GSoC Project Proposal: RFC- 6290 Quick Crash Recovery

About Me

I am Ujjawal Sharma, currently a student at the Johns Hopkins University, Baltimore, MD, USA. I am pursuing my masters in Information Security and would be completing my education in December 2018. Before beginning my education at JHU, I had worked for 3 years as a Security Consultant with Deloitte. Apart from this, I also have an experience of close to 1.5 years working as an e-Commerce developer with Sapient Corporation. Mostly, my Development experience of 4 years has been in Java and Python, but I have a good knowledge and understanding of C as well. (I have a professional training of C and Data Structures from Hewlett Packard and many coursework/pet projects).

LinkedIn GitHub Facebook

Overview of project: RFC- 6290 Quick Crash Recovery

Introduction:

According to the RFC 6290 document, I have understood that we need to develop a quick crash recovery method for the Internet Key Exchange Protocol (IKE). Any network communication happened between two nodes, the sender node and the receiver node.

Following are some of the important concepts to understand before diving into the details of the project:

- IPsec or the IP Security Protocol is a network protocol suite that authenticates and encrypts the packets of data sent over the network.
- IKE or the Internet Key Exchange is an IPsec standard protocol, which defines an automatic means of negotiation and authentication for IPsec Security Associations (SA).
- Security associations are security policies defined for communication between two or more entities and the relationship between the entities is represented by a key.

Our aim in the project is to implement an extension to the current IKEv2 (Internet Key Exchange version 2) that will detect fastly the de-synchronization of Security Association between 2 nodes. The situation occurs whenever the IPSec tunnel between two peers is disconnected because of restart of one of the peer. Restart takes time and so it takes time for the tunnel to recover and re-establish the connection.

Method Proposed:

1. Sending a QCD_Token in the IKE_Auth exchange that establishes the tunnel.
2. After reboot occurs, the rebooted peer can re-generate the token and send it to the peer, so as to delete the IKE SA.
3. The deletion here will cause a new establishment of IPsec tunnel.

Overview of proposed method:

1. The 'token makers' will send a notification called IKE_AUTH containing the QCD_Token.
2. The 'token takers' will receive the IKE_AUTH and will store the QCD_Token along with IKE SA.
3. If the peer is not a 'token taker', it will silently ignore the QCD_Token.
4. When a 'token maker' receives a protected IKE requests with unknown IKE SPIs, it should generate a new token that is identical to the previous token, and send it to the requesting peer in an unprotected IKE message.
5. When a 'token taker' receives the QCD token in an unprotected notification, it must verify that the TOKEN_SECRET_DATA should be matching with IKE SA.
   If the matching fails, it should be logged, if it succeeds, it must silently delete the IKE SA associated fields with the IKE_SPI fields and all dependent child SAs.
   The 'token maker' must accept such tokens from any of the IP address so as to be high on availability.
   Finally, the 'token taker' in the end, may wait for the subsequent traffic to create a new SA or it can itself create one based on initial key exchange.

Some facts about the construction:

1. The QCD_TOKEN notification is generally marked as optional as it is not necessary that every implementation will be both 'token maker' and 'token taker'.
2. If only one peer sends the QCD_TOKEN, the rebooted peer will not be detectable.
3. However, if traffic always originates from a particular peer, the requirement can be adjusted.
4. Lack of QCD_TOKEN will not guarantee that the peer does not support the standard, it may not be able to understand the notification.
5. The sending of notification, will thus be not dependent on other peer's status.
6. The QCD_TOKEN notification is related to IKE SA and it should follow the AUTH payload and precede the Configuration payload and all payloads related to the child SA.
7. After rekeying an IKE SA, the IKE SPIs are replaced, so the new SA also needs to have token.
   If the responder is the 'token maker' in the rekey exchange, it can be done in CREATE_CHILD_SA exchange in which only the responder will send the QCD_Token.
   If the initiator is a 'token maker', then we will need an extra informational exchange.
8. With some token generation method, a QCD token may get invalid, IKE SA being valid. In such a case, the 'token maker' must send the new token in a protected message under that IKE SA. The exchange can be INFORMATIONAL or MOBIKE INFORMATIONAL exchange.
9. The 'token taker' must accept such gratuitous QCD_TOKEN notifications as long as they are carried out in protected exchanges.
10. A 'token maker' should not generate them unless they can generate the old QCD_TOKEN.

