An Inconvenient Truth About Tunneled Authentications

Katrin Hoeper
Motorola Inc.
Schaumburg, IL 60196, USA
Email: khoeper@motorola.com

Lidong Chen
National Institute of Standards and Technology
Gaithersburg, MD 20878, USA
Email: llchen@nist.gov

***Abstract*—In recent years, it has been a common practice to execute client authentications for network access inside a protective tunnel. Man-in-the-middle (MitM) attacks on such tunneled authentications have been discovered early on and cryptographic bindings are widely adopted to mitigate these attacks. In this paper, we shake the false sense of security given by these so-called protective tunnels by demonstrating that most tunneled authentications are still susceptible to MitM attacks despite the use of cryptographic bindings and other proposed countermeasures. Our results affect widely deployed protocols, such as EAP-FAST and PEAP.**

***Index Terms*—Protective tunnel, authentication, tunnel-based EAP method, man-in-the-middle attack, cryptographic binding.**

I. Introduction

During the last decade, people have been searching for cost-effective solutions that allow the continued use of legacy protocols for client authentication with existing equipment while enhancing the protocols' security. Tunneled authentications were mainly invented for this purpose. Here, authentications are executed in a protective tunnel and it is generally believed that even though some of the legacy authentication methods may be weak, with the protective tunnel, they can still be securely used. Current standards efforts within the IETF aim to define a standard tunneled authentication protocol to support "password-based authentication mechanism, and additional inner authentication mechanisms" [1].

A tunneled authentication typically consists of two phases. First, the client and the authentication server establish a protective tunnel, usually through a tunnel protocol that employs public key-based schemes. This phase derives tunnel keys and generally includes the authentication of the server. TLS is commonly used as such a tunnel protocol [2]. In the second phase, the client executes an authentication protocol with the server inside the tunnel, *i.e.*, protected by the established keys which is referred to as *inner authentication* in the remainder of this paper. The protocol execution commonly ends with the derivation of a master session key that can be used later on to derive further keys, *e.g.*, traffic protection keys to protect subsequent communications.

Asokan *et al.* [3] identified a man-in-the-middle (MitM) attack on tunneled authentication protocols that exploit that tunnel protocol and inner methods are executed independently from each other. In the same paper, the authors propose binding the tunnel protocol and inner methods, which is commonly referred to as *cryptographic binding*. Such cryptographic bindings have become the de facto mitigation technique for MitM attacks on tunneled authentications and are implemented in widely deployed tunneled authentication protocols such as PEAP [4] and EAP-FAST [5]. In fact, cryptographic bindings are a mandatory requirement in the IETF standard for tunneled authentication that is currently defined.

In this paper, we shake the false sense of security given by tunneled authentications with cryptographic bindings by demonstrating that most of these tunneled authentications are still susceptible to MitM attacks. Our analysis shows that the standard binding approach can only protect strong client authentications with strong key derivation, while this as well as all other investigated binding methods fail to protect more common use cases, such as legacy password and any non-key deriving authentication schemes. In other words, passwords and other credentials, sent inside a tunnel and considered safe, are in fact still at risk.

While current solutions are agnostic to the inner methods and implementation environments, our analysis shows that the applicability and effectiveness of a particular binding method depend on the security properties of the inner authentication method as well as the network infrastructure. Furthermore, we show that using a fixed derivation scheme for traffic protection keys for different types of tunneled authentications may expose subsequent communications. Further results indicate that other proposed countermeasures in form of security policies are insufficient and/or impractical in most environments. These results are unsettling and affect widely deployed tunneled authentication methods, such as PEAP and EAP-FAST.

In the next section, we briefly review concepts and define terms used in the remainder of this paper. In Section III, we introduce various binding methods, while Section IV summarizes our threat model. In Sections V and VI, we analyze the applicability and effectiveness of cryptographic binding methods and security policies, respectively. Secure session key derivations are discussed in Section VII. In Section VIII, we discuss the security pitfalls of widely deployed tunneled authentication methods. And finally, in Section IX, we draw conclusions.

II. Review and Definitions

This section briefly reviews tunneled authentications and previous work.

A. Tunneled Authentications

A tunneled authentication, as depicted in Figure 1, consists of a tunnel protocol that is executed to establish a protective tunnel, an inner authentication method that is executed inside the tunnel and, finally, the derivation of a master session key that is used to derive traffic protection keys. These subroutines are defined in the following paragraphs.

1) *Tunnel protocol* - A tunnel protocol is a key establishment protocol between a client and a server that provides server authentication, mutual authentication or no authentication (anonymous tunnel). Since this paper focuses on client authentications inside a tunnel, we only consider the following two options in the remainder:
   a) Anonymous tunnels; or
   b) Tunnels with server authentication.

