DOTS
Internet Draft
Intended status: Standards Track
Expires: April 30, 2017

F. Andreasen
T. Reddy
Cisco

October 31, 2016

Distributed Denial-of-Service Open Threat Signaling (DOTS) Information and Data Model
draft-andreasen-dots-info-data-model-01.txt

Abstract

This document defines an information model and a data model for Distributed Denial-of-Service Open Threat Signaling (DOTS). The document specifies the Message and Information Elements that are transported between DOTS agents and their interconnected relationships. The primary purpose of the DOTS Information and Data Model is to address the DOTS requirements and DOTS use cases.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

A distributed denial-of-service (DDoS) attack is an attempt to make machines or network resources unavailable to their intended users. In most cases, sufficient scale can be achieved by compromising enough end-hosts and using those infected hosts to perpetrate and amplify the attack. The victim in this attack can be an application server, a client, a router, a firewall, or an entire network.

In order to mitigate a DDoS attack while still providing service to legitimate entities, it is necessary to identify and separate the majority of attack traffic from legitimate traffic and only forward the latter to the entity under attack, which is also known as "scrubbing". Depending on the type of attack, the scrubbing process may be more or less complicated, and in some cases, e.g. a bandwidth saturation, it must be done upstream of the DDoS attack target.

DDoS Open Threat Signaling (DOTS) defines an architecture to help solve these issues (see [I-D.ietf-dots-architecture]). In the DOTS architecture, a DDoS attack target is associated with a DOTS client which can signal a DOTS server for help in mitigating an attack. The DOTS client and DOTS server (collectively referred to as DOTS agents) can interact with each other over two different channels: a signal and a data channel, as illustrated in Figure 1.

```
    +---------------+                                  +---------------+
    |               | <------- Signal Channel ------> |               |
    |  DOTS Client  |                                  |  DOTS Server  |
    |               | <========  Data Channel  ======> |               |
    +---------------+                                  +---------------+
```

: DOTS signal and data channels

The DOTS signal channel is primarily used to convey information related to a possible DDoS attack so appropriate mitigation actions can be undertaken on the suspect traffic. The DOTS data channel is used for infrequent bulk data exchange between DOTS agents in the aim to significantly augment attack response coordination.

In this document, we define the overall information model and data model for the DOTS signal channel and data channel. The information and data models are designed to adhere to the overall DOTS architecture [I-D.ietf-dots-architecture] , the DOTS use case scenarios, and the DOTS requirements [I-D.ietf-dots-requirements].

2. Notational Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The reader should be familiar with the terms defined in [I-D.ietf-dots-architecture].

3. Information Model

The information model is broken into the following areas:

- General, which provides a general high-level overview of the overall information model and its structure

- Signal Channel specific, which provides an API-style interface description for the signal channel

- Data Channel specific, which provides an API-style interface description for the data channel

- Information Element specific, which describes the various information elements used by the above

3.1. General

DOTS supports request/response style interactions as well as asynchronous messages as illustrated below:

```
DOTS-Request(parameters) => DOTS-Response(parameters)

DOTS-Notification(parameters)
```

DOTS requests, responses and notifications are collectively referred to as DOTS messages, and associated with each DOTS message is one or more parameters. DOTS messages include one or more parameters, each of which belong to one of the following categories

- o Mandatory, which are base DOTS protocol parameters that MUST always be included with the message in question.

- o Optional, which are base DOTS protocol parameters that MAY be included with the message in question.

- o Extensions, which are extended DOTS protocol parameters that MAY be included with the message in question.

Mandatory and optional DOTS protocol parameters MUST be supported by all DOTS agents. Extension DOTS protocol parameters define extended functionality above and beyond the base DOTS protocol that MAY be supported by a particular DOTS agent. The DOTS protocol includes an extension framework that enables each DOTS agent to determine which extension a peer DOTS agent supports. The extension framework also enables a DOTS agent to specify how to handle any unsupported extensions.

DOTS supports versioning in the form of a DOTS protocol version, which is included in every DOTS message. If a DOTS agent receives a DOTS request with an unsupported protocol version, it will reply with a failure and an indication of the protocol version(s) supported. This enables a future version of the DOTS protocol to be defined in a backwards compatible manner. If a DOTS Notification with an unsupported protocol version is received, it will simply be discarded.

