Man-in-the-Middle in Tunneled Authentication Protocols

N. Asokan, Valtteri Niemi, Kaisa Nyberg
Nokia Research Center, Finland
{n.asokan,valtteri.niemi,kaisa.nyberg}@nokia.com

November 11, 2002

Draft version 1.3
Latest public version in http://eprint.iacr.org/2002/163/

Abstract

Recently new protocols have been proposed in the IETF for protecting remote client authentication protocols by running them within a secure tunnel. Examples of such protocols are PIC, PEAP and EAP-TTLS. One goal of these new protocols is to enable the migration from legacy client authentication protocols to more secure protocols, e.g., from plain EAP type to, say, PEAP. In these protocols, the security of the subsequent session credentials are based only on keys derived during the unilateral authentication where the network server is authenticated to the client. Client authentication is mentioned as an option in PEAP and EAP-TTLS, but is not mandated. Naturally, the PIC protocol does not even offer this option, because the goal of PIC is to obtain credentials that can be used for client authentication.

In addition to running the authentication protocols within such tunnel, it should also be possible to use them in legacy mode without any tunneling so as to leverage the legacy advantages such as widespread use. In this paper we show that in practical situations, such a mixed mode usage opens up the possibility to run a man-in-the-middle attack for impersonating the legitimate client. For those well-designed client authentication protocols that already have a sufficient level of security, the use of tunneling in the proposed form is a step backwards because they introduce a new vulnerability.

The problem is due to the fact that the legacy client authentication protocol is not aware if it is run in protected or unprotected mode. We propose to solve the discovered problem by using a cryptographic binding between the client authentication protocol and the protection protocol.

1 Introduction

Legacy protocols often constitute a preferred means for client authentication due to their existing key management infrastructure and widespread deployment. However, when run in the open environment they may be vulnerable to identity spoofing and other attacks against protocol exchange messages.

Recently new protocols have been proposed in the Internet Engineering Task Force (IETF) working groups for running remote client authentication protocols in a protected manner. Examples of such protocols are PIC [18], PEAP [2], EAP-TTLS [10] and more recently, PANA over TLS [14]. One goal of the new protocols, is to enable the migration from existing remote authentication protocols to more secure protocols, e.g. from plain EAP to, say, PEAP.

The proposed protocols are constructed as combinations of two protocols: an outer protocol, and an inner protocol. The inner protocol is the legacy client authentication protocol. The outer protocol is used to protect the exchange of the inner protocol messages. The outer protocol provides authentication of network to client, and the inner protocol provides authentication of client to network. The new drafts allow and even consider it as a significant advantage that now a client authentication protocol (e.g. an EAP type) can be used in multiple different ways.

In the current drafts, the outer protocol is solely responsible for the generation of session key material. Therefore the session key material is based only on unilateral authentication where the network server is authenticated to the client. Client authentication in the outer protocol is mentioned as an option in PEAP and EAP-TTLS, but is not mandated. If client authentication is indeed possible in the outer protocol, there will not be much need for the inner client authentication protocol. This is why, for example, the PIC protocol does not even offer the option of client authentication in the outer protocol.

To summarize, we can state three facts. First, the client authentication protocol can now be used in multiple ways, e.g., both with and without protective tunneling. Second, the session keys are derived solely on the basis of the network authentication protocol. Third, the client authentication protocol is not aware of the outer protocol. The combination of these facts opens up the opportunity for a man-in-the-middle to impersonate the legitimate client.

In this paper we propose to solve this vulnerability by cryptographically binding the inner protocol and the outer protocol. If the inner client authentication protocol produces a session key, this binding *does not require any changes* to the inner protocol. This is a very important requirement because the whole point of using legacy protocols is that they are already widely deployed. The basic idea is to use session keys produced by the inner client authentication protocol and the outer tunneling protocol in such a manner that each party can verify that its peer has knowledge of all the session keys. This check can be explicit or implicit, for example by deriving the session keys from both pieces so that, for example, they become available only to peers that know both pieces of the secret key material.

In section 2 some background information is provided. Brief overviews of PEAP, EAP-TTLS, PIC and the latest proposal of this type, POTLS, are given highlighting their common tunneling approach. Then the man-in-the-middle attack is described in section 3. It is also shown that all similar combined protocols, where an inner protocol is run through a protected tunnel provided by the outer protocol are potentially vulnerable to this attack. Ways of removing the vulnerability are presented in section 4. Related considerations like the use of weak password-based protocols are discussed in section 5. The conclusions are summarized in section 6. Finally, a brief status update is provided in section 8

2 IETF Drafts for Tunneled Authentication Protocols

The Extensible Authentication Protocol (EAP), described in RFC2284 [13], is a standard framework for support of multiple authentication methods. By using EAP in a system, it is possible to enable the system to use of a number of legacy authentication schemes, including smart cards, Kerberos, public key mechanisms, One Time Passwords, cellular authentication mechanisms like GSM [11] or UMTS AKA (Universal Mobile Telecommunication System, Authentication and Key Agreement protocol) [3], and many others.

EAP is a general authentication protocol run between a client and a server, possibly via a *front-end authenticator*. It does not require the front-end authenticator to authenticate the client itself. Instead, it allows the front-end authenticator to proxy authentication messages to an *authentication agent*, and inspect the packets to determine if the authentication was successful. The authentication agent may use yet another server to help authenticate the client. We call this third server the *home authentication server*. Use of EAP allows new authentication methods to be developed without requiring deployment of new code on the front-end authenticator. The front-end authenticator acts as a "pass-through", and need not understand specific EAP methods.

Since its deployment, a number of weaknesses in EAP have become apparent. These include the lack of protection of the user identity or of the EAP negotiation, and no standardized mechanism for key exchange [2].

One of the main purposes of the new protocols in IETF working groups is to fix these perceived weaknesses of EAP, while still retaining the primary benefit of EAP encapsulation: a standard interface between the inner client authentication protocol and the outer authentication protocol allowing support for multiple existing remote authentication protocols.

