SAND2017-8293C

Locally Operated Cooperative Key Sharing (LOCKS)

Sharing Session Keys to Enable Deep Packet IDS

Michael Bierma, Aaron Brown, Thomas M. Kroeger, Howard Poston, Troy Delano
*Sandia National Laboratories
Livermore, CA, 94550
mbierma,aabrow,tmkroeg,heposto,tedelan@sandia.gov

***Abstract*—Man-in-the-middle security systems recklessly and needlessly compromise authentication to gain access to information that the enterprise already has. The enterprise already has the session keys and should use them. Moreover, these systems necessitate the deployment and trust of a wildcard root certificate that enables the authentication as any domain. If compromised, this certificate enables attackers to validly authenticate as any domain, potentially for years.**

This paper presents the design, implementation and use of LOCKS (Locally Operated Cooperative Key Sharing). LOCKS enables local clients to share their session keys with the enterprise security monitoring systems to enable DPI without subverting authentication. LOCKS uses a modified cryptography library to enable the sharing of SSL session keys with a trusted agent. The agent works closely with the network monitoring system to enable real-time deep packet inspection of SSL encrypted traffic. This approach provides is fundamentally more secure than MITM introspection systems in widespread use. We discuss use of LOCKS in a diverse operational environment and the lessons learn. We show that the impact on latency is less that MITM and negligible when considered with normal traffic variations. Finally we show that our modified bro IDS system is able to perform real-time decryption and DPI with only a moderate impact on performance.

I. Introduction

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are the current standards for encrypting internet traffic.[1] Due to privacy and security concerns, the use of encryption for web-based communication has increased dramatically. This trend, led by internet giants such as Google, Facebook and Netflix, is expected to result in 30-40x increase in global SSL traffic between 2012 and 2018 [1]. Although encryption is critical for online safety and privacy, it complicates the defense of enterprise networks. Many techniques exist to identify adversaries within network traffic metadata, but deep packet inspection provides the most thorough analysis and best protection against attacks. To enable deep packet inspection on SSL traffic, many enterprises have deployed man-in-the-middle (MITM) proxies. These systems abuse SSL's authentication credentials to enable access to encrypted data. Additionally, MITM system necessities the trust and deployment of a wildcard certificate that can authenticate as any domain and has a typical lifespan of years. With this credential, a malicious actor can masquerade as any website, falsely authenticating to users who believe they are protected. Enterprises are stuck with a trade off between breaking SSL's authentication and/or seeing into their network.

Removing the user from participating in their own security and blinding their authentication is a fundamental step backwards. Everything that is needed to monitor encrypted traffic already exists within the enterprise, and the misuse of SSL authentication is unwarranted. We designed a system where users can share their SSL session keys with an escrow agent, using a modified web browser. Our system, LOCKS (Locally Operated Cooperative Key Sharing), enables the decryption of encrypted sessions without the security risks associated with MITM. Clients share their ephemeral encryption keys with a local secure agent. These keys can then be used to decrypt traffic in real-time, or they can be archived for use later if events warrant an investigation. Figure 1 provides a high-level overview of the LOCKS architecture. The client represents a machine operating within an enterprise. When the client visits amazon.com over SSL, their browser will send their ephemeral session keys to the escrow agent as soon as the handshake completes (if sharing is enabled). This provides the capability to perform real-time deep packet inspections of SSL traffic. The process is transparent to the user and gives the enterprise a finer grained control over their introspection policies compared to MITM.

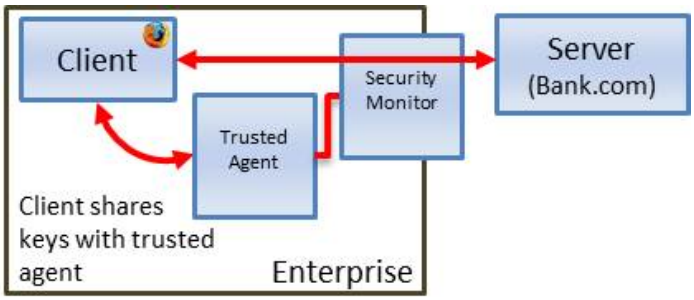


Fig. 1: Architecture diagram for LOCKS system.

A. Benefits

LOCKS has both security and performance advantages over existing encrypted traffic monitoring solutions. By cooperatively sharing keys with a local escrow, it is possible to enhance enterprise cyber protection by:

*Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

[1]This report uses the term SSL when referring to both TLS and SSL.

- Removing the need for a single master certificate that could be used to forge valid certificates
- Allowing the existing SSL protocol to operate as designed. Secure communications is a challenging problem. The many changes and updates to the SSL protocol have shown how subtle issues can easily weaken the security assurances users have. To this end our approach focused on leaving the existing SSL protocol untouched.
- Providing users the freedom to choose whether to share keys for sensitive websites like banking and healthcare.
- Improving performance by removing the MITM proxy's decryption and re-encryption step from client-to-server data path.
- Removing the ability for an attacker to use a compromised MITM device to to modify or spoof traffic.
- Enabling a much richer set of enterprise policies, ranging from just recording keys and only decrypting during an investigation to complete real-time monitoring and many way-points in between.

B. Evaluations and Lessons

LOCKS has been evaluated within an enterprise environment. This allowed us to obtain cross-platform performance characteristics in a variety of network settings. We detail our findings and the metrics we developed for accurately evaluating our monitoring system in a production environment.

