INSTITUTE FOR DEFENSE ANALYSES

Sharing Smart Card Authenticated Sessions Using Proxies

Kevin E. Foltz

William R. Simpson

January 2016

Approved for public release; distribution is unlimited.

IDA Non-Standard
NS D-5417
Log: H 15-000067
Copy

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882

IDA

The Institute for Defense Analyses is a non-profit corporation that operates three federally funded research and development centers to provide objective analyses of national security issues, particularly those requiring scientific and technical expertise, and conduct related research on other national challenges.

About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under contract HQ0034-14-D-0001, Task BC-5-2283, "Architecture, Design of Services for Air Force Wide Distributed Systems," for USAF HQ USAF SAF/CIO A6. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

Copyright Notice

© 2016 Institute for Defense Analyses
4850 Mark Center Drive, Alexandria, Virginia 22311-1882 • (703) 845-2000.

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (a)(16) [Jun 2013].

Sharing Smart Card Authenticated Sessions Using Proxies

Kevin E. Foltz and William R. Simpson

Institute for Defense Analyses, 4850 Mark Center Drive, Alexandria, VA 22311, USA

Abstract: This paper discusses an approach to share a smart card in one machine with other machines accessible on the local network or the Internet. This allows a user at a browser to use the shared card remotely and access web applications that require smart card authentication. This also enables users to access these applications from browsers and machines that do not have the capability to use a smart card. The approach uses proxies and card reader code to provide this capability to the requesting device. Previous work with remote or shared smart card use either requires continuous access to the smart card machine or specific client software. The approach in this paper works for any device and browser that has proxy settings, creates minimal network traffic and computation on the smart card machine, and allows the client to transfer from one network to another while maintaining connectivity to a server. This paper describes the smart card sharing approach, implementation and validation of the approach using real systems, and security implications for an enterprise using smart cards.

Key words: Smart card, IT security, authentication, key management, proxy, SSL, TLS, session stealing.

1. Introduction

This paper considers the smart card, a common hardware-based certificate and key store, and looks at ways to share access to the private key. This allows different individuals on different machines in different locations to access two-way authenticated TLS secured web applications using the same smart card.

The TLS (Transport Layer Security) protocol is used to provide authenticated, confidential communication with integrity for higher-layer protocols [1]. Based on the earlier SSL (Secure Sockets Layer) protocol [6], TLS has become the standard choice to provide these security properties on the web. The most common use of TLS is for HTTP, but other protocols that run over TCP can be modified to run over TLS. A simple TLS implementation uses a server certificate to authenticate the server's identity and exchange key material that is used to set up a secure channel for communication. An additional option involves the use of a client certificate to authenticate the requester. This two-way authenticated TLS provides a secure method for a user to log into a web application.

The certificates and keys used by the client can take different forms. Software certificates and keys are stored as files and loaded into the operating system key store or another file-based key store. Passwords may be used to encrypt the key store and private keys. Although private keys have many bits of entropy, the password protecting the key is often much shorter, so this is essentially password-based security plus some security by obscurity by protecting access to the encrypted key file.

Hardware-based certificates and keys offer stronger protection. The hardware is designed to provide only interfaces that use the private key but to never make the key itself available, even to the authorized user. There are methods to physically extract the key [7], but the hardware-based key is fundamentally stronger in protection than software-based keys with passwords since the private key is never exposed during use and is harder to duplicate.

Related work includes Refs. [8]. In Ref. [8],

Corresponding author: Kevin Foltz, Ph.D. in Electrical Engineering, research fields: cyber security and distributed systems. E-mail: kfoltz@ida.org.

alternatives to smartcard PKI are described which can facilitate sharing. In this paper we assume a standard smart card deployment.

In Ref. [9], which details the smart card proxy variant of Sykipot, the attack tool provides remote access to a smart card on a machine using an ActivClient DLL. It appears from Ref. [10] that the attack relies on collecting data at the victim and then transmitting it to the attacker. This is different from our paper, since we only require setting up the connection, but not transferring data, through the smart card machine.

In References [11, 12], a malicious USB driver provides an interface over TCP/IP to an attacker. The attacker can use a local copy of the smart card DLL to access a remote smart card by sending the raw USB commands from the attacker machine through TCP/IP to the remote USB driver. The attacker is required to run the same smart card software as the victim, which does not allow the use of alternative devices or operating systems, such as smart phones or tablets that do not support such software. Our approach does not require a specific DLL and works on any device with a proxy configurable browser.

What our approach provides that others do not is a way to share a smart card with any device running a browser using minimal network traffic to the smart card machine and no low-level hardware or operating system changes. This paper discusses the approach, its validation through real-world implementations, and enterprise-level security considerations and mitigations.

2. Description

This section lays some of the groundwork for our approach. The approach is based mainly on the use of proxies, and relies on certain technical implementation details of the TLS protocol and session management.

2.1 Proxy Terminology

The approach of providing access to smart-card-enabled sites described in this paper involves the use of proxies. The types of proxies used in this paper are the HTTPS proxy, TLS proxy, and TCP proxy, described below. Each proxy accepts multiple incoming connections concurrently and maintains a mapping of incoming connections to outgoing connections.

2.1.1 HTTP Proxy

The HTTP proxy function is defined as part of the HTTP protocol specification [13]. The client chooses the proxy and changes HTTP headers for use by the proxy. For example, the Host header tells the proxy where to forward the request. Requests are sent by the client to the HTTP proxy instead of the intended server. The proxy then relays client requests to the desired server and sends the responses back to the browser. The proxy speaks to the server on behalf of the client, and it speaks to the client on behalf of the server. The server may or may not know that it is speaking to a proxy, depending on the HTTP headers that are sent by the proxy to the server. Proxies can be chained so that there are multiple proxies in series between the client and server.

2.1.2 HTTPS Proxy

An HTTP proxy normally examines the HTTP headers to know where to forward requests. When HTTPS is used, the HTTP header and body are encrypted using TLS, so this header inspection is not possible. In this case a connect request is first used to tell the proxy to connect to the desired server. The proxy then sets up a TCP connection to the server, notifies the requester, and awaits further messages. When TLS traffic flows from the client to the proxy, the proxy forwards it unmodified to the server and sends server responses back to the client, also unmodified. This is also part of the HTTP specification, but it is identified separately because not all HTTP proxies support HTTPS.

2.1.3 TLS Proxy

There is no description of proxies in the TLS specification. For this paper, a TLS proxy is defined as an entity that receives incoming TLS traffic and acts as the server side of the TLS connection. It sets up a separate TLS connection with another fixed entity and forwards content from one connection to the other. Unlike an HTTPS proxy, which does not view content, the TLS proxy decrypts the content it receives and re-encrypts it before sending, so it has the ability to read and modify the content as a MITM (man-in-the-middle). For this paper, it is assumed that the location to which the proxy forwards traffic is fixed, but in general it could be dynamic and based, for example, on the initial Client Hello message server name extension.

2.1.4 TCP Proxy

The TCP proxy is an entity that accepts an incoming TCP connection and sets up a new connection to a fixed IP address and port. It then forwards all content between these connections.

2.1.5 Chaining Proxies

Proxies can be chained in different ways. One example is a TLS proxy followed by an HTTPS proxy. Together these act as an entity that terminates the TLS connection, decrypts and re-encrypts content, and forwards requests to the appropriate web site. Unlike an HTTPS proxy, which forwards all TLS traffic as-is, this combination can change TLS handshake messages and encrypted content. This combination forms the core of the smart card access approach in this paper by modifying a non-smart-card connection with the client to be a smart card connection with the server.

