DOTS T. Reddy
Internet-Draft D. Wing
Intended status: Standards Track P. Patil
Expires: September 17, 2016 M. Geller
Cisco
M. Boucadair
Orange
R. Moskowitz
HTT Consulting
March 16, 2016

# Co-operative DDoS Mitigation
draft-reddy-dots-transport-03

## Abstract

This document discusses mechanisms that a DOTS client can use, when it detects a potential Distributed Denial-of-Service (DDoS) attack, to signal that the DOTS client is under an attack or request an upstream DOTS server to perform inbound filtering in its ingress routers for traffic that the DOTS client wishes to drop. The DOTS server can then undertake appropriate actions (including, blackhole, drop, rate-limit, or add to watch list) on the suspect traffic to the DOTS client, thus reducing the effectiveness of the attack.

## 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, etc.

In a lot of cases, it may not be possible for an enterprise to determine the cause for an attack, but instead just realize that





certain resources seem to be under attack. The document proposes that, in such cases, the DOTS client just inform the DOTS server that the enterprise is under a potential attack and that the DOTS server monitor traffic to the enterprise to mitigate any possible attack. This document also describes a means for an enterprise, which act as DOTS clients, to dynamically inform its DOTS server of the IP addresses or prefixes that are causing DDoS. A DOTS server can use this information to discard flows from such IP addresses reaching the customer network.

The proposed mechanism can also be used between applications from various vendors that are deployed within the same network, some of them are responsible for monitoring and detecting attacks while others are responsible for enforcing policies on appropriate network elements. This cooperations contributes to a ensure a highly automated network that is also robust, reliable and secure. The advantage of the proposed mechanism is that the DOTS server can provide protection to the DOTS client from bandwidth-saturating DDoS traffic.

How a DOTS server determines which network elements should be modified to install appropriate filtering rules is out of scope. A variety of mechanisms and protocols (including NETCONF) may be considered to exchange information through a communication interface between the server and these underlying elements; the selection of appropriate mechanisms and protocols to be invoked for that interfaces is deployment-specific.

Terminology and protocol requirements for co-operative DDoS mitigation are obtained from [I-D.ietf-dots-requirements].

## 2. Notational Conventions

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].

## 3. Solution Overview

Network applications have finite resources like CPU cycles, number of processes or threads they can create and use, maximum number of simultaneous connections it can handle, limited resources of the control plane, etc. When processing network traffic, such an application uses these resources to offer its intended task in the most efficient fashion. However, an attacker may be able to prevent the application from performing its intended task by causing the application to exhaust the finite supply of a specific resource.





TCP DDoS SYN-flood is a memory-exhaustion attack on the victim and ACK-flood is a CPU exhaustion attack on the victim ([RFC4987]). Attacks on the link are carried out by sending enough traffic such that the link becomes excessively congested, and legitimate traffic suffers high packet loss. Stateful firewalls can also be attacked by sending traffic that causes the firewall to hold excessive state and the firewall runs out of memory, and can no longer instantiate the state required to pass legitimate flows. Other possible DDoS attacks are discussed in [RFC4732].

In each of the cases described above, some of the possible arrangements to mitigate the attack are:

- If a DOTS client determines it is under an attack, the DOTS client can notify the DOTS server using the DOTS signal that it is under a potential attack and request that the DOTS server take precautionary measures to mitigate the attack. The DOTS server can enable mitigation on behalf of the DOTS client by communicating the DOTS client's request to the mitigator and relaying any mitigator feedback to the requesting DOTS client.

- If a DOTS client determines it is under an attack, the DOTS client can notify its servicing router (DOTS relay) using the DOTS signal that it is under a potential attack and request that the DOTS relay take precautionary measures to mitigate the attack. The DOTS relay propagates the DOTS signal to a DOTS server.

  The DOTS server can enable mitigation on behalf of the DOTS relay by communicating the DOTS relay's request to the mitigator and

relaying any mitigator feedback to the DOTS relay which in turn propagates the feedback to the requesting DOTS client.