We demonstrate that the deployment of a cooperative key sharing system in an enterprise enviornment provides increased security with little additional overhead. We show that LOCKS outperforms MITM in our networking bechmarks and only moderately impacts packet loss (compared to a non-DPI system) while maintaining a System Usability Score of 85.6.

The rest of this paper is organized as follows: Section II provides analysis of alternative approaches and related work. The architecture and design of LOCKS is presented in Section III. We discuss the experimental results of LOCKS's evaluation in Section IV and present the lessions learned during our evaluation in Section V.

II. Background and Related Work

Here we present some background on the SSL protocol. We then discuss how many IDS systems are using man-in-the-middle techniques to enable inspection of encrypted traffic. We then present a survey or related research.

A. Secure Sockets Layer (SSL)

Because the primary use case for the LOCKS is inspecting SSL traffic, we will give a brief overview of the SSL protocol. The goal of this section is to help readers understand how LOCKS fits into the SSL channel.

At a high level, SSL creates a secure connection between two parties over an insecure network. It provides assurances of privacy, integrity and authenticity. It is this privacy protection that makes it difficult for security monitoring to examine the traffic. Figure 2 illustrates the process of how two clients set up a secure channel and communicate. For our work we used TCP as a reliable transport but nothing in our design prevents its use on SSL variants that use UDP.

- The client connects with the server sending *client-hello*.
- The server sends its response (*server-hello*) with a digital certificate that has been signed by a third party notary, known as a certificate authority.
- The client uses this digital certificate to verify the identity of the server, ensuring that the certificate is valid and has not been revoked.
- The server may request a digital certificate from the client to verify the client's identity.
- The client and server cooperatively establish session keys. These keys are are used to encrypt and validate all information between the client and server. They are only applicable for a single SSL session and periodically are rotated.
- Using these keys the client and server now share data that is encrypted to provide privacy, and has a message authentication code (MAC) that assure integrity.

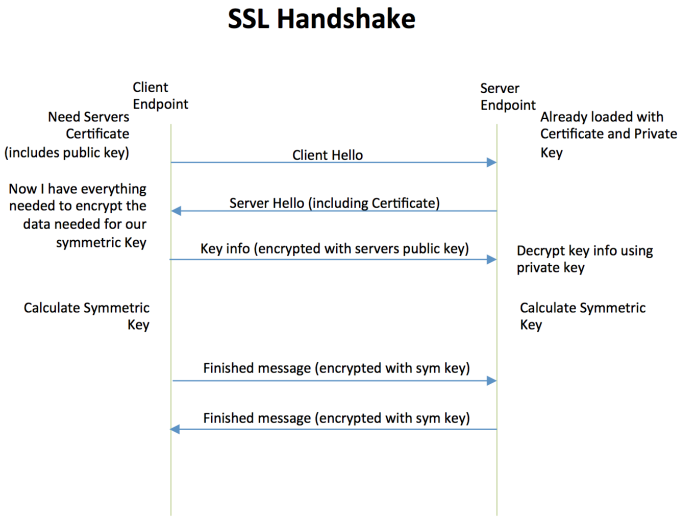


Fig. 2: SSL handshake network diagram.

The session keys are symmetric keys, meaning that the client and server must have the same set of keys in order to communicate. Many cipher suites separate decryption from integrity checking, necessitating two sets of keys: the encryption key and the MAC key. However, newer cipher suites combine the decryption and integrity calculation into a single step with a single key.

These session keys are temporal and don't live beyond this session. They are intended only as a means to provide privacy and integrity for this specific payload. Once this session is done they have no other values. By contrast the keys used for authentication typically live for years. They have the ability

to provide proof that you represent a specific domain. Usually this is quite restricted an only allows a certificate to represent something like `*.bank.com`. Unfortunately, MITM systems that we will explain next require every system in an enterprise to trust a cert whose domain is a complete wildcard (e.g. `*`).

B. SSL Introspection

The most common form of SSL introspection is to use a man-in-the-middle proxy. This proxy is positioned between two communicating parties, and relays the messages back and forth as shown in Figure 4.

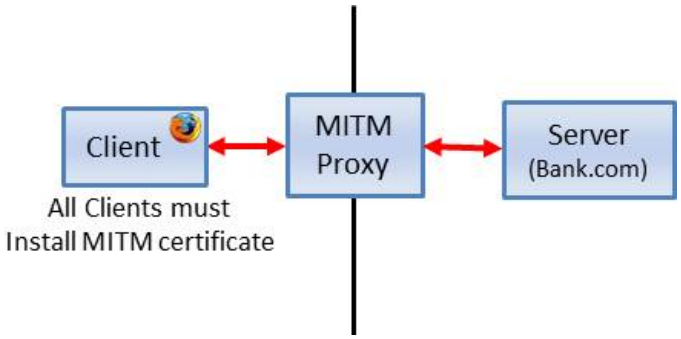


Fig. 3: Diagram of MITM introspection.

Instead of having a secure end-to-end connection to the server, the client negotiates a secure connection with the MITM proxy, which uses a generated certificate to masquerade as the server. The MITM proxy then negotiates a secure connection with the server. The MITM proxy proxies the data between these two connections.