2.2 TLS and Application Protocols

This section discusses aspects of TLS and application sessions relevant to this paper.

2.2.1 TLS Sessions, Connections, and Keys

A TLS session is established using a TLS handshake, which negotiates security parameters and performs authentication. Authentication of the client is accomplished by the client performing a private key operation on a hash of the handshake messages exchanged between the client and server. The server then uses the public key from a certificate signed by a trusted issuer to validate the client’s identity. This handshake process establishes a new TLS session and opens a connection within this session. The session has an associated “master secret,” which is used to generate key material for each new connection within that session. New connections can use a short handshake sequence to reuse the existing master secret and avoid expensive authentication and key exchange steps.

Fig. 1 shows the full handshake sequence to set up a client-authenticated TLS session. Fig. 2 shows the abbreviated handshake sequence, which is used to set up a new connection within an existing session. The two Certificate messages, the Client Key Exchange message, and the Certificate Verify message, which create most of the network traffic and computation, are skipped in the abbreviated handshake.

Fig. 3 shows the dependency graph when computing the encryption keys and MAC secrets for the full handshake. The random values are exchanged between the client and server. The premaster secret is either sent encrypted or computed in a distributed way between the endpoints. The master secret is a deterministic computation based on the random values and premaster secret. The key material is a similar deterministic computation based on the random values and master secret.

Fig. 4 shows the dependencies for the abbreviated handshake. The client and server random values are exchanged as in the full handshake, but the master secret is the value stored from the session initiation. The computation of key material is the same deterministic calculation as for the full handshake.

Fig. 5 shows the relationship within a session of the key material for different connections. The initial connection uses the full handshake to establish a new session, and subsequent connections in the same session use the abbreviated handshake.

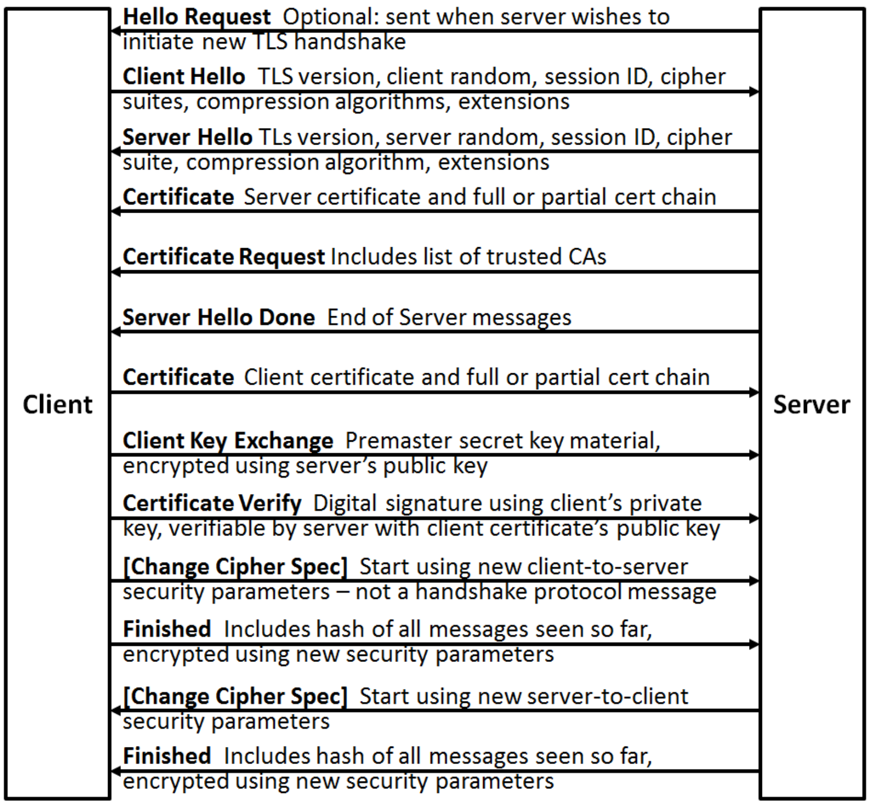
Client
Server
Hello Request Optional: sent when server wishes to initiate new TLS handshake
Client Hello TLS version, client random, session ID, cipher suites, compression algorithms, extensions
Server Hello TLs version, server random, session ID, cipher suite, compression algorithm, extensions
Certificate Server certificate and full or partial cert chain
Certificate Request Includes list of trusted CAs
Server Hello Done End of Server messages
Certificate Client certificate and full or partial cert chain
Client Key Exchange Premaster secret key material, encrypted using server's public key
Certificate Verify Digital signature using client's private key, verifiable by server with client certificate's public key
[Change Cipher Spec] Start using new client-to-server security parameters – not a handshake protocol message
Finished Includes hash of all messages seen so far, encrypted using new security parameters
[Change Cipher Spec] Start using new server-to-client security parameters
Finished Includes hash of all messages seen so far, encrypted using new security parameters

Fig. 1 TLS full handshake sequence.

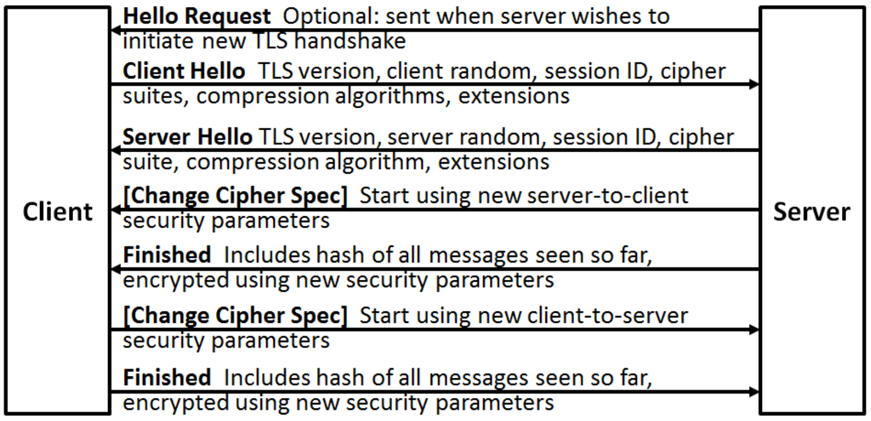
Client
Server
Hello Request Optional: sent when server wishes to initiate new TLS handshake
Client Hello TLS version, client random, session ID, cipher suites, compression algorithms, extensions
Server Hello TLS version, server random, session ID, cipher suite, compression algorithm, extensions
[Change Cipher Spec] Start using new server-to-client security parameters
Finished Includes hash of all messages seen so far, encrypted using new security parameters
[Change Cipher Spec] Start using new client-to-server security parameters
Finished Includes hash of all messages seen so far, encrypted using new security parameters

Fig. 2 TLS abbreviated handshake sequence.

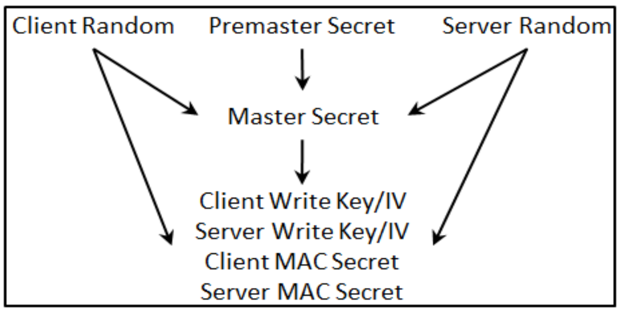

Fig. 3 Key calculation dependencies, full handshake.

