SIMULATION OF MITM IN PEAP WITH HOSTAP

SIARHEI SINIAK

Abstract. The original goal was the following: So if I am the very user who skips certificate verification, and the network uses the very exact configuration — PEAPv0,v1 with MSCHAPv2. Then why not to abuse other unsuspecting users, by writing a real life exploit.

1. Introduction

I'd like to present analysis of PEAP, with a consequent development of MITM exploit. The point is to simulate the attack. As a starting point was taken a paper [1], dated to 2002, retrieved and published in 2013. There we observe the research on tunneled authentication protocols. Many legacy, and consequently widespread, protocols are to updated to corrsepond to latest security standards, where as it is highly desirable we'd like to utilize the benefits of legacy code and infastructure. If the protocol is affected by weak-password or unprotected client's identity problems, then it can be tunneled by an outer protocol. In this case the network access server is authenticated to the client by the outer authentication protocol, then the client proceeds with inner authentication protocol (a legacy one) within a tunnel to authenticate himself to the server. The protocol is finished, both parties end up with some tokens, in case of successful authentication. If we are talking about encrypted network access, then the result tokens are session keys, that determine the encryption of further messages to be exchanged via link-layer being authenticated.

2. Cryptobinding

The inner protocol should be cryptographically binded with the outer protocol. The paper [1] mentiones two ways:

(1) an implicit one, i.e. the resulting session key K is obtained by a one-way hash function, that involves both outer protocol key material T, and the secret key S from the inner protocol.
(2) an explicit cryptobinding, when the special verification value V is generated the same way from T and S, and verified by some authentication entity for being equal among parties.

To decrease imposed alterations in legacy protocols, all this concerns are to be taken into account at the outer protocol level.

It's very important to add cryptobinding, because otherwise the inner authentication protocol is performed being unaware of whether the protected tunneling exists or not. Such a security scheme is vulnerable to man-in-the-middle attack.

Date: December 27, 2016.

3. PEAP with MSCHAPv2

Let's consider PEAP with MSCHAPv2, that is often used in Wi-Fi networks to authenticate clients. The main scheme is called WPA-Enterprise, and resembles usual WPA/WPA2 standard of encrypted wireless communication. The principal difference is in authentication and session key derivation. When the usual WPA/WPA/2 standard relies on a single passphrase, that is shared among all hotspot users, Enterprise variation allows full authentication via EAP state machine. It is a general purpose authentication framework for a diverse collection of legacy, widespread authentication schemes. For example, you can utilize Windows Server user database by configuring a network access server with PEAP with MSCHAPv2 that connects via RADIUS protocol to the server and authenticates user within a protected TLS tunnel through MSCHAPv2 scheme. At first a client authenticates a NAS by verifying his network certificate, it happens during ths TLS Handshake protocol. Having a successfuly authenticated and authorized server, parties initiate a TLS tunnel, by generating a common TLS master key. The TLS scheme allows to derive a common secret for both ends, and is secure against man-in-the-middle attacks, if only the client or the server does not skip the certificate verification. It is worth to note that many platforms allows this or that way, to ignore server's network certificate verification, leaving users defenseless. The practice of such a freevolent behaviour comes due to self-signed certificates being used for TLS tunnel authentication and operation. And if corrseponding authorities does not bother to provide the certificate to its users and sometimes even stimulate verification omitting, they are usually left with the choice to use insecure network, or to reject its resources. Some would note that we can apply certificate pinning, that is a bit more secure than not to verify it at all. Yes, we can indeed.

4. Tools analysis

The original goal was the following: So if I am the very user who skips certificate verification, and the network uses the very exact configuration - PEAP with MSCHAPv2. Then why not to abuse other unsuspecting users, by writing a real life exploit.

Among the available tools, hostap project [6] looked as very prominent. Because it is the implementation of both client and server side of wireless network encryption schemes. The codebase is up-to-date, widely sprea, actually all Android phones, and for sure almost every linux machine uses its wpa_supplicant, that is a client application for authentication and session key derivation in Wi-Fi networks. The hostapd application represents a server-side counterpart to wpa_supplicant. The application architecture is modular, and contains implemented EAP state machine for both peer and server. Even more, there is a working example, that simulates communication of two EAP state machines - peer and server. The configured protocol is the exact PEAP with MSCHAPv2.