By breaking the end-to-end connection, two SSL connections, one between the client and proxy and one between the proxy and server, are setup, in serial, for every end-to-end connection. This introduces a delay during the initial setup, and can cause confusing error messages if the MITM proxy is unable to setup a proper SSL connection to the server. This dual-connection proxying also increases end-to-end latency of data as the MITM proxy must decrypt the data coming in from one connection, and re-encrypt it to send it on the other connection.

This interdiction also introduces a number of security concerns. Since the client only ever interacts with the MITM proxy, it doesn't receive the real SSL certificate from the end server. It receives a certificate that has been generated by a certificate authority on the MITM proxy. This means that the client must rely on the proxy to properly validate the server's certificates.

Moreover the MITMs certificate must have the ability to authenticate as any possible web server. Accepting such a certificate throughout your enterprise creates a significant vulnerability if an adversary were to gain access to this certificate. Moreover authentication certificates run for years while session keys run for hours.

The MITM proxy provides an attractive target for attackers. If an attacker can compromise the MITM proxy, they can intercept and modify any traffic passing through the proxy. Even if the attacker is only able to obtain the MITM's private key, they could still decrypt all traffic to and from the proxy and could inject traffic into the end-to-end sessions.

Beyond simply intercepting or modifying existing traffic, an attacker with access to the MITM proxy can use the proxy's ability to masquerade as other servers to generate their own certificates for arbitrary sites that will be accepted as valid by the clients that make use of the MITM proxy.

C. Related Work

Jarmoc [2] discusses attacks on MITM interception proxies. Although this work does not discuss new ways to handle encrypted traffic, it does detail many of the dangers associated with MITM proxies. MITM proxies are a high value targets due to their access of sensitive data and critial certificates. Because MITM proxies disrupt authentication to introspect encrypted traffic, there are a variety of unintended consequences. These include weaker encryption, transitive trust and key pair caching attacks.

Vulnerabilities on MITM boxes, including CVE-2012-3372 have been exploited in the wild [3]. These exploitations reinforce the weaknesses of MITM proxies, their exposure to attacks, and the risk they pose to an enterprise.

In 2004, Blue Coat Systems Inc. [4] (a major developer of MITM proxy systems) filed US Patent 7543146 B1 for a protocol in which a client, upon requesting a secure connection via a MITM proxy, receives a request from the proxy for a certificate indicating whether or not it consents to monitoring of its connection by the MITM proxy. This modification to the protocol increases the level of privacy available to the user since they can withhold consent. However, this proposed protocol does not address the security issues caused by the MITM proxy breaking the end-to-end secure connection between the client and the destination server.

Naylor et al. [5] proposes an extention to TLS called Multi-Context TLS (mcTLS) to address the privacy and security issues caused by MITM proxies. In mcTLS, every party in the connection is assigned a privilege level (endpoint, read/write, read, or none) and middleboxes are provided with read and write keys (half of which is generated by each endpoint in order to ensure that both parties consent to the assigned privilege level) based upon their privilege level. A TLS record is broken into sections based upon content, encrypted, and has a MAC key appended by the endpoint. Every middlebox with read access can decrypt the data and checks the MAC key for unauthorized modifications. One with write access can make modifications and generate and append a new MAC key to the record. This protocol increases security by restoring the end-to-end authentication and encryption of TLS and provides both the client and server with fine-grained control over their level of privacy. However, use of mcTLS requires all endpoints and middleboxes to support the modified protocol and has increased overhead due to the need to generate and distribute read and write keys to the appropriate middleboxes.

Sherry et al. [6] proposes a new protocol for monitoring encrypted traffic and Judson Wilson [7] proposes a modification to the TLS protocol in order to monitor encrypted traffic. Sherry et al. [6] also introduces an extended handshake for the

encrypted connection, which adds penalties of up to 97 seconds. While these approaches enable SSL monitoring without MITM proxies, they require either new protocols entirely, or modifications to the existing protocol. These protocols should be thoroughly analyzed by the security community before they are safely deployed in an operation setting. Additionally, implementations of these protocols should be vetted for security vulnerabilities. Because LOCKS does not modify the existing SSL protocol (we are simply sharing the existing session keys within the enterprise), we leverage a protocol designed and analyzed by cryptography and security experts.

While LOCKS is specifically not designed to share session keys outside of the enterprise, the notion of session key sharing has been previously explored. Goh et al. [8] details an covert exfiltration attack where the ephemeral keys are shared with a party outside of the enterprise.

Finally, the issues of intercepting encrypted traffic has been quite active in the press. Our system here focuses on supporting security monitoring for consenting party. We do not advocate a "Keys Under Doormats" approach [9]. Rather, we believe every user is a part of the entire enterprise security solution. LOCKS enables willing parties to participate in enterprise security.

III. Architecture

LOCKS is designed to perform deep packet inspection (DPI) on decrypted SSL network traffic in an enterprise environment. Many enterprise networks use the SSL MITM attack in order to inspect encrypted web traffic traveling across their network borders. LOCKS provides SSL DPI while avoiding SSL MITM, leading to better performance and increased security.

The goal of LOCKS is to leave the SSL connection completely intact while enabling client support of DPI as desired (or as the enterprise policy specifies). As seen in Figure 4, our system is comprised of 5 main players: the client endpoint, LOCKS, IDS (Bro), a network proxy and the server endpoint.

First, the client endpoint initiates an SSL connection with the server endpoint via TCP connection[2]. Once the TCP handshake is negotiated (through the proxy), our client can initiate the SSL handshake directly to the server endpoint. As soon as the session keys are generated on the client endpoint, the keys are shared with the LOCKS registrar, before any additional information is sent on the SSL connection. We have modified the NSS security library within Firefox to share the SSL session keys.