The premaster secret is only shared or computed during the first connection within the session. The master secret is computed based on the random values, client random #1 and server random #1, exchanged during this first connection. The key material for a connection is computed from the master secret for the session and the random values for that connection. Although the master secret is identical for all connections, each connection has different random values, providing different key material for each connection.

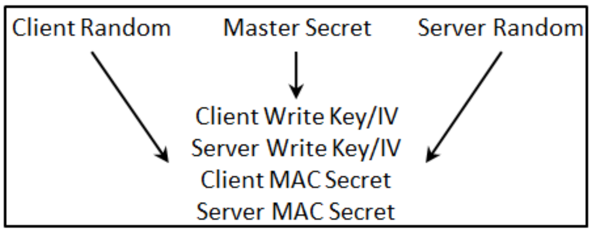

Fig. 4 Key calculation dependencies, abbreviated handshake.

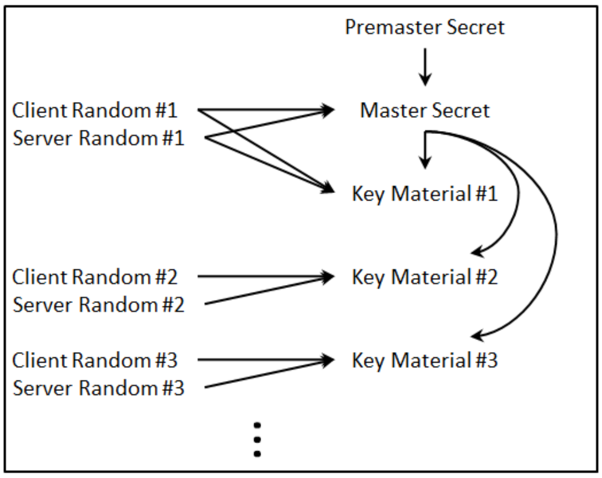

Fig. 5 Relationships between key material for different connections within a session.

2.2.2 Using Identical Key Material

This section examines the feasibility of using the same key material on both the client and server-side connections of the proxy. This behavior enables the client to use a smart card proxy to establish a TLS connection to the server and then drop the proxy and continue the connection directly to the server. The key material for a connection is determined by the client and server random values and the premaster secret. For RSA key exchange, a TLS proxy can view and modify all of these values. In particular, the proxy can establish identical key material for the client-side and server-side connections. This works even when the client certificate is changed by the proxy.

For key exchange methods using PFS (perfect forward secrecy), the proxy is generally unable to create identical key material on both connections. If the proxy modifies the Certificate or Certificate Verify message based on the smart card certificate and private key, then the Finished messages must also be modified, since they include an encrypted hash of all handshake messages, but the proxy will have no knowledge of the keys needed to compute the new Finished messages.

With some additional inputs, the proxy may be able to create identical key material. For example, if the client and server share their premaster secret inputs with the proxy, then the proxy can establish identical keys for both sides of the connection. However, in general the proxy has no control over the server and no ability to extract this information. If the client shares the master secret or key material with the proxy, then this can be used to compute the new Finished messages. A “browser” such as the OpenSSL s_client tool makes master secrets available with the debug flag, but it is not generally used for browsing. With Firefox and other NSS library browsers, it is possible to extract premaster secrets or master secrets by setting an environment variable as described in Ref. [14]. Another option that could work with existing software is to extract keys from memory as demonstrated in Ref. [15]. These additional options have different degrees of portability across devices, and they require a method of collecting and passing values to the proxy and using these values at the proxy, so it is generally easier to try to negotiate RSA or another non-PFS key exchange method with the server if possible.

2.2.3 Application Layer Sessions

TLS session establishment is computationally expensive, so when the first authenticated TLS connection is established, the web application often takes measures to avoid repeating this process. It is possible to reuse the same session to set up new connections with the abbreviated handshake sequence, but if there is any fatal error on any of these connections, such as a flipped bit between client and server, the connection is closed and no new connections are allowed within the same session, so a new full handshake is required, including client authentication. A common method to avoid client re-authentication is to use a session cookie. This allows additional TLS sessions to be established using the cookie in place of client authentication. This behavior is one of many options available to the server, which include the following:

- Use the session cookie for authentication [intelshare.intelink.gov];
- Use the session cookie and requester IP address for authentication (i.e., a request with an active session cookie from a different IP address invalidates the session and necessitates a new TLS client authentication) [www.my.af.mil];
- Require re-authentication through TLS when clicking on certain links [disa.deps.mil];
- Require re-authentication for all new TLS connections;
- Require re-authentication for each request.

These choices are ordered generally by increasing security and decreasing user convenience. The first three have been observed on active sites. The last two are included as high security options but not observed, likely due to the inconvenience of frequent re-authentication. For example, parallel requests, such as JavaScript or image files associated with a page, would require separate authentication, which is impractical when many such content items must be loaded for a single page.

It is possible to transfer a TLS session across IP addresses. This is not common practice for clients, but nothing in the TLS specification prevents this behavior. This could happen, for example, when a mobile device switches between the mobile network and Wi-Fi during an application session. Knowledge of the master secret and some other non-cryptographic information is enough to establish a new connection within an existing session, regardless of the source or destination of the connection.

For proxies that act as TLS endpoints, the client connects to the proxy through one TLS session and the proxy connects to the server through a separate TLS session. The proxy simply relays the content sent through each TLS connection from the client or server to the other endpoint. Although these two TLS sessions are nominally different, we can cause them to have nearly identical parameters by proper implementation of the proxy. The parameters of interest include the master secret and each connection’s encryption keys (and IVs if appropriate) and MAC secrets, along with the cipher suites, compression algorithm, TLS version, and other information about the session and connections. Other values in the TLS exchange may be different, such as the server certificate and “Finished” values, but the important thing is that the master secret and cryptographic key values are all the same.

If connections are established in this way, it is possible that the encrypted content flowing on each connection is identical. The encrypted data will differ under certain conditions. First, if the content is modified by the proxy in any way, the content will differ starting with the modified content. This will most likely change all future content as well. Second, if the record boundaries are changed the MAC values will change. Since the MAC value is appended to the content before encryption, this will change the encrypted value the same way a modification to content would.

If the record boundaries are preserved and the content is copied from one side of the proxy to the other, then the sessions essentially have the same state and the encrypted application data going to and from the proxy will be identical. This allows the proxy to be removed and the two endpoints connected directly. Since the encrypted content is identical and the keys are the same, the two endpoints can resume the connection as if nothing changed.

Table 1 Common HTTP and SSL/TLS proxies and their properties.

Name	Smart-Card-Enabled?	Identical Sessions?
Burp Suite v1.6	YES http://portswigger.net/Burp/help/options_ssl.html	NO different premaster secret different record sizes modifies HTTP headers
Paros v3.2.13	YES (Andiparos fork) https://code.google.com/p/andiparos/	NO different premaster secret different record sizes modifies HTTP headers
Zed Attack Proxy v2.4.1	YES? https://code.google.com/p/zaproxy/wiki/SmartCards	NO different premaster secret different record sizes modifies HTTP headers
WebScarab	YES? https://www.owasp.org/index.php/WebScarab_SSL_Certificates	NO different premaster secret different record sizes modifies HTTP headers
Fiddler v4.6.0.2	NO, does not appear to support smart cards.	NO different premaster secret different record sizes different cipher suites

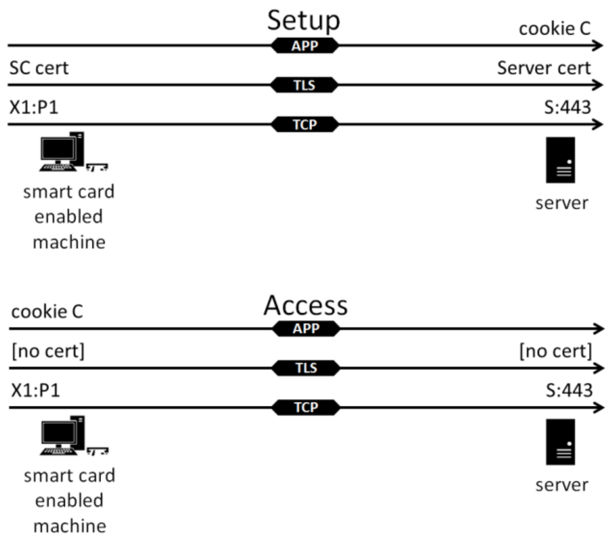

Fig. 6 Normal smart-card-based access.