I can't find any serious disadvantages in this direction. Though few facts should be mentioned. There is a presentation dated to 2008, from the security conference SHMOOCON 2008 [8]. Two people were talking about server impersontation to real clients, but their attack aims to collect victim's messages during MSCHAPv2 protocol, then sends authentcation completed successfuly to them. Those messages allows quick dictionary attack onto the password. They even presented a patched version of hostapd (hostapd-wpe [7]) that implements their exploit. The server gathers special hashes, that can be cracked by their utility, by name asleap. Though, it is definetely recalls to our goal, it is a different attack. They are cracking weaknesses of password hashing in MSCHAPv2, and we are going to proxy the whole protocol conversation and are not interested in its contents. So you see hostap was already used for similar purposes. I am not aware of how quickly the password attacking happens, but I think it takes more time than a simple pass-through of the messages between a client and a real NAS via MITM node.

5. MitM attack simulation and its code

The code [2] we present only simulates the attack, and is not yet a real life exploit. Let's left the estimation of how far it is away from one to others.

A straight-forward implementation you may find in [1], it is as follows:

(1) MitM waits for a legitimate device to enter an untunneled legtacy remote authentication protocol and captures the initial messages 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 messages through the tunnel.
(4) MitM unwraps the legacy authentication protocol messages received through the tunnel from the authentication agent and fowards 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.

The original eap_example was extended to 4 EAP state machines, i.e. Bob Peer, Alice Server, Eve Peer and Eve Server. Here Bob plays the role of a victim, and Alice is the original server. Eve Peer and Server state machines represent a man-in-the-middle, where it is onvious, that Bob Peer communicates Eve Server, and Alice Server authenticates Eve Peer.

In a real life, MitM can intercept by producing a hotspot with the same "outlook" but with more powerful signal than the original server. If the network has no encryption, then the quickest way to perform MitM attack is to enable mobile internet on your Android phone, then create a hotspot with a name equal to the access point under attack. After you need to come close to the victim. On the internet there is aircrack-ng package, that allows to send a direct deauthentication packet, so that client's supplicant is to reinitiate session, and with a high certainty, your phone will be a chosen one as NAS.

But such a scheme does not work for encrypted network communication, unless you know the password to impersonate a server, or any other relevant secret for correct authentication.

In our case, i.e. with WPA-Enterprise hotspot, configured with PEAP with MSCHAPv2, MitM can be applied, but mobile internet won't work, as the authentication occurs at the real network access server, that we can not simulate. We don't know user password.

We can impersonate NAS during TLS Handshake protocol. Because the only proof of server's identity is the network certificate, presumably self-signed or provided with a chain toward one of the trusted Certificate Authorities. But if the

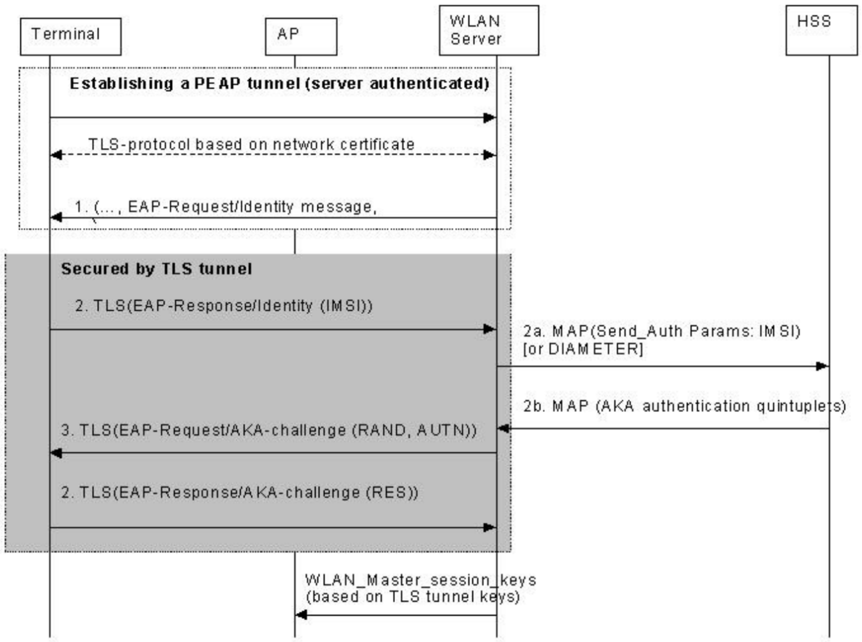


Figure 3: PEAP with EAP AKA: Example Message Flow

FIGURE 1. Original PEAP Session

client does not verify it, and such an option is available uniquetuously, then MitM attack is trivial. All we need is to participate in TLS Handshake communication.