Once the keys have been registered, they are stored in the LOCKS database and pushed to the IDS. Depending on the network architecture, the IDS can either decrypt traffic inline (as an intrusion prevention system), decrypt traffic in monitoring mode (not inline), or simply watch the encrypted traffic and use the keys only for post-event forensics.

A. Cooperative Key Sharing

The SSL session keys are generated frequently by the client endpoint. Visiting a website may initiate 10 new SSL

[2]We acknowlege that SSL connections can also use UDP with DTLS

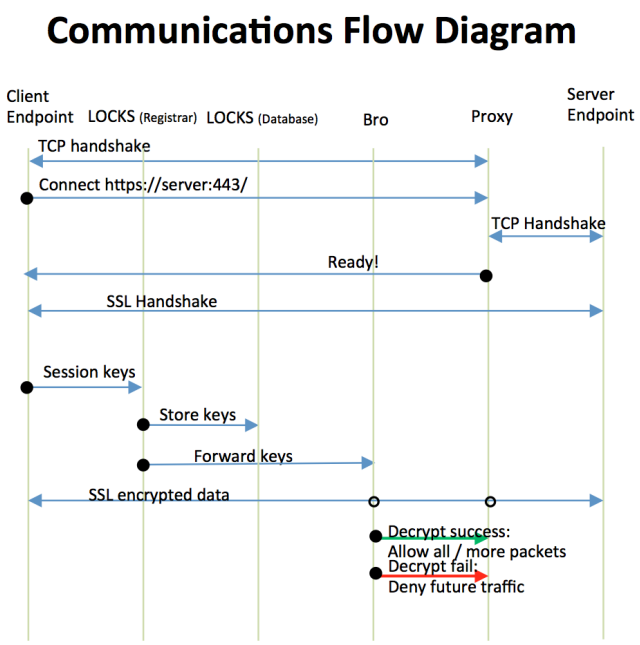


Fig. 4: LOCKS network flow diagram.

connections. Because these SSL connections happen often and are protecting sensitive information, keys must be shared securely and efficiently.

One solution is to encrypt the keys using the public key of the escrow server. Public-private key encryption incurs greater overhead than symmetric key encryption but does not have the additional overhead associated with the derivation of the shared symmetric key.

Using SSL to send the session keys incurs overhead during the initial session setup. However, this can be ameliorated by reusing the SSL session to send multiple key pairs to the escrow server. The logistics of this, including where to track this session within the SSL library and how long to keep it around, are outside the scope of this effort.

B. Selectively-Providing Keys

In addition to ensuring the keys are securely received by the agent, we are exploring ways for the client to easily specify which keys they wish to share. It is important that the end user is aware of the consequences for sending keys that protect sensitive personal information. While the escrow server can have a blacklist configured for sensitive domains, the user is better positioned to know which domains they consider sensitive.

C. SSL Flow Regulation

Due to their position between the clients and servers, MITM proxies are well-positioned to guarantee that they can intercept all traffic. LOCKS, however, does not mandate a specific network configuration to provide the same guarantee.

The nature of SSL mandates that LOCKS needs to, at mimimum, permit the SSL handshake to pass through to ensure

that the client and server can agree upon the keys. Once the keys have been agreed upon, the client can then register the keys with the escrow server.

However, after the SSL handshake has finished, the client and server may communicate before the network security monitor can decrypt the traffic. There are a few ways to handle this situation. These include:

- whitelist- allow for encrypted traffic to friendly domains
- blacklist- only block encrypted traffic to malicious domains
- IPS- block all encrypted SSL traffic until it is decrypted
- forensics- store the keys for post event forensics

A firewall between the client and server can drop encrypted traffic until the appropriate keys have been received by the escrow server. This configuration has some requirements that can make it awkward to deploy in practice. For a firewall to selectively drop encrypted SSL traffic, it needs to be able to discern encrypted SSL traffic from unencrypted SSL traffic. Assuming that each encrypted SSL record is in its own distinct TCP packet, it's possible for a firewall to do byte-level matching to determine whether or not the packet contains an encrypted record and drop it. However, if encrypted SSL records are included in the same TCP packet as the SSL handshake, it is impossible to only drop the encrypted traffic without doing substantial packet manipulation. Future versions of SSL are looking to increase this mixture of encrypted and plaintext SSL records in the same TCP packet [10]. Beyond the protocol issues, the firewall needs to provde some method for the escrow server or the network security monitor to dynamically whitelist flows. There are systems available like OpenFlow that can ease this approach [11].

Instead of having the firewall drop encrypted traffic by default, it's possible to have the network security monitor blacklist traffic once it has seen the SSL handshake proceed. This has the benefit of allowing the network security monitor, which is doing deep-packet inspection and has a better understanding of SSL, to discern when a given flow should be stopped from proceeding. However, this method has both timing and protocol issues. There will be a delay between when the network security monitor discovers that the flow should be stopped, and when the firewall actually implements the rule to drop the flow's packets. During this time period, it's possible for the client and server to continue communicating. The longer this delay, the more traffic can be exchanged, negating the benefits of the blacklist. Even if the blacklist timing issue didn't exist, this method also presumes that the traffic is going to be received in order. If due to loss, nefarious or innocuous, the packet signifying the SSL handshake has finished gets lost or delayed, the client and server will still have generated the session keys and may have sent a large quantity of encrypted data that the remote side is collecting in the TCP receive buffer. By the time the network security monitor has received the notification that the handshake is complete and can institute the blacklist, the client or server has already received the encrypted traffic.

