Virtual HSM Implementation in OpenVZ Containers

Dmitry Kartashov, Kirill Krinkin
Saint-Petersburg Academic University of Russian Academy of Sciences
Saint-Petersburg, Russia
mapseamoff@mail.ru

***Abstract*—Nowadays some host providers offer facilities to improve the security of sensitive data. It's achieved by using Hardware Security Modules (HSMs) – external pluggable devices that store data in the internal memory and performs cryptographic operations (encryption, decryption, digital signing, etc), on that data. Amazon CloudHSM [1] is an example of such a service, but maintaining these devices is expensive [2] for customers so it's desirable to have a solution which security is comparable to HSM, but utilization and maintenance costs are much lower. In this paper we discuss one of possible solutions to this problem – Virtual HSM that is based on the idea of encapsulation of sensitive data and cryptographic operations in an isolated virtual environment.**

I. Introduction

As mentioned above, the major disadvantage of the HSM is high price. Software token (e.g. OpenDNSSEC SoftHSM [3]) is an alternative to physical devices. It implements a secure storage accessible via PKCS #11 protocol [4]. Unfortunately, these solutions are less secure than HSM because cryptographic operations are performed in a client application environment.

The key idea of Virtual HSM is to store the sensitive data and operate on them in one environment and process the results of cryptographic operations in the other. In this solution runtime environments are virtual and represented by containers or virtual machines. Thereby client application cannot access the secret data directly though that data is physically located in RAM (or in files on hard drive) – this is achieved by OS mechanisms.

The architecture of Virtual HSM is represented in the Fig. 1. Key components of VHSM are:

- VHSM virtual environment (VHSM VE) is the isolated environment that contains the VHSM server and secure storage. The server performs operations on secret data and storage keeps encrypted user data;
- client API and accompanying utilities for accessing the VHSM server from a client environment;
- transport exchanges data between client and server virtual environments.

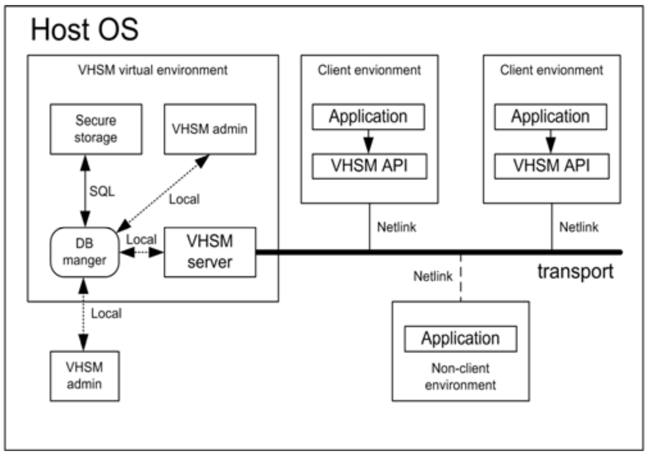

Fig. 1. Virtual HSM architecture

Virtual environments are provided by OpenVZ [5] because of this technology is widespread among host-providers.

The mechanism of the data exchange between virtual environments is based on Netlink [6]. The advantages of this way of inter-container communication are easy extensibility (without kernel recompilation) that makes it possible to implement a flexible and effective communication system. The transport layer is discussed in the section 3.

Client applications can't work in VHSM VE and interact with VHSM only through an API. The VHSM API supports bindings to different APIs, e.g. it's possible to use VHSM via OpenSSL [7] using the engine mechanism [8]. For example, this type of interaction with HSM is used in “Rutoken EDS” [9]. The OpenSSL engine for VHSM is discussed later.

The protocol of interaction between a client application and the VHSM is represented in the Fig. 2. Client applications use OpenSSL with the appropriate engine or directly call API methods that make subsequent calls to the transport layer. The transport-layer library converts cryptographic function calls into messages that are passed to the VHSM VE by the kernel transport module. The

VHSM server processes requests using the secure storage if necessary and sends the response in the same way.

