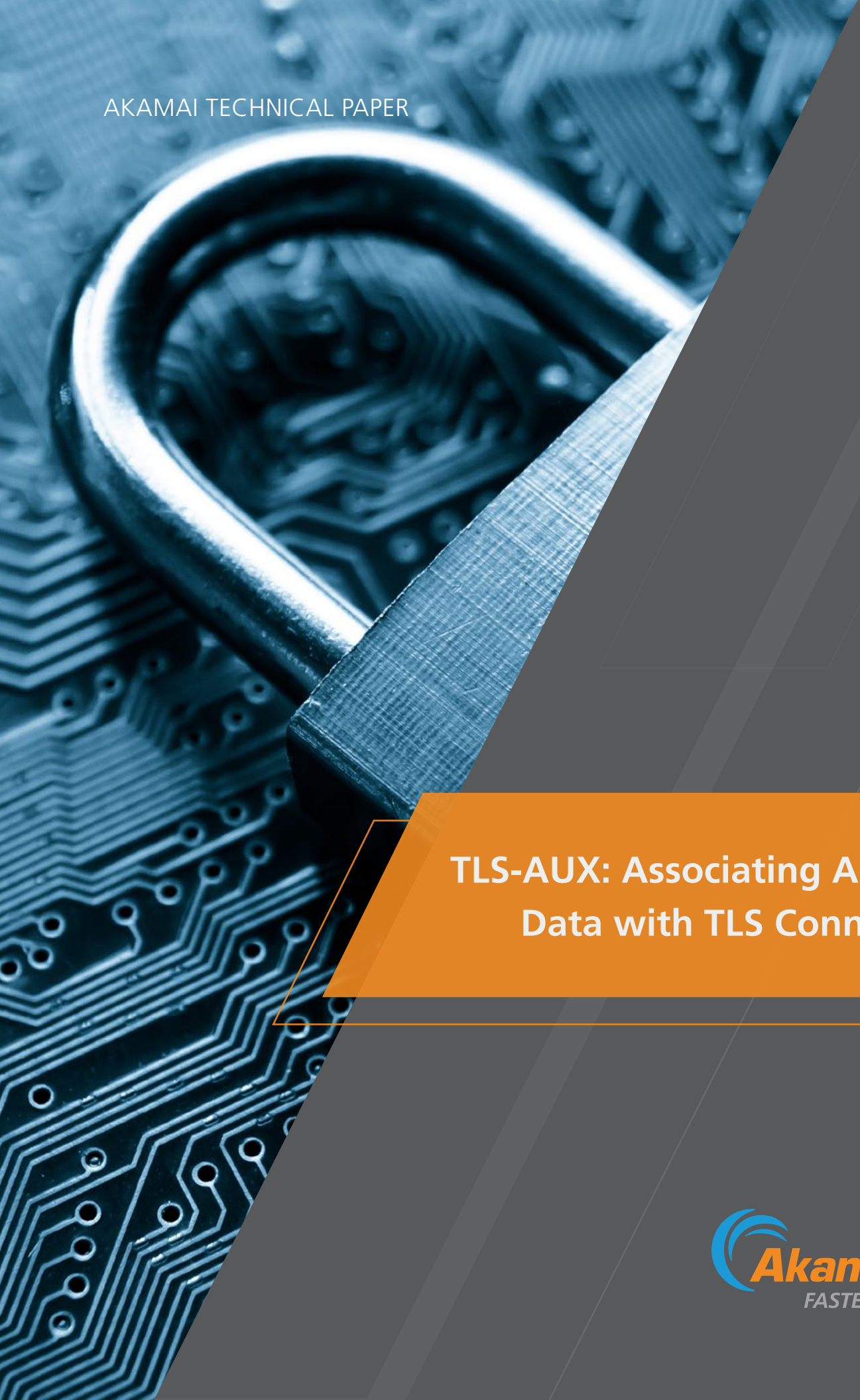

AKAMAI TECHNICAL PAPER

TLS-AUX: Associating Auxiliary Data with TLS Connections

Akamai
FASTER FORWARD

TABLE OF CONTENTS

1. Introduction

In the year 2014, concerns over the privacy of communication on the Internet accelerated the trend of websites serving all of their content over HTTPS. It is estimated that in the beginning of 2015, over 40% of web traffic was served encrypted[1]. The opinion that all the web traffic should be secured with an end-to-end technology (like TLS) that provides encryption, server authentication, and integrity, has become prevalent. At the same time, some communities pointed out the collateral damage that would be caused if all the traffic used such encryption technologies. Network operators use intermediate proxies for various services such as transparent caching, transcoding/trans-rating, real-time network intelligence sharing, etc. It is feared that such services would either cease to function in the encrypted communication environment, or they would become very difficult to implement. Many operators believe that this would have a severe impact on their resource management practices. How to reconcile the desire to use end-to-end encryption while not making worthy network services entirely obsolete continues to be a topic of debate at the time of writing this paper.

One could classify intermediate, proxy-based network services into two broad categories based on their data access needs.