A natural question is whether commonly available proxies that can read smart cards allow this behavior. Table 1 summarizes the behavior of some common proxies. Four of them claim to support smart cards, based on their documentation. However, testing them revealed that they change the cryptographic parameters and content from one TLS connection to the next. A TLS proxy was developed by the author that set up identical sessions with identical record boundaries and also enabled smart card client authentication. It is not known whether another proxy exists with these properties.

2.3 Normal Access with a Local Smart Card

Fig. 6 shows a normal interaction between a smart-card-enabled machine and a server. This is separated into the setup phase, where the initial authentication is established and the session cookie is provided by the server, and the access phase, where subsequent access is provided based on the session cookie without TLS authentication.

During setup, the browser on the smart-card-enabled machine establishes a TCP connection from its IP address and port, X1:P1, to the server IP address and port, S:443. A new TLS session is then established with a TLS connection over this TCP connection with the server. The initial connection may use only the server certificate, but at some point, such as a login screen, the server requests a client certificate for authentication in a TLS session. The requester uses the smart card to provide the certificate and verify identity with the private key. A session cookie, C, is sent by the server to the client so that future requests need not repeat the TLS-based client authentication.

After this initial setup stage, subsequent access to the application is granted using the session cookie, C, over the established TCP and TLS connections. The TCP, TLS, and application layers are separated in Fig. 6, with the IP address and port, certificates, and cookies listed at the appropriate layers.

The switch from one-way authenticated TLS to two-way authenticated TLS can happen in different ways. A completely new session can be created, such as by clicking a link to a new server. With a new session, no data is transmitted until the client is authenticated. Alternatively, TLS session renegotiation can be used to establish a new session while communicating on an existing TLS connection. The client certificate is requested and provided as part of this renegotiation. With renegotiation, client certificate information is encrypted within the original TLS connection. In either case, the smart card is required to set up the new two-way authenticated TLS connection.

2.4 Access with a Remote Smart Card

A number of methods of providing access to one machine by using a smart card on another machine are described in this section. The different architectures have varying complexity and difficulty of implementation, and they allow a requester to meet different access rules and restrictions at the server.

2.4.1 Access on the Same Network

Fig. 7 shows the network architecture to share a smart card using a single machine on the same LAN as the victim. The smart-card-enabled machine runs a TLS proxy and an HTTPS proxy in series. The remote machine is on the same LAN, connected to the Internet through a wireless router, which uses NAT (Network Address Translation). We assume the remote user is using a mobile device, but any device that can connect to the LAN works equally well.

The remote user communicates with the smart-card-enabled proxy through the local network. A new TCP connection is established from the device, X1:P1, to the TLS proxy, Xp1:Pp1. A TLS session may also be established using a proxy certificate and a client certificate. The smart-card-enabled proxy then communicates with the server through separate TCP and TLS connections. The TLS proxy authenticates through the HTTPS proxy to the server using the smart card. With the TLS connection successfully established, the browser sends HTTPS messages to the proxy, which forwards them to the server. The server sends a session cookie to the proxy, which forwards this to the requester. This completes the setup phase.

For subsequent access, the remote user disconnects from the proxy. New requests are sent directly to the server using a new TCP connection, a new TLS connection, and the same application layer session. The server sees only the external IP address of the wireless router, which is the same for the smart-card-enabled machine and device. As a result, the subsequent access simply looks like a new connection from the same machine using the same application layer session. If the server uses client authentication for the initial login and server authentication plus the session cookie for subsequent traffic, the remote user can continue the session, since the smart card will not be needed for further communication.

2.4.2 Switching Networks

In the previous subsection, the remote user is constrained to use the same network for setup and access. If the location of the smart card machine is not conducive for further access or if the network is not desirable, the remote user might want to leave the network between the setup and access phases. Fig. 8 shows the network architecture for a remote user who performs the setup as in Fig. 7 but then disconnects from the local network and reconnects to the server through a mobile connection for subsequent access.

In this case, the IP address of the remote user changes when switching networks, since it goes through a mobile provider instead of the local router. If the server allows new connections from a different IP address to use existing application layer sessions, the access is similar, but now the remote user can go anywhere with mobile coverage. If the server does not allow changes in IP address, then steps must be taken to comply with this server rule

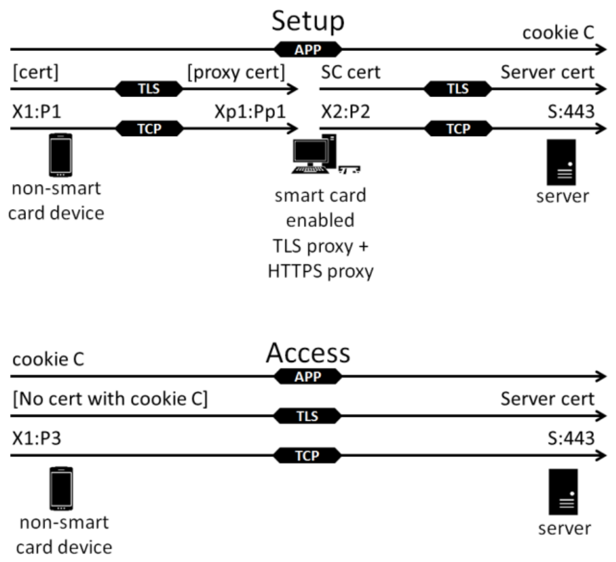

Fig. 7 Mobile device connects through proxy on the same network.

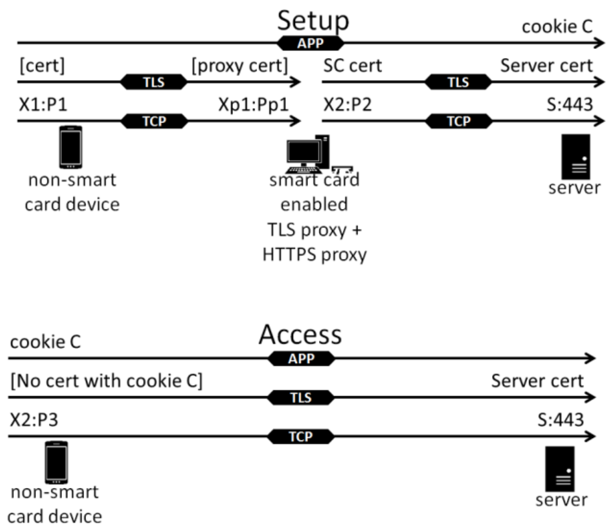

Fig. 8 Mobile device connects through local proxy, then switches to mobile network.

2.4.3 Switching Networks, IP Check at Server

When switching networks, the changing browser IP address provides a way for the server to detect the change between setup and access phases. Servers can be configured to tie a cookie to a requester IP address and invalidate the entire session if the cookie is presented from a different IP address. A more aggressive approach is to tie the cookie to an IP address and a TLS session.