   In an anonymous tunnel, neither client nor server authenticates to the other party and authentications are executed later on inside the tunnel. This is sometimes used to provide privacy, credential provisioning, or to protect pre-shared key-based authentications not directly supported by the tunnel protocol.

   This paper explores attacks on tunneled authentications inside cryptographically strong tunnels, *i.e.*, we make the following assumptions:
   - The key establishment used in the tunnel protocol cannot be compromised through solving the underlying hard mathematical problem.
   - In tunnels with server authentication, the client is assured that the server authentication is bound to the key establishment, *i.e.*, only the client and the server can obtain the established key.

   In this paper, we use K_T to denote a key established by the tunnel protocol. A tunneled message is denoted as $T(D_1, D_2, ..., D_m)$, where each D_i is a data field in the message. In our analysis, we assume that every tunneled message is encrypted and optionally integrity-protected. Note that TLS tunnels always provide integrity-protection and optionally encryption.
2) *Inner authentication* - An inner authentication method provides client authentication to an inner authentication server, which may or may not be the same server with which the tunnel has been established (see Figure 1). We make the following assumptions for inner authentications: 1) the long-term authentication credentials are either a secret key K^{in}_{auth} (or a password) or a pair of public/private keys $(pk^{in}_{auth}, sk^{in}_{auth})$, 2) public keys are certified by a trusted third party, 3) whenever a secret key is used, the client is authenticated by a keyed hash, such as a message authentication code (MAC), and 4) whenever the authentication is public key-based, the client is authenticated by generating a digital signature.

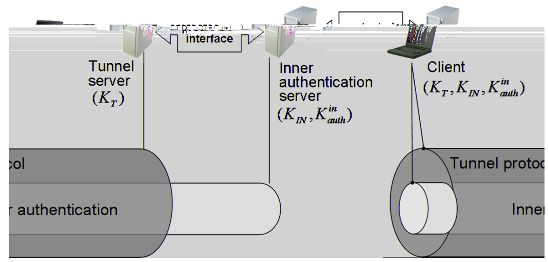

Fig. 1. Tunneled Authentication.

   We consider the following categories for tunneled client authentications:
   a) Strong authentication & strong key establishment;
   b) Strong authentication & weak/no key establishment;
   c) Weak authentication & strong key establishment;
   d) Weak authentication & weak/no key establishment.

   In this paper, we refer to authentications that are based on digital signatures or keyed hash with the required security strength[1] as *strong authentications*. On the other hand, we refer to an authentication as *weak authentication* if the secret or private authentication credential of a client and/or the server can be compromised whenever an attacker has access to the protocol exchange. Examples of weak authentication schemes are schemes that exchange secret credentials in the clear, are prone to dictionary attacks or use weak hash function such as MD5 or HMAC-MD5 that cannot resist pre-image attacks [7] to generate the keyed hashes.

   Furthermore, a key establishment as part of an inner method is referred to as *strong key establishment*, if an attacker cannot obtain the established key *during* an ongoing session by attacking the underlying mathematical problem of the key agreement scheme or the protocol itself. Unlike attacks on the authentication scheme, attacks on the key establishment are time-bound by the tolerated protocol delays. Conversely, an inner key establishment is considered as *weak key establishment*, if an attacker can obtain the established keys during the on-going session. If the inner authentication has a key establishment, we use K_{IN} to denote the established key.
3) *Master session key derivation* - The master session key is an output of the tunneled authentication and often used to protect subsequent communications between the client and another network entity, such as a wireless access point. We denote this key as K_{SES} and consider the following options for its derivation:
   a) $K_{SES} = f(K_T)$,
   b) $K_{SES} = f(K_{IN})$,
   c) $K_{SES} = f(K_T, K_{IN})$,

[1]Please refer to NIST SP 800-57 [6] for guidelines on security strengths of algorithms and adequate key lengths.

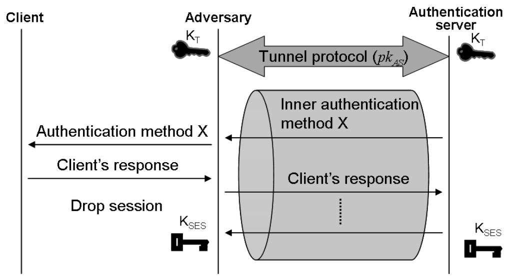

Fig. 2. Original Man-in-the-Middle Attack on Tunneled Authentications [3].

where f is a key derivation function. In the rest of this paper, f is always used to denote a key derivation function without special notation.