1. **Content-aware services:** Services such as transparent caches, transcoding/trans-rating, and parental guidance filtering fall in the first category. Without any end-to-end encryption, they can intercept and read/modify/write the payload of communication between the user and the servers (e.g. HTTP requests and responses).
2. **Content-unaware services:** This category of services deals with metadata related to the HTTP request/response rather than the payload of the communication itself. Sending network conditions intelligence such as real-time throughput and congestion guidance to the client and server is a widely cited example of a middle-box based service of this category. While the metadata is oblivious of the payload, the implementation of this type of service may choose to use an in-band technique such as inserting HTTP request/response headers in the communication as the vehicle to communicate the metadata to its destined endpoint.

If end-to-end encryption is used for all (or a large fraction) of web traffic, then the first class of services is rendered useless. The second class of services need not become obsolete due to encryption, but if its vehicle becomes unavailable, then the services of that class are unable to function in their current form. At the time of writing, it is unclear if there will be a broad consensus on doing work to re-enable both classes of services for end-to-end encrypted traffic or only the second class. It is also unclear if unified solutions exist that will solve the issue for both classes or if their various limited solution spaces will be addressed by specific point solutions. A good survey of potential mitigations, along with their pros and cons, can be found in[2].

As a CDN, we believe that network services that provide metadata for the optimization of content and its delivery can be very valuable, and we would like to see such services continue to develop even as more and more Internet traffic uses end-to-end encryption. Such network services are required to be designed to operate in a manner that neither weakens the privacy of communication nor leaks sensitive information as a side effect. We would like to encourage a collaborative approach towards that goal. A collaborative approach will allow CDNs and content providers to be aware of the features/optimizations that get applied to their traffic in the network, be able to approve/disapprove the application of these features, and work with other parties in the ecosystem to make the best use of these features.

In this paper, we consider the issue faced by the second class of network services described above, i.e., that their vehicle of communication is not available with end-to-end encrypted traffic. There are multiple ways to help re-enable such services:

1. The use of proxies in the network that carry TLS certificates. The proxies are able to terminate TLS connections for clients. The ability to terminate TLS connections enables the proxies to work with unencrypted data and thereby insert any metadata into the communication in the form of HTTP headers destined to the interested web servers located on the public Internet.
2. The use of out-of-band APIs, implemented in applications running near the gateway of the network. The interested web servers could query the metadata of interest through this API for every client connection.
3. Creating new in-band techniques for the communication of metadata either within the TLS model or by extending it. SPUD[3], McTLS[4], and throughput guidance through in-band signaling[5] in the transport layer are some examples of recent work in this category.

Each approach has pros and cons that can be evaluated in terms of their impact on the privacy of communication and network software layering conventions as well as the infrastructure and resource needs of the solution.

The direction outlined in this paper for this work falls under the 3rd category of the solutions listed above. We attempt to create an in-band metadata vehicle for end-to-end encrypted traffic which does not weaken the privacy of communication. Since networks have already deployed proxy-based services, we are developing this method in the hope that extension and reuse of the existing infrastructure will be possible and also relatively easy as compared to some of the other solutions. This method is not a promise of a future product or service from Akamai. It should be taken simply as an example, showing that in-band metadata exchanges are possible for end-to-end encrypted flows without warranting significant modification of their model.

In the design of this technique, we introduce a new layer between TCP and TLS, called TLS-AUX. By virtue of this position in the protocol stack, the TLS-AUX layer can recognize the data flowing on TLS connections in either direction as a sequence of structured TLS records[6]. We also introduce a new type of TLS record, called the AUX record, which is allowed to exist only at the TLS-AUX layer and not higher. The AUX records are to be used for communication of metadata between network services and webservers as described in the above use cases. Network services and webservers may directly insert and retrieve their metadata into the TLS connections in the form of AUX records, and retrieve them from the connections' flow. In this manner, the mechanism is very similar to network services inserting and removing HTTP request and response headers directly into the body of TCP connections.

Our prototype implementation of TLS-AUX is a user-level library. Two prototype network services for real-time throughput guidance are used to demonstrate the use of the library – one running on mobile networks' radio base stations and the other on home wifi routers. The webserver also supports TLS-AUX through a compile-time linking with the TLS-AUX library. TLS-AUX needs minor work in the application code itself (for inserting and retrieving metadata), but it needs no change to the TLS library (i.e. OpenSSL) or the operating system. The prototypes we built demonstrate how adaptive-bitrate MPEG-DASH media delivery can use throughput guidance from the network to improve the media QoE metrics[7].

The rest of the paper is organized as follows: In section 2, we present the requirements of TLS-AUX. In section 3, we present the design of the TLS-AUX layer as well as the out-of-band discovery and negotiation protocol between the middle-box and the server required to initiate communication at the TLS-AUX layer. In section 4, we show how some of the existing middle-box services may be redesigned to work with encrypted traffic. Section 5 concludes the paper.

2. Requirements

The TLS-AUX work is an attempt to create a mechanism for in-band metadata communication for use with end-to-end encrypted traffic flows which does not interfere with the encryption in any way. The requirements from TLS-AUX are divided into several categories:

1. **Functionality**
   - TLS-AUX should allow either endpoint of a TLS connection or an intervening middle-box to insert/retrieve/delete auxiliary data into the TCP flow of the connection.
   - It should be an application-layer mechanism.
   - The design should ensure that TLS-AUX is not used with incompatible endpoints or middle-boxes, which may cause denial of service.