In this section brief descriptions of PEAP [2], EAP-TTLS [10], PIC [18], and PANA over TLS [14] are given.

2.1 Protected EAP

Protected EAP (PEAP) [2] provides wrapping of the EAP protocol within TLS [8]. It claims to provide user anonymity and built-in support for key exchange.

The relationship between the EAP peer (client), front-end authenticator, known as the "network access server" (NAS) in PEAP, and an authentication agent, known as the "back-end authentication server" in PEAP, is depicted in Figure 1. As described in the figure, the EAP conversation "passes through" the NAS on its way between the client and the back-end authentication server. While the authentication conversation is between the EAP client and the back-end authentication server, the NAS and back-end authentication server need to have established trust for the conversation to proceed.

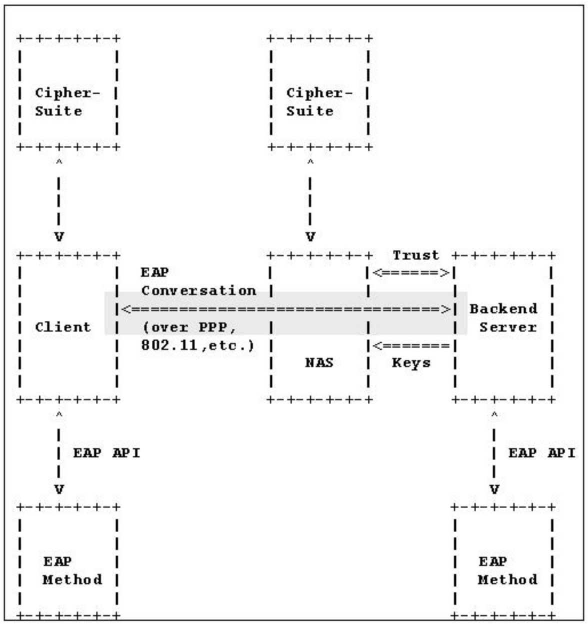

Figure 1: Relationship between EAP client, back-end authentication server and NAS in PEAP [2]

The client and the back-end server first set up a TLS channel over EAP. The client authentication protocol between the client and the back-end server is encrypted and integrity protected within this TLS channel. As a result, the NAS does not have knowledge of the TLS master secret derived between the client and the back-end authentication server, and cannot decrypt the PEAP conversation. The back-end server derives master session keys from the TLS master secret via a one-way function and conveys them to the NAS which can then use these session keys to protect subsequent link-layer communication between it and the client.

The PEAP draft [2] does not discuss the format of the attributes used to communicate the master session keys from the back-end authentication server to the NAS. AAA (Authentication, Authorization, and Accounting) carrier protocols such as RADIUS [17] or DIAMETER [7] can be used for this purpose.

The steps of PEAP operation are as follows.

1. Establish TLS connection over EAP.

   The TLS record protocol provides a secure connection between the client and the back-end authentication server

2. Authenticate TLS server.

   The TLS handshake protocol is used for server authentication.

3. Authenticate user.

   The user of the client authenticates by tunneling another EAP mechanism (e.g. Generic Token Card) inside the EAP-TLS connection. The back-end authentication server may have to contact another server, a home authentication server, to get the user authentication information validated.

4. Generate session keys.

   Using the TLS Pseudo-Random Function (PRF), the client and the back-end server generate key material for use between the NAS and the client.

(5. Transport session keys.

The session key is transported from the server to the NAS using e.g. Radius attributes and secure connection.)

Let us consider how PEAP is typically intended to be used with a legacy protocol. In a recent Internet draft, EAP SIM GMM authentication [6], an application of PEAP to GSM authentication is presented. The architectural overview of this new EAP method using SIM is depicted in Figure 2. As discussed, the session keys to protect the link between the client and the NAS are derived from the TLS master secret. The draft states "We rely on PEAP for session key derivation so that any other EAP client authentication method could be utilized without duplicating the complexity of generating a secure key hierarchy." In other words, the session key material agreed between the client and the back-end server as part of the GSM authentication are not used.

Remark on terminology. The PEAP back-end server is called EAP Server in the EAP SIM GMM draft (see Figure 2). The "Home PLMN GPRS Network" plays the role of the home authentication server referred to above in Step 3.

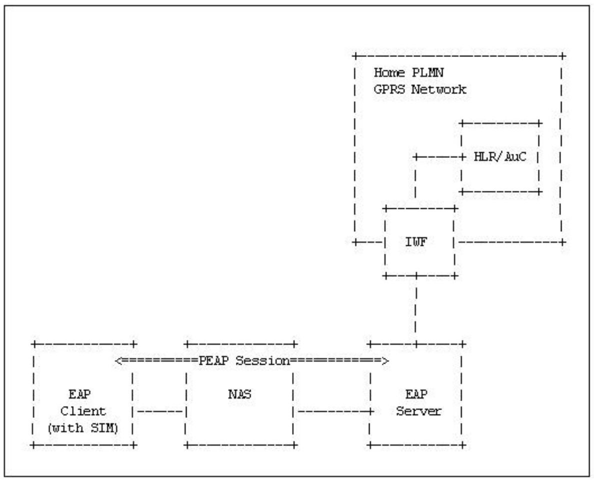

Figure 2: EAP SIM GMM Authentication: Architecture Overview [6]

As another example of PEAP usage, the message flow in PEAP with EAP AKA client authentication is depicted in Figure 3. The wireless local area network (WLAN) authentication server is in the role of the back-end server. The terminal has a smart-card containing a secret key. The same key is available in the subscriber database of HSS (Home Subscriber Server) located in the home network of the subscriber. The terminal sends a cellular identity like the International Mobile Subscriber Identity (IMSI) as part of EAP AKA. The WLAN server uses a UMTS protocol like MAP (Mobile Application Part) or DIAMETER to send the IMSI to the HSS. HSS returns an AKA authentication vector, which is a quintuplet containing a challenge (RAND), an authenticator for the challenge (AUTN), expected response (XRES), and session keys IK (for integrity) and CK (for confidentiality) to the WLAN server. The WLAN server forwards RAND and AUTN to the terminal via EAP AKA. The terminal can now verify AUTN to confirm that RAND comes from HSS, and compute its own response RES and the keys IK and CK. The terminal sends RES back to the WLAN server using EAP AKA. If RES and XRES are the same, WLAN server considers the terminal to be authenticated. The AKA protocol was defined as part of the third generation (3G) cellular standardization activities. More detailed descriptions of 3G protocols are available elsewhere, e.g., [12].