By placing the network security monitor inline to the packet stream (e.g. via IPS mode), it would be possible to stop flows at the exact moment they need to be stopped. They can also drop the encrypted packets received out-of-order to mitigate the protocol issues with blacklisting. However, this inline packet system could introduce performance problems into the network traffic and could provide an attack vector for DoSing the network by attacking the IPS itself.

The last option is to simply let all traffic through. Until the escrow server had the key, the network security monitor would buffer the encrypted data for the SSL connection. Once the key was available, the IDS could decrypt the buffered data. If the key was never made available, the IDS could generate an alert, and possibly archive the data. This option prevents sites from guaranteeing that they can intercept and decrypt all their traffic, and may delay the response slightly if the IDS discovers a problem. However, it is the only option that can be implemented without a corner case like the above, which end up devolving into this form of interception anyway.

IV. Evaluation

We evaluated LOCKS from three different perspectives: the impact LOCKS has on the client latency, the usability of LOCKS for end users, and the impact LOCKS has on sites' existing DPI infrastructure. In all test cases LOCKS did better than MITM but the differences was rather small when compared with the standard deviations. Our user testing provided a positive user experience that showed little change for users. Our Bro system saw increased overhead as it performed full packet decryption but was still able to achieve reasonable performance.

A. Download Latency

Browser latency is always a concern of users and developers. LOCKS requires a browser to share the session key as an SSL session is setup.

To measure the impact of locks on this communication latency we ran tests to download files that were 100 Kilobytes and 10 Megabytes. We did this both from a local web server and also from a web server hosted on an Amazon web services (AWS) instance. These comparison tests give us a sense of the effect of LOCKS in a variety of environments.

We set up our tests to run through similar network paths with MITM and LOCKS being the only difference. Figure 5 shows how this path looked for both tests. All browsers were pointed at our IDS proxy to provide a common starting point for timing and IDS monitoring. In the LOCKS case this system had a bro instance running IDS and decrypting the traffic. From this proxy, the packets were routed to a corporate HTTP proxy and then a Blue Coat IDS system with or without MITM interception. After the Blue Coat system packets were routed to their target server. This setup ensured a comparable path for packets in both tests, ensuring the same number of hops and isolating the differences to the method used for handling inspection of SSL traffic.

By capturing network packets (PCAPs) we were able to measure the how long each download took. Specifically, we measured the difference between the timestamps of the

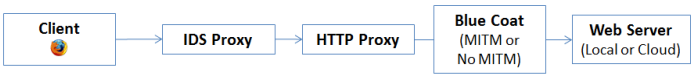


Fig. 5: Test Architecture for Browser Latency Testing

TABLE I: Browser Latency for Local Server

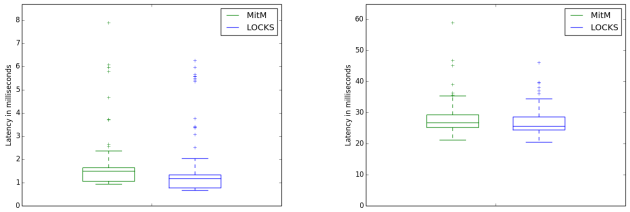
		Mean	Std. Dev.	Min.	Median	Max.
Small File	MITM	1.78	1.28	0.94	1.50	7.90
	LOCKS	1.62	1.40	0.68	1.18	6.26
Large File	MITM	28.43	5.56	21.22	26.74	58.95
	LOCKS	27.06	4.34	20.48	25.72	46.07

initial SYN packet of the connection and the ACK packet acknowledging the final data of the download. This provided a reasonable measurement for how long it took to complete this download. We ran each test a total of 100 times and dismissed the five worst values as outliers.

Table I shows the results of the tests running to the webserver in the same network. From these tables, we can say that on average LOCKS performs better than MITM in all cases, but the difference in the two methods is minor when compared with the variation in network latency.

Table II contains results for the test case with a webserver hosted on the cloud. As in the previous test, the LOCKS system created roughly the same browser latency as MITM interception.

To enable a side-by-side visual comparison of these tests we created box and whisker diagrams that show our latency results for each set of tests, Figures 6–7. Each box shows the middle third for the observed values, the line in that box represents the median, and the bars above and below show the fifth and ninety-fifth percentiles. From these results, we again show that the differences between LOCKS, MITM and no deep packet inspection are negligible when compared to the typical variations in network behavior.