2. **Security**
   - It should be possible to prove that TLS-AUX does not lower the security guarantees provided by TLS in any manner.
   - TLS-AUX should not require changes to the currently existing and proposed future TLS protocol versions. Consequently, changes should not be needed in any TLS libraries (such as OpenSSL).
   - An optional mechanism should be provided for the protection of the auxiliary data exchanged in the TLS-AUX layer. The mechanism should allow the communicating entities to exchange the data in plaintext, signed plaintext, or encrypted with an auxiliary key.

3. **Flexibility**
   - It should be flexible enough to work over any network without having to make special arrangements for entities like NAT, firewalls, and gateways, which may obscure network-wide visibility.
   - No new infrastructure (such as secure tunnels, de-NAT functions, APIs, and gateway applications) should be required to be set up for the service to work between network elements and webservers located in the public Internet.

3. Design of the TLS-AUX Layer

In this section, we present the design of TLS-AUX in two parts. First, we describe the use of AUX records as the communication vehicle used by the TLS-AUX layer. In the second part, we describe the out-of-band protocol used for discovery and negotiation. The protocol helps middle-boxes discover if it's safe to use TLS-AUX on any TLS connection and if so, to negotiate the auxiliary key with which the payload of AUX records should be signed or encrypted.

3.1 Insertion and removal mechanism

TLS-AUX Record Specification

The TLS protocol operates on top of TCP/IP and consists of the exchange of messages between the client and the server. All of the messages exchanged are formatted as TLS records[6]. The standard format of a TLS record is shown below in Figure 1.

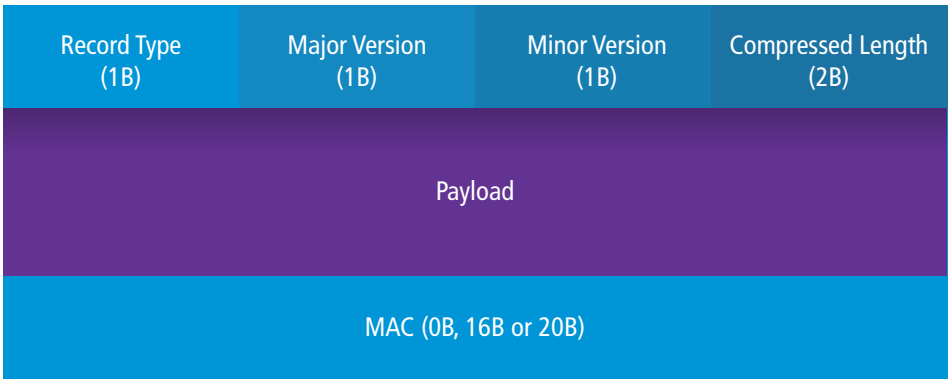

Figure 1: TLS Record Format

The 5-byte header of the TLS record specifies the type of record, TLS protocol, and the length of the record. The body of the record is interpreted based on the values of its type and version. For TLSv1.2, the major and minor versions have the values of 0x03 and 0x00. For the future TLSv1.3, the proposed values are 0x03 and 0x01.

We define an AUX record to be a type of TLS record for which the major and minor version fields have a value of 0x0000 and the record type field has the value of 0xFF. AUX records are not recognized by TLS and would be considered invalid records at the TLS layer. The special values used for version and type have not been officially reserved for this use.

Rules for the Insertion and Removal of AUX Records

Consider an environment in which a client creates a connection to a server and the packets of this connection pass through a network element in the middle. The rules for insertion and removal of AUX records into the connection are as follows:

1. AUX record may be inserted into or removed from a connection along either direction (towards the client or a server) by either endpoint or a network element in the middle.
2. The entity that inserts an AUX record into a connection must do so without violating the boundaries of any other TLS record on the connection.
3. The entity that inserts or removes an AUX record must ensure that the TCP sequence numbers do not become inconsistent from the addition or removal. Some implementation techniques may require a sequence number offsetting method while others may be immune to this issue.
4. Since AUX records are essentially invalid TLS records, they must not be allowed to reach the TLS layer at either endpoint of the TLS connection. To support AUX records at either endpoint, the applications at that end must have the TLS-AUX layer sandwiched between the TCP and TLS layers.

5. The TLS-AUX layer should be responsible for handling the AUX records and keeping them from reaching the TLS layer. The TLS-AUX layer may be implemented as a user-level library to be linked with the application at compile time. It should provide an API for the application to send or retrieve auxiliary data through the TLS-AUX mechanism.
6. TLS-AUX records must not be inserted into a TLS connection unless the sender is sure that they will not cause errors at their intended destination or any intervening network element.

3.2 Discovery and Negotiation protocol

Before metadata exchange can be initiated, two main functions need to be fulfilled.