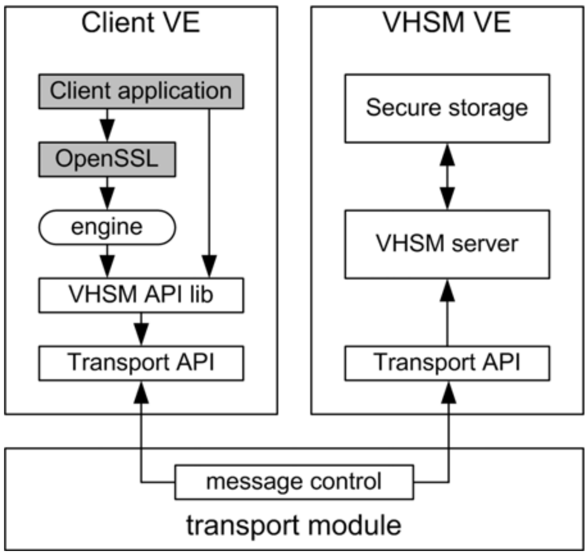

Fig. 2. Interaction between a client application and the Virtual HSM

II. VHSM VIRTUAL ENVIRONMENT

The VHSM VE contains the secure storage which is the database that stores the sensitive data in the encrypted form while other data (import date, purpose, etc) are stored unencrypted. The encryption key (master key) is generated from the user password using PBKDF2 [10]. The salt that is used in the function is stored unencrypted in database. Thereby the VHSM doesn't keep the encryption key and utilizing PBKDF2 reduces the brute-force attack rate significantly if the database is compromised. The Fig. 3 shows how sensitive data are encrypted and stored in the VHSM.

The VHSM server is responsible for user authentication, interaction with the secure storage, and performing cryptographic operations.

A client must be registered in the system to work with VHSM. Registration process is shown in the Fig. 4. First of all, a client reports its credentials to the VHSM admin. When the client is registered in the system a 256-bit authentication key is generated and encrypted in GCM mode [11] using the master key. The GCM mode guarantees integrity and confidentiality of the user data and therefore makes it possible to authenticate the encryption key derived from the user password. The VHSM uses this feature for user authentication.

A user is authenticated (see the Fig. 5) using the login/password pair and the container ID (VEID) where authentication request is received from. When a user is registered it's bound to the set of containers where one can

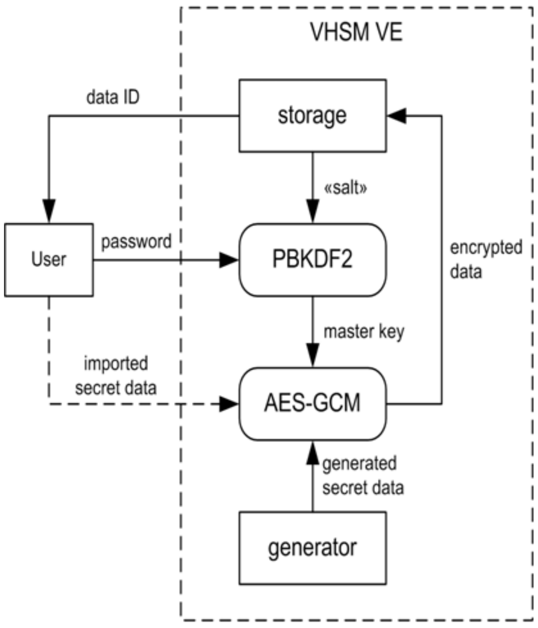

Fig. 3. Data encryption in the Virtual HSM

get access to the VHSM from. If the user attempts to access to the VHSM from a non-authorized container the request is refused.

III. TRANSPORT LAYER

The data exchanging protocol between clients and the VHSM is written in the Protocol Buffers language [12]. On the one hand, it allows to easily extend the protocol, on the other it's more space efficient compared to XML. In general, a protocol message contains parameters of the API function being called and some service information as well.

As mentioned above, inter-container communication is based on Netlink, so the OS kernel contains the module that passes messages between virtual environments while VEs contain the transport API library which interacts with this module via Netlink sockets. An example of the communication between containers is shown in the Fig. 6.