Token Generation:

Token generation is also a part of this RFC which we can choose from any option available to use, but they should follow the minimum guidelines of generating a random token, making it secure and not easy for attackers to break them. The tokens should be:

1. They must be 16 to 128 octets long.
2. Distinguishable from random data.
3. Should be generated using Pseudo Random Number generator or hash functions.
4. The 'token maker' should be able to re-generate the token based on the IKE SPIs even after the reboots.
5. The collision resistance of two tokens should be high.

Apart from the above points, the RFC also defines the concepts of Backup Gateways, Interactions in case of Session Resumption and other Security Considerations to be taken care for the construction.

Why the project will be useful?

This project will help us to develop a protocol, which will be useful for a network communication to recover quickly its connection when one of the peers between whom the connection was initially established undergoes a reboot. If the connection is bi-directional for the nodes, the recover is quick. Else, suppose a case of a VPN gateway where the connection is uni-directional, in such cases, the traffic will be uni-directional and the gateway may have a dynamic IP. So, the sender will be unable to keep a track of its tunnel.

The construction defined above will be helpful in such cases as the QCD_TOKEN will keep the track of the peers between whom it is being shared and will thus decrease the recovery time of the system.

For me the project will be useful as it would be the first time I would be working on development/modifications of networking protocols. I have developed modules related to data transfers and encryptions/token generation but it would be a great experience for me to implement my earlier learning into a single project.

Goals of the project?

The goal of the project is to come up with an algorithm to implement the modification proposed in the RFC document. The algorithm should be capable enough to tackle the following:

1. Implementation of IKE_AUTH containing QCD_TOKEN.
2. Handling of tokens.
3. Generation and verification of tokens.
4. Properly passing the token in Auth exchange/rekeying.
5. Proper creation and replacement of CHILD_SAs.
6. The presentation of tokens in protected/unprotected notifications.
7. Session recovery.

8. To consider all aspects of the key exchanges as there can be situations when one peer is based on a different standards and is unable to understand the notification protocols of the other peer.
9. We should also consider the scenarios of who will behave as a 'token maker' or a 'token taker' and under what conditions. For example, for remote access clients, it is useful to have it as a 'token taker' whereas for inter-domain gateways, implementing both is required.
10. Our implementation should not allow the attackers to gain any crucial information out of the transmission of the tokens.

Knowledge Areas

For completing the project, we will need the knowledge about following :

1. C (Proficient)
2. Writing Efficient Data Structures. (Proficient)
3. Overall knowledge of the libreswan project. (Started understanding the code)
4. Knowledge of Encryption schemes and randomness. (Comfortable)
5. Understanding of standard network protocols. (Comfortable)

Time Schedule

April 23 - 14th May (Community Bonding and Introduction with the Project)

- Meeting with the team and knowing each other.
- Understanding the requirements in detail.
- Understanding the architecture of syslog-ng and coding practices.
- Setting up the environment for development.
- Analyzing the module to be changed and the possible modifications.
- Discussion on which kind of encryption will be fruitful.

15th May to 31th May (First Phase of Work)

- A basic code development for the IKE_AUTH and the Notification Payload according to the diagram provided in the RFC.
- Development of algorithm for the token exchange.
- Observing the success/failure of the algorithm based on our requirement.
- Analyzing the completed work and observing for any roadblocks.
- Unit testing and bug tracking if any.

1st June to 15th June (Second Phase of Work)

- Analyzing the first phase of work and the effect of inclusion of new code.
- Fixing the bugs of first phase and developing modules for key replacements for both scenarios – resumption and for existing SA.
- Developing the code for presenting the protected/unprotected code, token generation, token handling and verification.
- Unit testing, bug tracking and analyzing the performance of the code.
- Identifying corner cases.

15th June to 20th June (Analyzing the work till now)

- Discussing the work till now.
- Understanding any new proposals/changes.
- First phase of Documentation (Draft work).
- Observing the metrics of bugs from the unit testing and planning for next phases of development.
- Time to thoroughly check the framework and identifying the remaining cases.
- Observing the effect on performance of the overall protocol because of the modifications.

20th June to 5th July (Third Phase of work)

- Developing the fixing the final bugs, if any.
- Preparing the final documentation and preparing for the presentation related to the changes done.
- Start of the next RFC.
- Identifying the requirement changes and development.

5th July to 20th July (Final Phase of work)

- Development, Bug Fixing, Documentation.

20th July to 3rd August (Last Modifications)

- Development, Bug Fixing, Documentation.

4th August - 6th August (Wrapping Up)

- Preparation of presentation.
- Presentation.
- Last meet with the team.