If the server records and checks the client IP address associated with an application layer session, it can forcibly end sessions that change requester IP addresses. This stops the remote access in Fig. 8 from working. To work around this restriction, a second proxy, accessible on the Internet, can be used to provide a fixed IP address to the server. Fig. 9 shows the network architecture for remote access that uses an external HTTPS proxy. In this case, the server IP check will pass and the session will not be dropped, even though the original client's local IP address changes. The remote user could run the external proxy, but any reliable Internet accessible proxy is sufficient.

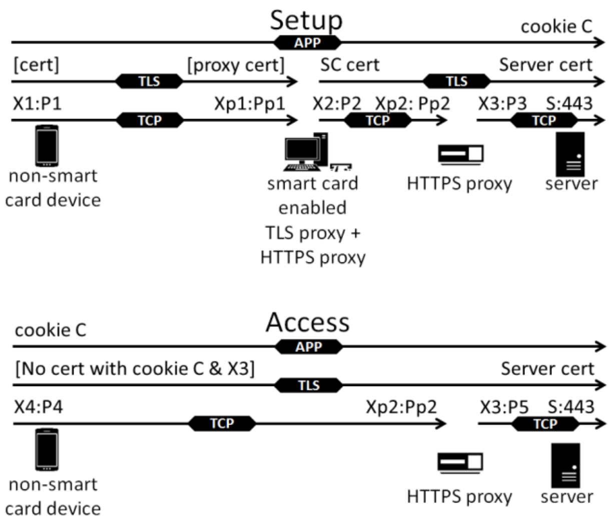

Fig. 9 Mobile device connects through local proxy and remote proxy, then switches to mobile network and remote proxy.

An alternative would be to host the external proxy on the remote user's device itself. The smart-card-enabled proxy would use the remote device as its proxy, and the remote device proxy would receive requests on the LAN and forward them to the server on the mobile link. This removes the dependence on an external entity. Also, since the proxy is hosted on the user's device, the remote user could simply remove the proxy and connect directly through the mobile link using the same IP address. This approach would require a proxy application for the device, which does not appear to currently exist.

2.4.4 Switching Networks, IP and TLS Checks

In all the previous scenarios, the browser establishes new TCP and TLS sessions with the server at the start of the access phase. It is possible that the server not only requires the same IP address, but also requires the same TLS session or connection. This generally limits functionality, since each new TCP connection requires a new TLS connection, and a single bit error in a single TLS connection requires setting up a new session for new connections. For highly secure applications, this is an approach that would stop all of the previous methods of access. To address this challenge, a new flow must be used that hides proxy changes from the client and server, so both the client and server see no changes between the access phase and the setup phase. This is where the proxy with identical key material on both connections is used.

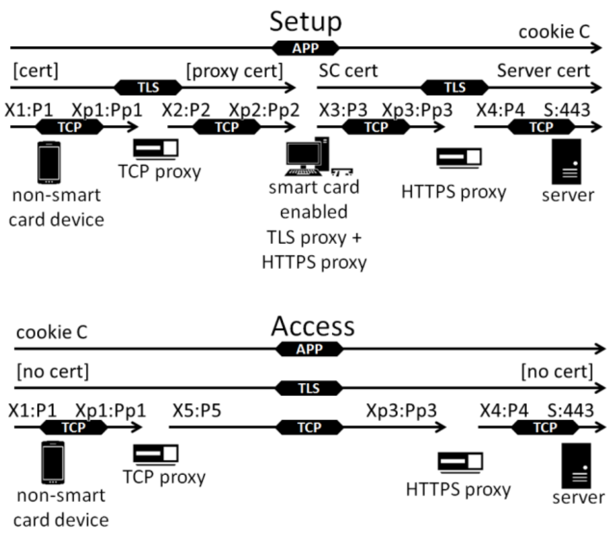

Fig. 10 Client connects through client, smart-card-enabled and server proxies, and then bypasses smart-card-enabled proxy.

Fig. 10 shows a setup where the smart-card-enabled proxy is shielded between two fixed proxies. This addresses any client-side issues when changing proxy settings, since the changes are made at the client proxy, not the client. The client-side proxy simply forwards incoming requests directly to either the smart-card-enabled proxy or the server-side proxy, so it could be implemented as a TCP proxy instead of an HTTPS proxy. The client-side proxy must be under the control of the remote user, since its settings must be changed to point to the server-side proxy after initially pointing to the smart-card-enabled proxy. When removing the smart card proxy, the client and server both continue using the existing connections. The remaining proxies create a new direct connection between them and remap incoming and outgoing connections to preserve the end-to-end flow from client to server.

After removing the middle proxy, the client and server will not realize that the TLS endpoint at the smart-card-enabled proxy has been removed. This creates a situation where the client side of one TLS connection is used with the server side of another TLS connection. The browser and server will both attempt to use sessions established with the smart-card-enabled proxy when they talk directly to each other. In order for this to work, the session keys for the two original sessions must be the same, so that when this intermediary is removed the client and server are using the same keys. In addition, any state information relating to block cipher initialization vectors, sequence numbers, and record boundaries must be matched between the two TLS connections.

The key material can be synchronized by using the same random numbers during the TLS handshake and the same premaster secret value during key exchange for each TLS connection. Some care is required to keep the other values in synch, but because there is only a brief exchange during setup, this is reasonable to achieve.

Instead of synchronizing all TLS connections, it may be easier to synchronize the session data, including the master secret. In this case existing connections will not be functional, since keys and other state information will be out of synch, but new connections within the same session can be established using the short handshake sequence based on this shared master secret. The short handshake uses the existing master secret and avoids use of the smart card for authentication. For this to work, all existing connections from the browser must be left open in an unused state. Any attempt to use them will result in an error which could terminate the entire TLS session and potentially require a new session with smart card authentication.

Another approach, instead of matching master secrets, is to compute different master secrets but then export the server-side TLS state information, including cryptographic keys, to the client [16]. In general, TLS session information is well protected. Methods exist to retrieve it, but they require low-level access on the smart-card-enabled proxy machine [14]. With custom proxy software and a custom browser, the remote user could provide a way to transfer TLS session information between proxy and browser.

Whatever method is used to recombine the two TLS sessions into a single session, the result is that the browser and server are communicating through two passive proxies. Similar to what was described previously, the proxies, with proper configuration, could both be implemented on the remote device, eliminating the dependence on external components during the access phase.

2.4.5 Separating Smart Card and Proxy

The TLS + HTTPS proxy combination does not need to run on the smart card machine. All that is needed on this machine is code to request certificates and private key operations and a method to communicate with an external machine. A lightweight application could do this by receiving queries on an open port and sending back the results of smart card operations. This splits the functionality of the smart-card-enabled proxy into the proxy function and the smart card access function. Fig. 11 shows the logical connections using the single smart-card-enabled proxy, and Fig. 12 shows the split functions of proxy and smart card access. The smart card reader code is invoked only to perform smart card specific actions, which minimizes the use of the smart card machine.

This separation of proxy and smart card reader functionality can be applied to any of the variants mentioned above. It is particularly useful for the setup in Fig. 10. In this case it removes the need to disconnect from the TLS proxy, since only the private key usage request is sent to the smart card machine. However, for performance and stability reasons, it may still be beneficial to bypass the proxy after client authentication.

2.4.6 Potential Vulnerabilities and Attacks

The variants above would be attacks if the smart card machine user does not know that the remote user is using the smart card. The technical implementation is identical.