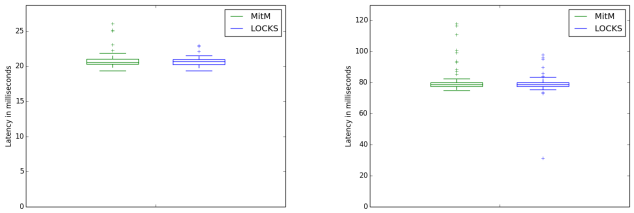


(a) 100KB files (b) 10MB files

Fig. 6: Downloads from local server

TABLE II: Browser Latency for Cloud Server

		Mean	Std. Dev.	Min.	Median	Max.
Small File	MITM	20.77	1.05	19.33	20.54	26.07
	LOCKS	20.67	0.60	19.34	20.65	22.97
Large File	MITM	80.79	7.71	74.87	78.77	117.98
	LOCKS	78.94	6.34	31.42	78.77	97.93



(a) 100KB files (b) 10MB files

Fig. 7: Downloads from cloud server

B. LOCKS Usability Testing

To evaluate our user experiences, we used the System Usability Scale (SUS) created by John Brooke from Digital Equipment Corporation in 1986 [12]. The SUS evaluates the subjective ease of use as perceived by individual users through 10 multiple choice questions. Each question is answered on a 1 through 5 scale, where 1 is Strongly Disagree and 5 is Strongly Agree. These answers are combined to give a scalar result. While the final questionnaire results range from 0 to 100, they are not a percentile ranking. Based upon a fair bit of research, a score above 68 is considered to be above average, and a score below 68 is below average. Figure 8 shows how SUS scores are translated to a normalized percentile.

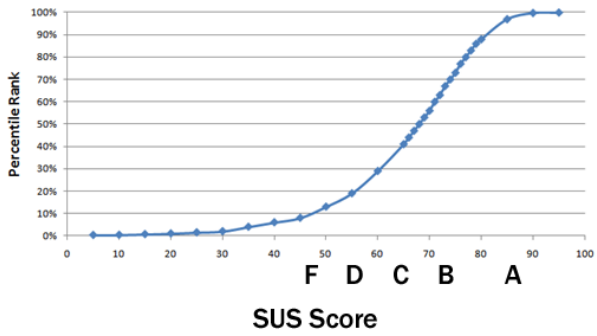


Fig. 8: Normalized percentile for SUS scores

A survey of initial alpha users produced an average score of 85.6. This correlates to a normalized percentage of around 97%. This implies that, given previous research and calibration, only 3% of software products are easier to use than our product, and 97% are less easy to use. (According to the chart above, this means we get an A). Note that the Ease of Installation question is separate from the SUS. The average Ease of Installation response was a 5.25 out of 7 points maximum (where 7 is Very Easy and 1 is Very Difficult).

C. Measuring Impact On DPI

The benefit of LOCKS is that it augments existing DPI solutions with the ability to inspect SSL encrypted sessions. To measure LOCKS' impact on these DPI solutions, we compared the performance of the Bro IDS both with and without SSL decryption enabled.

We setup a testbed on a controlled network consisting of a client, a machine running a Squid proxy and Bro, and a web server with the same benchmark as was used in the latency measurements. We used the traffic shaping tools available in Linux to vary the bandwidth between nodes and ran multiple browser instances to vary the number of simultaneous clients.

To gauge the effect of decryption on Bro, we made use of the included "capture loss" facility [13]. As a passive monitor, Bro reconstructs the SSL sessions from the network traffic. Bro tracks events like ACK for packets it did not see, and other similar gaps in the traffic flows, and uses this to estimate the loss rate it saw. This loss rate tells us how much traffic, that transitted between the clients and servers, was dropped before it was seen by the sensor. In testing, our testbed did not induce any monitoring loss, and since all our traffic was routed through the host running Bro, the only way for monitoring packet loss to be introduced was if Bro was not able to keep up with the traffic being generated.

The nature of handling, and decrypting, SSL sessions is embarassingly parallel. As more flows are added, they can be handled by simply adding more resources to handle them. Bro has its own limitations on scalability and, to avoid measuring the limitations of Bro's scalabilty, we took advantage of the embarassingly parallel nature of SSL decryption to measure the effect on a single Bro instance [14]. As the Bro instance becomes less able to handle the workload without introducing loss, we can gauge, with the limitations laid out by Weaver and Sommer, how a cluster of Bro instances would handle a scaled up workload.

The effect of LOCKS was measured by independently varying the number of clients and the bandwidth of the connection being monitored by Bro. The number of clients was varied from 10 to 50 performing downloads in parallel in steps of 10 clients. The connection bandwidth was varied from 100 Mbps to 400 Mbps in steps of 100 Mbps.

Figure 9 shows a sample of the results of these measurements. For this test, 40 clients performed downloads in parallel through a connection whose bandwidth was throttled at different rates. As shown in the figure, the decryption enabled by LOCKS (blue, solid line) puts enough additional load on Bro to increase the packet loss it induces as compared to a system not running LOCKS (red, dashed line). To negate these losses will either necessitate increasing the computational resources available to Bro, or to accept the small additional loss.

D. Weaknesses

Although LOCKS outperforms existing SSL DPI techniques, it is not without flaws. LOCKS is dependent on a few large systems working together, namely Bro, OpenSSL, NSS and Firefox. Firefox, the web browser we modified to share session keys with the LOCKS server, uses Network Security Services (NSS) for its SSL implementation. Bro, on the other hand, uses OpenSSL. While these two libraries are functionally equivalent, they both require modification to integrate with LOCKS.