All DOTS messages include a message-id and an agent-id. The agent-id identifies the originator of the message, whereas the message-id provides an identifier for DOTS Request/Response correlation and DOTS Notification identification.

- o [Editor's note: There are different views on whether an agent-id is needed at the DOTS level or whether the underlying security mechanisms (incl. authenticated identity via (D)TLS) suffices. For now, the draft includes an explicit agent-id at the DOTS level].

- o [Editor's note: Need to look more closely at how we ensure message-id uniqueness. One option is to include the agent-id from the request or notification as part of it. Also, not clear (yet) if this is really information/data model or protocol]

DOTS messages are exchanged over the DOTS Signal channel and the DOTS Data Channel. In either case, there is a channel establishment procedure taking place initially whereby:

1. The DOTS client determines the address of a DOTS server to establish the DOTS channel with. The base DOTS protocol assumes the DOTS client has already obtained either this address, or a domain name [RFC1034] or DNS SRV name [RFC2782] that can lead to it. One way of achieving that is by provisioning.

2. The DOTS agents mutually authenticate each other. The Information and Data model assumes that the DOTS protocol will run on top of a secure transport (e.g. TLS [RFC5246] or DTLS [RFC6347]) that also provides mutual authentication. The DOTS agent-id will need to be tied-to or covered by this transport-level mutual authentication. If not, explicit mutual authentication of the DOTS agent-id at the DOTS protocol level will be needed

   o [Editor's note: The mutual authentication aspect of the DOTS agent-id is a bit sketchy; needs more consideration]

Note that the mutual authentication of the DOTS agents are needed to verify the service relationship between the DOTS client and server as well as govern relevant authorization policies with respect further DOTS operation (e.g. scrubbing of traffic for the client in question).

o [Editor's note: The below is work in progress - main focus is on overall approach and messages right now. The actual parameters are expected to evolve in future versions]

3.2. Signal Channel Messages

In this section, we describe the signal channel messages.

o [Editor's note: Most (all ?) messages can be extended - currently not clearly shown in the following]

3.2.1. Open Signal Channel

The OpenSignalChannel request establishes a signal channel from the LocalAgent to the RemoteAgent:

```
   OSCreply OpenSignalChannel(LocalAgentId, RemoteAgent
                              [, ExtensionsSupported])
```

LocalAgentId is the local DOTS client id.

RemoteAgent is one or more of IP-address(es), domain-name and DNS SRV name for the remote DOTS agent with which the signal channel is to be established. When more than one of these is present, the priority order for client use is DNS SRV, domain-name and IP-address(es). A lower priority MUST NOT be used unless a higher priority fails to produce a response.

o Note that the intent of this priority order is not to replace any DNS caching but rather to provide a "last resort" in case of DNS failure. If domain name use is not desired, then RemoteAgent MUST only include IP-address(es).

ExtensionsSupported may optionally be included to indicate which extensions are supported by the DOTS client.

A successful OSCreply contains the following information:

o Response code, which indicates the outcome of the request

o RemoteAgentId, which is the peer DOTS agent Id.

o SignalChannel, which is a handle for the new signal channel assuming the request succeeded

o [optional] ExtensionsSupported, which lists the extensions supported by the remote DOTS agent.

A failure OSCreply contains the following information:

o Response code, which indicates the outcome of the request

o RemoteAgent, which indicates a different agent to establish the signal channel with (redirect)

o [optional] ExtensionsSupported, which lists the extensions supported by the remote DOTS agent.

3.2.2. Close Signal Channel

The CloseSignalChannel request closes a previously established signal channel:

```
CSCreply CloseSignalChannel(SignalChannel)
```

SignalChannel is the handle for the signal channel to be closed.