It is the first phase in PEAP. After that the MSCHAPv2 protocol takes place. It is tunneled through TLS protocol.

On the figures 1, 2, 3 you may see an original PEAP session, the general session of MitM attack, and clarified MSCHAPv2 proxying in our implementation.

5.1. **Codebase analysis.** To acheive the desired behaviour it is important to understand how the particular implementation operates. Luckily, the code correspondes wll with RFC 4317, where the architecture of EAP state machine is defined in details. There are both EAP Peer and EAP Server state machines. For the EAP Server the stand-alone authenticator was considered.

First of all we've tested the dfeault behaviour, it was of great importance to see when the vulnerability exists and when it does not. Cryptobindings mentioned in [1] are implemented indeed: the session key is derived by Pseudo Random Function Plus (PRF+), it is based on TLS master secret T and phase2 key S derived near the end of inner protocol authentication. The detailed algorithm is available in [4]. It is worth mentioning that the version 5 [3] of the same draft states for session key to be derived solely based on TLS master secret T. The search on the internet says that Cisco routers has crypto binding since 2004, but the functionality is optional. In hostap, we've discovered that TLS Cryptobinding (the actuall name of binding protocol) is always initiated by the server, and includes not only explicit binding,

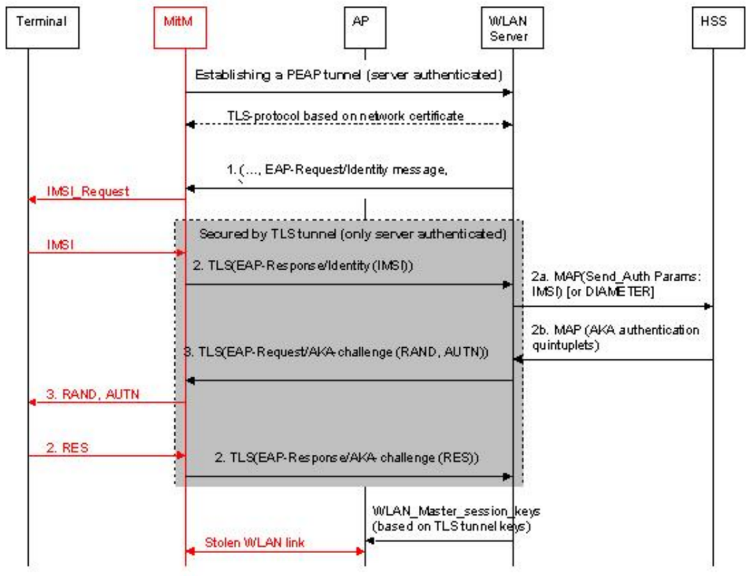


Figure 7: Man-in-the-Middle in PEAP, e.g., with EAP AKA

FIGURE 2. General session of MitM attack in PEAP

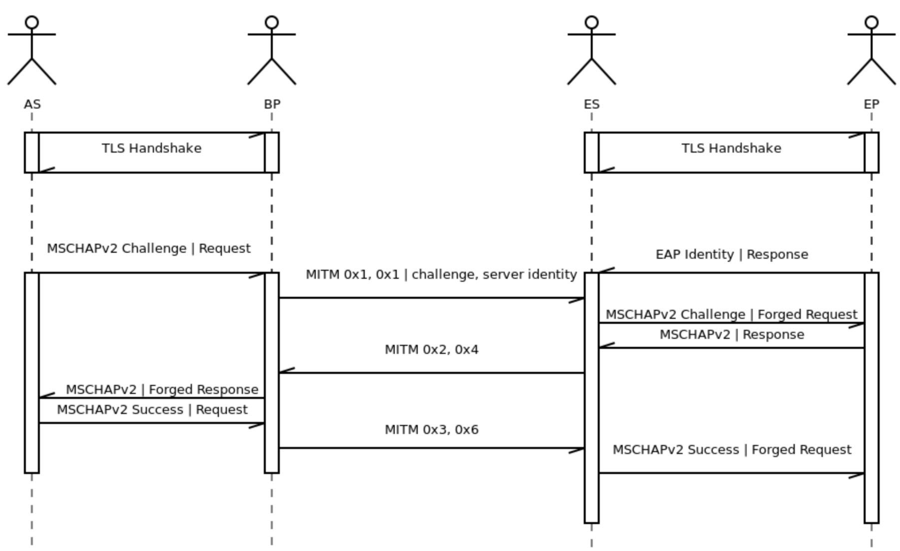


FIGURE 3. Clarified MSCHAPv2 Proxying

but as well generates session key based on T and S. If the crypto binding are disabled, the behaviour is from [3], i.e. vulnerable to MitM attack, of course if only the client does not verify the network certificate.

From the code I see that:

(1) cryptobinding are only present for PEAPv0, PEAPv1 implementation does not have this feature, and in many aspects resembles PEAPv0.
(2) client can't ask the server to proceed with cryptobinding, he can only reject the connection. It happens, when client's settings force cryptobinding, but the server does not use them, or makes it optional in hostap config.