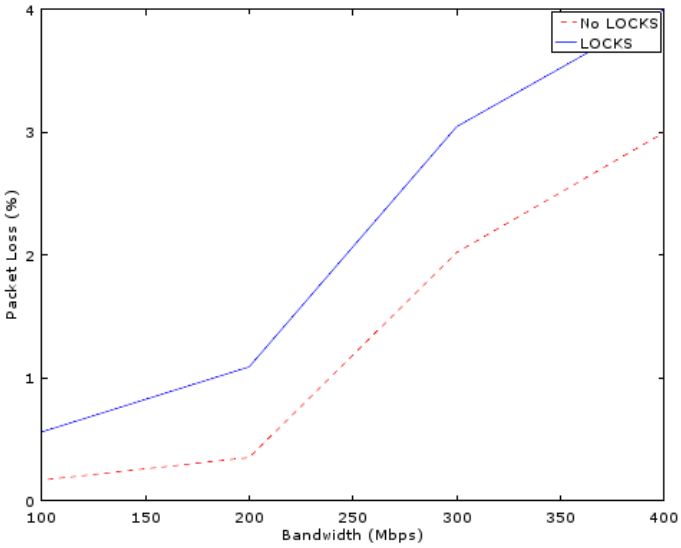


Fig. 9: Packet loss vs. Traffic bandwidth for 40 parallel clients

E. Protocol Complexity

The TLS standards are relatively complex, and ensuring that the semantics for all possible uses can be rather difficult. For example, in the standard usage, there is a single client/server handshake, and after that, all subsequent messages are application data that would be decrypted, with a final message to indicate the connection is being closed. However, nothing in the protocol says that this must be the case. For example, it's possible to do a second handshake routine after the first one to renegotiate session parameters.

To make matters worse, new versions of TLS get released periodically (1.3 is currently in development), that add or deprecate ciphers, and can make changes to the protocol semantics, in both obvious and subtle ways. As an example, when TLS 1.1 was released, it changed the CBC-mode cipher text message to include an explicit initialization vector, whereas in TLS 1.0 and earlier, the IV was simply the last block from the previous message. Supporting older cipher suites (using the previous last block as an IV) requires adding buffering capabilities to the IDS system in order to enable successful decryption of traffic using these older cipher suites. Additionally, newer GCM mode ciphers that combine the encryption and MAC keys into one key are becoming more popular. This makes it impossible to allow decryption without modification.

The net effect of this is that with each new release of a TLS standard, or even with new releases of HTTP daemons and clients who may use different ciphers or have different TLS usage patterns, there may be new ciphers or new semantics that the LOCKS IDS/IPS will need to be updated to handle. Presumably, these will be relatively infrequent, but the result may be a lag time during which new browsers may not work with the system.

F. Distribution

Unlike MITM, LOCKS requires new software on each client that is part of the enterprise. This means that it is necessary to support Linux, Windows, OSX, Android, iOS and any other operating systems in use within the network. We used Firefox for our proof of concept, which has cross-platform support, but other software packages that use SSL may ship with a custom network stack. This means that those vendors will have to rewrite their software to support LOCKS deployments.

Fortunately, most software packages use SSL linked libraries that can be replaced relatively easily. Distribution management systems within the enterprise will need to be continuously updated with the latest SSL libraries, modified for use within LOCKS.

G. Maintainence

To use the LOCKS design, a custom version of an SSL library is necessary. In the case that we only wish to introspect on Firefox browser traffic, then we will need a custom version of NSS. The custom SSL library will export the information about a given TLS flow to the Key Escrow server, where Bro can then use that information to decrypt the session.

This has the benefit of making the LOCKS concept easy to integrate into Firefox itself (e.g. it could be part of the preferences), and can make it work more effectively (e.g. Firefox could stop processing SSL until it passed the key information to the LOCKS server). Unfortunately, this means that with every new release of Firefox, the LOCKS component will need an update. Based on the 2014 Firefox release schedule, it looks like releases happen at least once a month, which makes this endeavor somewhat difficult.

V. Lessons Learned

Deploying LOCKS within an enterprise environment provided valuable insight into the necessities and pitfalls of a SSL DPI system.

A. Browser support

Adapting a browser to share keys went smoothly. Firefox uses the NSS security library for SSL, and we were able to easily modify NSS to share session keys with our trusted server upon generation. Although we statically linked NSS to Firefox for our implementation, it should be possible to dynamically link a modified NSS version for existing browsers. This would enable LOCKS to support other programs that utilize dynamically linked NSS for thier SSL connections, without having to recompile the application.

Instead of modifying NSS, it is also possible to leverage the browser's built-in SSL debugging support. This feature allows the user to set an environment variable in order to save information to allow for an external program to decrypt the SSL connection. With a few well-crafted hooks, it might be possible to enable key sharing without modifying the browser.

B. Cipher suite complexities

While MITM systems can limit the cipher suites available, LOCKS systems leave the user in control of the SSL handshake. As result, this approach must support a broader range of ciphers.

There are some situations where it may be more efficient to drop support for specific cipher suites. Some block ciphers require packet buffering for successful decryption across packets. This can be done on the decryption appliance, but we may wish to limit these ciphers if the buffering congests the network.

Additionally, some of the cipher suites use one key for both integrity and privacy. With these ciphers it's not possible to split out privacy and integrity. Authenticated encryption (e.g. AES-GCM) uses the same key for encryption and integrity. If we wish to only decrypt the session (without the ability to modify traffic), it may be desirable to require cipher suites to support separate encryption and integrity keys.

While it is possible for us to block ciphers that we do not wish to support, we cannot gracefully terminate the connection in these conditions.

C. Key Identification and Verification

Once the keys are registered with the key escrow, it can be challenging to determine the correct key for a given stream or which stream the key belongs to.

