLite(Certificate Revocation List Lite), an efficient and easily-deployable system for proactively pushing all TLS certificate revocations to browsers.

CRLite servers aggregate revocation information for all known, valid TLS certificates on the web, and store them in a space-efficient filter cascade data structure. Browsers periodically download and use this data to check for revocations of observed certificates in real time. CRLite does not require any additional trust beyond the existing PKI(Public Key Infrastructure), and it allows clients to adopt a fail-closed security posture even in the face of network errors or attacks that make revocation information temporarily unavailable.

Comparing CRLite to an idealized browser that performs the correct CRL(Certificate Revocation List)/OCSP(Online Certificate Status Protocol) checking, they show that CRLite reduces latency and eliminates privacy concerns.

Chrome and Firefox only make CRL and check via OCSP and requests for EV(Extra Validation), and not all mobile browsers check for revocation. And this leads to several problems while secure browsing.

CRLite is implemented in two parts: a server-side system that aggregates revocation information for all known, valid TLS certificates on the web and places them in a filter,and a client-side component that downloads filters and uses them to check for revocations of observed certificates. And it deals with majorly six challenges, which are efficiency, timeliness, failure, privacy, auditability, deployability.

Background:

A. The TLS Ecosystem

Certificate Validation, Certification Transparency, Measuring TLS Ecosystem

B. TLS Certificate Revocation

CRL Certificate Revocation List, OCSP Online Certificate Status Protocol

C. Revocation Checking

Revocation Checking in Practice, Fail-open vs. Fail-closed, CRLSet and OneCRL

D. Other Revocation Distribution Schemes

Micali's Certificate Revocation System, multi-certificate revocation, revocation trees, or combinations of these techniques.

E. Proposals to Replace PKI

AKI, PoliCert, ARPKI and PKISN aim to replace the existing PKI with a new hierarchy that avoids centralizing trust, is transparent, and supports seamless revocation.

At a high level, this method aggregates all revocation information for every known certificate, compactly represents them in a filter cascade, and provides a means by which clients can publicly audit them.

The server side of the application works in this order:

Obtaining Raw Certificates ⟶ Validating Certificates ⟶ Obtaining All Revocations ⟶ Filter Cascade Construction ⟶ Delta Updates ⟶ Audit Logs ⟶ Hosting

1.Obtaining Raw Certificates- To create the filter of revoked certificates, CRLite first needs a list of all valid certificates.

2.Validating Certificates- CRLite validates all certificates by looking for non-expired, well-formed leaf and intermediate certificates that cryptographically chain to a trusted root.

3.Obtaining All Revocations- To collect revocations, CRLite extracts CRL and OCSP responders from all valid certificates.

4.Filter Cascade Construction- CRLite uses Bloom Filters for storing and retrieving information as the rate/probability of finding the required certificates is more efficient, has higher probability and is protected.

5.Delta Updates- CRLite produces delta updates that allow clients to incrementally update their copy of the filter(a small percentage of certificates change on a daily basis).

6.Audit Logs-The audit log contains (1) copies of all CRLs and OCSP responses that were used to construct the corresponding filter cascade, and (2) copies of all certificates included in the whitelist

7.Hosting- This is the display of all delta updates and audit logs through web servers hosted on cloud storage.

Security Analysis:

Man-in-the-middle, forcing fail open, DoS, BackDated Certificate and Rogue Aggregator are taken care of by CRLite.

This paper helps one to keep track of the revocation of the TLS certificates, by adding an extension to the browser and they're really helpful for private organisation, who use certificates, as the list generated keeps updating every 24 hours and let's one know if it's safe to use/continue with the current in use certificate and the list is of reasonable size due to the usage of bloom filters and cascading filters.

C. Safe Configuration of TLS Connections Beyond Default Settings

This paper, authored by Michael Atigetchi, Nathaniel Soule, Partha Pal and Joseph Loyall, focuses on TLS (Transport Layer Security), its precursor SSL (Secure Sockets Layer), drawbacks of "out-of-the-box TLS configurations" and optimal settings to minimize the risk of TLS-compromise.

Since the deployment of SSL in 1995 and the subsequent release of TLS in 1999, the TLS-SSL standard has been the frontline for communication encryption, by countering attackers who try to sniff live traffic or even man-in-the-middle attacks, that try to compromise the confidentiality of information being transmitted. Despite the wide-scale deployment of TLS 1.3 (usage: 2014-current), it remains vulnerable to a number of threats at the protocol layer and does not provide robust security when used "out-of-the-box", hence requiring tweaks to its configuration in order to attain the expected security benefits. The paper goes over these tweaks and elucidates the subsequent security enhancements.

To begin with, TLS is a security standard that makes use of certificates that have been issued by standardized bodies, called certificate authorities (CAs). These certificates serve the purpose of identity cards, that allow a page or a server to verify their identity with someone trying to communicate with them. The client, upon receiving the certificate from the server, checks the certificate for authenticity, and upon verification of the same, initiates the second step of the handshake by sending a copy of its own certificate. In this way, both parties have full awareness and full confidence in who they are communicating with.