The DOTS client must authenticates itself to the DOTS relay, which in turn authenticates itself to a DOTS server, creating a two-link chain of transitive authentication between the DOTS client and the DOTS server.

- o If a network resource detects a potential DDoS attack from a set of IP addresses, the network resource (DOTS client) informs its servicing router (DOTS relay) of all suspect IP addresses that need to be blocked or black-listed for further investigation.

  The DOTS client could also specify a list of protocols and ports in the black-list rule.  That DOTS relay in-turn propagates the black-listed IP addresses to the DOTS server and the DOTS server blocks traffic from these IP addresses to the DOTS client thus reducing the effectiveness of the attack.





  The DOTS client periodically queries the DOTS server to check the counters mitigating the attack.  If the DOTS client receives a response that the counters have not incremented then it can instruct the black-list rules to be removed.  If a blacklisted IPv4 address is shared by multiple subscribers, then the side effect of applying the black-list rule will be that traffic from non-attackers will also be blocked by the access network [RFC6269].

A network diagram showing a deployment of these elements is shown below.  This shows the DOTS server operating on the access network.

```
   Network
   Resource        CPE router        Access network      __________
 +------------+    +-----------+       +--------------+     /          \
 |            |____|           |_______|              |___ | Internet |
 |DOTS client|     |DOTS relay|        | DOTS server  |    |          |
 |            |    |           |       |              |    |          |
 +------------+    +-----------+       +--------------+     \__________/
```

Figure 1

The DOTS server can also be running on the Internet, as depicted below:

```
    Network
    Resource        CPE router         __________  DDoS mitigation svc
  +-----------+    +----------+       /          \    +-------------+
  |           |____|          |_______|          |___ |             |
  |DOTS client|    |DOTS relay|       | Internet |    | DOTS server |
  |           |    |          |       |          |    |             |
  +-----------+    +----------+       \__________/    +-------------+
```

Figure 2

In typical deployments, the DOTS client belongs to a different administrative domain than the DOTS server. For example, the DOTS client is a web server serving content owned and operated by a company, while the DOTS server is owned and operated by a different company providing DDoS mitigation services. That company providing DDoS mitigation service might, or might not, also provide Internet access service to the website operator.

The DOTS server may (not) be co-located with the DOTS mitigator. In typical deployments, the DOTS server belongs to the same administrative domain as the mitigator.





The DOTS client can communicate directly with the DOTS server or indirectly with the DOTS server via the DOTS relay.

## 4. Happy Eyeballs for DOTS Signal Channel

DOTS signaling can happen with DTLS over UDP and TLS over TCP. A DOTS client can use DNS to determine the IP address(es) of a DOTS server. The DOTS client must know a DOTS server's domain name; hard-coding the domain name of the DOTS server into software is NOT RECOMMENDED in case the domain name is not valid or needs to change for legal or other reasons. The DOTS client performs A and/or AAAA record lookup of the domain name and the result will be a list of IP addresses, each of which can be used to contact the DOTS server using UDP and TCP.

If an IPv4 path to reach a DOTS server is found, but the DOTS server's IPv6 path is not working, a dual-stack DOTS client can experience a significant connection delay compared to an IPv4-only DOTS client. The other problem is that if a middlebox between the DOTS client and DOTS server is configured to block UDP, the DOTS client will fail to establish a DTLS session [RFC6347] with the DOTS server and will, then, have to fall back to TLS over TCP [RFC5246] incurring significant connection delays. [I-D.ietf-dots-requirements] discusses that DOTS client and server will have to support both connectionless and connection-oriented protocols.

