Implementation of EAP authentication into IKEv2 protocol

Jelena Vučak, Leonardo Jelenković, Marin Golub
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
Address: Unska 3, 10000 Zagreb, Croatia
Tel: +385 1 6129-935, {jelena.vucak, leonardo.jelenkovic, marin.golub}@fer.hr

Abstract - IKEv2 is a protocol for exchanging keys in the IPsec architecture. In it's specification, EAP was proposed as one of the authentication mechanisms. EAP is extensible authentication protocol based on client/server architecture and allows introduction of additional EAP methods. Implementation of this protocol is complex and in our project it was decided to include one of the existing implementations of EAP into IKEv2 protocol. WPA_supplicant implementation is chosen for peer and this article mainly describes how it was included. IKEv2 responder, on the other side, will rely on RADIUS server for EAP. Therefore it should provide protocol traverse between IKEv2 and RADIUS, both encapsulating EAP packets.

I. INTRODUCTION

Development of EAP methods for IKEv2 project [6] from scratch is long process, as with any other network protocol. To simplify and speedup development we choose to reuse some existing EAP implementation. After considering all available options WPA_supplicant [8] was found as the best solution: it has the most complete EAP implementation, it is well documented and very common among various Linux distributions. WPA_supplicant is a component that is running on the clients' stations. Its main purpose is to authenticate clients to the server who will grant them access to the network. This component uses EAP as an authentication method. Whole environment in which WPA_supplicant runs is shown on the Figure 1.

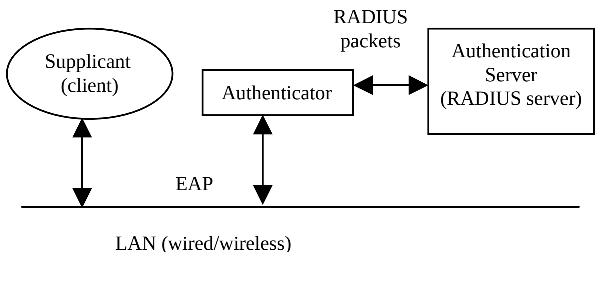

Figure 1: Environment of the WPA_supplicant

Process of the authentication consists of three parts: supplicant, authenticator and the server. Client and authenticator are connected trough the same network (wired/wireless). WPA_supplicant is running on the client. Supplicant and authenticator exchange EAP packets (sometimes encapsulated in some network layer packets). Authenticator communicates with authentication server, which in our implementation is RADIUS [1] server. Authenticator and RADIUS server exchange RADIUS packets containing EAP packets [2, 3].

Since we only need EAP methods, only EAP components from WPA_supplicant will be reused in the implementation in IKEv2 project.

II. EXSTENSIBLE AUTHENTICATION PROTOCOL

Extensible authentication protocol (EAP) [4] supports many EAP methods but all of them have similar behavior. Basically, EAP packets can be divided into two types: Requests from EAP authenticator and Responses from EAP supplicant. Generally, first message in EAP authentication is sent by authenticator and requires EAP authentication from the supplicant (*Request message)*. In the second message supplicant sends it's response in which he introduces himself by sending his ID (*Response message)*. After that authenticator may, according to method used, send additional requests on which supplicant should reply. Exchanging of EAP messages continues until supplicant is not authenticated or is rejected. In the case of successful authentication authenticator sends Success message, otherwise, in case of unsuccessful authentication, Failure message is sent. Messages flow is shown on the Figure 2.

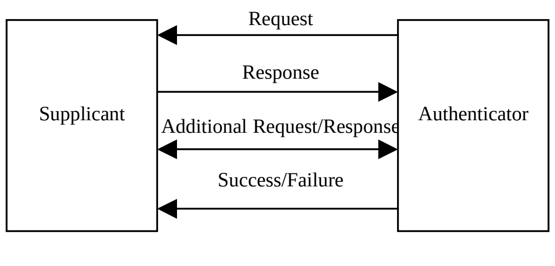

Figure 2: EAP messages exchange

EAP packets format, specified in [4], consists of four fields: Code, Identifier, Length and Data. Code contains the Type of the EAP packet and can be: Request (1), Response (2), Success (3) or Failure (4). Identifier is the number used for matching Responses with Requests. Length is the length of the whole EAP packet. Field Data is specifically defined for each packet type.