But the role of the public key does not end there, as the client then encrypts the session using public key cryptography i.e., it uses the server's public key to encrypt the data. This encrypted data, upon reaching the server, can only be decrypted using the server's private key, which is stored confidentially on the server's end. This way, using public-key cryptography (aka symmetric cryptography), the data is encrypted and can only be decrypted by the server.

Beyond the working of TLS, the paper discusses additional means of protecting data encrypted by TLS/SSL, first of which is called the Crumple Zone.

Crumple zone is a safety technique used by cars to minimize the damage to core components and passengers inside the automobile in the event of a crash. It consists of reinforcing empty space inside a car with roll cages and strengthened steel bars in order to absorb and distribute the impact of a crash, diminishing its intensity before it is able to reach the passengers and cause any harm. This same idea is what inspired a "Crumple Zone" (CZ) for data encrypted by TLS-SSL.

The crumple zone is intended to reduce and absorb the effects of malicious attacks before it is able to cause any damage to the confidential data.

This necessity of creating a CZ for data arose out of the popularity of Service Oriented Architectures (SOAs). SOAs feature loose coupling, high dynamism and composition-oriented system constructions which make this architecture

highly appealing for application creation, but equally complicated to create viable security for, as they create more points of entry, as compared to self-contained architectures or monolithic systems.

In the context of SOAs, the CZ consists of a layer of intelligent filter proxies which present a higher barrier of entry for attackers, which minimize the damage made to critical data, and increases the chances of detection in the event of such a malicious attack.

These intelligent proxies behave as endpoints for any inbound TLS connections attempting to reach the service, and no data can reach the service before passing through the CZ. The proxies comprising the CZ are equipped to scan, filter, and even partially execute the data to verify its non-maliciousness before it can pass through and reach the service. Since the components present in the CZ also sometimes execute the information, they are expected to fail occasionally, and are hence monitored by a watchdog process which restarts them whenever necessary.

The CZ then finally establishes all outbound TLS connections to then pass on the approved data from the CZ to the protected service.

The three main principles that need to be maintained for proper CZ functionality are as follows:

- Consistency: All data flowing in and out of the CZ must be protected via TLS
- No bypass: No data flows exist such that they can reach the protected service without passing through the CZ
- Security: The TLS configuration used must be capable of defense against malicious attacks and must be cryptographically strong.

Beyond these ideas, the paper discusses the TLS threat model, and the best practice configuration recommendations for TLS to be utilized for information interchange.

The TLS Attack Surface:

Since the rollout of SSL/TLS in 1999, it has been used widely to secure information interchanges on a variety of communication channels, be it browser authentication or service verification. Due to the large number of browsers, browser versions, SSL/TLS protocol versions and implementations, the tuple space of <version, implementation, configuration> is so terribly large, which leads to an expansive attack surface. In short, the authors are trying to convey that the regular, out of the box TLS will never be fully equipped to handle the large number of possible vulnerabilities across all members of this tuple space. The authors also mention that in the past few years, even the most recent versions of TLS have been successfully breached and results have been published on various networking journals.

Furthermore, the authors also mention that due to the slow and piecewise upgrades of the different browsers to later versions, as well as the non-uniform upgrade of TLS versions used on different servers, there exist a huge number of TLS version incompatibilities even to this day. To overcome this, the client and server must negotiate on a common protocol so as to facilitate communication between the older and newer version of TLS. This down-negotiation means that even though many servers have upgraded to later versions of TLS, they may still be forced to use older versions of TLS, with all their vulnerabilities.

The author mentions the example of BEAST (Browser Exploit Against SSL/TLS) in which attackers exploit a vulnerability present in the CBC (Cipher Block Chaining) cryptography method used in TLS 1.0, which can allow an attacker to perform a plaintext extraction of data secured by CBC. This meant that even if servers upgraded to later versions of TLS, if they

were communicating with services using TLS 1.0, they would be exposing themselves to this vulnerability present in the CBC encryption format. The authors state that according to SSL-Pulse, a global monitoring service, 65.2% of sites surveyed by them were susceptible to this attack, even though many had upgraded to later versions of TLS.

Similarly, another attack method known as CRIME (Compression Ratio Info-leak Made Easy) which relies on information leakage from observations of compression behavior to allow attackers to break SSL/TLS encryption. According to SSL-Pulse, 24.1% of sites they surveyed were vulnerable to this attack, and these sites used compression techniques alongside TLS, which is what made them vulnerable to CRIME.

To counter above mentioned attack surface of TLS, the following recommendations are mentioned by the author:

- Configuration restriction
- Mutual authentication
- Hostname verification

Configuration Restriction: -

Due to the expansive number of configurable TLS parameters, cipher combinations, and multiple provider options, there are a large number of doors left open for attackers to exploit. The authors recommend the below mentioned techniques to reduce the attack surface of TLS arising from configuration variation:

Cipher restriction: Due to the large number of available cipher techniques (such as MD5, SHA, RC4, etc.), many of these do not provide the best possible security but are still used as they take lesser time for encryption/decryption hence making lesser overheads. These leave servers open to BEAST attacks as mentioned earlier, or collision attacks orchestrated by a malicious client. If TLS were to be configured to use only a small set of ciphers that are known to be resilient against current exploits, then this would considerably reduce the attack surface of TLS.