To overcome these connection setup problems, the DOTS client can try connecting to the DOTS server using both IPv6 and IPv4, and try both DTLS over UDP and TLS over TCP in a fashion similar to the Happy Eyeballs mechanism [RFC6555]. These connection attempts are performed by the DOTS client when its initializes, and the client uses that information for its subsequent alert to the DOTS server. In order of preference (most preferred first), it is UDP over IPv6, UDP over IPv4, TCP over IPv6, and finally TCP over IPv4, which adheres to address preference order [RFC6724] and the DOTS preference that UDP be used over TCP (to avoid TCP's head of line blocking).





```
DOTS client                                               DOTS server
   |                                                          |
   |--DTLS ClientHello, IPv6 ---->X                           |
   |--TCP SYN, IPv6-------------->X                           |
   |--DTLS ClientHello, IPv4 ---->X                           |
   |--TCP SYN, IPv4------------------------------------------>|
   |--DTLS ClientHello, IPv6 ---->X                           |
   |--TCP SYN, IPv6-------------->X                           |
```

```
|<-TCP SYNACK---------------------------------------------|
|--DTLS ClientHello, IPv4 ---->X                          |
|--TCP ACK----------------------------------------------->|
|<------------Establish TLS Session---------------------->|
|----------------DOTS signal----------------------------->|
|                                                         |
```

Figure 3: Happy Eyeballs

In reference to Figure 3, the DOTS client sends two TCP SYNs and two DTLS ClientHello messages at the same time over IPv6 and IPv4. In this example, it is assumed that the IPv6 path is broken and UDP is dropped by a middle box but has little impact to the DOTS client because there is no long delay before using IPv4 and TCP. The IPv6 path and UDP over IPv6 and IPv4 is retried until the DOTS client gives up.

## 5. Performance Considerations

DOTS client and server can also use the following techniques to reduce the delay required to deliver a DOTS signal:

- o DOTS client can use (D)TLS session resumption without server-side state [RFC5077] to resume session and convey the DOTS signal.

- o TLS False Start [I-D.ietf-tls-falsestart] which reduces round-trips by allowing the TLS second flight of messages (ChangeCipherSpec) to also contain the DOTS signal.

- o Cached Information Extension [I-D.ietf-tls-cached-info] which avoids transmitting the server's certificate and certificate chain if the client has cached that information from a previous TLS handshake.

- o TCP Fast Open [RFC7413] can reduce the number of round-trips to convey DOTS signal.

- o While the communication to the DOTS server is quiescent, the DOTS client may want to probe the server to ensure it has maintained cryptographic state. Such probes can also keep alive firewall or





NAT bindings. This probing reduces the frequency of needing a new

handshake when a DOTS signal needs to be conveyed to the DOTS server.

* A DTLS heartbeat [RFC6520] verifies the DOTS server still has DTLS state by returning a DTLS message. If the server has lost state, it returns a DTLS Alert. Upon receipt of an unauthenticated DTLS Alert, the DTLS client validates the Alert is within the replay window (Section 4.1.2.6 of [RFC6347]). It is difficult for the DTLS client to validate the DTLS Alert was generated by the DTLS server in response to a request or was generated by an on- or off-path attacker. Thus, upon receipt of an in-window DTLS Alert, the client SHOULD continue re-transmitting the DTLS packet (in the event the Alert was spoofed), and at the same time it SHOULD initiate DTLS session resumption.

* TLS runs over TCP, so a simple probe is a 0-length TCP packet (a "window probe"). This verifies the TCP connection is still working, which is also sufficient to prove the server has retained TLS state, because if the server loses TLS state it abandons the TCP connection. If the server has lost state, a TCP RST is returned immediately.

## 6. DOTS Signal Channel

A DOTS client can use RESTful APIs discussed in this section to signal/inform a DOTS server of an attack.

TBD: Constrained Application Protocol (CoAP) [RFC7252] is used for DOTS signal channel. COAP was designed according to the REST architecture, and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. CoAP has been defined to make use of both DTLS over UDP and TLS over TCP. The advantages of COAP are: (1) Like HTTP, CoAP is based on the successful REST model, (2) CoAP is designed to use minimal resources, (3) CoAP integrates with JSON, CBOR or any other data format, (4) asynchronous message exchanges, etc.

JSON [RFC7159] payloads is be used to convey signal channel specific payload messages that convey request parameters and response information such as errors.