Again, only the TLS master secret is used to derive the session keys to be used to protect the WLAN link. The secret key material carried within the 3G AKA [12] authentication quintuplets is not used.

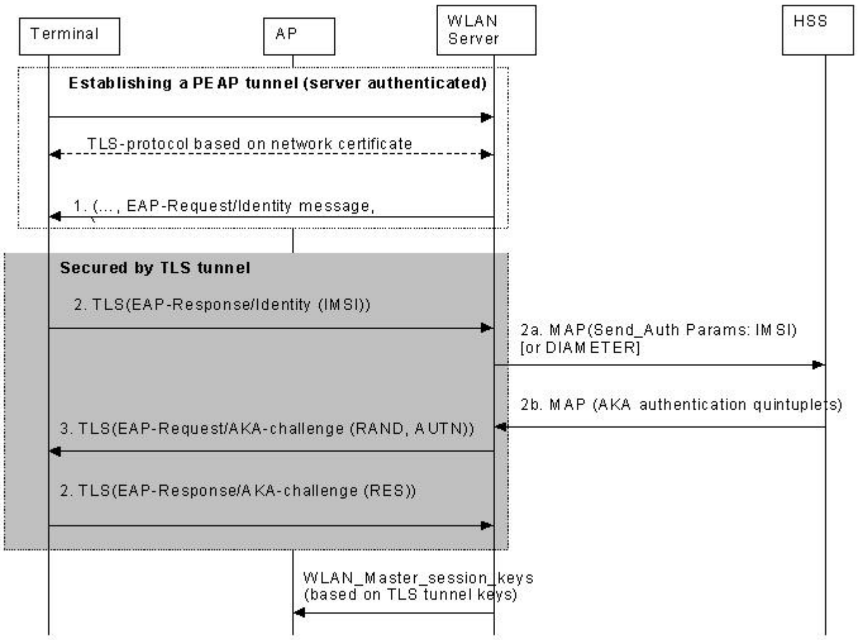

Figure 3: PEAP with EAP AKA: Example Message Flow

2.2 Tunneled TLS

The architectural view of EAP-TTLS [10] is very similar to that of PEAP, see Figure 4. EAP-TTLS claims to allow legacy password-based authentication protocols to be used with existing authentication databases, while protecting the security of these legacy protocols against eavesdropping, man-in-the-middle and other cryptographic attacks.

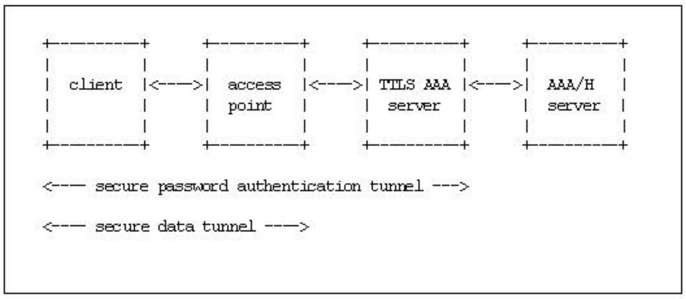

Figure 4: The network architectural model for EAP-TTLS [10]

EAP-TTLS also allows client and the back-end server to establish keying material for use in the data connection between the client and the front-end authenticator. The keying material is established implicitly between client and the back-end server based on the TLS handshake. As with PEAP, EAP-TTLS derives sessions keys by applying a pseudo-random function to the TLS master secrets and other input. The back-end server distributes derived sessions keys to the front-end authenticator using the AAA protocol. The client derives the same keys on its own.

Remark on terminology. The TTLS terminology is mapped to our terminology as follows. The front-end authenticator is called the “access point”. The authentication agent is the “TTLS AAA server”. The home authentication server is the “AAA/H server”.

2.3 PIC: A Pre-IKE Credentials Provisioning Protocol

The PIC protocol [18] is a method to bootstrap IPsec authentication via an “Authentication Server” (AS) and user authentication mechanisms (e.g., based on EAP and RADIUS [17]). The client machine communicates with the AS using a key exchange protocol where only the server is authenticated. The session keys derived by this process are used to protect the user authentication between the client and the “back-end authentication server” (see remark on terminology below). Once the user is authenticated, the client machine obtains credentials from the AS that can be later used to authenticate the client in a standard IKE exchange with an IPsec-enabled security gateway. The latter stage does not require user intervention.

A primary design goal of PIC is to support legacy authentication protocols without requiring any changes in them. In particular, a basic assumption is that“the back-end authentication server software and database cannot be modified.” [18] Further, the primary motivation focus of PIC is on legacy authentication based on passwords or other similar weak key material.

Remark on terminology. The network entity called “back-end server” in PIC plays a different role compared to the “back-end server” in the description of PEAP, see Figure 1. The common terminology we have been using above maps to the terminology of the PIC draft as follows. The PIC draft does not have a front-end authenticator. The “AS” in the PIC draft is what we call an authentication agent. The “back-end auth server” in the PIC draft is what we call a home authentication server. The IPsec-enabled gateway does not map to any entity in our terminology because it comes into play after PIC is completed.