Protocol Restriction: As mentioned earlier, incompatible versions of TLS can cause down-negotiation which can increase the vulnerability of even the latest versions of TLS. To prevent this, older versions of TLS must be phased out and newer versions must be enforced.

Options Restriction: The CRIME attack was mainly a result of servers using compression techniques alongside TLS, which exposed them to the attack. Eliminating the option of TLS compression completely will help reduce the attack surface of TLS.

Mutual Authentication:

TLS provides confidentiality and message integrity, but is also used to verify the identity of servers that are being contacted by the use of digital certificates as shown earlier. However, TLS does not provide client certification, and there is no way to find out if a server is communicating with a legitimate client. TLS has mainly relied on company specific password schemes for credentials.

Military, Army, Defense and other high priority channels however, use client authentication methods such as smart cards or Common Access Cards (CACs) which allow the client as well as the server to establish crypto-strong trust. If this is enforced internet-wide, it could significantly mitigate attacks on TLS's confidentiality and integrity.

Hostname Verification

TLS performs certificate verification, and not endpoint verification. This means that if a party X where to get its hands on a compromised but valid/trusted certificate belonging to party Y, and party X presents this certificate to any client trying to communicate with party Y, then the

clients would accept the certificate and begin trusted communication even though the endpoint is not party, but party X. This endpoint verification must be explicitly enabled on TLS, and external code blocks can be run on received communications in order to verify endpoint authenticity, over certificate authenticity. This creates a layer of security over boilerplate TLS, and reduces the risk of attack due to compromised TLS certificates.

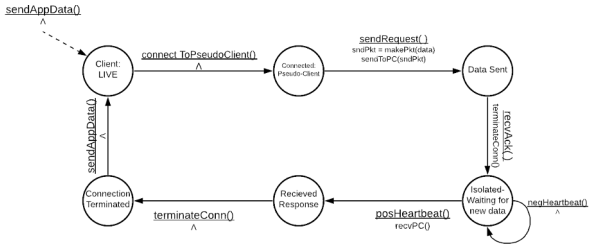
Overall, the paper illustrates the role of TLS in modern day communications, the vulnerabilities present in its use, the past attacks made on TLS based communications using cryptographic decryption, and the various tweaks that can be used over "out-of-the-box" TLS to make it more secure against today's exploits.

Our project PSPIT, which will be using TLS to secure its communication between the pseudo clients and servers can be significantly helped by these tweaks mentioned, and we would definitely want to explore these options, post completion of the final multi-system prototype.

IV. Proposed Methodology

The Pseudo-system Protocol for Information transfer makes use of 2 extra network nodes on either side of a communication session which serve as "masks" to the original identity of the client and the server systems. The functioning of these 4 systems in a basic communication example is as follows.

A. Client

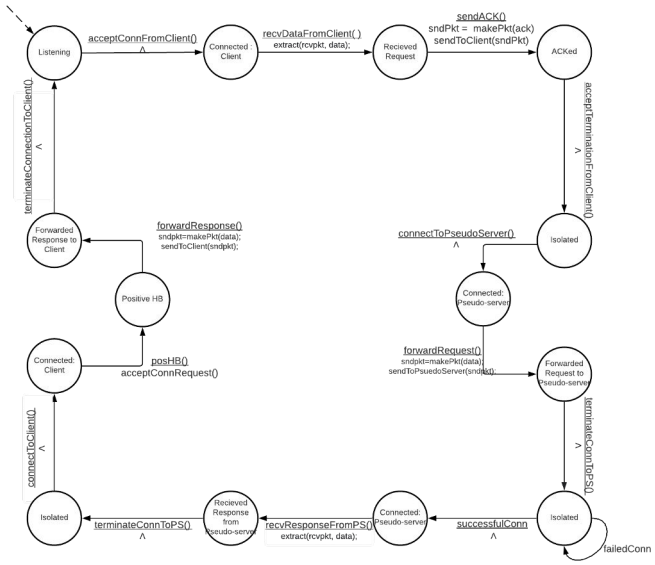


This is the state diagram for the client/host system. There are 6 states including that of the terminal isolated state. The client system triggers the entire protocol when it receives a send command from the upper layer protocol. The first step in this cycle is the establishment of a connection with the pseudo-client. This is done using standard TCP procedures. Once the connection is established(and secured) the client sends the request packet through the data flow link to the pseudo-client. This is again handled by the underlying TCP which takes care of the delayed or unsuccessful transmission. Upon receipt of acknowledgment of successful transmission of the request packet, the client terminates its connection with the pseudo-client. At this point, the client is isolated from the pseudo-client and hence from the internet.

Immediately after isolation of the client, the heartbeat phase of the client takes over. The client sends repetitive beacons in order to check for the availability of response to its request at the pseudo-client. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-client is closed, the client is unable to send the beacons(this connection is later established by the pseudo-client).