## 6.1. Mitigation Service Request

The following APIs define the means to convey a DOTS signal from a DOTS client to a DOTS server. POST request is used to convey the DOTS signal from a DOTS client to a DOTS server over the signal channel, possibly traversing a DOTS relay, indicating the DOTS client's need for mitigation, as well as the scope of any requested mitigation (Section 6.1.1).

DELETE requests are used by the DOTS client to withdraw the request for mitigation from the DOTS server (Section 6.1.2).

GET requests are used by the DOTS client to retrieve the DOTS signal(s) it had conveyed to the DOTS server (Section 6.1.3).

PUT requests are used by the DOTS client to convey mitigation efficacy updates to the DOTS server (Section 6.1.4).

### 6.1.1. Convey DOTS Signals

An HTTP POST request is used to convey a DOTS signal to the DOTS server (Figure 4).

```
POST {scheme}://{host}:{port}/.well-known/{version}/{URI suffix for DOTS sign
Accept: application/json
Content-type: application/json
{
   "policy-id": "number",
   "target-ip": "string",
   "target-port": "string",
   "target-protocol": "string",
   "lifetime": "number"
 }
```

Figure 4: POST to convey DOTS signals

The header fields are described below.

policy-id: Identifier of the policy represented using a number. This identifier MUST be unique for each policy bound to the DOTS client, i.e. ,the policy-id needs to be unique relative to the active policies with the DOTS server. This identifier must be generated by the DOTS client. This document does not make any assumption about how this identifier is generated. This is a mandatory attribute.

target-ip: A list of IP addresses or prefixes under attack. This is an optional attribute.





target-port: A list of ports under attack. This is an optional attribute.

target-protocol: A list of protocols under attack. Valid protocol values include tcp, udp, sctp, and dccp. This is an optional attribute.

lifetime: Lifetime of the mitigation request policy in seconds. Upon the expiry of this lifetime, and if the request is not refreshed, the mitigation request is removed. The request can be refreshed by sending the same message again. The default lifetime of the policy is 60 minutes -- this value was chosen to be long enough so that refreshing is not typically a burden on the DOTS client, while expiring the policy where the client has unexpectedly quit in a timely manner. A lifetime of zero indicates indefinite lifetime for the mitigation request. The server MUST always indicate the actual lifetime in the response. This is an optional attribute in the request.

The relative order of two rules is determined by comparing their respective policy identifiers. The rule with lower numeric policy identifier value has higher precedence (and thus will match before) than the rule with higher numeric policy identifier value.

To avoid DOTS signal message fragmentation and the consequently decreased probability of message delivery, DOTS agents MUST ensure that the DTLS record MUST fit within a single datagram. If the Path MTU is not known to the DOTS server, an IP MTU of 1280 bytes SHOULD be assumed. The length of the URL MUST NOT exceed 256 bytes. If UDP is used to convey the DOTS signal and the request size exceeds the Path MTU then the DOTS client MUST split the DOTS signal into separate messages, for example the list of addresses in the 'target-ip' field could be split into multiple lists and each list conveyed in a new POST request.

Implementation Note: DOTS choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a

consideration and path MTU is unknown, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes, as per [RFC0791] IP packets up to 576 bytes should never need to be fragmented, thus sending a maximum of 500 bytes of DOTS signal over a UDP datagram will generally avoid IP fragmentation.

Figure 5 shows a POST request to signal that ports 80, 8080, and 443 on the servers 2002:db8:6401::1 and 2002:db8:6401::2 are being attacked.





```
POST https://www.example.com/.well-known/v1/DOTS signal
Accept: application/json
Content-type: application/json
{
  "policy-id":123321333242,
  "target-ip":[
      "2002:db8:6401::1",
      "2002:db8:6401::2"
  ],
  "target-port":[
      "80",
      "8080",
      "443"
  ],
  "target-protocol":"tcp"
 }
```

Figure 5: POST for DOTS signal

The DOTS server indicates the result of processing the POST request using HTTP response codes. HTTP 2xx codes are success, HTTP 4xx codes are some sort of invalid request and HTTP 5xx codes are returned if the DOTS server has erred or is incapable of performing the mitigation. Response code 200 (OK) will be returned in the response if the DOTS server has accepted the mitigation request and will try to mitigate the attack. If the request is missing one or more mandatory attributes then 400 (Bad Request) will be returned in the response or if the request contains invalid or unknown parameters then 400 (Invalid query) will be returned in the response. The HTTP response will include the JSON body received in the request.