So we disable Bob Peer's verification of server's certificate, i.e. provide no certificate in config. Then to disable cryptobinding we force PEAP version on both sides to be the first. To make some fun, MitM verifies server's certificate, but forces PEAPv1. Additionally, Eve Peer and Serer are left without any user password. Though to make it easier, username is left unchanged as in original eap_example code. Anyway it can be obtained , as the client sends his identity twice, before TLS Handshake (according to PEAP), and within TLS tunnel before the inner authentication protocol. And it is in plain text.

5.2. **Specific notes about hostap implementation.** Now we are going into more details with explanation of hostap implementation.

Originally EAP state machine does not allow pending responses and requests. But in the MitM attack Eve Peer and Server are to wait for Bob Peer and Alice Server from time to time. It appears that such a behaviour was necessary and hostap developers provided this functionality. On our side we are using it a lot, though modification to the PEAP code was necessary to correct behaviour. Pending functionality played an important role in interruption of state machine with its consequent resuming. Usually TLS has the protection against replay attacks, and we were not able to replay the PEAP message for Eve's state machines, when we've obtained necessary data, to continue with MSCHAPv2 proxying. But pending allows it by a simple callback, that keeps decrypted data from the correct message, and resumes later the method with the same data, stored at the previous step. The new packet is ignored if the machine was in pending state. But it does a favour by triggering all necessary state machine actions, that perform the resuming. Without that, the resulting code would be quite cumbersome.

State machine means a graph, were states (vertices) are connected via directed transitions (arcs), that prescribe the conditions and the actual direction of state changing. A machine might perform some actions when it enters a state, or when it performs transition. EAP state machines contain actions within states. In the MitM state machine, we perform actions during state changing.

On figures 4, 5 and 6, 7 are presented EAP Peer, EAP Server and MitM Peer and Server state machines.

The eap_example code triggers state machines to make a step one by one, and if there is some new message produced, the loop iteration repeats. In a real life, all of the state machines will operate in parallel, and on different devices. The physical communication is the task of EAPOL state machine, i.e. EAP over LAN. It is responsible for operation of EAP state machine and the delivery of its messages. The communication between pairs of Bob Peer and Eve Server, and Eve Peer and Alice Server state machines occurs naturally as Eve Peer likely to be wpa_supplicant