The kernel module identifies containers by their VEIDs that are provided by OpenVZ. The VHSM environment ID is passed as a parameter to the module. Messages from other containers are routed to that VEID. The passed message contains serialized protobuf data and the header with the message type, container VEID and process PID. Meaning of these parameters depends on the transfer direction. If the message comes from a client container then the module checks its VEID and forwards the message to the VHSM VE if the container is allowed to send requests to the VHSM. When the kernel module forwards the message it sets the VEID in the message header to the sender VEID and the PID to the sender process PID derived

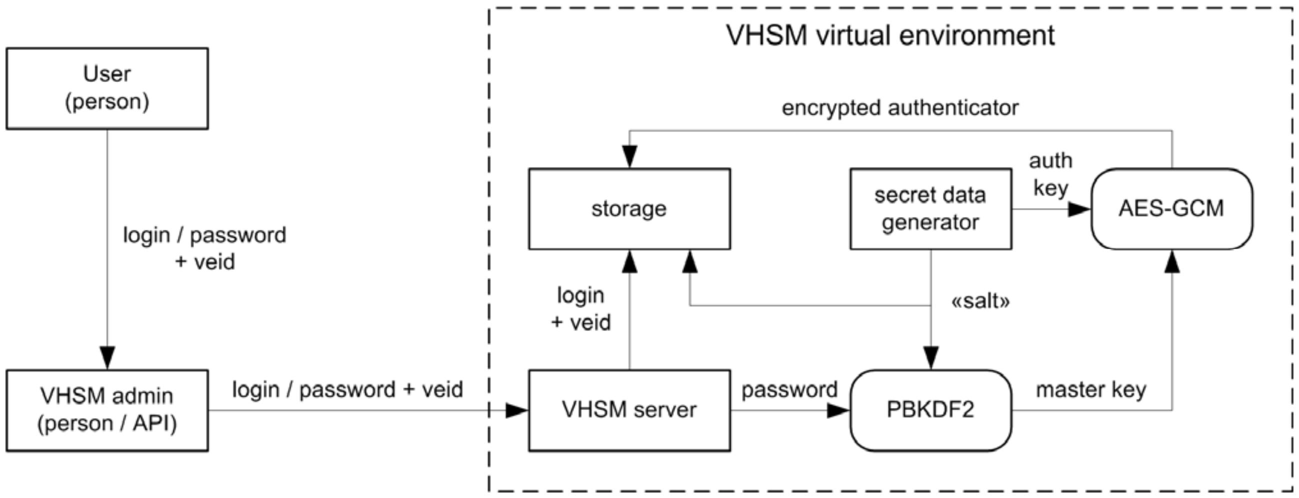

Fig. 4. Client registration in the Virtual HSM

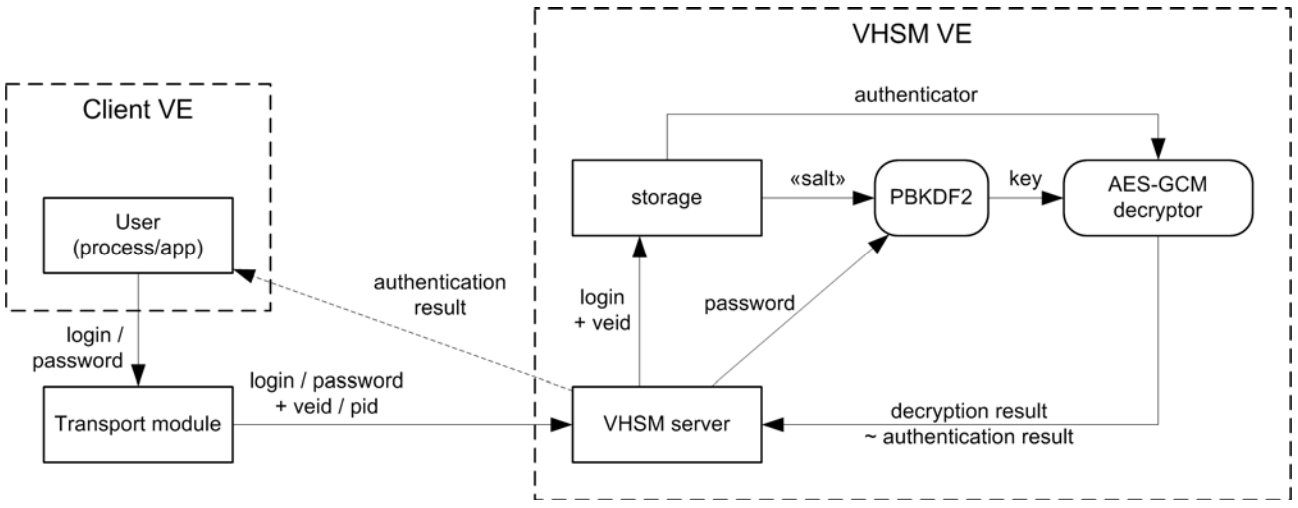

Fig. 5. User authentication in the Virtual HSM

from the Netlink message header. So the VHSM can address the response to the client. If the message is transferred from the VHSM then the VEID and PID in the message header are set by the VHSM itself. It's possible because the VHSM sends responses to the client requests only and never initiates communication with a client.

IV. Client virtual environment

The client part of the VHSM consists from the API-library and some auxiliary tools such as OpenSSL engines or PAM-modules (see section V).

The VHSM API library provides PKCS#11-like functions for managing sessions and user keys and for digital signing.

Client applications may directly call library functions or may use OpenSSL extensions. The VHSM API allows the user to import or generate some secret data (keys). These data is encrypted by AES-GCM algorithm using the user master key and then stored in the database. Then the user gets the data id which allows to use these data in the future.