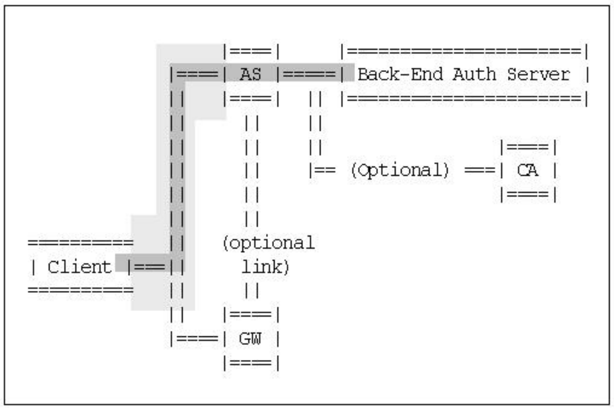

Figure 5: Relations between the PIC entities [18]

PIC embeds EAP messages [13] in ISAKMP payloads to support multiple forms of user authentication. If this user authentication succeeds, the client machine can request and obtain credentials from the AS. The term “credentials” is used to mean both digital certificates and shared secret keys. It is possible to define other types of credentials in the future. The credentials are intended to be used by the client to perform regular IKE authentication with an IPsec-enabled gateway.

As with the other protocols discussed above, the PIC draft does not specify the protocol used between the AS and the back-end authentication server. The PIC protocol is defined between the Client and the AS.

The PIC draft [18] describes the four main stages of the proposed PIC protocol as follows:

1. An optional round of messages provides partial protection of the AS from denial-of-service attacks by verifying that the initiator of the exchange is reachable at the purported source IP address. This is done before any significant CPU or memory resources are consumed by the AS.
2. The protocol establishes a one-way authenticated channel from the client to the AS in which only the server is authenticated.

3. User authentication is performed over this secured channel. User authentication information is transported using EAP tunneled within ISAKMP.
4. The AS sends the client a (typically short-term) credential, which can be used in subsequent IKE exchanges. This credential can be thought of as a certificate, or a private key generated or stored by the AS and accompanied by a corresponding certificate. It may also be a symmetric secret key, or other information for deriving such a key. In this case, the secret key is made available to the IPsec-enabled gateway using the optional link between it and AS.

In stage 4 the created ISAKMP tunnel is used for the secure provisioning of credentials for successfully authenticated users.

An example message flow for PIC with EAP AKA is given in Figure 6.

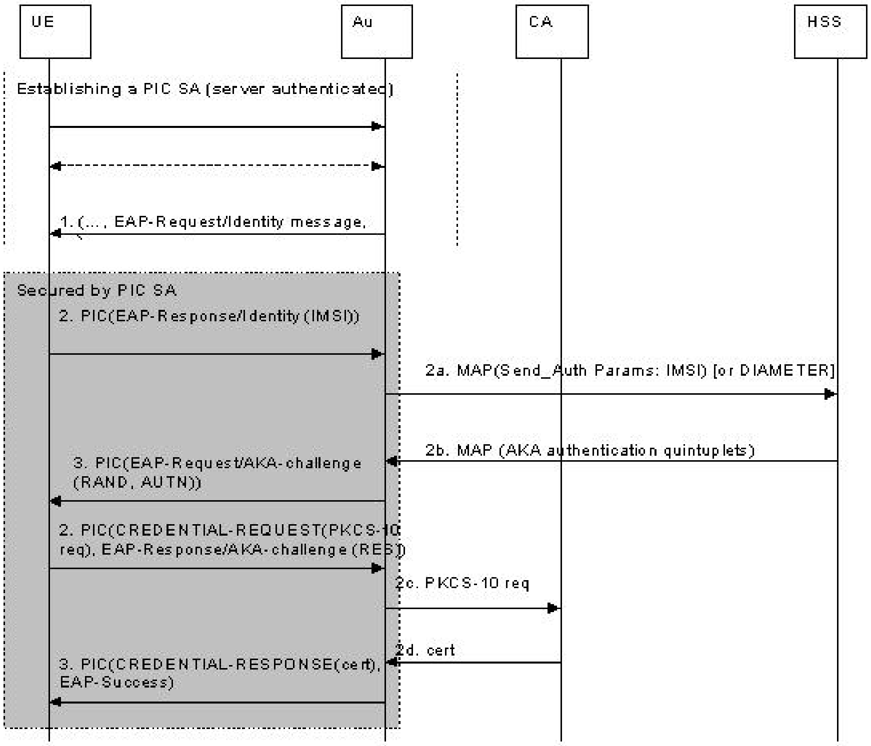

Figure 6: PIC with EAP AKA: Example Message Flow

2.4 PANA over TLS

PANA over TLS (POTLS) [14] specifies a method to carry authentication information over TLS between PANA Client (PaC) and PANA Authentication Agent (PAA).

According to the POTLS draft [14], POTLS is designed for carrying any client authentication protocol information including EAP messages. It is also possible to use a TLS certificate for authenticating a PaC without using any other authentication protocol. PANA over TLS supports combining multiple types of authentication to authenticate a PaC. For example, it is possible to use a TLS client certificate for authenticating an IP address of the PaC and then use EAP for authenticating the user of the PaC.

In our terminology, the PaC is the client, and the PAA is the authentication agent. As with the other methods, the TLS master secret agreed between the PaC and the PAA are used for deriving subsequent security associations. Although POTLS can be used with TLS client certificates, the draft states that

"First, unlike IKE, TLS does not require mutual authentication for establishing a secure communication channel between peer entities. It would not be a realistic requirement for assuming mutual authentication especially in roaming environments." Thus, the common use of POTLS is likely to consist of carrying out a legacy client authentication protocol through a server authenticated TLS connection.

3 Man-in-the-Middle Attack

The security properties of TLS are analyzed in Appendix F of [8]. It is noted that "whenever the server is authenticated, the channel is secure against man-in-the-middle attacks". Further, it is noted that "if the server is authenticated, its certificate message must provide a valid certificate chain leading to an acceptable certificate authority." Moreover, it is stated as a necessary requirement that "each party is responsible for verifying that the other's certificate is valid and has not expired or been revoked". Also in a highlighted warning it is reminded that "server authentication is required in environments where active man-in-the-middle attacks are a concern."