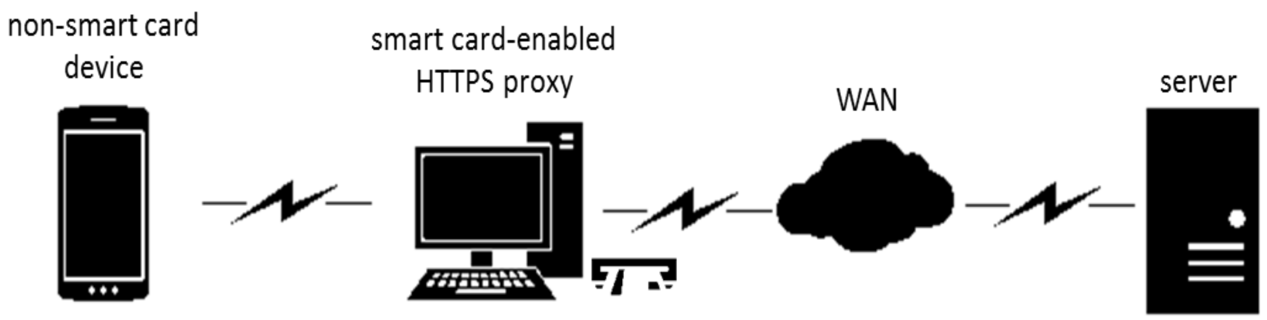

Fig. 11 Combined smart card and proxy machine.

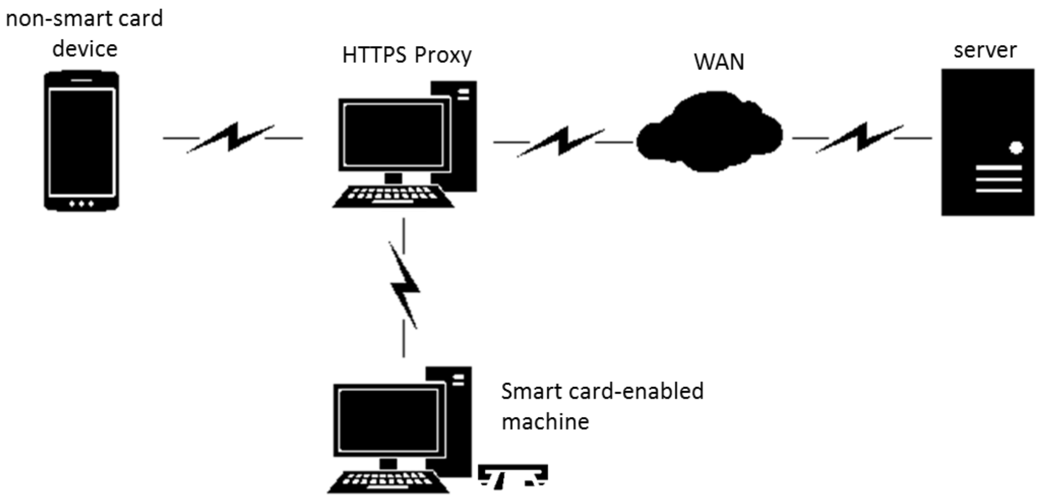

Fig. 12 Separate smart card and proxy machines.

The keys to the attack would be running the proxy or smart card access code on the victim's machine, connecting to the machine to use the proxy or smart card access code, and hiding this activity from the smart card user, scanners, and other detection appliances.

One approach is for the attacker to set up the proxy on a public machine and wait for someone to use their smart card on it. Another is to convince someone to use the attacker's machine, while the attacker or another attacker accesses the smart card remotely. These rely on social engineering.

Another approach is to install the proxy or smart card reader code on the victim's machine. This could be done using malicious email attachments, physical access, or by leaving USB devices in places where people with smart cards might pick them up and plug them in. The USB device would actually be the Rubber Ducky USB, which can use keystrokes to configure a remote machine for smart card access by an attacker [17].

3. Results and Discussion

Some of the variants described have been implemented and tested. Others remain theoretical. This section describes the scenarios that have been implemented and tested, with a discussion of the implications to an enterprise using smart cards for access.

3.1 Session Cookie with No IP Check

The first scenario that was tested was the use of a TLS and HTTPS proxy on a Windows 7 desktop with a smart card reader. The desktop was connected by Ethernet to a wireless router. An iPhone 5S was connected to the wireless router through Wi-Fi, with the desktop configured as a proxy. A web site that uses client smart card authentication was opened on the phone. When prompted for a certificate, a dummy certificate was sent to the proxy, which was configured to replace the client certificate with the smart card certificate on the server connection. The proxy prompted for a PIN, which was provided on the desktop machine, and the TLS connections were created between phone and proxy and between proxy and server. Then the phone browser received and displayed the smart card restricted content.

The phone Wi-Fi was then disabled, switching the interface to a 4G LTE connection. After this change, the site's restricted content remained available through continued page loads.

The site was periodically refreshed, with increasing delay between page loads, starting at 5 minutes, increasing to 30 minutes, and eventually to many hours, and access was maintained. For this test, access was maintained for a total of almost three days.

However, during other tests the session was reset after long periods of inactivity. It appears the server policy was to use the session cookie for long periods of time without requiring re-authentication by smart card. It is not clear based on testing whether the session had an expiration time or was simply removed from the cache.

3.2 Session Cookie with IP Check

A second site was tested using the same method. It loaded during the setup phase, but when the network connection was switched from Wi-Fi to mobile, the session was reset and access was lost. Repeating the test by simply removing the Wi-Fi proxy instead of switching networks allowed the setup phase to continue to the access phase while maintaining the connection. This is consistent with the IP check hypothesis, since the IP address of the proxy machine and phone through Wi-Fi were identical due to NAT, while the mobile IP was different.

It was desired to choose an external proxy with a different IP address and switch networks while going through that proxy. However, there was no easy way to provide a proxy configuration for a mobile connection, so this was not tested. Instead the TLS proxy was chained with an external HTTPS proxy with a different IP address. The phone connected through both of these proxies during the setup phase and then switched to just the external HTTPS proxy for the access phase. This provided access. However, the remote user in this case is limited to using the local Wi-Fi instead of the mobile network. Note that this is not a full test, since the IP address of the phone did not change. However, it appears that this server implements the session cookie with an IP check, since access fails when the server-facing IP address changes and succeeds when it does not.

3.3 Re-Authentication Required within Site

A third site, chosen for its high security, was tested for comparison. Access was gained as for the other sites, but certain links on the site required re-authentication using the smart card. Also, the session timeouts were fairly aggressive, allowing only a few minutes of inactivity. This prevented casual browsing of the site, since only a small section of the site was available after each authentication.

To gain access to this site, two additional steps would be required, neither of which was tested. First, the remote user would have to map out the site and its re-authentication boundaries to send a quick sequence of requests, one to each authentication zone. Second, the user would repeatedly and automatically reload each of these pages frequently enough to maintain access to all parts of the site simultaneously. From such a setup, the remote user could browse the site manually for as long as the automated refreshing kept the session open.

3.4 Separate Smart Card and Proxy

The setup where the smart card machine is separate from the proxy was also tested. A small Java program was written to implement PKCS11 requests to the smart card, including a query of available certificates, retrieval of a specific certificate, and use of the private key to generate the Certificate Verify message signature value. The Java program listens on a specified port for requests and returns appropriate responses after accessing the smart card. The TLS proxy was modified to send requests to the smart card machine on the appropriate port in order to perform smart card operations.

The reason Java was chosen was due to ease of implementation. The Net library contains built-in smart card operations, which were initially tried due to their integration. However, the operation required for the Certificate Verify message is not available, which is a raw private key operation on padded content. The only private key operations allowed were the signature operation, which computed a hash before performing the padding and private key operation, and decryption, which failed on content that was not properly padded after the private key operation.