If the user wants to perform a cryptographic operation then one tells key id for this operation to the VHSM. The VHSM extracts the key from the storage, decrypts it and performs computations in its isolated virtual environment while the user gets the result of the operation only.

OpenSSL can act as a layer between a client application and the VHSM API library. To achieve that OpenSSL engines can be used. Engines extend the functionality of the OpenSSL and allow to override implementation of some cryptographic algorithms. For example, OpenSSL engines make it possible to use a physical cryptographic device without modifying the code of the client program. If the engine provides implementation of the algorithm that doesn't have the specialized API in OpenSSL then the

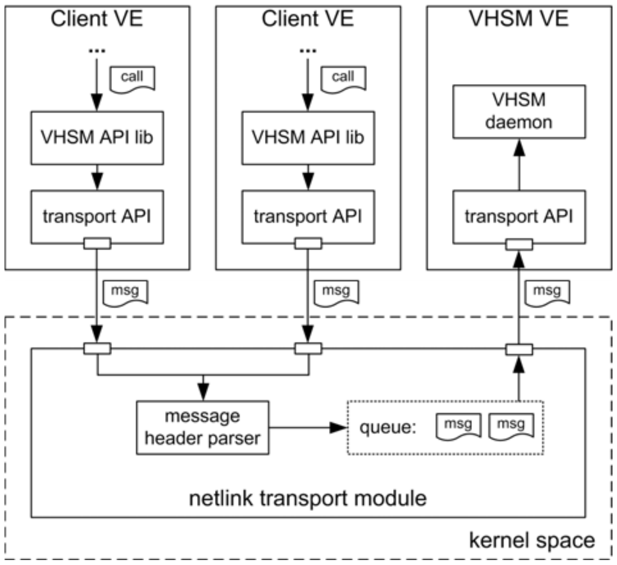

Fig. 6. Inter-container communication

standard API is used. The current implementation of the VHSM engine overrides the hashing algorithm, so the standard set of OpenSSL HMAC functions can be used. So implementation of support for the VHSM in an end user program doesn't require significant efforts.

V. Use-case example: Linux PAM module

Use-cases of the VHSM are not limited to the standard cryptographic operations. For example, the VHSM can be used for user authentication in the OS or for enhancing OS-users password security. For these purposes the PAM mechanism [13] is used in Linux. In particular, PAM allows to add some additional steps in the authentication process and/or in user creation.

The VHSM provides the PAM-module for improving password security. This module is designed to be used with the programs login and passwd. The idea is to store the hash of the signed with the special VHSM key password in /etc/shadow instead of the hash of the real user password. For this purpose the special user that signs user passwords and the corresponding special key must be created in the VHSM. The PAM-module must be configured with the credentials and the key id of that user.