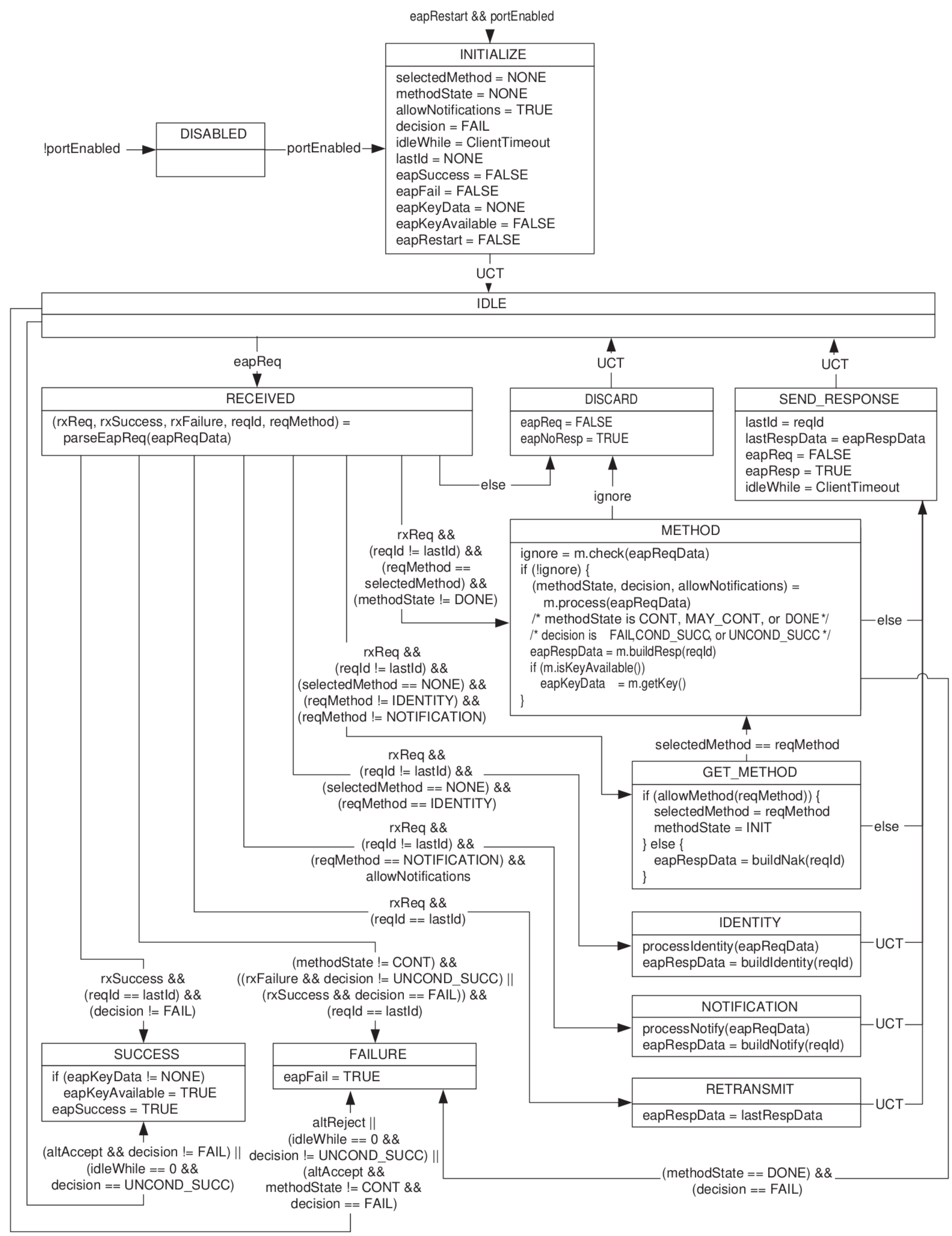


Figure 3: EAP Peer State Machine

FIGURE 4. EAP Peer state machine, from RFC4137

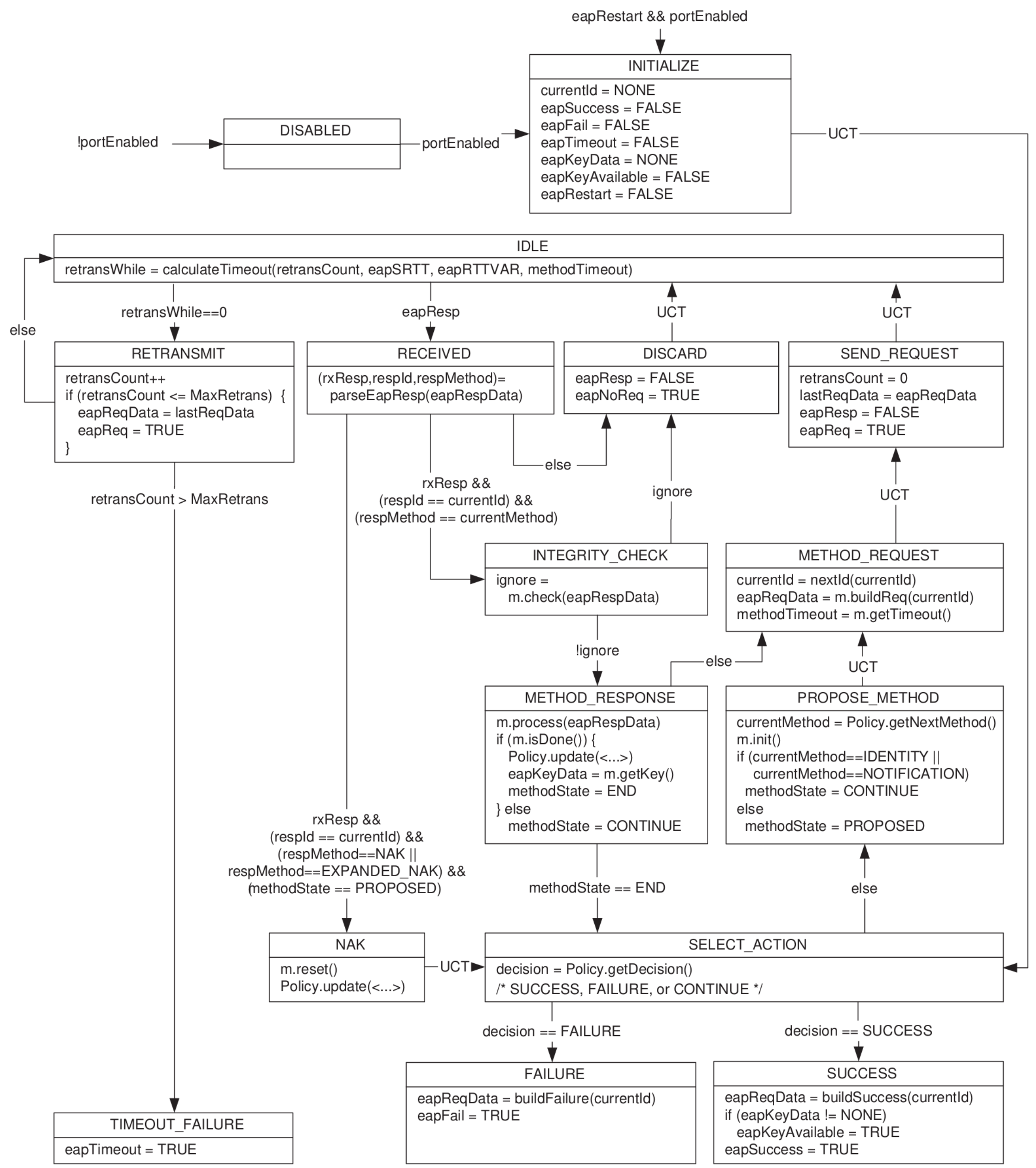
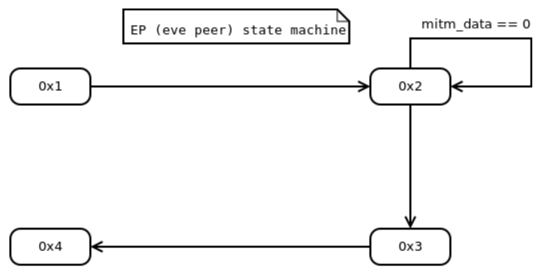


Figure 4: EAP Stand-Alone Authenticator State Machine

FIGURE 5. EAP Server state machine, from RFC4137



0x1 → 0x2

(1) Transmit MITM protocol message with MSCHAPv2 Challenge Request from AS(alice server)

0x2 → 0x3

(1) Receive MITM protocol message with MSCHAPv2 Challenge Response from BP (bob peer)
(2) Build Forged MSCHAPv2 Challenge Response using obtained challenge response

0x3 → 0x4

(1) Transmit MITM protocol message with MSCHAPv2 Challenge Response form BP(bob peer)
(2) Build MSCHAPv2 Success Response without verification of authenticator response in success request

Figure 6. MitM Peer state machine

instance, and Eve Server is to be the instance of hostapd. The rest two machines, are of no concern to the attacker. To connect MitM state machines, we are likely to use socket communication. But I repeat again, that we present only simulation. And here all packets are transmitted via simple buffer copying within one process.

6. Development summary

Here is presented a summary of how simulation behaves with every commit applied on top of the previous one.

commit 242fc738a057

Peer is authenticated by Server within a TLS 1.2 tunnel, with a help of PEAPv0 with MSCHAPv2. By the end of conversation TLV Cryptobinding protocol is performed.

commit 3d38acc54e62

EAP behaviour is the same. Commits contains not related changes.