B. Previous Work

MitM attacks on tunneled authentications and authentications in anonymous tunnels have been identified in [3] and [8], respectively. For the sake of simplicity, we review the attacks for scenarios in which tunnel server and inner authentication server are the same entity. The attacks, illustrated in Figure 2, consist of the following steps:

1) An attacker executes a tunnel protocol with a tunnel server, either by intercepting a tunnel protocol initiated by the server with a client or by initiating a protocol with the server pretending to be the client. Here, the server's certified static or authenticated ephemeral public key pk_{AS} is used to establish the tunnel key K_T, *i.e.*, the tunnel protocol provides server authentication[2]. As a result, both the server and attacker obtain K_T.
2) Then the attacker initiates authentication method X with a client pretending to be an authentication server (or waits until the client initiates such a session). The attacker redirects the client's responses inside the tunnel that has been established in Step 1. From the server's perspective, the client authenticates to the server using inner method X.
3) Upon a successful authentication, the master session key K_{SES} is derived. At this point, the protocol terminates and the authentication server considers the client authentication as successful.

The authors in [3] observed that in order to thwart the described MitM attack, the server needs assurance that the client is the same entity that executed both, the tunnel protocol and the inner authentication method. As a possible countermeasure, the authors suggest deriving a compound key by combining either the tunnel key K_T with the inner key K_{IN}, if available, or the tunnel key K_T with the client's long-term inner authentication key K^{in}_{auth}. So-called explicit protocol variants provide key confirmation of these compound keys. Both methods are referred to as cryptographic binding in [3]. PEAP and EAP-FAST have both adopted the first approach with key confirmation, *i.e.*, combining K_T and K_{IN}. As an alternative to cryptographic bindings, the authors in [3] suggest enforcing a policy that prevents authentication methods that are tunneled from being executed outside a protective tunnel. Similarly, the authors in [8] propose a number of security policies, including to prohibit anonymous tunnels, to thwart the reviewed attacks.

EAP-FAST supports mutually authenticated tunnels for session resumption which prevent MitM attacks. However, the initial full protocol execution is still at risk and, if compromised, can compromise subsequent session resumptions as well.

In our analysis, we show the insufficiencies and practical issues of cryptographic bindings and security policies in thwarting MitM attacks.

III. Types of Cryptographic Bindings

We present four general methods for cryptographic bindings in the following subsections. Cryptographic bindings can be computed as soon as the required information is available and need to be verified before the master session key K_{SES} is derived. Latter is important to prevent attackers from computing and using K_{SES} before a failed cryptographic binding indicates the occurrence of an MitM attack.

A. Method A: Standard Binding

In this method, the keying material K_T established through the tunnel protocol and K_{IN} established through the inner authentication are combined to derive a compound key K_c, *i.e.*,

$$K_c = f(K_T, K_{IN}).$$

For a proof of the binding, the client uses K_c to generate a message authentication code

$$MAC(K_c, R),$$

where R can be a random factor provided by the server, a sequence number, a time stamp or a combination of these, and sends it to the server.

Method A is similar to one of the cryptographic bindings with explicit authentication defined in [3] and is used in PEAP and EAP-FAST.

B. Method B: Inner Binding

To remove the dependency with inner key establishments, a compound key could be computed using the tunnel key K_T and the client's long-term authentication credential K^{in}_{auth}, *i.e.*,

$$K_c = f(K_T, K^{in}_{auth})$$

As in Method A, the derivation of the key needs to be confirmed by the client.

This cryptographic binding method is similar to one of the bindings with explicit authentication in [3] and, to the best of our knowledge, has not been implemented yet. Instead of using the long-term credential directly, a key derived from it could be used in the binding to comply with the "single purpose" principle of cryptographic keys.

[2]The attack also works for anonymous tunnels. In that case the server's anonymous public key pk_{anom} is used to establish the tunnel.

C. Method C: Inside-Out Binding

As Method B, Method C uses the client's long-term authentication credential to generate a binding with the tunnel protocol. However, instead of deriving a compound key, the long-term credentials are used to sign the tunneled data. In particular, if the inner authentication is public key-based, the client's private key sk_{auth}^{in} is used to sign the tunneled message $T(D_1, D_2, ..., D_m)$, *i.e.*, the proof of binding is presented as

$$Sig(sk_{auth}^{in}, T(D_1, D_2, ..., D_m)).$$

On the other hand, if the inner method is secret key-based, then the client's long-term authentication key K_{auth}^{in} is used to generate a MAC over $T(D_1, D_2, ..., D_m)$, *i.e.*, the proof of binding is presented as

$$MAC(K_{auth}^{in}, T(D_1, D_2, ..., D_m)).$$

Here, instead of K_{auth}^{in}, a key derived from K_{auth}^{in} could be used as the MAC key for the same reason as pointed out for Method B.