The same security consideration as in Appendix F of the TLS specification has been the motivation and the security basis of the new proposed protocols for tunneled client authentication. As we saw, all protocols using TLS mandate the use of a server certificate for Full Authentication (see [8]). Thus TLS provides protection against man-in-the-middle attacks on contents carried over the TLS connection.

So why can the tunneling approach go wrong? There are two reasons:

1. The legacy client authentication protocol is used in other environments, e.g., plain EAP without any tunneling, or without any encapsulation at all, e.g., direct use of one-time passwords, or cellular authentication protocols.

   The current IETF drafts do not provide any means for the inner protocol to verify if it is used with tunneling or not. It is considered as a major benefit of the protection protocols that no changes are required in the legacy protocols to be tunneled.

2. The client fails to verify the server certificate properly.

   Even if in this situation the client is to be blamed for the failure, it is not acceptable for the network services that the security depends to such extent on a single action of the client.

The active attack by a Man-in-the-Middle (MitM) proceeds as follows:

1. MitM waits for a legitimate device to enter an untunneled legacy remote authentication protocol and captures the initial message sent by the legitimate client.
2. MitM initiates a tunneled authentication protocol with an authentication agent
3. After the tunnel is set up between MitM and the authentication agent, the MitM starts forwarding legitimate client's authentication protocol messages through the tunnel.
4. MitM unwraps the legacy authentication protocol messages received through the tunnel from the authentication agent and forwards them to the legitimate client.
5. After the remote authentication ended successfully, MitM derives the session keys from the same keys it is using for the tunnel.

Now we illustrate how the MitM attack can be instantiated in the various protocols discussed in section 2.

Figure 7 shows how the MitM attack works in PEAP with EAP AKA as the example client authentication protocol in a WLAN access authentication setting. The victim terminal assumes that the MitM is a UMTS radio access network. It is important to note that UMTS AKA, and EAP AKA, are both well-designed protocols in themselves. They provide mutual authentication of the user terminal and the radio access network and result in strong symmetric session keys. But the attempt to improve the security of EAP AKA by tunneling it through PEAP opens up the new vulnerability.

Figure 8 shows an example of how the MitM attack can work in PIC environment. The protection tunnel of PIC is created not by TLS but using a simplified ISAKMP which provides unilateral authentication of the server to the client. In PEAP and POTLS client authentication in the TLS handshake is mentioned as an option. In PIC, client authentication tunnel establishment is not possible.

Generalizing from these examples, we conclude that all tunneled authentication protocols potentially exhibit the same vulnerability. The MitM attack becomes possible when the protection tunnel is based on unilateral server authentication only and the security of the session credentials depends only on the tunnel. For such protocols, the attack can be launched either when the client is authenticated based on

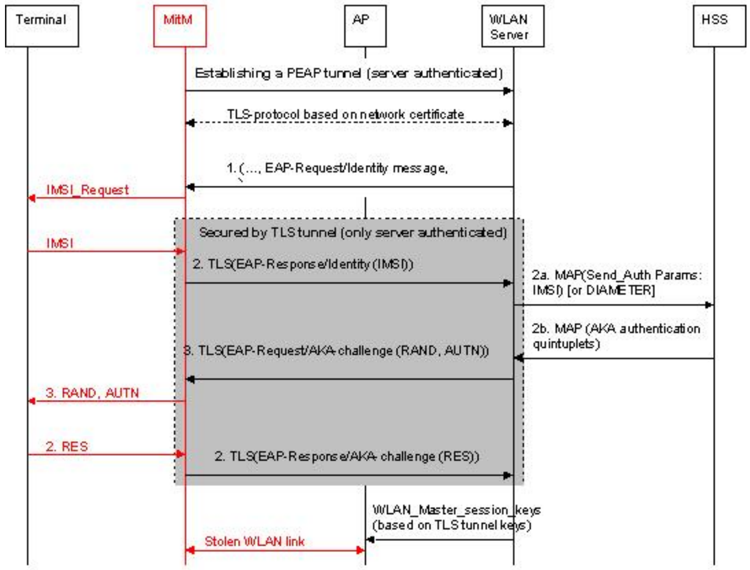

Figure 7: Man-in-the-Middle in PEAP, e.g., with EAP AKA

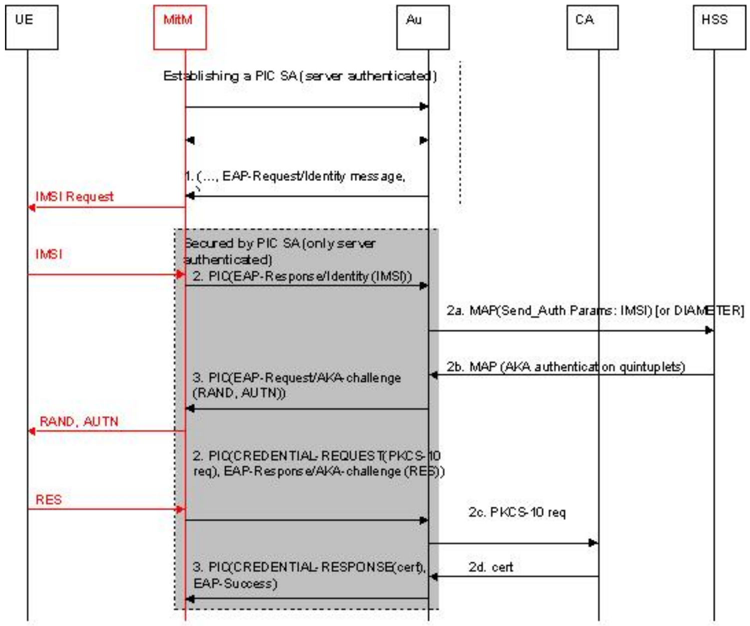

Figure 8: Man-in-the-Middle in PIC, e.g., with EAP AKA

the same identity and the same authentication token in different environments, or when the client fails to properly authenticate the server while building the tunnel. As noted earlier in section 3 that the attack on the composed protocol does not imply that the inner authentication protocol is insecure.