Another option considered was directly calling the DLL file that accesses the smart card. The list of operations was obtained, but not enough information was available to create valid requests without significant effort.

Java requires some configuration to set up the PKCS11 provider and identify the correct DLL file, but it was the simplest way identified to provide this functionality. [18]

The first test above was repeated with this separate proxy and smart card setup, with successful setup and access to the protected site.

3.5 Mock Attack

A mock attack was performed using a co-worker with a smart card. The co-worker was told to log on to a specific smart-card-enabled site on a computer running a proxy, while the author connected an iPhone to the proxy through Wi-Fi on the same LAN. The co-worker entered his PIN to access a site, and then the author sent requests to another site through the proxy, using the smart card that was now unlocked by the PIN entry. The author was able to gain access to a different site than the one the co-worker loaded. After showing the results to the co-worker and discussing the attack, the author logged out and closed the browser and the co-worker removed his smart card from the machine. This mock attack demonstrated the ease with which social engineering could be combined with the use of a proxy and smart card reader code to gain unauthorized access to a protected site.

3.6 Capabilities and Attacks

The main capability provided is using a device without a smart card, such as a smart phone, tablet, or laptop, to interact with web sites that require smart card authentication for access. After the initial access is provided through a smart card proxy, the user can use their preferred device for continued access. This could include logging in from work and then working from home, working while traveling, or simply working on a device with different applications and capabilities than the smart-card-enabled machine has available. With proper server configuration, these could all be useful capabilities that would not be difficult to execute.

Although mobile devices are popular, smart card use for such devices is not as common. The proxy technique is one way that access could be provided to mobile devices. This provides a single point through which all mobile devices receive access. For example, when issuing mobile devices for a day of work in the field, they could log in through a local proxy with a smart card. The server could be configured to allow 8-hour sessions before requiring re-authentication. This could be an accountability issue if a device is lost or shared. However, it may enable activity that otherwise would not be possible, so the additional security risk might be acceptable.

The primary attack is to use someone else’s smart card to log into a site. This is similar to a session-stealing attack, except there is no existing session to steal. Instead, the attacker creates the sessions using the victim’s smart card. This is undesirable in an enterprise, since accountability is compromised.

For attacks using the separate smart card and proxy machines, a separate control channel could be established, either for the attacker to query whether a smart card is ready for use or for the machine with the smart card to call out when a smart card is available for use. This would enable an automated attack on a shared machine to which the attacker has privileged access. The smart-card-reading code would be pre-positioned, and when someone uses the machine with a smart card, the attacker would initiate connections to sites to which the victim has access, and then transfer to a mobile network. Over time, the attacker could potentially access many different sites using different users’ smart cards.

Another attack would involve requesting help from an administrator who uses a smart card to perform certain administrative functions. The attacker would run the smart-card-reader code and proxies before requesting help. The administrator inserts and uses a smart card to perform privileged functions, at which point the attacker uses a mobile device, automated tool on another machine, or partner on another machine to gain access to sites that the administrator can access.

These attacks will be more effective on sites that the victim does not access frequently. This is in contrast to a normal session-stealing attack, which relies on the victim accessing the site. When the user actually logs in, the server is likely to close the attacker session when it creates the new one, so the attacker will have only a limited window of access.

For all attack scenarios, the attacker relies on the victim to enter the PIN to unlock the smart card and for this PIN entry to provide a window of access to the attacker without the need for further PIN entries. PIN caching is fairly complicated, relying on interactions among policies in the application, operating system, device driver configuration, and smart card itself. For the system used in testing, the net result was that the smart card was locked upon removal and reinsertion, and after 10 minutes of idle time while remaining inserted.

A simple script was written to periodically access a smart card by connecting to the Java smart card program. After a single initial PIN entry, this script was launched with a request period set to 5 minutes, and this provided many hours of PIN-free access while the card remained inserted. Such a script could be combined with the Java smart card program to prevent the need for another PIN entry.

3.7 Mitigations, Analysis, and Extensions

For capabilities, the idea of using a proxy with a smart card to log into a web site is not new. This is a common test scenario [19]. However, using multiple proxies, both local and external, to provide a seamless transfer from a connection with a local proxy to a connection without a local proxy appears to be new. In particular, using a common master secret in the TLS sessions to allow transparent proxy removal appears to be new, since the proxies tested do not support this function. In addition, this technique allows access on almost any device that has a browser with proxy settings without continued use of the smart card machine. Other techniques to use a smart card remotely appear to require continuous access to the smart card machine or require installation of drivers or other code on the device making the requests.

If existing TLS connections are used after removing the proxy, they are likely to report fatal decryption or other errors, since the TLS connection-specific cryptographic information and other state information are not likely to be identical. In this case, the existing connection is closed and the session is not allowed to start new connections, thus ending the attack.

To avoid this, the client must stop use of all existing connections after proxy removal, which could be accomplished by blocking communication to and from the client ports used by those existing connections. The solution is, again, adding a proxy, possibly a TCP proxy, which gracefully closes all but one connection and then halts communication on this connection until a new one is established, at which point it gracefully closes the last original connection. This could be implemented on the local machine of the attacker.

The idea of stealing sessions to launch an attack is not new. However, most session hijacking involves stealing a user’s application layer session as stored in an HTTP cookie [20]. The attacks described in this paper create a new session instead of hijacking an existing one. This does not rely on a user logging into the target site but only on a user using his or her smart card, even if for some other purpose. The end result of most of these attacks is similar to session-hijacking attacks, since the attacker has control over a session with the server while logged in as the victim.

The main effect of these attacks is that smart card login security is reduced from needing the smart card and PIN to just needing someone to use a smart card on a machine to which the attacker has privileged access. Typically, the smart card is considered safe because the private key is protected, but with the ability to read the certificate from the smart card and use the private key to encrypt the Certificate Verify TLS handshake message, an attacker can log in and stay logged into a smart-card-enabled site.

Mitigation is often simple conceptually but difficult in practice. One simple solution is to periodically require a user to log in again. This could be done at the TLS layer by requiring a new two-way authenticated session for continued access after some time period. The user would periodically see a prompt for a PIN. This would require support by the server.

Another simple partial solution is to not cache PINs. There is usually a time period during which the PIN is not required to access the smart card after a successful PIN entry. This allows an attacker to use the smart card without the victim's knowledge. This is mentioned as only a partial solution, because users often simply enter their PINs when prompted, and timing the attack to correspond in time with another use by the victim would probably not raise any alarms. Also, PIN caching is complicated in real systems. The smart card has a caching policy, the smart card driver program has its own caching policy, the operating system has caching policies, and applications that use smart cards may have their own caching policy. Interactions between these policies can be tricky to understand and apply correctly [21].

Another simple solution is to limit application layer session durations, which would require going through the two-way authenticated login again. However, without underlying TLS support, this solution would be susceptible to TLS session re-use without requiring smart card access.

A policy-based partial fix would be for people to use smart cards only on trusted machines. This would prevent the use of the proxy or smart-card-reader code from using the smart card to set up sessions for the attacker. However, this is only a partial solution, because establishing trust is difficult, especially for the end user who is responsible for smart card use.

All of these simple fixes require changes to existing information technology infrastructure or policies. These are difficult to implement because none of them are centralized, so management and implementation are difficult, complicated, and prone to errors and inconsistencies.

One central policy that is a partial fix is to blacklist known proxies. This is effective against attackers who use public proxies, but an attacker who sets up his own unpublished proxy would be difficult to detect. A proxy is not hard to set up, so this is only a partial mitigation.