Method C is loosely related to the channel binding mechanism for EAP methods executed inside an IKEv2 tunnel [9]. However, unlike Method C, the method in [9] requires the inner authentication to derive a key.

D. Method D: Outside-In Binding

Here, the tunnel key and the client's long-term inner authentication key are used to derive a new inner authentication key, *i.e.*,

$$K_{auth}^{in-T} = f(K_{auth}^{in}, K_T).$$

The new inner authentication key K_{auth}^{in-T} is used in the inner authentication in place of K_{auth}^{in}. Note that K_{auth}^{in-T} is an ephemeral key that will be used only in this session.

IV. Threat Model

In our analysis we do not only consider the MitM attacks proposed in [3] and [8], but also a new *extended MitM attack* that is a modified combination of both attacks. While the original attack relies on the fact that clients may execute inner methods outside a protective tunnel, the extended attack exploits that clients may accept the establishment of anonymous tunnels. The extended MitM attack is illustrated in Figure 3 and can be described as follows:

1) As in the original MitM attack, the attacker executes a tunnel protocol with a tunnel server, where the server is authenticated. The established tunnel is referred to as tunnel 2 and is protected by tunnel key K_T^2.
2) Then the attacker initiates an anonymous tunnel protocol with the client posing as the tunnel server. Here the attacker's anonymous public key pk_{anom} is used to derive tunnel key K_T^1 and establish tunnel 1.
3) Now the attacker listens to all messages sent inside one tunnel and then redirects the messages into another tunnel, making client and server believe they share a protective tunnel with each other.

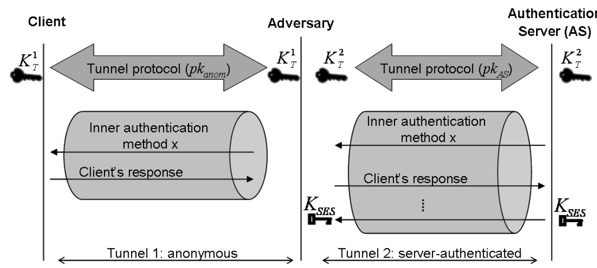

Fig. 3. Extended MitM Attack.

Throughout our analysis, we consider an MitM attack on a tunneled authentication as successful if at least one of the following conditions hold:

- the attacker successfully impersonates a client, *i.e.*, the attacker can authenticate to the server as the client,
- the attacker is able to obtain session-specific confidential information, such as session keys, and use this information to launch attacks in the same session, or
- the attacker is able to obtain confidential long-term credentials in an on-line or off-line attack that can be used to compromise future sessions.

V. Security Analysis of Cryptographic Bindings

We now discuss the applicability and effectiveness of the cryptographic binding methods presented in Section III. Throughout our analysis, we distinguish between the four categories of inner authentication methods defined in Section II-A. The analysis also covers different implementation scenarios as well as the execution of multiple authentications within the same tunnel. Our results are summarized in Table I.

A. Applicability and Other Constraints

An obvious limitation of Method A is that it only works for inner methods that derive keys. On the other hand, this results into the benefit that Method A, unlike the other binding methods, uses only ephemeral keys for the binding computation.

Methods B and D only work for secret key-based inner methods. Methods B and C require that the inner authentication credentials can be used by the tunneled authentication protocol.

The implementation environment of the tunnel protocol may not allow to generate tunneled messages, sign them, and then send them trough the tunnel. In this case, the signatures in Method C need to be sent outside the tunnel, which may not be feasible in some applications, *e.g.*, whenever a client needs to send all data through a VPN tunnel. Furthermore, Method C does not work for inner methods in which inner authentication and inner key establishment are inseparably combined, such as in password-authenticated key establishments (*e.g.*, [10]).

The design of Method D has two consequences: 1) it requires the modification of the inner authentication method contradicting one of the original design goals of tunneled authentications, and 2) the inner authentication method can only be executed if a tunnel key K_T is available. The second property prevents the original MitM attack, because the inner method needs to be executed inside a tunnel. However, the extended MitM attack presented in Section IV still applies.

B. Strong Inner Authentication & Key Establishment

First we consider the ideal case, *i.e.*, the inner method provides strong authentication and strong key establishment. This refers to inner methods of category (a) as defined in Section II-A.

Whenever applicable (see previous subsection), all cryptographic bindings methods prevent the considered MitM attacks. Method A prevents the attacks because the attacker cannot obtain K_{IN}, and Methods B, C and D because the attacker cannot get K^{in}_{auth} to compute the respective binding.

C. Strong Inner Authentication & Weak Key Establishment

We now study a less ideal case, where the client authentication is strong with weak or no key establishment. This refers to inner methods of category (b) as defined in Section II-A and covers protocols that are exclusively designed for entity authentications, *e.g.*, using a token or a personal identity verification (PIV) card.

