Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 26, 2020

B. Carpenter
Univ. of Auckland
October 24, 2019

Quick and Dirty Security for GRASP
draft-carpenter-anima-quads-grasp-01

Abstract

A secure substrate is required by the Generic Autonomic Signaling Protocol (GRASP) used by Autonomic Service Agents. This document describes QUADS, a QUick And Dirty Security method using symmetric cryptography and preconfigured keys or passwords. It also describes a simplistic QUADS Key Infrastructure based on asymmetric cryptography used over insecure instances of GRASP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

As defined in [I-D.ietf-anima-reference-model], the Autonomic Service Agent (ASA) is the atomic entity of an autonomic function, and it is instantiated on autonomic nodes. When ASAs communicate with each other, they should use the Generic Autonomic Signaling Protocol (GRASP) [I-D.ietf-anima-grasp]. It is essential that such communication is strongly secured to avoid malicious interference with the Autonomic Network Infrastructure (ANI).

For this reason, GRASP must run over a secure substrate that is isolated from regular data plane traffic. This substrate is known as the Autonomic Control Plane (ACP). A method for constructing an ACP at the network layer is described in [I-D.ietf-anima-autonomic-control-plane]. Scenarios for link layer ACPs are discussed in [I-D.carpenter-anima-l2acp-scenarios]. The present document describes a simple method of emulating an ACP immediately above the transport layer, known as QUADS (QUick And Dirty Security) for GRASP.

It also describes a simplistic key infrastructure known as QUADSKI, using asymmetric cryptography embedded in GRASP objectives used over insecure instances of GRASP.

2. QUick And Dirty Security Method

Every GRASP message, whether unicast or multicast, is encrypted immediately before transmission, and decrypted immediately after reception, using the same symmetric encryption algorithm and domain-wide shared keys. This applies to all unicast and multicast messages

sent over either UDP or TCP. Typically encryption will take place immediately after a message is encoded as CBOR [RFC7049], and decryption will take place immediately before a message is decoded from CBOR.

There is no attempt to specify an automatic algorithm choice or key distribution mechanism. Every instance of GRASP in a given Autonomic Network (AN) must be pre-configured with the choice of encryption algorithm and any necessary parameters, and with the same key(s).

An alternative to configuring the keys is that every instance of GRASP is pre-configured with a fixed salt value and the keys are created from a locally chosen keying password, using a pre-defined hash algorithm and that salt value. Note that the salt value cannot be secret as it must be the same in all QUADS for all GRASP implementations. In this model the secrecy depends on the keying password.

The choice of algorithms should follow best current practice, e.g. [RFC8221]. At present the following choices are recommended: AES/CBC, key length 32, initialisation vector length 16, padding PKCS7(128).

3. QUick And Dirty Security Key Infrastructure

A QUADSKI key server exists in one instance in a given AN. It supports two GRASP objectives, provisonally named "411:quadskip" and "411:quadski". It runs via an instance of GRASP that is not running QUADS, i.e. its traffic is not encrypted except as defined below.

"411:quadskip" is a synchronization objective that is flooded out to all nodes in the AN. Its value is the PEM encoding of the public RSA key of the QUADSKI server. In fragmentary CDDL [RFC8610], it is defined as follows:

```
quadskip-objective = ["411:quadskip", objective-flags, loop-count, value]
objective-flags = ; as in the GRASP specification
loop-count = ; as in the GRASP specification
value = server-PEM
server-PEM = bytes
```

The recommended frequency of flooding is once per minute with a valid life time of two minutes. By this means, every autonomic node can learn the public key of the server.

"411:quadski" is a negotiation objective that is used by an autonomic node that wishes to enrol securely in the AN, known as a "pledge" to

align with BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] terminology. In fragmentary CDDL, it is defined as follows:

```
quadski-objective = ["411:quadski", objective-flags, loop-count, value]
objective-flags = ; as in the GRASP specification
loop-count = ; as in the GRASP specification
value = pledge-value / server-value
pledge-value = [encrypted-password, pledge-PEM]
server-value = encrypted-keys
encrypted-password = bytes
pledge-PEM = bytes
encrypted-keys = bytes
```

The encrypted-password is a previously agreed domain password (which should not be the same as the keying password used in Section 2), RSA-encrypted using the public key of the server.

The pledge-PEM is the PEM encoding of the public RSA key of the pledge node.

The encrypted-keys value is the result of the following process:

1. Assume the symmetric cryptography in use is AES/CBC, key length 32, initialisation vector length 16, padding PKCS7(128).

2. Let the key bytes be 'key' and the initialisation vector bytes be 'iv'.

3. Construct the array object [key, iv].

4. Encode this object in CBOR.

5. Encrypt the resulting CBOR bytes with RSA using the public key of the pledge ("pledge-PEM").

6. The result is the value of "encrypted-keys".

The QUADSKI server must have possession of the domain keys (Section 2) and the domain password when it starts up, by a method not specified here. It then proceeds as follows:

1. Create an RSA key pair, store the private key, and prepare the PEM encoding of the public key ("server-PEM").

2. Start flooding out the "411:quadskip" objective with the "server-PEM" value, using the GRASP M_FLOOD message.

3. Start listening for negotiation requests (GRASP M_NEG_REQ) for the "411:quadski" objective.

4. Whenever it receives such a request, RSA-decrypt the "encrypted-password" using its private key.

5. If the password matches, recover the pledge's public key from the "pledge-PEM".

6. Generate the "encrypted-keys" value as described above, and reply (GRASP M_NEGOTIATE) with that value.

7. Normally, the pledge will reply with GRASP M_END and an O_ACCEPT option.

Error conditions such as a password mismatch will be handled like any GRASP error condition, with GRASP M_END and an O_DECLINE option.