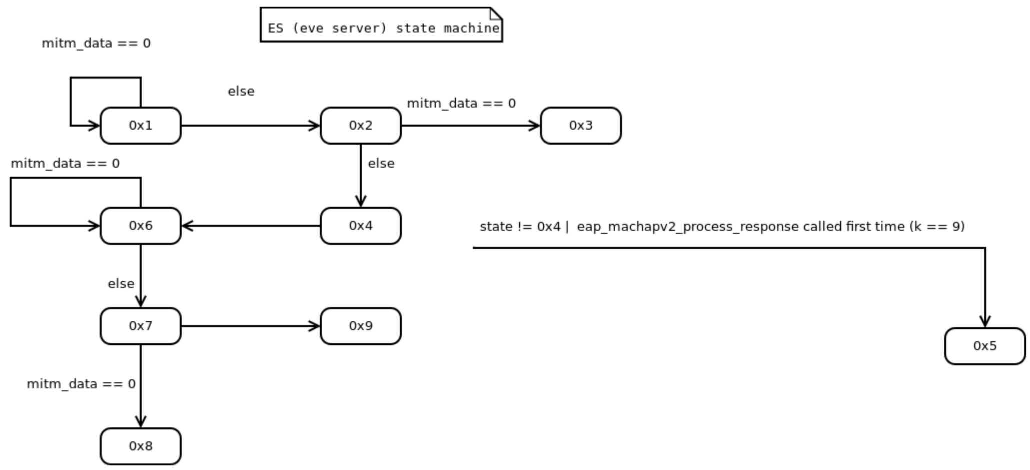
commit cf8b14eb9c93

EAP behaviour is the same. But secret material is revealed in log, i.e. TLS master secret, EAP keying material and etc.

commit 26d71ce309d3

EAP behaviour is the same. EAP state machine data was incapsulated into instance_data structure. See commit message for more details.

commit c08e56344833



0x1 → 0x2

(1) Recieve MITM protocol message: MSCHAPv2 Challenge Request from AS (alice server)

0x2 → 0x3, 0x* → 0x5, 0x7 → 0x8

(1) Failure

0x2 → 0x4

(1) Build Forged MSCHAPv2 Challenge Request using obtained auth_challenge and server_id

0x4 → 0x6

(1) Transmit MITM protocol message with MSCHAPv2 Response from BP(bob peer)

0x6 → 0x7

(1) Receive MITM protocol message MSCHAPv2 Success Request from AS (alice server)
(2) Skip Challenge Response verification, state = SUCCESS_REQ, master_key_valid=1

0x7 → 0x9

(1) Build Forged MSCHAPv2 Success Request using obtained success request

Figure 7. MitM Server state machine

The EAP behaviour was duplicated. Two pairs of state machines communicate between each other - Bob Peer and Eve Server, and Eve Peer and Alice Server.

commit 024a2b3685aa

Bob Peer was configured to skip network certificate verification obtained from Eve Server.

commit 9a2cd78574c6

Bob Peer forces PEAPv1, the same actions are taken by Alice and Eve Server. It alleviates TLV Cryptobinding as well as forces EAP keying material to be derived from TLS master secret only.

commit bc0150b8dadb

Eve Peer and Eve Server are delayed for some time. To achieve the behaviour pending request and pending response functionality of hostap implementation was utilized.

Both Eve Peer and Eve Server are waiting 10 iterations processing the very same packets – MSCHAPv2 Challenge Request for Eve Peer, and EAP-Identity Response for Eve Server. After 10 iterations they proceed with usual behaviour.

commit b88c9c348287

Eve Peer puts MSCHAPv2 Challenge and server_id into mitm_data buffer and remains in pending state as usual. Eve Server waits for a new message, after that he continues usual behaviour of EAP-Identity Phase2 method.

So despite of the message transmission, Eve's state machines end up as usual after waiting iterations.

Both pairs still communicates independently as in default eap_example.

commit 7964f349cbbe

Eve Server generate MSCHAPv2 Challenge Request with a challenge from Alice Server.

The communication ends up succefully, as Eve Server possesses user identity and password.

There is no simulation, except for generating random challenge by Alice Server only. And Eve Server takes it, by receiving a message from Eve Peer MitM state machine, to generate challenge request.

commit 764f22d88a12

EAP behaviour is the same. Commit enables CONFIG_TESTING_OPTIONS to generate asleap utility commands. Such a behaviour resembles the attack from SHMOOCON presentation.