1) Protocol Quintuple: In the simple case, it's possible to use the quintuple of: source address, destination address, source port, destination port and protocol. However, in an enterprise setting, this may not be sufficient. This may be affected by internal networks that are NATed or by eventual port re-use by a client and server. The NAT problem can be worked around by deploying NSMs at each internal network so the quintuple is the same as the clients. The quintuple re-use issue can be ameliorated by using timestamps to narrow down the possible keys for a given flow.

2) Session IDs/Session Tickets: It is also possible is to use the session IDs or session tickets. Session IDs and tickets are used by many SSL clients and servers to reduce the latency of connection setup when a client is reconnecting to a given server. Sessions with the same IDs or tickets will not use the same session keys, as these are generated using a combination of the session-specific randoms, and the original negotiated password. To successfully use either, the NSM would need to generate the new keys for a given session using the new client and server randoms, along with the previously negotiated password specified via the session id or ticket. Future versions of SSL plan to make this session resumption even more prevalent.

3) Client/Server Randoms: Given the dependency of the session keys on the client and server randoms as well as their preservation across NAT configurations, these are a reasonable candidate to use to match a given session with the registered session keys. These randoms implicitly include timestamps, if constructed as per the protocol, giving them a natural resistance to accidental reuse. Another benefit is that they are constructed by both the client and the server, meaning that an attacker needs to control both in order to produce duplicate session matches.

D. Key Verification

Once a key has been selected, or multiple keys if collisions were present, it can be good to verify that a given key is correct. If a given session has the MAC key registered, or is using a ciphersuite that doesn't require a ciphersuite, it is easy to verify the decryption. If the ciphersuite requires a MAC key, and it is not registered, it's possible to run entropy analysis heuristics in an attempt to validate the decryption. However, a determined attacker may be able to create valid-looking data, even in the face of invalid decryption.

Note, single packet handshakes in TLS 1.3 may make this more difficult.

VI. Future Work

While our study has shown that such a key escrow system can be implemented with modest effort and minimal user impact, there are several areas that warrant further exploration. Many of the weaknesses and pitfalls of LOCKS can be resolved with a new protocol or an extension to SSL. In the future, we would like to explore the proper protocol for encrypted traffic DPI. More practically, we are exploring hands-on ways to deploy such a system within a broader user base to enable better enterprise protections.

Additionally, LOCKS enables a very broad range of possible enterprise policies. These could vary widely from "must be used at all times" to "keys are only used after the fact for post-event forensics". There are a broad range of deployment options we believe should be explored.

Moreover, the session keys themselves are a very sensitive thing to store. There exists a broad range of other ways to store data among multiple parties [15] that could provide significant policy values here. One could envision a system where the service to access a users keys required two separate parties consenting, the same way most retail stores require two people to refund a customer with cash.

VII. Conclusion

The successful use of LOCKS has shown that this approach can be an effective way to enable security monitoring on encrypted traffic. Many of the limits and difficulties we encountered were not due to issues in our architecture or difficulties with deployment. Instead, these challenges involved minor items in the SSL protocol. We believe a key conclusion we can draw from our work in developing LOCKS is the need for an extension to the SSL protocol that consciously supports users sharing their session keys with a trusted monitoring party. The prevalence of man-in-the-middle systems has shown that there is a clear need for introspection. Such an extension would go a long way towards reliving the danger from current approaches and also bringing this important issue to the table for standards groups to provide a safe and secure solution for enterprise security monitoring.

References

[1] Intel, "Upsurge in encrypted traffic drives demand for cost-efficient ssl application delivery," http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cost-efficient-ssl-application-delivery-paper.pdf, 2013.

[2] J. Jarmoc, "Ssl/tls interception proxies and transitive trust," 2012, Black Hat Europe.

[3] "CVE-2012-3372." Available from MITRE, CVE-ID CVE-2012-3372., 2012.

[4] S. Karandikar and T. Kelly, "Using digital certificates to request client consent prior to decrypting ssl communications," Jun. 2 2009, uS Patent 7,543,146. [Online]. Available: http://www.google.com/patents/US7543146

[5] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steennkiste, "Multi-context tls (mctls) enabling secure in-network functionality in tls," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 199–212. [Online]. Available: http://dx.doi.org/10.1145/2785956.2787482

[6] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 213–226. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787502

[7] J. Wilson, "Tls 1.3 pr 426 keyupdate," 2016.

[8] E.-J. Goh, D. Boneh, P. Golle, and B. Pinkas, "The design and implementation of protocol-based hidden key recovery," in *Proceedings of the 6th Information Security Conference*, Oct 2003, pp. 165–179.

[9] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. W. Diffie, J. Gilmore, M. Green, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, M. A. Specter, and D. J. Weitzner, "Keys under doormats," *Communications of the ACM*, vol. 58, no. 10, pp. 24–26, 2015. [Online]. Available: http://doi.acm.org/10.1145/2814825

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-tls13-12, Mar. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-12

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[12] J. Brooke, "Sus: A quick and dirty usability scale," 1996.

[13] T. B. Project, "policy/misc/capture-loss.bro," https://www.bro.org/sphinx/scripts/policy/misc/capture-loss.bro.html, 2013.

[14] N. Weaver and R. Sommer, "Stress testing cluster bro," in *Proceedings of USENiX DETER Community Workshop on Cyber Security Experimentation and Test (DETER 2007)*, 2007.

[15] T. M. Kroeger, J. C. Frank, and E. L. Miller, "The case for distributed data archival using secret splitting with percival," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 204–209.