Different EAP methods in Data field transport method specific data. Some of the EAP methods often used are: EAP-MD5, EAP-OTP (One time password), EAP-TLS (Transport Level Security), LEAP (Lightweight EAP), EAP-SIM (Subscriber Identity Module), EAP-PEAP (Protected Extensible Authentication Protocol), EAP-MSCHAPv2 (Microsoft Challenge Handshake Authentication Protocol version 2), EAP-PSK (Pre Shared Key), EAP-PAX (Password Authenticated Exchange) etc.
EAP implementation in WPA_supplicant consists of several modules. The main module is EAP state machine, described in [2]. Other modules include implementations of various EAP methods and necessary crypto functions. EAP state machine changes its state depending on received messages during the process of the authentication. On the initiator's side authentication starts with initializing EAP state machine. That state machine communicates with state machines of each EAP method supported by the implementation. At the end of the authentication EAP state machine enter into state Success or Failure, depending on the last message received from the authenticator.

III. IKEv2 PROTOCOL

Internet Key Exchange protocol version 2 [6] is used for exchanging shared keys between two sides that need to communicate (initiator and responder). When two parts want to communicate they have to establish *security association* - SA, with which they will define the protection of their communications. At the beginning, there were problems with exchanging shared keys because the keys had to be exchanged upon insecure network, vulnerable to different attacks. With IKEv2 protocol keys are exchanged automatically. Daemon, which runs IKEv2 protocol, randomly generates symmetric keys using shared secret and some source of randomness. Also, daemon does rekeying after some period of time and reestablishes IKE SA between initiator and responder.
IKE messages are being exchanged in pairs and can be divided in two phases: IKE_SA_INIT and IKE_AUTH [7]. First phase consists of one pair of messages, with which initiator and responder define cryptographic suite, exchange nonce numbers and perform Diffie-Helman exchange (shared secret exchange with asymmetric algorithm). Messages in second phase are used for the authentication of the previous exchanged messages. In these crypted messages initiator and responder introduce to each other by sending IDs and certificates (if required). With these messages first IKE SA is created and traffic between initiator and responder is protected.
In some cases second phase require more than one pair of messages, as for example, when EAP is used for authentication. EAP messages exchanges are additional IKE_AUTH exchanges and must be done before IKE SA is created. By protocol, initiator (EAP peer) requires EAP authentication by leaving out AUTH payload from IKE_AUTH message it sent to responder, as 3rd message in IKEv2 message exchange. Responder (EAP server) then sends EAP payload but leaves out parameters needed to create IKE SA. Following messages contain only EAP payload. When authentication is finished and initiator is successfully authenticated IKE_AUTH messages exchange can be finished and IKE SA is created. In the case when the authentication fails, responder terminates further IKE SA and CHILD SA establishment. Some EAP methods create shared key during authentication, called Master Session Key (MSK). When using these methods for authentication, MSK must be used for creating AUTH payload in the IKE_AUTH messages. IKE_AUTH exchange with EAP as authentication method is shown on Figure 3 from [6].

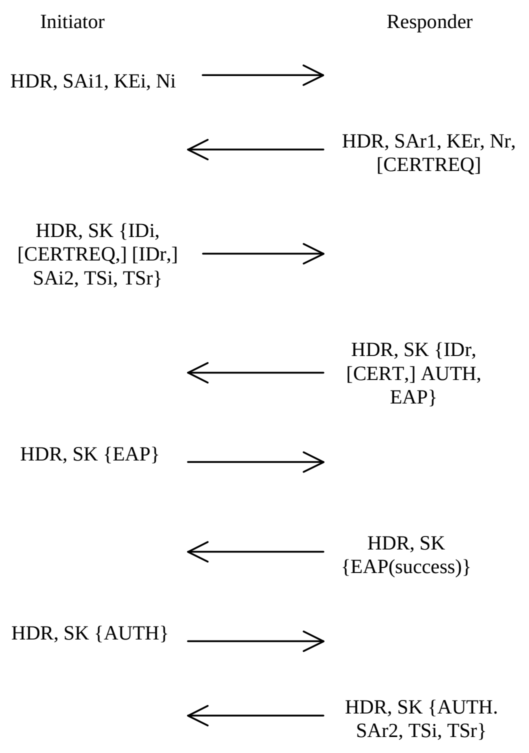
Initiator
Responder
HDR, SAi1, KEi, Ni
HDR, SAr1, KEr, Nr, [CERTREQ]
HDR, SK {IDi, [CERTREQ,] [IDr,] SAi2, TSi, TSr}
HDR, SK {IDr, [CERT,] AUTH, EAP}
HDR, SK {EAP}
HDR, SK {EAP(success)}
HDR, SK {AUTH}
HDR, SK {AUTH. SAr2, TSi, TSr}