The pledge proceeds as follows:

1. Create an RSA key pair, store the private key, and prepare the PEM encoding of the public key ("pledge-PEM").

2. Wait until it detects the flooded "411:quadskip" option, at which point it can recover the QUADSKI server's public key from the "server-PEM" value.

3. Request the domain password from the user.

4. RSA-encrypt the password using the server's public key.

5. Use GRASP discovery (M_DISCOVER "411:quadski") to locate the QUADSKI server.

6. Construct a "411:quadski" objective whose value is [encrypted-password, pledge-PEM] as described above.

7. Start the negotiation process (M_NEG_REQ).

8. When it receives a successful reply (M_NEGOTIATE), RSA-decrypt the value using its own private key, decode the result from CBOR, and thus recover the QUADS keys [key, iv].

9. Close the GRASP session with M_END + O_ACCEPT.

As noted, this process uses unencrypted GRASP, since there are no QUADS keys available until it ends. Unlike BRSKI [I-D.ietf-anima-bootstrapping-keyinfra], it does not rely on any

limitation to link-local traffic, since it is protected by asymmetric cryptography. However, for this to work on an evolving network where nodes may enrol at any time, GRASP must run encrypted for nodes that have acquired QUADS keys and simultaneously unencrypted for the QUADSKI process. The simplest way to achieve this is to run two GRASP instances as necessary. In particular, a node that acts as a GRASP relay needs to be able to relay encrypted traffic (for enrolled nodes) and unencrypted traffic (for nodes needing to run the QUADSKI process). Note that such instances will receive GRASP broadcasts that they cannot interpret (encrypted packets reaching an unencrypted GRASP instance, and vice versa). These packets can be harmlessly discarded.

4. Implementation Status [RFC Editor: please remove]

QUADS for GRASP has been implemented as a small extension to the Python GRASP prototype, using the Python 'cryptography' module. The algorithm choices were:

- o Encryption: AES/CBC, key lengths 32/16, padding PKCS7(128).

- o Password hash: PBKDF2HMAC SHA256, length 32, 100000 iterations.

- o Salt used for keying password hash:
  0xf474526a2e74accee189f1fbc1c34ceb.

QUADSKI for GRASP has been implemented as two Python ASAs, known as 'quadski.py' for the server and 'quadspledge.py' for the pledge node. These also use the Python 'cryptography' module.

I probably need to specify some RSA parameters here...

The code will be posted to https://github.com/becarpenter/graspy when stable.

5. Security Considerations

QUADS provides effective secrecy for all GRASP messages, against any party not in possession of the relevant shared keys. However, before a GRASP message is encrypted or after it is decrypted, it is not protected within the host. Therefore, secrecy is only effective against nodes that do not contain a GRASP instance in possession of the keys. Those nodes cannot send valid GRASP messages, and they cannot interpret intercepted GRASP messages, including multicasts. However, they might attempt traffic analysis.

QUADS provides authentication of GRASP instances to the extent that they must be in possession of the relevant shared keys.

QUADS depends on pre-configuration of keys, or on password entry and a public salt value, for each autonomic node, unless QUADSKI is in use.

QUADS offers no defence against denial of service attacks.

QUADSKI securely avoids the need for pre-configuration of keys except in a central server. Nevertheless it requires each joining node to be in possession of a domain password, and there is presently no rekeying procedure without rebooting the whole autonomic network.

6. IANA Considerations

This document makes no request of the IANA.

7. Acknowledgements

Excellent suggestions were made by TBD

8. References

8.1. Normative References

[RFC8221] Wouters, P., Migault, D., Mattsson, J., Nir, Y., and T. Kivinen, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 8221, DOI 10.17487/RFC8221, October 2017, <https://www.rfc-editor.org/info/rfc8221>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

8.2. Informative References

[I-D.carpenter-anima-l2acp-scenarios]
Carpenter, B. and B. Liu, "Scenarios and Requirements for Layer 2 Autonomic Control Planes", draft-carpenter-anima-l2acp-scenarios-01 (work in progress), October 2019.

[I-D.ietf-anima-autonomic-control-plane]
Eckert, T., Behringer, M., and S. Bjarnason, "An Autonomic Control Plane (ACP)", draft-ietf-anima-autonomic-control-plane-20 (work in progress), July 2019.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-28 (work in progress), September 2019.

[I-D.ietf-anima-grasp]
Bormann, C., Carpenter, B., and B. Liu, "A Generic Autonomic Signaling Protocol (GRASP)", draft-ietf-anima-grasp-15 (work in progress), July 2017.

[I-D.ietf-anima-reference-model]
Behringer, M., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", draft-ietf-anima-reference-model-10 (work in progress), November 2018.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

Appendix A. Change log [RFC Editor: Please remove]

draft-carpenter-anima-quads-grasp-00, 2019-10-16:

Initial version

draft-carpenter-anima-quads-grasp-01, 2019-10-24:

Added QUADSKI

Author's Address

Brian Carpenter
The University of Auckland
School of Computer Science
University of Auckland
PB 92019
Auckland 1142
New Zealand

Email: brian.e.carpenter@gmail.com