If the inner authentication has a weak key establishment, then Method A is applicable. However, an MitM attacker could break the key establishment scheme during the ongoing session, derive compound key K_c, and successfully impersonate the client. Hence, Method A cannot prevent MitM attacks for this category of inner methods.

Methods B, C, and D all prevent the MitM attacks, because the attacker has no access to the client's long-term authentication credentials K^{in}_{auth}, and, thus, cannot compute the cryptographic binding.

D. Weak Inner Authentication & Strong Key Establishment

In this section, we analyze bindings for protocols with weak authentication but a strong key establishment. The authors are currently not aware of deployed protocols that fall into this category (c).

The standard binding does not successfully prevent attacks for inner methods of category (c). An MitM attack would be detected due to a failure in presenting the cryptographic bindings, however, by then the attacker could be already in possession of the client's long-term authentication credentials.

Methods B and C both depend on the long-term authentication credentials and thus cannot mitigate the MitM attacks. In particular, if the credentials are weak, then the attacker can break the cryptographic binding. In any case, the weak authentication algorithm enables an MitM attacker to recover the client's authentication credential in an on-line or off-line attack.

Applying Method D, launching the original MitM attack is not feasible, however, an attacker could attempt an extended MitM attack. Here, the client computes its updated authentication key $K^{in-T}_{auth} = f(K^{in}_{auth}, K^1_T)$, where the server computes $K^{in-T}_{auth} = f(K^{in}_{auth}, K^2_T)$, which will be typically used in a MAC to compute and verify the inner authentication data as well as the proof of binding, respectively. The attacker knows both K^1_T and K^2_T and has access to the exchanged data. Hence, authentication schemes exchanging secrets in the clear are broken. If the inner authentication method is weak because K^{in}_{auth} has low entropy, then the attacker can obtain the key in an on-line or off-line dictionary attack. On the other hand, if the weakness is due to a weak authentication algorithm but K^{in}_{auth} has sufficient entropy, then the attacker can only obtain the updated authentication key K^{in-T}_{auth} through an off-line attack on the MAC. This key K^{in-T}_{auth} is only used in the current session. Hence, if given K^{in-T}_{auth} and K_T one cannot recover K^{in}_{auth}, then Method D prevents the attack and is a suitable proof of binding.

E. Weak Inner Authentication & Key Establishment

Last but not least, we will discuss the worst case, where the authentication is weak with a weak or no key establishment, *i.e.*, category (d) in Section II-A. Actually, this worst category inspired the introduction of tunneled authentications and constitutes the most common application. For example, some of the EAP methods specified in the original EAP standard [11], such as One-Time Password (OTP) and MD5-Challenge, fall into this category. Other examples include the widely deployed MS-CHAP v1 and v2 authentication protocols ([12], [13]) that are vulnerable to dictionary attacks and are nowadays tunneled using PEAP.

In this scenario, the standard binding does not provide a proof of binding since an MitM attacker can obtain K_{IN} and derive the compound key K_c.

Methods B, C and D all depend on the security of long-term authentication credential K^{in}_{auth} and, thus, for providing a valid proof of binding it does not matter whether the inner key establishment is strong, weak or does not exist at all. Hence, the same discussions as in the previous subsection for inner methods in category (c) apply. As a result, only Method D may be able to provide a proof of binding under the conditions listed in the previous subsection.

F. Multiple Inner Authentications

Multiple client authentication methods may be executed concurrently or sequentially inside a tunnel, *e.g.*, allowing clients to provide different levels of authentication using different sets of credentials.

In general, concurrently executed inner authentication methods are independent from each other and as such each require individual protection by a cryptographic binding method. Hence, the same discussions as in Sections V-B-V-E apply.

On the other hand, cryptographic bindings of sequentially executed inner authentication methods may be combined to a *chained cryptographic binding*. In particular, given n sequentially executed inner methods supporting the same cryptographic binding, an *intermediary cryptographic binding* can be computed upon the completion of each inner method i, such that it binds all completed inner methods to the tunnel and to each other. Such a chained cryptographic binding has been proposed for standard cryptographic bindings (Method A) [5],

TABLE I
EFFECTIVENESS OF CRYPTOGRAPHIC BINDING METHODS FOR DIFFERENT INNER AUTHENTICATION METHODS.