In other words, we can start up a hostapd instance, which has the name of a target NAS. And to repeat the attack all we need is to enable asleap commands generation.

By the end of communication log will contain proper dictionary attack commands for all clients, that were trying to authenticate at our server.

commit 0ccda46808674

Eve Server MitM state machine transmits MSCHAPv2 Challenge Response obtained from Bob Peer to Eve Peer MitM state machine. Eve Server ignores all response verification routins as well as MSCHAPv2 master key derivation. It will be user later, when replying with a forged MSCHAPv2 Success Request.

The simulation is still not correct, as by the end of waiting loop, Eve Server takes user password to verify Challenge Response and generate Success Request.

Eve Peer also authenticates correctly, because the only alteration is transmission of Challenge Request from Alice Server to Eve Server. Eve Peer as well uses the same password as Bob Peer.

commit 1a1149e963aca

Eve Peer sends to Alice Server correct challenge response, botained by Eve Peer MitM state machien from Eve Server MitM state machine. Alice Server authenticates Eve Peer. But Eve Peer fails to accept success requst from Alice Server as the default eap_example behaviour should do. Because MSCHAPv2 Peer Challenge is not equal to the one produced by Bob Peer. In further commits, Eve Peer will simply ignore this verification as the protocol relies on user consciousness only. And it is right, since Eve Peer is the attacker, and the legitimate client.

For now we may say that the simulation is partially succeeded. Since the attacker was authenticated by original server, and all we need is to properly finish MSCHAPv2 Sucess Request verification and reply with a proper MSCHAPv2 Success Response, that doesn't require any special knowledge from Eve Peer.

The only reason to forge MSCHAPv2 Success Request for Bob Peer is to make him tunneling his network traffic through MitM node.

commit 4a46fa992e7ca

Eve Peer catches Success Request from Alice Server. Eve Peer MitM state machine transmits the message to Eve Server MitM state machine, which receives the message and resume Eve Server Eap state machine operation.

The next step by Eve Server is to built a forged MSCHAPv2 Success Request, with a help of the one generated by Alice Server and obtained by Eve Peer.

To the moment, both Eve Peer and Bob Peer marks MSCHAPv2 Success Request as invalid. The one doesn't posses correct secret material, whereas another one, Bob Peer, receives an incorrect request.

commit 9cebe623049fe

Eve Server applies obtained MSCHAPv2 Success Request to generate a proper Request for Bob Peer. That the one happily accepts. The conversation between Eve Server and Bob Peer finishes successfully. Both parties derive the same keying material based solely on TLS master secret. From this moment Bob Peer should begin its network activity, encrypted by keying material known to Eve Server.

To finish the attack simulation, we need to authenticate Eve Peer, because the network resource is in possession of Alice Server.

commit 45d1094c7494c

Eve Peer ignore MSCHAPv2 Success Request verification and replies with MSCHAPv2 Success Response. The conversaion between Eve Peer and Alice Server finishes successfully. Both parties derive the same keying material based solely on TLS master secret. From this moment Alice Server should accept network traffic, and the data will be crypted with keying materail known to Eve Peer.

In a real life MitM attack, Eve Server and Eve Peer should start forwarding messages between Bob Peer and Alice Server. They has all required secret keys to compromise TLS tunnels. As a benefit, Eve can send any additional network messages, as it has fully authenticated and authorized access to the network. In MitM attack the primary goal is to analyze the traffic of the victim. This goal is achieved successfully.

7. Conclusion

I'd like to say that simulation was successful and due to the good hostap codebase, have not taken a lot of time to be implemented. Though we can only dream about its application in a real life.

References

[1] Man-in-the-Middle in Tunneled Authentication Protocols http://eprint.iacr.org/2002/163/
[2] https://github.com/nartes/hostap-mitm-mschapv2-peapv1
[3] https://tools.ietf.org/html/draft-josefsson-pppext-eap-tls-eap-05
[4] https://tools.ietf.org/html/draft-josefsson-pppext-eap-tls-eap-10
[5] https://tools.ietf.org/pdf/rfc4137.pdf
[6] http://w1.fi/

[7] https://github.com/OpenSecurityResearch/hostapd-wpe
[8] http://www.willhackforsushi.com/presentations/PEAP_Shmoocon2008_Wright_Antoniewicz.pdf