1. AUX records must be sent only when the destination connection endpoint and all intervening network elements can handle them without causing TLS errors. The steps of the protocol discover whether it is safe to send AUX records to the intended destination.
2. The entity inserting metadata must know the format in which the destination expects it to arrive. The protocol provides a way to negotiate the format of this data. The auxiliary data may be sent in plaintext, signed plaintext, or encrypted, depending on the need to ensure its privacy and integrity. The protocol provides a way to negotiate the encryption method and key.

It is possible to achieve both of these functions through basic techniques, such as configuration provisioning at the network and server side. It is also possible to use a discovery and negotiation protocol for this purpose that helps replace or augment the basic technique. In this section, we present the steps of this prototype.

Steps of the protocol

The steps of the protocol are presented in the context of an environment in which clients make HTTPS requests to a server (S) on the Internet and the data flow passes through a network element in the middle (NE). The protocol describes what NE should do to initiate metadata communication with S. Appendix A shows an example of the messages exchanged between NE and S for illustration purposes.

A. Discovery protocol

1. If S wants to send and/or receive metadata to/from NE, then it should publicly advertise TLS-AUX support through two custom options in its SSL certificate.
   - The option TLS-AUX-Support may have the values of SEND, RECEIVE, or SENDRECEIVE, indicating whether the server is interested in only sending metadata, only receiving metadata or both.
   - The option TLS-AUX-NegotiationURL may contain the URL where any interested NE can negotiate the format of the metadata and cryptographic keys to be used (if applicable) for the payload of the AUX records.
2. If NE has the ability to observe traffic flow, then it may discover all the server IPs that serve TLS traffic to clients downstream from NE, as well as the domain names hosted by them, from either the certificate or the SNI part of the handshake. Further, NE may identify the subset of the servers that advertise TLS-AUX support in their server certificates. For TLSv1.2 or below, NE should do so by observing the server certificates on the wire. For TLSv1.3 or higher, the certificates sent by a server are encrypted and thus cannot be examined by NE. In this case, NE should explicitly initiate a TLS handshake with SNI to S for each known domain name served by S. It should thus retrieve all the domains' certificates from S and examine them to discover which domains support TLS-AUX. Business rules configured in NE may be used to further short-list the servers that NE should engage in TLS-AUX communication.

B. Negotiation protocol

Once NE has decided to send and/or receive metadata to/from a server (S), NE should then initiate the negotiation protocol. The protocol is run out-of-band from live-user traffic.

1. All of the protocol negotiations must happen over a TLS connection that NE initiates to S. NE must insert a zero-length TLS-AUX record into this connection after the TLS connection is established, before any further communication. If the insertion of this record gives rise to a TLS error, then NE must conclude that either S or an invisible network element on the network path to S does not support TLS-AUX. The protocol fails and NE must not insert any TLS-AUX records into connections destined to S.

2. If step 2 does not fail, then NE should make a POST request to the negotiation URL provided in the server's certificate. In the POST body, NE must include a request for the metadata schema and encryption methods supported by the server.
3. Upon receiving the POST request body from NE, S must decide if it wants to engage in metadata communication with NE. To decline, it must respond with an HTTP 403 (Forbidden) response. If S decides to accept this request, it should create a new TLS-AUX session for metadata exchange between NE and S. To notify acceptance, it must respond with an HTTP status 200 (OK). In the response body, S must provide a session-id, the types of metadata that it is willing to send and/or receive, the crypto algorithm, a key, and the expiration time of the key. The key is for the encryption of the payload of TLS-AUX records exchanged as part of the session. Both NE and S must associate the key with the session. If S wants the payload of the TLS-AUX record to be in plaintext, then the crypto algorithm should indicate so, and the key would be irrelevant.
4. If NE's POST request indicated that it might want to receive metadata from S, then S must include a zero-length TLS-AUX record into this connection before the response is sent.
5. Upon receiving an HTTP 200 response, NE must identify the subset of the server's metadata types that it wants to send and/or receive. It then must make a second POST request to the same URL, identifying the subset of interest in the send-and-receive direction in the session.
6. Upon receiving the second POST, S must respond either with an HTTP 200 (OK) to confirm or HTTP 403 (Forbidden) to deny, which concludes the negotiation.

Upon the completion of the steps of the negotiation protocol, S and NE have an established TLS-AUX session with a common knowledge of the crypto scheme, key, and expiration time of the session. The sender of the metadata should use the key to encrypt or sign the metadata sent through TLS-AUX records. The key cannot be used after the expiration date. The TLS-AUX session may be renegotiated by NE any time to get a valid key.

Internet routes are not stable and can change over time. This means that any new incompatible network element may get introduced in the path between NE and S within the lifetime of an existing TLS-AUX session, which can deny traffic until the end of the established session's lifetime. A renegotiation of the session after expiration would remedy the situation. It can be argued that network elements that examine TLS record structures may be present near the endpoints of the connection (e.g., services in the Gi-LAN of mobile networks or security appliances in the datacenter where servers are hosted), rather than in the public Internet paths. Therefore, path changes on the Internet are far less likely to bring about this situation than a change of network configuration near the endpoints. Since the networks and servers are willful participants in TLS-AUX sessions, they can be expected to avoid this situation through their operations processes.