Figure 3: IKE_AUTH exchange when EAP is included

IKEv2 protocol defines its own state machine. Actually, there are state machines for the initiator and the responder. In our implementation of EAP into IKEv2 protocol the most important state machines are the first level state machines – state machines that describe behavior of IKE SA. All states of these machines can be divided into three groups: states that are specific for the initiator, the one specific for the responder and the states common to both.
States specific for the initiator are:

- IKE_SMI_INIT
- IKE_SMI_INVALIDKE
- IKE_SMI_COOKIE
- IKE_SMI_AUTH
- IKE_SMI_INSTALLCSA

States specific for the responder are:

- IKE_SMR_INIT
- IKE_SMR_AUTH

States common for both, initiator and responder, are:

- IKE_SM_MATURE
- IKE_SM_REAUTH
- IKE_SM_DYING

Second level state machines are used for rekeying IKE SA, creating new CHILD SAs, rekeying existing CHILD SAs and won't be considered here because they are not important for the implementation of EAP into IKEv2 protocol. In the next chapter it is described what changes have been made on the IKE state machine on the initiator.

IV. IKEv2 DAEMON AND WPA_SUPPLICANT MODIFICATION

While implementing EAP into IKEv2 protocol some changes have been made into IKEv2 initiator state machine. One new state is introduced: IKE_SMI_EAP. Also, IKEv2 daemon now includes EAP state machine whose purpose is to generate EAP responses to the responder. In the following paragraphs, transitions of modified IKEv2 daemon state machine, starting from the initial state are described.

When initiator has to establish new CHILD SA (e.g. receives kernel request) it creates new state machine, initially in state IKE_SMI_INIT (as always). In that state daemon creates and sends IKE_SA_INIT request to responder and changes its state into IKE_SMI_AUTH, where it waits for IKE_SA_INIT response.

After receiving IKE_SA_INIT response, IKEv2 daemon on initiator, processes response, creates IKE_AUTH request and sends it. In sent IKE_AUTH request there is no AUTH payload, since IKEv2 daemon on initiator wants to authenticate with EAP. Immediately after sending IKE_AUTH request initiator commences into IKE_SMI_EAP. This is a new state, introduced to allow for EAP authentication in IKEv2 daemon. In this state initiator remain until response is received.

First response from responder should contain initial EAP packet - EAP identity request, if responder agrees on EAP authentication. On that response, IKEv2 daemon initializes EAP state machines which process EAP request and generate EAP response packet. Initiator forwards EAP packet to responder and waits for next request. After receiving next response from responder, it is processed and the EAP payload is passed to EAP state machine [5]. Response of EAP state machine is then sent to responder. This exchange is repeated until responder sends either EAP SUCCESS or FAILURE message.

In case SUCCESS message is received, IKEv2 initiator state machine sends final IKE_AUTH request, and transitions into IKE_SMI_INSTALLCSA state where it waits for final IKE_AUTH response from responder.

If initiator receives FAILURE message, then it transitions into IKE_SM_DEAD state where session structure for that peer is removed from memory.

Except on the IKEv2 initiator state machine, code changes were necessary also on the WPA_supplicant. From WPA_supplicant only EAP state machine, crypto functions and EAP methods were taken into IKEv2 daemon. WPA_supplicant has to be compiled as a library whose parts are then used in IKEv2. Besides original code form it, some new "glue" functions were written. Their purpose is to set and get values of control variables when EAP state machine changes it's states, i.e. it is processing requests and generating responses. These functions need to communicate with supplicant code - allow transfer of EAP messages between IKEv2 daemon and the EAP state machines. We consider EAP state machines as a black box and to control them we use callback functions. Callback functions are listed in table 1.

TABLE 1.
CALLBACK FUNCTIONS USED BY EAP STATE MACHINES

Function	Description
get_config (…)	Called by EAP state machines to retrieve configuration structure.
get_bool (…)	Get value of boolean variable.
set_bool (…)	Set value of boolean variable.
get_int (…)	Get value of integer variable.
set_int (…)	Set value of integer variable.
get_eapReqData (…)	Called by EAP state machines to retrieve received request.