While creating a new OS user adduser calls passwd which loads the VHSM PAM-module according to the Linux PAM configuration. The module works like pam_unix and only different is signing of the user password with special VHSM-user key. The signed password is hashed and placed in /etc/shadow. When the user created in this way wants to login into the OS, the login program loads the PAM-module too and the module signs the password and checks it against the password hash that stored in /etc/shadow.

VHSM PAM-module dataflow is presented in Fig. 7. It's obvious that an adversary can't crack the user password in this authentication scenario even if the /etc/shadow is compromised.

VI. Threats elimination

The main purpose of the VHSM is improving user data security, so let's consider major possible security threats and countermeasures provided by VHSM. User data privacy threats are presented in the following list:

- reading secret user data from the client application memory: secret data are processed in the isolated and trusted environment only. Memory isolation is provided by OS mechanisms;
- database leakage: secret data are stored in the encrypted form. The encryption key is not stored in the persistent storage and derived from the user password;

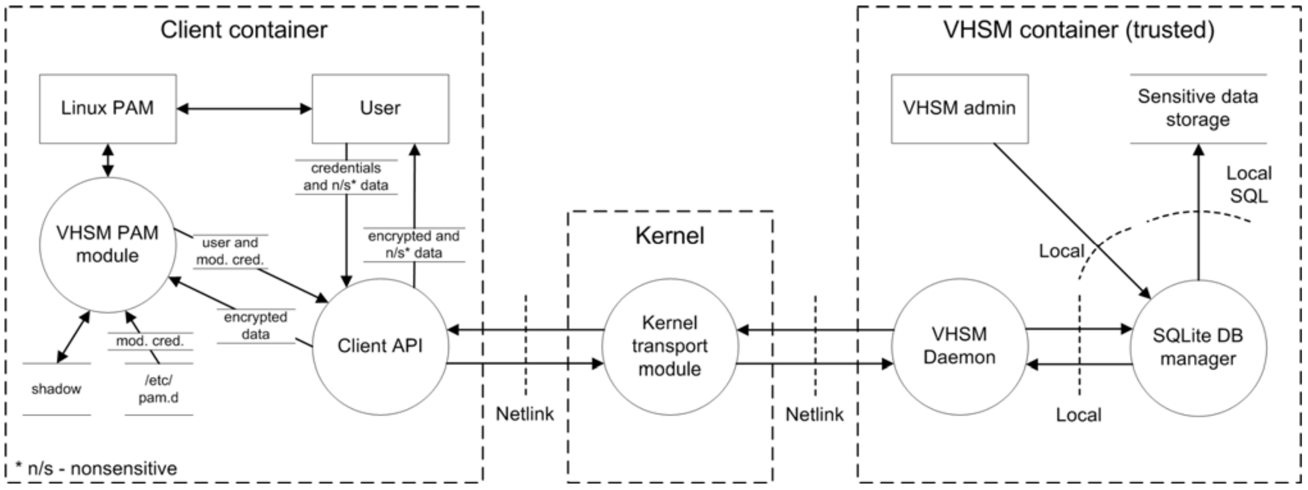

Fig. 7. Virtual HSM dataflow

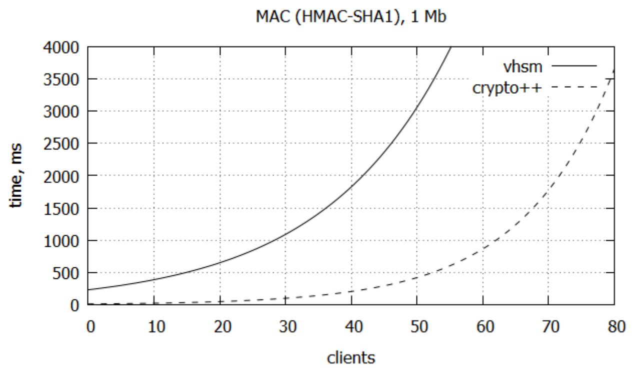

Fig. 8. Performance test: VHSM and Crpyto++

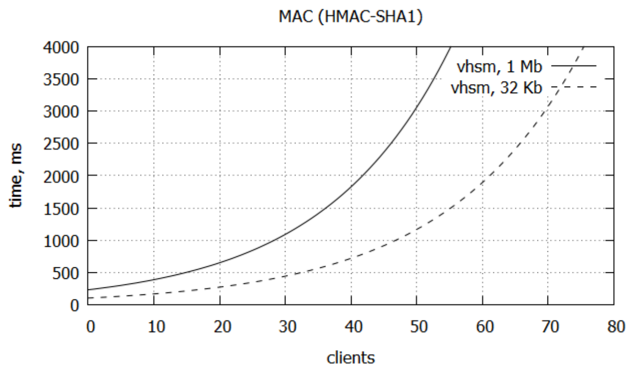

Fig. 9. Performance test: 1 Mb and 32 Kb files

- Executing an arbitrary database query (including a SQL injection): the database is stored in isolated environment so an adversary must escalate once privileges to the host OS root to read the database. The VHSM uses prepared statements to access the database which are resilient against SQL injection, because the SQL query parameter values need not be correctly escaped.

Some attacks result in an adversary privileges escalation or denial of the VHSM service. For example:

- Sending ill-formed messages (attacks on protobuf parser): transport module checks the message header. Attacks on protobuf parser are difficult because of fixed message structure so ill-formed messaged lead to deserialization error;
- DoS-attack by calling API functions or sending messages frequently: no protection is implemented at this moment.

VII. Performance

The VHSM testing was performed to evaluate overall performance of typical operations depending on the number of clients, because there are no similar solutions at this moment to compare with. A computer with the following configuration was used in tests: 2.5 GHz processor, 1 GB RAM, OS Debian 6.0.7 x64 with OpenVZ patch. Tests were also performed using only Crypto++ library [14] to estimate an overhead introduced by the VHSM. Here the list of tests:

- digital signing the 1 Mb file using HMAC-SHA1 algorithm (Crpyto++ vs. VHSM) performed by clients simultaneously. Average results are shown in the Fig. 8;
- digital signing the 1 Mb and 32 Kb files by VHSM using HMAC-SHA1 algorithm (Fig. 9);

As can be seen from graphs, the VHSM introduces a great overhead in cryptographic operations and currently can be used only by 20-30 clients simultaneously.

VIII. Conclusion

We have discussed one of possible implementations of the software HSM where logical execution environments are separated and isolated. Indeed, it's less secure than a real HSM and has many potential security threats because it has a lot of user data interaction points (sockets, parsers, buffers, etc.). On the other hand, the main problem of the software HSMs – non-isolated execution environment – has been solved. Furthermore, the host-providers don't require additional resources to maintain this solution. Currently we published preliminary solution [15, 16] which can be downloaded.

References

[1] AWS CloudHSM, Web: http://aws.amazon.com/cloudhsm.
[2] AWS CloudHSM Pricing, Web: http://aws.amazon.com/cloudhsm/pricing.
[3] OpenDNSSEC SoftHSM, Web: http://www.opendnssec.org/softhsm.
[4] RSA Laboratories – PKCS #11: Cryptographic Token Interface Standard, Web: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm.
[5] OpenVZ Linux Containers Wiki, Web: http://openvz.org.
[6] P.N. Ayuso, R.M. Gasca, L. Lefevre "Communicating between the kernel and user-space in Linux using Netlink sockets", *Software – Practice and Experience,* Aug. 2010, pp. 797-810.
[7] OpenSSL: The Open Source toolkit for SSL/TLS, Web: https://www.openssl.org.
[8] OpenSSL, Documents: engine, Web: http://www.openssl.org/docs/crypto/engine.html.
[9] Rutoken EDS support in OpenSSL, Web: http://forum.rutoken.ru/topic/1639.
[10] B. Kaliski "PKCS #5: password-based cryptography specification", *IETF, RFC 2898,* Sep. 2000, pp. 9-11.
[11] M. Dworkin "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC", Web: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.
[12] Protobuf – Protocol Buffers – Google's data interchange format, Web: https://code.google.com/p/protobuf.
[13] A Linux-PAM page, Web: http://www.linux-pam.org.
[14] Crypto++ Library 5.6.2 – a Free C++ Class Library of Cryptographic Schemes, Web: http://www.cryptopp.com. VHSM repository, Web: http://git.openvz.org/?p=vhsm.
[15] Virtual HSM – OpenVZ Linux Containers Wiki, Web: http://openvz.org/Virtual_HSM