C. Session identification

As long as a valid TLS-AUX session exists between NE and S, they may insert/remove TLS-AUX records into users' data flows. However, network middle-boxes are often transparent or are located in private network space, which makes them invisible to S. As a result, when a user's TLS connection arrives at S, it may not be possible for S to tell which TLS-AUX session it may belong to, which is required to locate the decryption key for deciphering the metadata contained in the TLS-AUX record and knowing which metadata parameters have been negotiated for that session. To help S identify the session, if NE intends to exchange metadata with S for any TLS connection, then it should send a TLS-AUX record to S containing the session-id in the beginning of the TLS connection establishment. S must not send any TLS-AUX records on any connection unless it is able to identify the TLS-AUX session for which the connection belongs.

Although the protocol is described above in the context of one NE and one S, it is extensible to an environment with multiple NEs along the path between clients and S. In such an environment, each NE must negotiate its own TLS-AUX session with the server, and a TLS connection between the client and the server may belong to multiple TLS-AUX sessions. The library description below assumes a single TLS-AUX session for each connection. An extension to support multiple sessions is straightforward.

3.3 Implementation of a TLS-AUX library

At any entity that terminates a TLS connection, AUX records must be processed and removed before they can reach the TLS layer. We present here a method to implement the library for reference. It is a user-level library to be linked with the application at compile time with a few specific system-call wrapper options.

Figure 2 below shows the software layering arrangement. Figure 3 shows the library API that the AUX layer provides to the application to send/retrieve metadata.

In order to identify a connection's TLS-AUX session and to send/receive metadata on a connection, the application should call the API functions and provide the file-descriptor of the TLS connection. Any responsibility of handling the encryption or signing of the metadata in TLS-AUX records lies with the library, and the application is expected to work with unencrypted metadata.

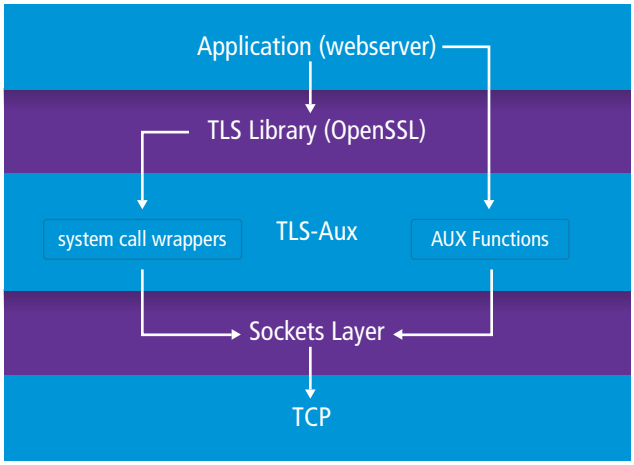

Figure 2: Layering of the TLS-AUX library

```
void setSessionDetails (int authType, int key) ;
int get_tlsaux_item_count(int fd) ;
int read_tlsaux_item_size(int fd, int item_id) ;
int read_tlsaux_item(int fd, int item_id, void *buf, size_t count) ;
int delete_tlsaux_item(int fd, int item_id) ;
int write_tlsaux_item(int fd, void *buf, size_t count) ;
```

Figure 3: TLS-AUX functions

System-call wrapping

TLS libraries, such as OpenSSL, are user-space libraries, and they use TCP sockets for network communication. They use the standard system calls such as open(), close(), read(), readv(), write(), writev(), listen(), and accept() with underlying sockets. The TLS-AUX layer intercepts the calls between the TLS library and sockets by using the mechanism of system call wrapping. The TLS-AUX library hijacks all of the above system calls so that when the TLS library makes one of these system calls, the corresponding function of the TLS-AUX layer gets called. The layer can then invoke the original system call in addition to performing additional actions, as described below.

Receiving metadata

When the TLS layer calls read() to receive data from the network, the TLS-AUX layer's read wrapper function gets called. This function, in turn, invokes the original system call read() to receive data from the network. The read() wrapper function then operates on the data received from the network and separates TLS-AUX records from the original TLS records. The metadata received from the TLS-AUX records on any TLS connection is decrypted or stored in a memory table corresponding to that connection. The leftover data is then passed on to the TLS layer as part of the read() call. When the application calls the API functions such as get_tlsaux_item_count(), read_tlsaux_item_size(), read_tlsaux_item(), etc., the TLS-AUX layer answers the calls using the table that stores these items for each connection.

Sending metadata

As mentioned previously in section 3.2C, metadata may not be sent on a TLS connection unless its TLS-AUX session(s) can be identified. When it is known, and the application layer calls write_tlsaux_item(), The metadata to be sent is stored in another table that stores outgoing metadata for each file descriptor. When the TLS layer makes a write() call on the connection, the TLS-AUX layer's write wrapper function gets called. This function, in turn, is responsible for writing TLS-AUX records into the connection, followed by the original encrypted TLS records received, as the argument of the write() function.

Handling encryption of the AUX frames payload

Once the TLS-AUX session has been established by the application with an NE, it can set the session's encryption details in the library using the setSessionDetails() function. The library can then use the encryption method and key to decrypt metadata in the TLS-AUX messages or to check signatures for validity.

Housekeeping

The wrappers for system calls accept() and close() perform all housekeeping tasks.