Cryptographic Binding	**Inner Authentication Method** Category (a)	Category (b)	Category (c)	Category (d)	Limitations
Method A	✓*	X†	X	X	
Method B	✓	✓	X	X	- secret key-based inner authentication - re-usable long-term credentials - not desirable for separate servers‡
Method C	✓	✓	X	X	- sending signed tunnel data - re-usable long-term credentials - stand-alone authentication - MAC variant not desirable for separate servers‡
Method D	✓	✓	✓ with conditions§, else X	✓ with conditions§, else X	- secret key-based inner authentication - modification of inner method - not desirable for separate servers‡

*✓:This combination of cryptographic binding method and inner authentication method resists the MitM attacks.
†X: This combination of cryptographic binding method and inner authentication method does not resist all MitM attacks.
‡See our discussion in Section V-G.
§The conditions are: K^{in}_{auth} has sufficient entropy and given K^{in-T}_{auth} and K_T one cannot recover K^{in}_{auth}.

where for inner methods that do not have a key establishment, an all zero string is used in place of the inner key to derive the cryptographic binding. Here the intermediary binding is computed as intermediary compound keys

$$K^i_c = f(K^{i-1}_c, K^i_{IN}), \text{ for } 0 < i \leq n \text{ and } K^0_c = K_T,$$

where K^i_{IN} is the key established by the i-th inner method. Similar chained cryptographic bindings can be designed for the other binding Methods B/C/D.

One may intuitively assume that the security conditions for at least some of the tunneled inner methods can be relaxed due to the chained cryptographic binding. However, for a binding method to prevent the considered MitM attacks on *every* inner method, the properties of the inner methods are not the only factor and co-dependent on various implementation factors. The problem is quite complex as we will illustrate by means of example in the remainder of this section.

Considering the above formula, let's suppose the first $i \leq n$ inner methods do not provide strong key establishments, while inner method $i + 1$ does. In this case an MitM attacker can compute all intermediary compound keys K^1_c to K^i_c, but cannot derive any subsequent compound keys K^{i+1}_c to K^n_c. What does this mean for the security of all inner methods? If the protocol instantly verifies each intermediary cryptographic binding upon the completion of an inner method and immediately aborts after detecting a failure, attacks on weak inner authentications and key establishments are limited to the first $i + 1$ and i inner methods, respectively. Otherwise, the authentication credentials and key establishments of *every* inner method are exposed to attacks. As an alternative to instant intermediary cryptographic binding verification, the tunnel key K_T could be replaced by compound key K^j_c to protect the next inner method $j + 1$, creating a *chained compound tunnel*.

From this example it can be observed that chained bindings alone cannot help to relax the requirements on inner methods, but in combination with several other implementation factors, such as instant intermediary cryptographic binding verification or chained compound tunnels, inner methods i with $i > 1$ could be from other categories than indicated in Table I without compromising the overall security. However, these implementation factors render cross-platform implementations insecure and require a secure sequence negotiation that ensures that the first method in a sequence is chosen according to Table I.

G. Server Implementation Scenarios

It has been considered as one of the implementation scenarios for tunneled authentications that the server for the tunnel protocol, called *tunnel server*, may not be the same as the server for the inner authentication, called *inner server*. Verifying cryptographic bindings requires a protected interface between the tunnel server and the inner server, so that they can pass the keys that are necessary for the verification to each other (see Figure 1). While one goal of tunneled protocols is the preservation of legacy systems, it can be observed that most implementations supporting cryptographic bindings require changes to the inner server, either to pass on keys to the tunnel server or carry out the binding verification. In the following we discuss for each binding method under which condition the tunnel server and inner server can be separated and which server preferably acts as the verifier[3].

Verifying the standard cryptographic binding requires that either the inner server passes the K_{IN} to the tunnel server or the tunnel server passes K_T to the inner server. Both servers can act as the verifier.

For verifying the cryptographic bindings of Method B, either K^{in}_{auth} (or its derivative) or K_T need to be passed to the respective server. However, passing K^{in}_{auth} to another server is undesirable because it is a long-term credential. Here, the inner server should act as a verifier. On the other hand, if

[3]In [3], the verifier of a standard binding is called a "binding agent", where the discussion was developed w.r.t. whether the binding agent should be co-located with the inner server or tunnel server.

a derivative of this key was used to compute the binding, then only this ephemeral derivative key needs to be passed to the tunnel server. In this case, the binding verification can be performed by either server.

If Method C is used with digital signatures, then verifying the binding includes verifying the signature over tunneled data. This is the only case that does not require the modification of the inner server, because the tunnel server knows tunneled message $T(D_1, D_2, \ldots, D_n)$ while public key pk_{auth}^{in} and its certificate are publicly available. However, if Method C is applied with a MAC, the verifier needs access to the other input key to derive the compound key, *i.e.*, K_{auth}^{in} or its derivative. For the same reasons as discussed for Method B, the inner server should act as a verifier if K_{auth}^{in} or a key derived from K_{auth}^{in} is required for the binding verification.