Once the heartbeat beacon returns, indicating that the pseudo-client has a new response to send to the client, the connection is re-established with the pseudo-client and the client receives the pending response. After this is done, the client terminates its connection with the pseudo-client and is thus isolated from the network.

B. Pseudo-Client



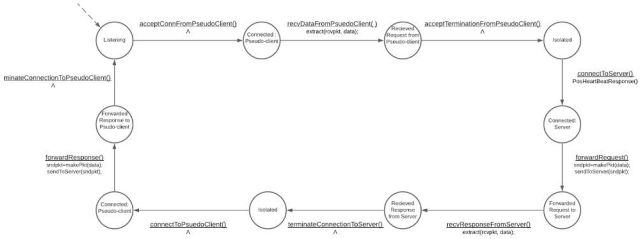
This is the finite state diagram for the pseudo-client system of PSPIT. This system can be in 7 states in 2 directions throughout a request-response cycle. The initial state is achieved when the client system(depicted in the previous FSM) tries to establish a connection with the pseudo-client. The pseudo-client accepts this connection request. Once a connection is established using standard TCP procedures, the pseudo-client receives the request packet from the client. Upon successful receipt of this request, the system sends an ACK message as an acknowledgment. Once the client receives the ACK, the connection is terminated between itself and the pseudo-client. At this point in the protocol, the pseudo-client is isolated from the other systems as there are no active connections to or from itself.

As its next phase, the pseudo-client connects to the pseudo-server and forwards the request packet to the pseudo-server. Upon receiving an acknowledgment of successful transmission of the same, the pseudo-client terminates its connection to the pseudo-server. Immediately after this step, the pseudo-client starts attempting to establish another connection with the pseudo-server. Since the latter is not accepting connections at the moment, all these connection requests fail. When the pseudo-server has new response packets to send, the connection requests succeed and a new connection for receiving the response

packet is established. After receiving the response packet from the pseudo-server, the connection to the pseudo-server is terminated. Note that the transmission of the packet is done using normal TCP procedures.

The next phase is to relay the newly received packet to the client. All this while, the client would have been sending heartbeats probing for the existence of a new response message. Once the pseudo-client is prepared to forward the response to the client, it accepts a connection request from the client and responds to the heartbeats sent by the client. After this, it transmits the response message hence completing its part in the protocol.

C. *Pseudo-Server*

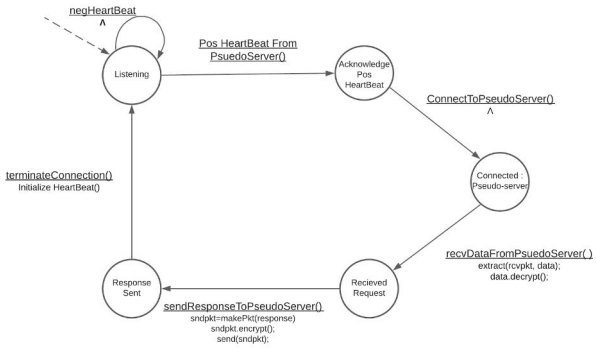


This is the finite state diagram for the pseudo-server system of PSPIT. The initial state for this machine is that it is in a listening and isolated state waiting for a connection request from a pseudo-client. From the previous FSM given for pseudo-client, we can see that once the connection is established between pseudo-client and pseudo-server, the data transmission takes place via TCP protocol procedure. Once the data packet is received, the pseudo-server terminates the connection and goes into an isolated state(no internet connection).

In the next phase, the pseudo-server keeps receiving heartbeats from the server, but only sends a positive response when it has data packets to be transmitted to the server and a connection with the server is established. The request is forwarded to the server using normal TCP procedures and the server responds back to the request. After receiving a response from the server, it terminates the connection with the server leading it to an isolated state.

And in this phase, the isolated pseudo-server acknowledges connection requests from the pseudo-client, as the response has been received from the server, via a normal TCP three-way handshake connection. The response is forwarded to the pseudo-client and the connection with the pseudo-client is terminated. Leading it back to the first state where it is in an isolated state waiting for the pseudo-client to send the request to establish the connection.

D. *Server*



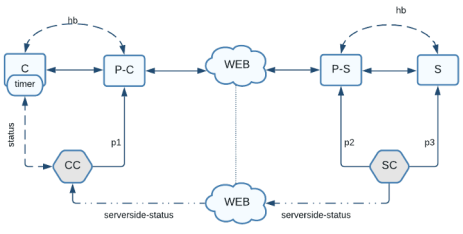
This is the finite state diagram for the server system of PSPIT. This machine has 5 states including the listening/isolated state. This is the final machine to which all the data packets are being sent. In the initial state of this machine, the server is an isolated state, sending heartbeats to the pseudo-server. The server sends repetitive beacons in order to check for the availability of requests at the pseudo-server. This is done using UDP in order to reduce the latency of the entire protocol. Since the connection with the pseudo-server is closed, the server is unable to send the beacons(this connection is later established by the pseudo-server).