3.4 Security considerations

The intent of TLS-AUX is to allow an in-band communication of auxiliary data without weakening the secrecy guarantees of the end-to-end encrypted communication. Whether the TLS-AUX layer suggested above weakens the secrecy guarantees of end-to-end encrypted communications is one of the most important questions that must be answered as part of its proposal.

First, this layer does not need the SSL certificate for its operation at the connection's endpoints or in the network elements in the middle. Second, by virtue of its position below TLS, the TLS-AUX layer can be considered a man-in-the-middle entity for the TLS connection. In this position, it has the same vantage point as that of other entities in the middle, such as wifi devices, firewalls, switches, routers, DPI boxes, etc. TLS-AUX, like any of these entities, is neither able to get access to the session keys of the TLS connection, nor influence the mechanism used to negotiate them. Its ignorance of the session key prevents it from deciphering the contents of encrypted communication and writing correctly encrypted data into the connection on behalf of either endpoint.

The lack of need to host any SSL certificate in this layer and the ignorance of the session key means that TLS-AUX cannot weaken end-to-end encryption.

3.5 Privacy considerations

It can be readily concluded from the material presented in sections 3.1 and 3.2 that TLS-AUX is a vehicle for free exchange of any negotiated types of metadata between NE and S, and that any undesirable usage of this mechanism (like disclosure of privacy-sensitive metadata from NE to S) is not prevented. In this section, we discuss how to avoid TLS-AUX from getting used for undesired purposes.

First, we observe that TLS-AUX, the in-band vehicle for metadata, does not enable NE and S to exchange metadata (sensitive or otherwise) any more than they already are. Presently, NE and S do possess the ability to exchange metadata through out-of-band vehicles such as ICMP messages, TLS tunnels, or APIs specially hosted for this purpose.

One method to prevent an undesirable use of TLS-AUX is to create an unpermissive environment for TLS-AUX to operate within. It is possible to implement TLS-AUX as a layer below TLS only because the type and version fields of TLS records are not encrypted. If in a future version of TLS these fields are encrypted, then AUX records cannot be defined by any layer external to TLS. This would shut down the TLS-AUX layer entirely, thereby preventing it from getting used for undesirable purposes. A more interesting research problem is that of keeping the vehicle but preventing undesirable usage.

Metadata privacy model

We break down the task of preventing the undesirable use of TLS-AUX into two parts - (1) defining the limits on the metadata that may be exchanged by NE and S, and (2) implementing a mechanism to enforce these limits.

To address (1), we use the work of Nottingham et. al.[11], who prescribes restricted vocabulary and transparency for metadata that NE and S may exchange. This model restricts metadata to only the parameters that are agreed upon by an inclusive community that represents all of the relevant interests from network management to users' privacy. We expect that sensitive metadata types will be kept out of the vocabulary through this process. TLS-AUX is compatible with constrained vocabulary and transparency. Constrained vocabulary can be implemented by making the negotiation schema an established open standard. As part of the discovery and negotiation protocol, S is required to be transparent. It needs to include the metadata of its interest into its SSL certificate, open for anyone to examine. Further, if plaintext and signed metadata are the only two modes allowed for TLS-AUX records, then it allows for third-party examination and verification of the metadata communicated between NE and S.

Collusion prevention through transparency

This model establishes a standard for metadata, but it does not prevent colluding implementations of NE and S from exchanging extra-vocabulary metadata that may violate users' privacy. The part (2) of this model needs to be a deterrent to the use of extra-vocabulary metadata. We found that a deterrent can only be implemented with the help of a checker external to NE and S. The TLS-AUX model described so far seeks no additional functionality in the standard TLS protocol, and it does not assign any role to entities other than NE and S. For implementing a deterrent, we need to assume that TLS-AUX is a part of the TLS layer and define a new TLS extension (aux_disclosure). We assign the checker's role to the client (C). Browsers and apps are the intended instances of C. By doing this metadata transparency is extended and enforced all the way to C.

As part of the initial TLS handshake, S must disclose to C the types of metadata that were negotiated for the TLS-AUX session for which the connection belongs. The new TLS extension (aux_disclosure) must be used for this purpose. C may then apply its privacy policy to determine if the metadata types are acceptable. If not, it can choose to take some action based on policy, such as warning the user for privacy violation in the network or terminating the TLS handshake. The possibility of a large number of clients receiving warnings on privacy violations and/or refusing to communicate should deter both NE and S from exchanging extra-vocabulary metadata.

Client control

Users of mobile devices such as phones and tablets are familiar with privacy control settings of sensitive information such as the geographic location of the device. Devices have a default policy on whether the information should be accessible to an app on the device, and the users have an ability to override the default for chosen apps. Similar to this, we propose that networks should provide a well-publicized default metadata privacy policy to users with respect to the allowed vocabulary and provide an overriding control that allows users to flip the policy for any metadata type for any server (S) of their choosing. The networks should require all of their network-based services to follow the user's privacy policy.

4. Applications of TLS-AUX

In this section, we describe some network-based services that are implemented using HTTP headers as the vehicle for metadata exchange, which are incompatible with end-to-end encryption.