Except those changes that were made on the initiators side (EAP state machine changes and additional callback functions for supplicant), changes were made on the responder's side.

V. IKEv2 DAEMON AND RADIUS RELATED MODIFICATION

Our IKEv2 responder doesn't implement EAP methods locally, instead we use RADIUS server as EAP authentication server. Responder has to extract EAP packet from IKEv2 messages, forward them to RADIUS, using RADIUS packets, get responses from RADIUS, extract EAP packet from received packet and forward them to initiator in next IKEv2 message. Responder should also be aware of EAP packet transmitted, it should detect successful and unsuccessful authentication.

IKEv2 daemon source is composed of several modules with well defined structure. As successor to IKEv1 daemon project it inherited most characteristic form it. New modules, such as EAP had to be designed to fit into rest of code. Interaction with RADIUS server, however, can not be easily achieved with old modules. New module, called RADIUS subsystem is designed, which purpose is to serve as a stub for sending and receiving RADIUS packets to and from RADIUS server. A few other functionalities are also included in module, like packet composition, parsing and information extraction. RADIUS module relies on network module for UDP connectivity.

Sockets are dynamically created and reused, depending on actual module demands. Module has its own thread waiting on packets from RADIUS server(s). Its job is to perform only minimal processing, and then push received packet as part of a new message in main message queue. Further processing is to be done in message subsystem. Message subsystem has to match received packet with existing session. Packets that can't be matched with some session are silently discarded. After matching packet with session a new message is created and pushed for further processing by thread pool. At this point, state machine module that got a new state, takes next actions. In other direction, from IKEv2 daemon to RADIUS server, packet is directly sent by using functions from RADIUS module.

VI. CONCLUSION

Most of open source projects in their development reuse some existing libraries from other open source projects aiming at shorter development time and better interoperability. The same scenario was used in this IKEv2 project, when EAP and other functionalities have been introduced into IKEv2 protocol. IKEv2 protocol is complex protocol and to achieve its functionality while preserving some degree of modularity we have to be careful when importing foreign code. When it was decided that EAP for peer functionality would be implemented by reusing existing EAP implementations, a suitable module had to be found. WPA_supplicant was chosen, as most advanced and modular at that time. Since this module will be used only on peer performance, issues are not critical. However, since after successful EAP authentication some data has to be extracted from WPA_supplicant state machines, a comprehensive knowledge of its code and data structure must be obtained. On the other side, on responder, RADIUS module's difficulties are in design not coding. Responder is to be built for heavy loads and performance must always be high priority. Hopefully, EAP implementation design presented in this paper will be successfully fully integrated into IKEv2 daemon project contributing with a sophisticated authentication method to protocol that strive to achieve better network security.

ACKNOWLEDGMENT

This work has been carried out within projects 036-0361994-1995 Universal Middleware Platform for Intelligent e-Learning Systems funded by the Ministry of Science and Technology of the Republic Croatia, and IKEv2 Step2 project funded by Siemens Networks.

LITERATURE

[1] C. Rigney, et all, Remote Authentication Dial In User Service (RADIUS), URL: http://www.ietf.org/rfc/rfc2865.txt

[2] C. Rigney, et all, RADIUS Extensions, URL: http://www.ietf.org/rfc/rfc2869.txt

[3] B. Aboba, P. Calhoun, RADIUS support for EAP, URL: http://www.ietf.org/rfc/rfc3579.txt

[4] B. Aboba, et all, Extensible Authentication Protocol (EAP), 2004., URL:www.ietf.org/rfc/rfc3748.txt

[5] J. Vollbrecht, P. Eronen, N. Petroni, Y. Ohba, State Machines for Extensible Authentication Protocol (EAP) - Peer and Authenticator, 2005., URL: www.ietf.org/rfc/rfc4137

[6] C. Kaufman, Ed., Internet Key Exchange Key (IKEv2) Protocol, 2005., URL: www.ietf.org/rfc/rfc4306.txt

[7] P. Eronen, P. Hoffman, IKEv2 Clarifications and Implementation Guidelines, 2006., URL: www.ietf.org/rfc/rfc4718.txt

[8] Jouni Malinen, Linux WPA/WPA2/IEEE 802.1X Supplicant, URL: http://hostap.epitest.fi/wpa_supplicant/

[9] S. Groš, J.Vučak, Supplicant design document, 2006., unpublished work