In the next phase, the pseudo-server sends a positive response to those heartbeats, which are then acknowledged by the server and a connection is established. Request/Data packets are sent from the pseudo-server to the server and the server responds back with the necessary data packets/response requested by the client to the pseudo-server. Then it goes back to the isolated state, wherein it restarts sending heartbeats to the pseudo-server, and the cycle repeats itself.

E. *Node Failure Handling*

The pseudo-system protocol for information transfer, despite enforcing stringent host security measures, is prone to failures or nodes or delays of request-response messages like any other protocol. After all, the nodes in the system are physical machines running on sources of power with the possibility of being miles apart from each other which can contribute towards packet delays and losses. While delays are already handled by lower layer protocols like TCP, there are a few methods to implement delay detection/handling. Additionally, there are multiple methods to handle system downtimes/failures.

1. System Check Nodes



Here, two extra nodes(routers) are added to the entire topology. These are the Client Check(CC) on the client side and Server Check(SC) on the server side. These nodes maintain the availability of the entities in the protocol. Server Check handles the availability of the two server side machines, namely the pseudo-server and the server machine and Client Check handles the availability of the client side node pseudo-client. Additionally, Client Check maintains the status of the server side machines in the form of bit strings. Eg, 101 means that the pseudo-client and the server are active but the pseudo-server system is down.

Client Check sends probe p1 to pseudo-client while Server Check sends probe p2 and p3 to pseudo-server and server, respectively. These probes are sent through a TCP channel which is initially set up when the systems come online for the first time. So, these channels are up and running as long as both sides of these systems are up and running. This means that at any point of time, Server Check is able to send a probe to check the availability of the systems under its supervision. The same also applies to Client Check.

The moment Server Check is unable to send a probe to it's child systems, it implicitly assumes that the system it was trying to probe is down. In the occurrence of such an event, the Server Check sends a SYSTEM_DOWN status message to the Client Check through the world wide web using conventional TCP protocols. The SYSTEM_DOWN message is simply a bitstring showing the availability of the systems under the Server Check node supervision. The next time(given that a SYSTEM_DOWN message has been sent) it is able to send the probes successfully to both the server side systems, it sends a SYSTEM_UP message to the Client Check machine.

Similarly, the Client Check continually probes the liveness of the pseudo-client system by probing it at regular intervals. Upon encountering the failure of the pseudo-client, it internally sets the status bit of the pseudo-client machine to 0. The next time it is able to probe the pseudo-client successfully, it resets the bit for that particular system.

Upon receiving the SYSTEM_DOWN message from Server Check, it changes the status of the respective machines to 0.

These are the internal workings of Server and Client Check machines.

Every request sent by the client starts a timer exclusive to that request. The expiration of the timer could imply the occurrence of two events.

2. Dedicated Timers

In this proposed solution, each of the client-side machines have a dedicated timer built into each of them. These timers are asynchronous with respect to each other. Additionally, these timers are always running which means that, the moment each of those systems start up, the timer associated with that system is activated. The timers are depicted in the above diagram as follows:

c-timer: This is the client timer exclusive to the client machine.

pc-timer: This is the pseudo-client timer exclusive to the pseudo-client machine.

These timers are responsible for only the requests leaving their respective machines and do not keep track of the delay/latency of the requests handled by the other machines.

When the client system sends a request to the pseudo-client, its timer is started. This timer ends only when the response for that particular request is received back at the client. When the timer expires, the client resends the request message and restarts the timer. In case the client receives a delay indicator message from the pseudo-client before the expiry of its own timer, it ends the timer immediately and resends the request to the pseudo-client. In case, the client is unable to establish a connection with the pseudo-client initially, it implies that the latter is dysfunctional and is unable to receive requests at the moment. However, the problem with using this methodology is that the client attempts to send a heartbeat immediately after sending a request message as well, and the inability to send the heartbeat could be mistaken for the dysfunctionality of the pseudo-client. In this case, the timer of the client is used as a fall back. The moment the timer expires, the client assumes that there is a delay or one of the systems has failed.

When the pseudo-client system sends a request to the pseudo-server, its timer is started; this timer ends only when the response for that particular request is received back at the pseudo-client. When the timer expires, the pseudo-client relays a delay indication message to the client. The client can then resend the request. In case the pseudo-client receives a delayed indicator message from the pseudo-server before its timer expires, it immediately ends its timer and relays this delay message to the client. Again, the client can attempt to resend the message. If the pseudo-client is unable to send the request in the first place, it can be implied that the pseudo-server or the server is disabled and a delay indicator message is sent to the pseudo-client. However, since the pseudo-client

utilizes the inability to connect to the pseudo-server as a means to identify if there are new responses at the pseudo-server, this could cause havoc. The solution to this is that, if the request is sent and the pseudo-client is unable to connect to the pseudo-server, it is assumed that the server side machines are performing correctly. In case this assumption turns out to be incorrect, the already running timer for that request eventually expires and the pseudo-client can then send a delay indicator to the client.