4.1 Throughput and data-plan guidance to media servers

This is an example of the case in which a mobile network wants to send metadata to media servers (CDN and otherwise) located on the public Internet so that media delivery can be optimized. The two main uses cases are:

a. As is well known, in mobile networks, users experience highly varying radio conditions over time, which gives rise to rapidly fluctuating achievable bandwidth on their connection. This factor significantly hurts media QoE for mobile users. Various network elements in the radio network may have the instantaneous bandwidth estimates for every mobile user, and they may be able to provide this bandwidth guidance to both client and server. MPEG-DASH's SAND initiative[9] is a good example, which needs an ability to exchange metadata to make network elements DASH aware.

b. Mobile end users typically have limited data plans. Watching long high-resolution videos can exhaust the data plan very quickly. Providing the user's data-plan info and status to media servers may allow them to adapt the quality of media delivered to suit the user's data plan. Media content providers that make media delivery sensitive to a user's data-plan status stand to gain in user engagement, as users are likely to watch more video on these sites instead of competitive sites that are data-plan unaware.

4.2 Cell-id information to web servers

This is another use-case in which a mobile network sends metadata to media servers located on the public Internet to optimize network resource usage and deliver a better QoE to its subscribers. 3GPP specifications on radio networks offer an on-demand operation of the multicast service, which is termed as MOOD[8]. This facility is meant for webservers to identify if a large user population in a mobile network is simultaneously consuming the same content and if suitable, move them all over to multicast delivery. Since radio multicast is highly spectrum efficient, such an opportunistic move to multicast helps not only the network conserve its radio resources, but also helps end users get better performance than on unicast.

The webserver knows the URLs of popular content it serves and the client IP addresses. But in order to initiate MOOD, it needs to identify the cells where the users should be advised to switch to multicast delivery. If the network provides the users' cell-identifier metadata with the requests, then the media server can perform clustering of the users, identify if any cells qualify, and invoke MOOD in those cells. The cell-id here only needs to be an opaque handle, which need not reveal any location information of the users.

4.3 TCP parameters hints for middle-boxes

This is an example of the case in which a web-server on the public Internet wants to send metadata to a proxy located inside the private network space of a mobile network.

The path from the server to the end user may be thought of in two segments – a wired network segment from the server to the radio transmitter and a last-mile radio segment. The two segments have very different performance characteristics. Segmenting the end-to-end TCP into two pieces corresponding to these network segments and using different flow-control parameters on the two segments is desirable from a performance perspective. Such TCP optimization proxies are commercially available. In the future, they may be available in the future as MEC applications[10], which extend the server's functionality on the radio transmitter itself. Web servers may want to provide these middle-boxes some hints on the TCP flow-control parameters that should be used for the radio segment.

For all of the above use cases, if TLS-AUX is adopted by the network services and the webservers, it may be seen as the replacement for the functionality provided by HTTP header enrichment for the end-to-end encrypted traffic. For the vendors of such services, extending the service to work with encrypted traffic involves only a round of software development. For the operators, it does not translate to additional infrastructure.

5. Conclusion

In this paper, we considered the issue of some network services becoming obsolete from the rapid rise in the use of end-to-end encryption for web traffic. We expect to see research and development activity in this area, resulting in proposals for new solutions in which network services are made to work with encrypted traffic without compromising on communication privacy needs. In this paper, we presented TLS-AUX, an application-layer method for in-band communication of metadata with end-to-end encrypted traffic. This technique is implemented as a software layer between the TCP and TLS layers. It requires no changes to either layer. By virtue of having no access to the private key of the SSL certificate or the keys of end-to-end encryption, this technique cannot lower the level of communication privacy in any manner. The out-of-band discovery and negotiation protocol ensures that the exchanged auxiliary information itself cannot be accessed or modified by unauthorized middle entities. Further, we show how this method may implement a deterrent for misuse (such as sharing of sensitive user data without consent). We learned that the deterrent must be implemented within the TLS layer, not below it. This means a satisfactory implementation of TLS-AUX must not be done independently of TLS, but rather within it.

At the time of writing this paper, we do not know how the main issue with end-to-end encrypted traffic and network services will finally be settled. Our solution presented in this paper demonstrates that many of the valuable use cases do not need the end-to-end encryption model to make the major change of formally accommodating network services in the middle. We present this solution not as a future product or protocol in itself, but simply as an example of a limited solution that can help the ongoing discussion of network services and encryption move forward.