Surprisingly, this type of protocol composition is very popular. There are other examples of this type. The most widespread of these may be the use of HTTP authentication [9] through a TLS tunnel (i.e., to an `https` URL). Many web sites use this type of a scheme for controlling access to their resource;, for example, on-line access to bank accounts are usually controlled this way. Naturally, if the authentication method is usable without a TLS tunnel (e.g., to a plain `http` URL), a MitM attack is possible.

Next, we show that it is possible to design the protected tunneling protocols so that they are not vulnerable to the MitM attack, if the inner authentication protocol produces a session key. We do this by designing the tunneling in such a manner that the session keys and credentials become known only to the properly authenticated parties.

4 Cryptographic Binding

The vulnerability discussed above is due to the fact that the entity that after the protocol gets hold of the client's copy of the session keys or credentials is not properly authenticated by the protocol. Two approaches to provide the necessary authentication are possible: either by implicit authentication or explicit authentication. The goal of implicit authentication is to ensure that the correct session keys or credentials are accessible only to the legitimate parties. In explicit authentication a separate authentication step is performed to verify that the master secrets from which the session keys are derived, are in possession of legitimate parties only. In both cases, the authentication is provided by a cryptographic binding between the inner and the outer protocols.

These solutions are applicable to mutual authentication protocols that are constructed as a combination of two authentication protocols. The outer protocol is assumed to support the construction of a secure tunnel based on server authentication. As we saw, this is the case with all the protocols cited in section 2.

The inner authentication protocol must satisfy one of the following requirements:

1. The protocol results in a session key; i.e., it is an authentication *and* key agreement protocol; or
2. The client's long-term authentication key is accessible for derivation of session keys directly, outside the authentication protocol.

A typical example of a protocol of the first type is EAP AKA. Figure 6 illustrates how EAP AKA can be used within the PIC protocol. The AKA quintuplets contain 256 bits of secret session keys which would be readily available for session key derivation by the authentication agent. These AKA session keys are also available in the client's device. Similarly, most existing EAP types provide a method for session key derivation. It is likely that a common key agreement framework will be defined for EAP [1]. If such a framework is defined, all EAP types can result in session keys. For such protocols, no changes are required to provide the necessary cryptographic binding.

In the second case, it is necessary to specify an additional key derivation application that has direct access to client's authentication key. This would be in addition to the existing authentication protocol and is to be judged separately for each case.

Let K be the ultimate session key. In PEAP and EAP-TTLS the session key K is the master key that finally becomes available to the local authentication agent (access server), which uses K to derive further session keys. In PIC the key K is the key that is used to protect requesting and transportation of IKE credentials to the client.

Let S be the secret key material known to the client and the client's home authentication server (Home AAA server, or HSS, or similar). We assume S to be a session key derived in the authentication and key agreement protocol, or it can be the client's long-term authentication key. Let T denote the master key that is used to derive the secret keys for the protection tunnel. For example, the TLS master key derived in the TLS handshake of PEAP is a typical example of T.

There are two ways of using S to achieve the necessary binding between S and K. In the first method the binding is established directly by taking S in addition to T as input to the session key computation. This provides implicit authentication of the client. The second method is to make use of a cryptographic check value to verify that the client who is in possession of T is also in possession of S. This second type of binding provides explicit authentication of the client.

Both methods rely on secure communication between the network entities. On the network side some entity, let us call it "binding agent", is responsible for collecting the secret key information S and T and creating the binding value. Typically the binding agent is either the local authentication agent or the home authentication server. If S is a long term authentication key of the client, then the binding agent is preferably co-located with client's home authentication server, to avoid transfer of S across the network.

In implicit binding, the binding agent and the client each compute its copy of the session key K from S and T using a pseudo random function suitable for key derivation. The binding agent distributes the session key to the network entities that need to use it for further communication with the client.

In explicit binding, the binding agent and the client each compute its copy of a verification value V from S and T using a cryptographic hash function or a message authentication function. The client and the binding agent transfer their verification values to some network entity responsible for comparing the two verification values. If they are equal, the client is granted access to the network service. The comparing entity can be the authentication agent or the home authentication server.

It is also possible to implement both implicit and explicit binding. In such a case the explicit binding verification acts as a key confirmation for the agreed session key.

Let us consider PIC with EAP AKA as an example of how to implement the binding. A natural choice is to select the local authentication agent (Au) to be the binding agent and also the network entity to do the binding verification. In EAP AKA the AKA quintuplets are sent from home authentication server (HSS) to Au, which can use them for the binding.

In PIC, explicit binding is sufficient to guarantee the integrity of the tunnel. Note that this is not necessarily the case for all tunneling protocols. Assume MitM is running two sessions of the outer protocol, one with the client and the other with the local authentication server. Assume MITM is using T1 with the client and T2 with AS. If the outer protocol allows MitM to influence the value of the session key, then MitM can arrange T1 to be the same as T2. In this case, explicit binding of session keys alone is not sufficient to detect existence of the MitM. Therefore some data input which is specific to the client should be used in the computation of the explicit binding value.

In the case of PIC and EAP AKA the explicit verification value V can be specified for example as follows:

$$\begin{aligned} K &= h(IK, T) \\ V &= \mathrm{MAC}(K, \texttt{credential\_request}), \end{aligned}$$

where IK is the UMTS AKA integrity key, h is a pseudo random function (which use all the bits of IK and T), and the MAC computation is performed using HMAC_SHA1 [4] with key K on client-specific data input `credential_request`.

Figure 9 illustrates the position of the binding verification value in the message flow. Usually it is the case that the binding values can be transferred as an additional field in one of the existing messages of the tunnel protocol.