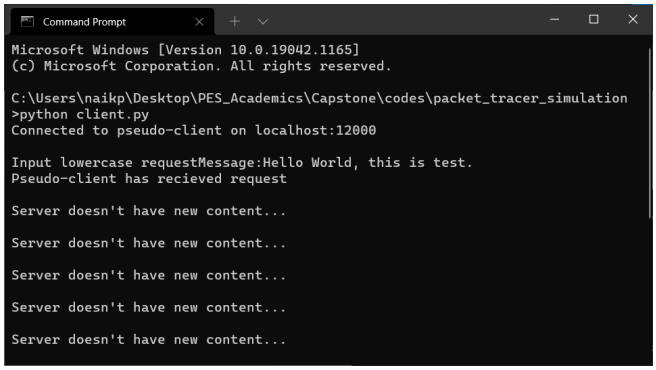
The pseudo-server also has a fall back mechanism in case of errors. In case the pseudo-server is unable to send the request to the server, it can send a delay indicator to the pseudo-client which is relayed upstream to the client which can then resend the request once the server is online.

Drawbacks: The major drawback of this solution is that it is not easy to decide if the timeout is due to the delay in transmission of the messages or due to the failure of the machines.

V. Results

The functioning of the protocol is illustrated using 4 terminals to simulate the 4 systems in the protocol: client, pseudo-client, pseudo-server and server in the local system prototype.
The prototype is initiated by running in the following order.

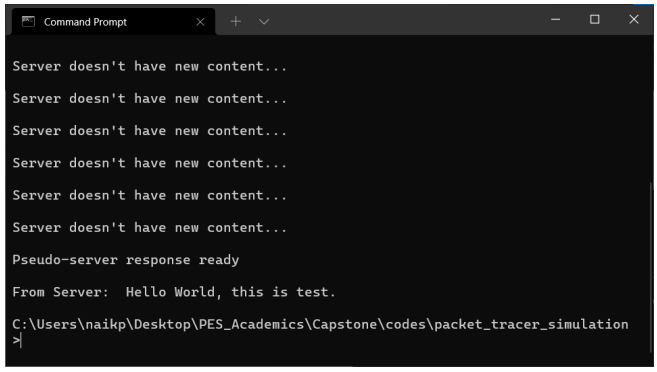
- Server
- Pseudo-server
- Pseudo-client
- Client



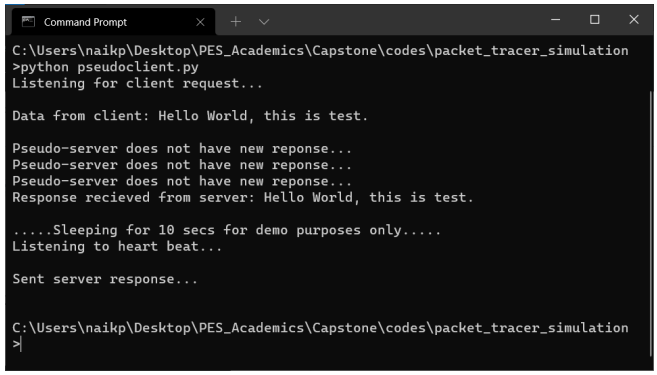
The above screenshot belongs to the simulated client terminal.

This is the result obtained after executing the client protocol script. In the present prototype, the client requests for input from the user. Immediately after this phase is done, the client starts probing the pseudo-client for the presence of a new response from the server.

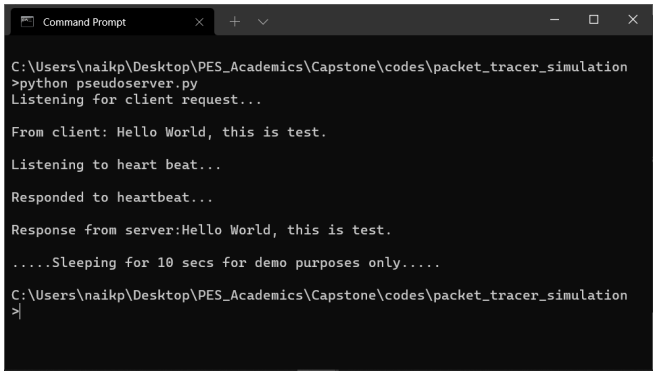
The consecutive “Server doesn’t have new content” messages are the outputs that indicate that the pseudo-client does not have new responses. This is done by attempting to send a packet through an open socket on the client.



As long as the pseudo-client has no response to provide to the client, it continues to probe the pseudo-client. The moment the pseudo-client has a new response, it responds to the probe. This step is indicated by the “Pseudo-server response ready” message being displayed on the client terminal. Once this is done, the pseudo-client sends the server response to the client. This is indicated by the message “From Server: Hello, World, this is test”.