When Method D is applied, the binding must be verified by the inner server, because it is undesirable to pass long-term credential K_{auth}^{in} to the tunnel server. In this case, the tunnel key K_T is passed from the tunnel server to the inner server. Since Method D requires the modification of inner methods, the modification of the inner servers does not add any additional burden.

VI. Analysis of Configuration Policies

Finally, we discuss the feasibility and effectiveness of security policies that have been discussed as countermeasures for MitM attacks and could be used whenever cryptographic bindings are ineffective or impractical (see Table I).

Security policies demanding clients to execute inner methods only inside a tunnel [3] or servers not to accept anonymous tunnels [8] do not prevent the extended MitM attack presented in this paper. In order to prevent the original as well as the extended MitM attacks, we derive the following configuration policy:

- Inner authentication methods can only be executed inside a server-authenticated protective tunnel.

The above policy needs to be enforced by the client, because the server is unaware of clients engaging in non-tunneled or anonymous sessions with an attacker (see Figures 2 and 3). However, client-side policy enforcement suffers from several difficulties. First of all, a client device may be used inside an enterprise intranet as well as in public environments such as airports, coffee shops, or home offices for remote access. In the former setting, an authentication can be executed without tunnel while in the latter, it must be tunneled. Hence, the used authentication methods have to allow two modes: tunneled and non-tunneled. Furthermore, whenever a password is used for user authentication, one cannot rely on the user to distinguish whether he is sending data through a tunnel or not. But exactly this client capability is necessary to enforce the security policy.

In addition, anonymous tunnels are still supported by some clients. Finally, client devices are more susceptible to attacks (*e.g.*, compared to servers) and thus an attacker could tamper with the device to hinder the enforcement of the policy or make the client believe that data is submitted inside a protective tunnel, where in fact it is submitted in the clear.

TABLE II
SECURE MASTER SESSION KEY K_{SES} DERIVATION FOR DIFFERENT INNER AUTHENTICATION METHODS.

Inner Authentication Method	**Key Derivation Input** K_T	K_{IN}	(K_T, K_{IN})
Category (a)	O *	✓†	✓
Category (b)	O	X‡	O
Category (c)	O	O	O
Category (d)	O	X	O

*O: This input key can be used to derive K_{SES} in combination with a suitable cryptographic binding method or enforced configuration policy.

†✓: This input key can be used to derive K_{SES} with or without cryptographic binding.

‡X: Using this input key to derive K_{SES} puts communication protected under K_{SES} at risk of compromise.

VII. Secure Session Key Derivation

This section discusses the dependencies between the secure derivation of master session key K_{SES} and different inner authentication methods with or without cryptographic binding. Only if derived securely, K_{SES} can be used to protect subsequent communications. Our results are summarized in Table II and it can be observed that only for inner methods with strong authentication and strong key establishment, K_{SES} can be securely derived from K_{IN} or a combination of K_{IN} and K_T if no binding method has been used. In all other cases, the secure key derivation requires the use of a suitable binding method. For example, without cryptographic bindings, tunnel key K_T cannot be used as only input to derive K_{SES} because it is known to MitM attackers. In addition, inner keys K_{IN} from weak inner key establishments should never be used to derive K_{SES}, while keys from strong key establishments paired with weak authentications (category (c)) cannot be securely used as sole input because the authenticity of the keys cannot be ensured.

We conclude, that if a suitable cryptographic binding method is applied, it is advisable to derive K_{SES} either from both tunnel key K_T and inner key(s) K_{IN} or from the tunnel key K_T alone. When both inner authentication and key establishment are strong (category (a)), K_{IN} may be used as a sole input to derive K_{SES}. In case of category (c), if the weak inner authentication is due to low entropy credentials, only when the configuration policy in Section VI is truly enforced, K_{IN} can be used as the only input to derive K_{SES}. If multiple inner methods i are tunneled, any combination of inner keys K_{IN}^i may be used to derive K_{SES} as long as the overall key derivation complies with the previous discussions.

VIII. Security Pitfalls in Current Deployments

Most of the currently deployed tunneled authentications are tunnel-based EAP methods such as EAP-TTLS [14], EAP-FAST [5], and PEAP [4]. Based on our findings in this paper, we point out some security pitfalls of these tunneled authentication protocols.

A. Standard Bindings

As we have shown in this paper, the applicability and effectiveness of a cryptographic binding method highly depend on the inner method executed inside the tunnel (see Table I). However, tunnel-based EAP methods allow any type of inner method to be tunneled, while either applying the standard cryptographic binding (EAP-FAST and PEAP) or no binding at all (EAP-TTLS). Obviously, without any binding, MitM attacks are feasible unless other countermeasures are applied (see next subsection). On the other hand, as demonstrated in this paper, using the standard cryptographic binding can only protect inner methods with strong authentication and strong key establishment. However, such methods are an exception. For example, commonly tunneled client authentication protocols are MS-CHAP v1 and v2, that are, when not tunneled, prone to dictionary attacks and fall into our category (d) of inner authentication methods. These and any other inner methods without key establishment lead to the computation of a trivial compound key K_c. In particular in EAP-FAST and PEAP, K_{IN} is replaced with a zero string resulting into a cryptographic binding that cannot serve as a proof of binding and, thus, cannot mitigate MitM attacks.