A stronger version of this idea is to use a whitelist of known and approved systems to allow access. It is not always easy to know in advance which systems should have access to a site, so this presents significant challenges. Also, it is not easy to prove to the server that a system is legitimate. The initial connection is set up through a valid system, so the system parameters can be copied and reused by the attacker in future requests. Unless there is a hardware-based cryptographic module, like a TPM, this whitelist policy can be difficult to enforce. Even then, the TPM is much like a smart card, in that the initial connection may be the only time it is used for security.

SSO (Single Sign On) is used to sign into one site as a way to access others. If the SSO site uses smart card authentication, this has the potential to expand the scope of the initial attack. Instead of having access to just the sites for which connections are established while having access to the smart card, an attacker now only needs a single connection using the smart card to the SSO site, from which connections to many other sites can be established through the SSO site session. If the individual sites use two-way smart card authentication, this attack will fail, but connections with one-way authenticated TLS through which an SSO token is sent would allow this type of attack.

This attack is similar to stealing SSO cookies. In the SSO cookie-stealing attack, the cookie itself is used as authentication. In this attack, the session with the SSO site is stolen and used to generate any number of valid tokens. In each case the application layer authentication token is used for access.

The examples mentioned in this paper are based on accessing a web application. However, other uses of smart cards include encryption and decryption of data and digital signatures. For example, email messages can be both signed and encrypted using smart card keys and certificates, and some documents can be electronically signed using a smart card. These are offline actions, though, so the Java code that allows remote access to smart card functions would be sufficient to complete many of these. The web application is challenging because it uses the TLS protocol, which requires a rapid execution of a private key operation, and the Certificate Verify operation is not one of the standard operations like sign, verify signature, encrypt, or decrypt.

4. Conclusions

Proxies can be used to share access to a smart card. This enables people without a smart card or without a smart-card-enabled device to access smart card protected resources. This may be useful when one person is using a smart card to access a resource and another person without a smart card needs the same access, or someone needs access from a device that does not accept their smart card. In either case, the standard route of issuing a new smart card or getting a device to accept a smart card may be too time-consuming to be practical. For remote usage and quick turn-around, proxies may offer a practical temporary solution.

Longer term or widespread adoption of these tactics poses several security issues. By designing the TLS proxy to create identical keys on incoming and outgoing connections and adding additional proxies to shield the client and server from the TLS proxy and each other, different server validation checks, such as IP address consistency, can be bypassed. This weakens accountability, since one person is gaining access using another person's smart card credentials. Because the actual entity accessing resources is not strongly authenticated, attackers can use these techniques to gain unauthorized access using a valid smart card.

Acknowledgements

Special thanks to the US Air Force and DoD for funding this and related work and providing smart cards and smart-card-enabled servers that were used for exploration and testing of concepts and implementations.

References

[1] The Transport Layer Security (TLS) Protocol Version 1.3 (draft). Accessed August 28, 2015. https://tlswg.github.io/tls13-spec/.

[2] Request for Comments: The Transport Layer Security (TLS) Protocol Version 1.2. Accessed August 2008. http://tools.ietf.org/html/rfc5246.

[3] Request for Comments: The Transport Layer Security (TLS) Protocol Version 1.1. Accessed April 2006. http://www.ietf.org/rfc/rfc4346.txt.

[4] Request for Comments: The TLS Protocol Version 1.0. Accessed January 1999. https://www.ietf.org/rfc/rfc2246.txt.

[5] Request for Comments: Transport Layer Security (TLS) Extensions. Accessed April 2006. http://tools.ietf.org/html/rfc4366.

[6] The SSL Protocol Version 3.0. Accessed November 18, 1996. https://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00.

[7] Ross, A., and Kuhn, M. 1997. "Low Cost Attacks on Tamper Resistant Devices." In *Security Protocols*, 5th International Workshop, Paris, France, April 7-9, Proceedings, Springer LNCS 1361, 125-36, ISBM 3-540-64040-1. Accessed September 3, 2015. Available at http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf.

[8] http://www.cs.dartmouth.edu/~pki02/Sandhu/paper.pdf.

[9] http://www.sans.org/reading-room/whitepapers/malicious/detailed-analysis-sykipot-smartcard-proxy-variant-33919

[10] https://www.alienvault.com/open-threat-exchange/blog/s

ykipot-variant-hijacks-dod-and-windows-smart-cards.

[11] http://www.computerworld.com/article/2493077/malware-vulnerabilities/proof-of-concept-malware-can-share-usb-smart-card-readers-with-attackers-ove.html.

[12] http://www.spamfighter.com/News-18066-POC-Malware-Wins-Control-Over-USB-Smartcards.htm.

[13] Request for Comments: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Accessed June 2014. http://tools.ietf.org/html/rfc7230.

[14] NSS Key Log Format. Available at https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.

[15] Dolan-Gavitt, B., Leek, T., Hodosh, J., and Lee, W. *Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection.* Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2013 November.

[16] Foltz, K., and Simpson, W. R. "Wide Area Network Acceleration in a High Assurance Enterprise." *World Congress on Engineering (WCE) 2015*, London, England, July 2015.

[17] http://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe?variant=353378649.

[18] Java™ PKCS#11 Reference Guide. Accessed September 3, 2015. Available at http://docs.oracle.com/javase/1.5.0/docs/guide/security/p11guide.html.

[19] http://blog.taddong.com/2012/04/owasp-zap-smartcard-project.html.

[20] OWASP. "Session Hijacking Attack." Available at https://www.owasp.org/index.php/Session_hijacking_attack.

[21] http://stackoverflow.com/questions/1800745/cac-smartcard-reauthenticate.

REPORT DOCUMENTATION PAGE						*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**						
1. REPORT DATE (DD-MM-YY) 00-01-16			2. REPORT TYPE Non-Standard			3. DATES COVERED (From – To)
4. TITLE AND SUBTITLE Sharing Smart Card Authenticated Sessions Using Proxies						5a. CONTRACT NUMBER HQ0034-14-D-0001
						5b. GRANT NUMBER
						5c. PROGRAM ELEMENT NUMBERS
6. AUTHOR(S) Kevin E. Foltz						5d. PROJECT NUMBER BC-5-2283
						5e. TASK NUMBER
						5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES Institute for Defense Analyses 4850 Mark Center Drive Alexandria, VA 22311-1882						8. PERFORMING ORGANIZATION REPORT NUMBER NS D-5417 H 15-000067
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Frank P. Konieczny USAF HQ USAF SAF/CIO A6						10. SPONSOR'S / MONITOR'S ACRONYM SAF/CIO A6
						11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION / AVAILABILITY STATEMENT This draft has not been approved by the sponsor for distribution and release.						
13. SUPPLEMENTARY NOTES Project Leader: William R. Simpson						
14. ABSTRACT This paper discusses an approach to share a smart card in one machine with other machines accessible on the local network or the Internet. This allows a user at a browser to use the shared card remotely and access web applications that require smart card authentication. This also enables users to access these applications from browsers and machines that do not have the capability to use a smart card. The approach uses proxies and card reader code to provide this capability to the requesting device. Previous work with remote or shared smart card use either requires continuous access to the smart card machine or specific client software. The approach in this paper works for any device and browser that has proxy settings, creates minimal network traffic and computation on the smart card machine, and allows the client to transfer from one network to another while maintaining connectivity to a server. This paper describes the smart card sharing approach, implementation and validation of the approach using real systems, and security implications for an enterprise using smart cards.						
15. SUBJECT TERMS Smart card, IT security, authentication, key management, proxy, SSL, TLS, session stealing.						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Frank P. Konieczny
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	Unlimited		18	19b. TELEPHONE NUMBER (Include Area Code) (703) 697-1308