The CSCreply contains the following information

o Response code, which indicates the outcome of the request

3.2.3. Redirect Signal Channel

The RedirectSignalChannel request instructs the peer DOTS agent to change the signal channel to the alternative DOTS agent indicated.

o [Editor's note: At the IETF Berlin meeting, there was discussion around using anycast to possibly avoid redirection. Anycast however would not be mandatory, and even when supported, it may be desirable to "redirect" to a non-anycast address after initial discovery for stability]

o [Editor's note: Data channel is currently associated with the data channel. Either they both have to get redirected or we need a way of (re)associating the data channel with the new signal channel; maybe a MoveSignalChannel() ?]

o [Editor's note: When redirecting the channel, do we redirect the existing one or create a new one and close the old one ? The latter seems cleaner, albeit more explicit.]

```
RSCreply RedirectSignalChannel(SignalChannel, NewAgent)
```

SignalChannel is the handle for the signal channel to be redirected.

NewAgent is an IP-address, domain-name or DNS SRV name for the new remote DOTS agent with which the signal channel is to be redirected to.

The CSCreply contains the following information

- o Response code, which indicates the outcome of the request. Note that a success response merely indicates successful receipt and reply to the request; successful redirection of signal channel is not implied.

3.2.4. Request Status Update

The RequestStatusUpdate message is a notification to the peer agent asking it to provide a status update as indicated

```
RequestStatusUpdate(Target [, Heartbeat] [, Health] [, Attack]
                              [, Mitigation] [, Efficacy])
```

Target specifies the attack target for which status updates are requested

- o [Editor's note: Need to come up with a more detailed model for how we identify and describe targets]

Heartbeat requests a status update at the heartbeat interval specified.

Health requests health information for the target.

Attack requests attack information for the target.

Mitigation requests mitigation status information for the target.

Efficacy requests mitigation efficacy information for the target.

3.2.5. Status Update

The StatusUpdate message is a notification to the peer agent. StatusUpdate provides heartbeat functionality and updates around health, attack status, mitigation status and mitigation efficacy

```
StatusUpdate(Target, [, HealthStatus] [, AttackStatus]
               [, MitigationStatus] [, MitigationEfficacy])
```

Target specifies the attack target for which status updates are provided.

HealthStatus provides overall health for the target

AttackStatus provides information about an ongoing attack for the target

- [Editor's note: Do we need to identify and refer to specific attacks for a given target or just the target itself. For now, assume just the target itself. Question applies to other parameters here as well.]

MitigationStatus provides information about current mitigation(s) in place for the target. The status reflects information from the mitigator's point of view.

MitigationEfficacy provides information about the efficacy of the mitigation. The efficacy reflects information from the attack target's point of view.

3.2.6. Request Mitigation

The RequestMitigation message is a notification to the peer agent to invoke mitigation for the attack target specified. If the request is received and honored, mitigation will commence and a StatusUpdate message will be sent to reflect that. Note that either message is especially susceptible to loss during an active DDoS attack

```
RequestMitigation(Target)
```

Target specifies the attack target for mitigation is requested.

3.2.7. Stop Mitigation

The StopMitigation message is a notification to the peer agent to stop further attack mitigation for the target indicated. The message is sent on behalf of the attack target towards the mitigator.

```
StopMitigation(Target)
```

Target specifies the attack target for which mitigation is to be stopped.

3.3. Data Channel Messages

In this section, we describe the data channel messages.

- o [Editor's note: Most (all ?) messages can be extended - currently not clearly shown in the following]

3.3.1. Open Data Channel

The OpenDataChannel request opens a Data Channel to be used in conjunction with a previously established Signal Channel

```
ODCreply OpenDataChannel(SignalChannel [, ExtensionsSupported])
```

SignalChannel is the handle for the existing signal channel.

ExtensionsSupported may optionally be included to indicate which extensions are supported by the DOTS client.

A successful ODCreply contains the following information

- o DataChannel, which is a handle for the new data channel

- o [optional] ExtensionsSupported, which lists the extensions supported by the remote DOTS agent.

A failure ODCreply contains the following information

- o Response code, which indicates the outcome of the request

- o RemoteAgent, which indicates a different agent to establish the data channel with (redirect)

3.3.2. Close Data Channel

The CloseDataChannel request closes a previously established data channel:

```
CDCreply CloseDataChannel(DataChannel)
```

DataChannel is the handle for the data channel to be closed.