In addition, EAP supports the execution of multiple inner methods and we showed in Section V-F that chained cryptographic bindings as implemented in EAP-FAST only protect against MitM attackers if *every* inner method provides strong authentication and strong key establishment or several other security techniques are in place such as instant intermediary cryptographic binding verification or chained compound tunnels. However, such additional security measures are not specified in the EAP framework or EAP-FAST.

B. Client-Enforced Security Policies

With the standard cryptographic binding applied by EAP-FAST and PEAP only preventing the known attacks in a few rare applications, the enforcement of the configuration policy derived in Section VI becomes crucial. However, the proposed policy is difficult to enforce in networks using EAP for tunneled authentications, because EAP specifies that inner methods should not be modified. Without modifications, existing authentication methods cannot be aware of whether they are executed inside a tunnel or not, because they cannot process any input from the tunnel protocol. Furthermore, unmodified existing methods do not output binding-related information. As a result, the correct enforcement of the policy cannot be verified by the tunneled authentication protocol itself and must be ensured by the client alone outside of the protocol execution. This is difficult to ensure for the reasons discussed in Section VI.

Note that TLS, the tunnel protocol used by tunnel-based EAP methods, supports anonymous tunnels, enabling the extended MitM attack presented in this paper and violating the configuration policy.

IX. Conclusions

This paper reveals the inconvenient truth about tunnels used to protect client authentications, namely that currently used tunneling methods are at risk to MitM attacks. None of the analyzed cryptographic binding methods is able to prevent MitM attacks on tunneled legacy password authentication schemes. In fact, the current standard binding only prevents attacks on tunneled methods providing strong authentication and strong key establishment.

Our results also indicate that the identified pitfalls of current deployments won't be easy to address. A secure deployment of tunneled authentications needs to implement the right combination of cryptographic bindings, traffic key derivations and configuration policies based on the tunneled methods. However, while the presented binding method D is the most effective one w.r.t. different categories of inner methods, it has the most implementation limitations making it impractical in many settings. In fact, method D could not be used in EAP methods without breaking existing implementations. In further results we showed that client-enforced policies that could prevent MitM attacks whenever cryptographic bindings fail are impractical in many environments. Concluding, tunneling weak client authentication methods is and likely remains an insecure practice, further emphasizing the need to ultimately replace legacy authentication methods with cryptographically stronger methods.

References

[1] IETF EAP Method Update (EMU) Working Group, http://www.ietf.org/dyn/wg/charter/emu-charter.html

[2] RFC 5246, "The Transport Layer Security (TLS) Protocol Version 1.2", (obsoletes 3268, 4346, 4366, updates: 4492), T. Dierks and E. Rescorla, August 2008.

[3] N. Asokan, V. Niemi and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols", Proceedings of Security Protocol Workshop 2005, LCN 3364, pp. 38-41.

[4] "Protected Extensible Authentication Protocol (PEAP) Specification", Microsoft White Paper, August 2009.

[5] RFC 4851, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", N. Cam-Winget, D. McGrew, J. Salowey and H. Zhou, May 2007.

[6] NIST Special Publication 800-57, "Recommendation for Key Management", March 2007.

[7] G. Leurent, "Message Freedom in MD4 and MD5 Collision: Application to APOP", Proceedings of Fast Software Encryption 2007, LNCS 4593, pp. 309-328.

[8] K. Hoeper and L. Chen, "Where EAP Security Claims Fail", The 4th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, 2007.

[9] RFC 4306, "Internet Key Exchange (IKEv2) Protocol", (obsoletes 2407, 2408, 2409), C. Kaufman, December 2005.

[10] D. P. Jablon, "Strong password-only authenticated key exchange", SIGCOMM Comput. Commun. Rev., vol. 26, no. 5, pp. 5-26, 1996.

[11] RFC 3748, "Extensible Authentication Protocol (EAP)", B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson and H. Lefkowetz, June 2004.

[12] RFC 2433, "Microsoft PPP CHAP Extensions", G. Zorn and S. Cobb, October 1998.

[13] RFC 2759, "Microsoft PPP CHAP Extensions, Version 2", G. Zorn, January 2000.

[14] RFC 5281, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", P. Funk and S. Blake-Wilson, August 2008.