This fix may be viewed as a contradiction of the design goals of PIC because PIC assumes that the inner authentication protocol can be used as a black-box, whereas the solution proposed above requires that PIC has access to the session key or other key material from the inner protocol. Also, the primary target of PIC are weak password based authentication protocols. These typically do not support session key agreement. We discuss the special requirements posed by weak authentication protocols in section 5.2 in more detail.

Given these constraints, one possibility is to limit the use of PIC to those instances where the security association for user authentication is not used except through a protected tunnel. This is a policy decision involving both the client and user sides. It is difficult to enforce such a policy, especially in cases where the enforcement relies on the correct behavior of the user. Such restriction will also clearly reduce the applicability of PIC.

Another possibility is to relax the design assumptions of PIC so an *optional* explicit binding verification field is added to PIC which can be used when the underlying authentication mechanism provides session keys.

5 Other Considerations

5.1 Identity Privacy

In passing, we also remark on another motivation for tunneling client authentication protocols: identity privacy. When a client authentication protocol is tunneled, the identity of the client is protected from all passive eavesdroppers. However, there is no way to prevent an active attacker from attacking identity

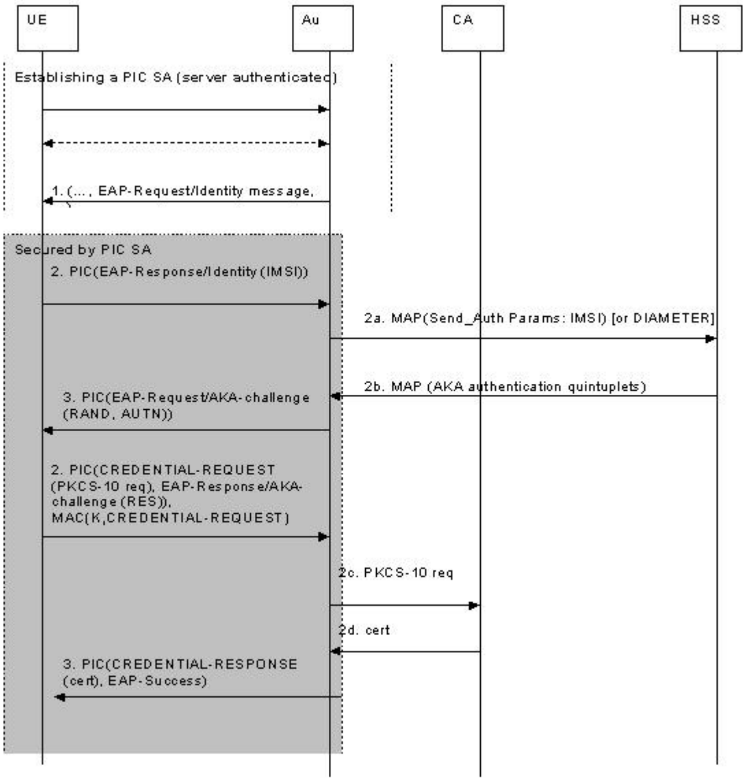

Figure 9: Binding for PIC and EAP AKA

privacy by setting up the legacy network environment, e.g., using a false GSM base station. Therefore, this kind of attack against client's identity privacy cannot be prevented if the legacy environment allows it. On the other hand, if the legacy environment has mechanisms to preserve privacy, tunneling does not bring in any benefit in this respect.

5.2 Weak Password-based Protocols

Consider a client authentication protocol based on weak secrets like passwords. Suppose further that the protocol does not use any techniques to protect the secret against guessing attacks. Many of the tunneling protocols described earlier, e.g., the PIC protocol, are particularly intended to address such weak authentication protocols, which are insecure without added protective tunneling. One may ask what is the impact of the cryptographic binding described in section 4 in such a case. Does the binding possibly improve the overall security? It is clear that if the client is unable to perform server authentication for the outer tunnel protocol, then the binding does not improve the security: an attacker could masquerade as the server in protocol run, and use the information gained in a dictionary attack on the password.

However, all the examples we considered in section 2 do assume that the client can perform server authentication for the tunnel protocol. In this case, the attacker cannot pretend to be the server end of the tunnel towards the client. If the attacker is a passive eavesdropper, he cannot perform the dictionary attack on the binding because the binding includes the outer tunnel key which is assumed to have sufficient entropy. If the attacker pretends to be the client towards the server and if the server does the binding first (e.g., if the server sends the binding verification value to the client, or if it starts using the key K or S to encrypt or authenticate messages), then he can perform a dictionary attack. But this is easily prevented by requiring that the client does the binding first and demonstrate knowledge of S and T to the server. If this step fails, the server must terminate the session.

Thus, the cryptographic binding does not *reduce* the security of a weak password-based protocol. However, to be fully secure, weak password-based protocols used with server authenticated tunnels must satisfy two conditions:

A1 *Correct server authentication*: the client **must** perform server authentication correctly, and

A2 *No mixing of authentication modes*: if a client uses tunneled authentication, it **must not** use the same authentication protocol outside secure tunnels.

Assumption A2 is unavoidable for weak password-based protocols. But it clearly diminishes the capability of the tunnelling protocol to leverage the advantages of already deployed legacy authentication protocols. Well-designed authentication protocols do not need assumption A2.

If assumption A2 can be made in general, cryptographic binding is not necessary, but the legacy advantages are lost. On the other hand, cryptographic binding is not applicable to the legacy authentication protocols that do not yield a key suitable for cryptographic binding.

Generic tunneling protocols, by definition, should be able to work with all types of authentication protocols while making as few assumptions about them as possible. Therefore a generic tunneling protocol must not take A2 as a blanket assumption to be imposed on all authentication protocols. Instead it allow the possibility of using the type of cryptographic binding described in section 4 where appropriate. This way, the tunneling protocol can be secure, generic (supporting both weak and strong authentication protocols), and non-invasive (avoiding unnecessary restrictions on strong authentication protocols).