References

1. D. Naylor, A. Finamore, I. Leontiadis, et al. The Cost of the "S" in HTTPS. In Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, CoNEXT '14, pages 133–140, New York, NY, USA, 2014. ACM.
2. Network Management of Encrypted Traffic, position paper of the GSMA, 28 February, 2015, http://www.gsma.com/newsroom/wp-content/uploads/WWG-04-v1-0.pdf
3. David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego Lopez, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, Peter Steenkiste, multi-context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS, SigCOMM 2015
4. J. Hildebrand, B. Trammell, Substrate Protocol for User Datagrams (SPUD) Prototype, http://datatracker.ietf.org/doc/draft-hildebrand-spud-prototype/
5. Mobile Throughput Guidance Inband Signaling Protocol, A. Jain, A. Terzis, H. Flinck, N. Sprecher, S. Arunachalam, K. Smith, https://datatracker.ietf.org/doc/draft-flinck-mobile-throughput-guidance/
6. The Transport Layer Security (TLS) Protocol, Version 1.2https://tools.ietf.org/html/rfc5246
7. Saguna press release, http://www.saguna.net/news-events/press-releases/akamai-and-saguna-win-best-innovation-based-on-network-intelligence-award-at-mwc-2015/
8. Multimedia Broadcast/Multicast Service (MBMS) improvements; MBMS operation on demand, http://www.3gpp.org/DynaReport/26849.htm
9. MPEG DASH Requirements for a webpush Protocol, https://tools.ietf.org/html/draft-begen-webpush-dash-reqs-00
10. Mobile Edge Computing - Introductory Technical White Paper, https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf
11. M. Nottingham, J. Hall, N. ten Oever, W. Seltzer, User Impact of Transport Metadata, draft-nottingham-transport-metadata-impact-00, https://github.com/mnot/I-D/blob/gh-pages/transport-metadata-impact/draft-nottingham-transport-metadata-impact-00.txt

Appendix A: Messages exchanged in the discovery and negotiation protocol

In this section, we show the negotiation messages exchanged between the server (S) and the network element (NE). We do not prescribe a specific data format on the wire, XML messages are used here for easy illustration purposes only.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="uri:akamai.com/tlsaux/1.0"
        xmlns="uri:akamai.com/tlsaux/1.0"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

<xs:complexType name="TLSAUXNegotiation">
   <xs:sequence>
    <xs:element name="SessionId" type="xs:String" minOccurs="0"/>
    <xs:element name="SendNegotiation" type="Item" minOccurs="0"/>
    <xs:element name="ReceiveNegotiation" type="Item" minOccurs="0"/>
    <xs:element name="CryptoNegotiation" type="Crypto" minOccurs="1"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Item">
   <xs:sequence>
    <xs:element name="Name" type="xs:string" minOccurs="1"/>
    <xs:element name="Type" type="xs:string" minOccurs="1"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Crypto">
   <xs:sequence>
    <xs:element name="Key" type="xs:string" minOccurs="0"/>
    <xs:element name="Algorithm" type="xs:string" minOccurs="0"/>
    <xs:element name="Expires" type="xs:dateTime" minOccurs="0"/>
    <xs:element name="Renew" type="xs:boolean" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>

</xs:schema>
```

Examples of negotiation messages

S to NE message advertising the entire set of metadata that are meaningful to the server and the crypto details

```
<TLSAUXNegotiation>
 <SessionId>06478a21ad5408a81d348dc89fbf3360</SessionId>
  <SendNegotiation>
   <Item Name="Bitrate" Type="Integer">
   <Item Name="TrafficClass" Type="String">
```

```
    <Item Name="RadioTCPOPT" Type="String">
  </SendNegotiation>
  <ReceiveNegotiation>
    <Item Name="ThroughputGuidance" Type="Integer">
    <Item Name="Userid" Type="String">
  </ReceiveNegotiation>
  <Ctypto>
    <Key>30f07b1567883785 6f878c43bc183735</Key>
    <Algorithm>RSA</Algorithm>
    <Expires>2014-11-05T00:10:00Z</Expires>
  </Crypto>
</TLSAUXNegotiation>
```

The response from NE to S confirming that in this session, it would like to send only the throughput guidance metadata to S and would want S to send only the bitrate to NE.

```
<TLSAUXNegotiation>
 <SessionId>06478a21ad5408a81d348dc89fbf3360</SessionId>
  <SendNegotiation>
    <Item Name="Bitrate" Type="Integer">
  </SendNegotiation>
  <ReceiveNegotiation>
    <Item Name="ThroughputGuidance" Type="Integer">
  </ReceiveNegotiation>
</TLSAUXNegotiation>
```

NE's request to S to renew the key after expiration

```
<TLSAUXNegotiation>
 <SessionId>06478a21ad5408a81d348dc89fbf3360</SessionId>
 <Crypto>
   <Renew>true</Renew>
 </Crypto>
</TLSAUXNegotiation>
```

S's response with a renewed key

```
<TLSAUXNegotiation>
 <SessionId>06478a21ad5408a81d348dc89fbf3360</SessionId>
 <Crypto>
   <Key>4b43f15f838925564d3cc0d0ddd0e40a</Key>
   <Algorithm>RSA</Algorithm>
   <Expires>2014-11-05T00:20:00Z</Expires>
 </Crypto>
</TLSAUXNegotiation>
```

Authors and Contact Information

Mangesh Kasbekar
Principal Architect, Platform
Akamai Technologies, Inc.

Vaishnav Janardhan
Principal Software Engineer, Media Development
Akamai Technologies, Inc.

Vinay Kanitkar
Chief Network Architect and Akamai Fellow
Akamai Technologies, Inc.

Manish Jain
Senior Architect, Platform
Akamai Technologies, Inc.

Corporate Headquarters: 150 Broadway, Cambridge, MA 02142

General Contact: +1 877 325 2624 (U.S. Only), +1 617 444 3000