The CDCreply contains the following information

- o Response code, which indicates the outcome of the request

3.3.3. Redirect Data Channel

- o [Editor's note: Resolve open questions in Redirect Signal Channel first]

3.3.4. SendData

The SendData request sends data to the peer DOTS agent over the data channel.

```
SDreply SendData(DataChannelData)
```

DataChannelData may contain one or more of the following:

- o Blacklists, whitelists, filters, aliases\names)

[Editor's note: The above needs much more work and overall structure. Break up into individual pieces and make sure it's modular, extensible and we have clarity on which data is mandatory versus optional to support]

3.4. Information Elements

o [Editor's note: The following should merely be considered as a bucket of possible IEs at this point; further work needed, including reconciling with message definitions in sections above]

3.4.1. Protocol version

3.4.2. Attack Target

o [Editor's note: may be superfluous given Mitigation Scope below"]

3.4.3. Agent Id

(identity for each DOTS client and server, contains a least a domain portion that can be authenticated)

3.4.4. Blacklist

(define, retrieve, manage and refer to)

3.4.5. Whitelist

(define, retrieve, manage and refer to)

3.4.6. Attack telemetry

(collected behavioral characteristics defining the nature of a DDoS attack) o [9/27: Probably extended functionality.]

3.4.7. Mitigator feedback

(attack mitigation feedback from server to client, incl. mitigation status [start, stop, metrics, etc.], attack ended and information about mitigation failure)

3.4.8. Mitigation efficacy

(attack mitigation efficacy feedback from client to server)

3.4.9. Mitigation failure

(unsupported target type, mitigation capacity exceeded, excessive "clean traffic", out-of-service, etc.)

3.4.10. Mitigation Scope

Classless Internet Domain Routing (CIDR) prefixes, BGP prefix, URI, DNS names, E.164, "resource group alias", port range, protocol, or service

o [Editor's note: comes from requirements - not clear how "protocol" and "service" are defined. Also, consider which URI schemes]

o [Editor's note: It would probably be useful to structure mitigation scope and related information (like telemetry, blacklist, etc.) into different "types", since different types of targets will have different parameters and different DOTS servers may support different types of attack targets]

Information about the attack (e.g. targeted port range, protocol or scope)

o [Editor's note: Not clear this is really different from "Mitigation Scope" below - taken from requirement OP-006]

- [9/27: Roland doesn't think we need this as baseline information.]

3.4.11. Mitigation Scope Conflict

Nature and scope of conflicting mitigation requests received from two or more clients

3.4.12. Resource Group Alias

(define in data channel, refer to in signal/data channel; aliases for mitigation scope)

3.4.13. Mitigation duration

(desired lifetime - renewable)

3.4.14. Peer health

(? - measure of peer health)

3.4.15. Filters

3.4.16. Filter-actions

(rate-limit, discard)

3.4.17. Acceptable signal lossiness

(for unreliable delivery)

3.4.18. Heartbeat interval

3.4.19. Data Channel Address

- Editor's note: For discussion (not entirely aligned with current architecture draft text); assumes establish signal channel first and learn data channel address through it (would be useful for redirection as well and makes it easier for signal and data channel to terminate on different entities)]

3.4.20. Extensions

(standard, vendor-specific, supported)

4. Data Model

TODO

5. IANA Considerations

TODO

6. Security Considerations

TODO

7. Acknowledgements

TODO

This document was prepared using 2-Word-v2.0.template.dot.

8. References

8.1. Normative References

[RFC1034] Mockapetris, P.V., "Domain Names - concept and facilities", RFC 1034, November 1987.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[I-D.ietf-dots-architecture] Mortensen, A., Andreasen, F., Reddy, T., christopher_gray3@cable.comcast.com, c., Compton, R., and N. Teague, "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture", draft-ietf-dots-architecture-00 (work in progress), July 2016.

[I-D.ietf-dots-requirements] Mortensen, A., Moskowitz, R., and T. Reddy, "Distributed Denial of Service (DDoS) Open Threat Signaling Requirements", draft-ietf-dots-requirements-02 (work in progress), July 2016.

8.2. Informative References

[RFC5246] Dierks, T., and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008

[RFC6347] Rescorla, E., and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012

Authors' Addresses

Flemming Andreasen
Cisco Systems, Inc.
USA

Email: fandreas@cisco.com

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com