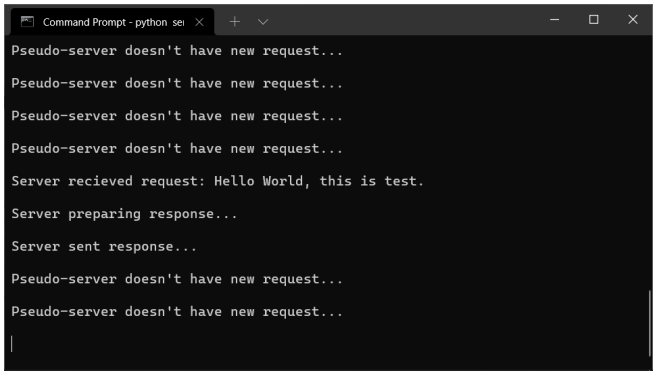
This is the pseudo-client screenshot. Initially, it receives the request from the client. This is indicated by “Data from client: Hello World, this is test”. Immediately after that, it closes the client port, forwards the request to the pseudo-server, closes the pseudo-server port and tries to re-establish the connection with the pseudo-server. The re-establishment is indicated by “Pseudo-server does not have new response”. Once the connection has been re-established, the pseudo-client receives the response and forwards it to the client. This is indicated by “Response received from server: Hello World, this is test” and “Sent server response”



This terminal initiates the pseudo-server python file. After the initiation, it starts listening for requests from pseudo-client. “From client: Hello World, this is test.” This is the input provided by the user at the client end.

“Listening to heartbeat. . .” is the step wherein the server keeps sending heartbeats to the pseudo-server, to check if there are any requests for itself. “Responded to heartbeat. . .” is the pseudo-server responding to the server with a positive message.

“Response from server: Hello World , this is test.” is the final response from the server back to the client.



This terminal corresponds to the server wherein the server keeps sending heartbeats to the pseudo-server, as a result, we can see the following message “Pseudo-server doesn’t have new request. . .”. After this step, the pseudo-server connects to the server when a message is received by it.

The next step is when the message is received, it prepares a response and sends it back to the client, as we can see the following steps are also displayed in the terminal. The next step is for the server to disconnect itself from the pseudo-server and start transmitting heartbeats through the UDP protocol.

In the multi-system implementation, four individual systems have been used, instead of one single simulation system, and packets will flow through the use of real-world physical media such as the internet (gateway routers and transmission links) and local area connections (MAC-Address communication).

The systems and their respective pseudo-systems will be connected to a Local Area Network (LAN), through the use of ethernet cables and RJ-45 connectors, to create a secure closed circuit communication medium between the two. Once packets have been sent from the pseudo-machine or vice versa, the local area connection is closed, and hence the main system is never connected to the internet.

The pseudo-system then establishes a connection to the internet and sends the packet to the destination pseudo-system via the internet and the ISP’s router network. Similarly, on the receiving end, the pseudo-system receives the packets, shuts itself off from the internet, and connects to the destination system via LAN, in the same way as the sender and the pseudo-sender communicated.

The protocol also sees the use of “heartbeats”, which is a term used for when a machine, unprompted, checks for a status change. If no status change is found, then the machine stays in the periodic check state, but if a change is detected, then a series of actions is carried out.

In order to ensure complete host isolation, we could not store the host’s communication details on the pseudo-machine, as it connects to the internet, and if any malicious attacker is able to get access to the pseudo-machine, they could easily start communications with the main system, posing as a pseudo-system.

To prevent this, we made the use of heartbeats in our protocol. Since the communication details are not stored on the pseudo-system, the main system must initiate the communication all on its own. Since the pseudo-system is connected to the internet to check for incoming messages, it cannot be constantly connected to the main system, and hence periodically opens the LAN port to check for any queued messages from the main system, therefore the term “heartbeats”.

The multi system implementation of the protocol utilizes a VPN service for demonstration purposes. Sans VPN, the traffic would be required to be routed through the public internet to the nodes that are concealed behind a NAT router. This made it inconvenient to demonstrate the proper working of the protocol as the NAT router configurations of the commercial routers were difficult to edit. The routers being used by the developers were made by Huawei, D-Link, TP-Link and Dragon. The router firmwares had sophisticated configurations which could not be changed easily. Using a VPN service eased the process of communications while simulating the behaviour of transport of packets across the public internet.

In the multi-system implementation two physical systems were used to simulate the client and the server sides respectively. The first system ran the client and the pseudo-client scripts, while the second system ran the pseudo-server and the server scripts. The two processes on each of the machines communicated using local connections

as the protocol requires them to be in a local network. The VPN tunnel between the two systems simulated the public internet through which the protocol requires the pseudo-client and the pseudo-server systems to be connected. The results of this set up were similar to the local system implementation of the protocol which indicated a successful request and response cycle.

VI. Conclusion

In a world where the value of data rises exponentially with every passing moment, its security becomes of paramount importance. Hackers and data breaches are taking new faces and finding ingenious ways to breach even the most secure protocols. In such a situation, it feels as though systems are better off not even connecting themselves to the internet. Although infeasible, PSPIT finds the best of both worlds, the power of the internet, as well as the safety of staying off the grid. This is what we call Host Isolation. The system with sensitive data sits off the grid, whereas a pseudo-system does the heavy lifting of data download/upload, and establishment of encrypted channels over the internet. This way, PSPIT has a lot of real-world applications where sensitive data is to be secured over a primary system, such as defense, finance, medicine, etc.