6 Conclusion

In this paper we have shown that when a client authentication protocol is tunneled within another protocol, it is necessary for each endpoint to demonstrate that it has participated in both protocols within the authentication exchange. If this is not demonstrated then the tunneled authentication protocol is vulnerable to a Man-in-the-Middle attack.

We have also shown that the required demonstration can be provided in an implicit or explicit way in a form of a cryptographic binding between the tunnel protocol and the authentication protocol. In our proposals the binding facility is implemented in the outer tunnel protocol. It requires the authentication protocol to provide some secret key values for the use of the binding. This approach is preferred since it requires minimal or no changes to the EAP protocols. It allows for flexible and secure usage of an authentication protocol in multiple authentication environments without the authentication protocol being aware of the specific environment.

The cryptographic binding proposed in this paper does not diminish the security of the tunneled protocols in any case. If the inner authentication protocol is a weak authentication protocol based on a weak client secret, the tunnel must be constructed based on server authentication, and the client should not use the same secret in different environments. Otherwise, the protocol is vulnerable to dictionary attacks, with or without cryptographic binding. Strong authentication methods are not vulnerable to dictionary attacks, and hence should not be restricted to tunneled environments only. Hence the need for the type of binding described in this paper.

7 Acknowledgments

We thank Henry Haverinen for many interesting and useful discussions on this and other related topics. We also thank Jan-Erik Ekberg, Dan Forsberg, Hugo Krawczyk, Jarno Rajahalme, Heikki Riittinen and the participants of the IST-SHAMAN project for discussions and valuable feedback on our ideas.

8 Epilogue

We pointed out this problem to the authors of some of the proposals in early October, 2002. Since then a number of positive developments have taken place. The IETF EAP working group has recognized the problem of securely binding a sequence of authentication methods together. It is now marked it as an open issue to be solved [5]. The latest version of the POTLS [15] also acknowledges the problem and defines the necessary messages required for an explicit key binding. Another draft [16] describes the type of solutions prescribed in this paper.

We can therefore expect that in due course of time the problem will be solved in all the identified specifications. The surprising aspect was the sheer number of different protocol combinations where the same problem has manifested. These are not academic toy protocols, but protocols with a good prospect of being deployed and used widely. The problem typically occurs when an authentication protocol is used in a new environment. Such use is inevitable. A hallmark of a successful protocol is when it is used for purposes that the authors did not explicitly foresee, but which are within the defined scope and satisfy the specified requirements.

References

[1] Bernard Aboba and Dan Simon. The EAP Keying Problem, October 2002. IETF personal draft `draft-aboba-pppext-key-problem-03.txt`.

[2] H. Andersson, S. Josefsson, Glen Zorn, Dan Simon, and Ashwin Palekar. Protected EAP Protocol (PEAP), September 2002. IETF personal draft `draft-josefsson-pppext-eap-tls-eap-05.txt`.

[3] J. Arkko and H. Haverinen. EAP AKA Authentication, October 2002. IETF personal draft `draft-arkko-pppext-eap-aka-05.txt`.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology – Crypto 96*, number 1109 in Lecture Notes in Computer Science, pages 1–15. Springer Verlag, 1996.

[5] L. Blunch, J. Vollbrecht, and Bernard Aboba. Extensible Authentication Protocol (EAP), October 2002. IETF *pppext* working group draft `draft-ietf-pppext-rfc2284bis-07.txt`.

[6] Adrian Buckley, Prasanna Satarasinghe, Vladmir Alperovich, Jose Puthenkulam, Jesse Walker, and Victor Lortz. EAP SIM GMM Authentication, August 2002. IETF personal draft `draft-buckley-pppext-eap-sim-gmm-00.txt`.

[7] Pat R. Calhoun et al. Diameter Base Protocol, June 2002. IETF *aaa* working group draft `draft-ietf-aaa-diameter-11.txt`.

[8] T. Dierks and C. Allen. The TLS Protocol Version 1.0, January 1999. IETF RFC 2246.

[9] J. Franks et al. HTTP Authentication: Basic and Digest Access Authentication, June 1999. IETF RFC.

[10] Paul Funk and Simon Blake-Wilson. EAP Tunneled TLS Authentication Protocol (EAP-TTLS), February 2002. IETF *pppext* working group draft `draft-ietf-pppext-eap-ttls-01.txt` (expired).

[11] H. Haverinen and J. Salowey. EAP SIM Authentication, October 2002. IETF personal draft `draft-haverinen-pppext-eap-sim-06.txt`.

[12] Heikki Kaaranen, Siamäk Naghian, Lauri Laitinen, Ari Ahtiainen, and Valtteri Niemi. *UMTS Networks: Architecture, Mobility and Services.* John Wiley & Sons, 2001.

[13] J. Vollbrecht L. Blunk. PPP Extensible Authentication Protocol (EAP), March 1998. IETF RFC 2284.

[14] Yoshihiro Ohba, Shinichi Baba, and Subir Das. PANA over TLS (POTLS), September 2002. IETF personal draft `draft-ohba-pana-potls-00.txt`.

[15] Yoshihiro Ohba, Shinichi Baba, and Subir Das. PANA over TLS (POTLS), October 2002. IETF personal draft `draft-ohba-pana-potls-01.txt`.

[16] Jose Puthenkulam, Victor Lortz, Ashwin Palekar, Dan Simon, and Bernard Aboba. The compound authentication binding problem, October 2002. IETF personal draft `draft-puthenkulam-eap-binding-01.txt`.

[17] C. Rigney, A. Rubens, W. Simpson, and S. Willens. Remote Access Dial In User Service, April 1997. IETF RFC 2138.

[18] Y. Sheffer, H. Krawczyk, and Bernard Aboba. PIC, A Pre-IKE Credential Provisioning Protocol, October 2002. IETF *ipsra* working group draft `draft-ietf-ipsra-pic-06.txt`.