VII. Future Scope

Pushing PSPIT over the well-accepted standards of TCP and TLS shows a strong trust factor on the protocol, and can even be made an optional security recommendation for high-security interconnected systems. Collaboration with international certification organizations such as IETF is critical in the future work of the protocol, and we intend on getting the protocol certified as a trusted security service framework.

The future work for the protocol involves working on further safeguards localized in the pseudo-systems which includes but is not limited to

- Partial execution of code
- Anti-malware screening of payload
- Stability testing of payload
- Encrypted sandbox testing

This would require extensive research, further to the research we have done to develop the protocols, as this forays into the domain of the operating system. Implementing OS safeguards over the network level safeguards would be our final stage of completion, post publishment of this research paper, and post IETF certification of PSPIT as a viable security standard.Furthermore, these multi-system communications will be encrypted with the use of TLS, which is Transport Layer Security. TLS is a secure communication encryption standard used worldwide for encrypting internet communications from unintended and/or malicious attackers. TLS uses keys in the form of certificates, cipher and decipher messages, and the certificates are confidentially maintained by each pseudo-system. This TLS security layer acts on top of TCP, with the systems sending their certificates to each other via the protocol, following which messages are encrypted with the destination machine's public key, and decrypted with its private key.

We will be researching the best configurations of TCP-TLS that can be used as part of our protocol, which was a part of our literature survey as well, to find the most secure and convenient settings package for our use, as well as which configuration works most efficiently on the largest number of systems.

The protocol shows great promise for real-world applications, due to its closeness to the issues that real-world systems face, and the simple solution of enforcing host isolation can mitigate a large number of security volatilities. We picture PSPIT to be one among the industry standards in host security - a domain yet unexplored.

References

[1] Elsadig, Muawia & Fadlalla, Yahia. (2016). Survey on Covert Storage Channel in Computer Network Protocols: Detection and Mitigation Techniques. International Journal of Advances in Computer Networks and Its Security. 6.

[2] Pawar, Mohandas & Anuradha, J.. (2015). Network Security and Types of Attacks in Network. Procedia Computer Science. 48. 10.1016/j.procs.2015.04.126.

[3] Pandey, Shailja. (2011). MODERN NETWORK SECURITY: ISSUES AND CHALLENGES. International Journal of Engineering Science and Technology. 3.

[4] Manu, A. & Rudra, Bhawana & Reuther, Bernd & Vyas, O.. (2011). Design and Implementation Issues of Flexible Network Architecture. 10.1109/CICN.2011.59.

[5] Abbasi, Abdul & Muftic, Sead. (2010). CryptoNET: security management protocols. 15-20.

[6] Satapathy, Ashutosh & Livingston, Jenila. (2016). A Comprehensive Survey on SSL/ TLS and their Vulnerabilities. International Journal of Computer Applications. 153. 31-38. 10.5120/ijca2016912063.

[7] Cheng, Pau--chen & Garay, Juan & Herzberg, Amir & Krawczyk, H.. (1998). A security architecture for the Internet Protocol. IBM Systems Journal. 37. 42 - 60. 10.1147/sj.371.0042.

[8] Sirohi, Preeti & Agarwal, Amit & Tyagi, Sapna. (2016). A comprehensive study on security attacks on SSL/TLS protocol. 893-898. 10.1109/NGCT.2016.7877537.

[9] Chakravarty, Sambuddho & Portokalidis, Georgios & Polychronakis, Michalis & Keromytis, Angelos. (2015). Detection and analysis of eavesdropping in anonymous communication networks. International Journal of Information Security. 14. 10.1007/s10207-014-0256-7.

[10] James F. Kurose and Keith W. Ross. 2012. Computer Networking: A Top-Down Approach (6th Edition) (6th. ed.). Pearson.

[11] Behrouz A. Forouzan and Sophia Chung Fegan. 2002. TCP/IP Protocol Suite (2nd. ed.). McGraw-Hill Higher Education.

[12] Atighetchi, Michael & Soule, Nate & Pal, Partha & Loyall, Joseph & Sinclair, Asher & Grant, Robert. (2013). Safe Configuration of TLS Connections - Beyond Default Settings. 6th IEEE Symposium on Security Analytics and Automation (SafeConfig 2013).

[13] Castelluccia, Claude & Mykletun, Einar & Tsudik, Gene. (2006). Improving secure server performance by re-balancing SSL/TLS Handshakes. 2006. 26-34. 10.1145/1128817.1128826.

[14] Larisch, James & Choffnes, David & Levin, Dave & Maggs, Bruce & Mislove, Alan & Wilson, Christo. (2017). CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. 539-556. 10.1109/SP.2017.17.

[15] Borboruah, Gayatry & Nandi, Gypsy. (2014). A Study on Large Scale Network Simulators. International Journal of Computer Science and Information Technologies. 5. 7318-7322.