### 6.1.2. Withdraw a DOTS Signal

An HTTP DELETE request is used to withdraw a DOTS signal from a DOTS server (Figure 6).

```
DELETE {scheme}://{host}:{port}/.well-known/{URI suffix for DOTS signal}
Accept: application/json
Content-type: application/json
 {
   "policy-id": "number"
 }
```

Figure 6: Withdraw DOTS signal

If the DOTS server does not find the policy number conveyed in the DELETE request in its policy state data, then it responds with "404" HTTP error response code. The DOTS server successfully acknowledges





a DOTS client's request to withdraw the DOTS signal using 200 (OK) response code, and ceases mitigation activity as quickly as possible.

#### 6.1.3. Retrieving a DOTS Signal

An HTTP GET request is used to retrieve information and status of a DOTS signal from a DOTS server (Figure 7). If the DOTS server does not find the policy number conveyed in the GET request in its policy state data then it responds with a 404 HTTP error response code.

1) To retrieve all DOTS signals signaled by the DOTS client.

```
GET {scheme}://{host}:{port}/.well-known/{URI suffix for DOTS signal}/list
```

2) To retrieve a specific DOTS signal signaled by the DOTS client. The policy information in the response will be formatted in the same order it was processed at the DOTS server.

```
GET {scheme}://{host}:{port}/.well-known/{URI suffix for DOTS signal}/<policy-i
```

Figure 7: GET to retrieve the rules

Figure 8 shows the response of all the active policies on the DOTS server.

```
   {
     "policy-data":[
       {
         "policy-id":123321333242,
         "target-prtoocol":"tcp",
         "lifetime":3600,
         "status":"mitigation in progress"
       },
       {
         "policy-id":123321333244,
         "target-protocol":"udp",
         "lifetime":1800,
         "status":"mitigation complete"
       },
       {
         "policy-id":123321333245,
         "target-protocol":"tcp",
         "lifetime":1800,
```

```
        "status":"attack stopped"
      }
    ]
  }
```

Figure 8: Response body

The various possible values of status field are explained below:

mitigation in progress: Attack mitigation is in progress (for e.g., changing the network path to re-route the inbound traffic to DOTS mitigator).

mitigation complete: Attack is successfully mitigated (for e.g., attack traffic is dropped).

attack stopped: Attack has stopped and the DOTS client can withdraw the mitigation request.

#### 6.1.3.1. Mitigation Status

A DOTS client retrieves the information about a DOTS signal at frequent intervals to determine the status of an attack. If the DOTS server has been able to mitigate the attack and the attack has stopped, the DOTS server indicates as such in the status, and the DOTS client recalls the mitigation request.

A DOTS client should react to the status of the attack from the DOTS server and not the fact that it has recognized, using its own means, that the attack has been mitigated. This ensures that the DOTS client does not recall a mitigation request in a premature fashion because it is possible that the DOTS client does not sense the DDOS attack on its resources but the DOTS server could be actively mitigating the attack and the attack is not completely averted.





### 6.1.4. Efficacy Update from DOTS Client

While DDoS mitigation is active, a DOTS client MAY frequently transmit DOTS mitigation efficacy updates to the relevant DOTS server. An HTTP PUT request (Figure 9) is used to convey the mitigation efficacy update to the DOTS server. The PUT request MUST

include all the header fields used in the POST request to convey the DOTS signal (Section 6.1.1). If the DOTS server does not find the policy number conveyed in the PUT request in its policy state data, it responds with a 404 HTTP error response code.

```
PUT {scheme}://{host}:{port}/.well-known/{URI suffix for DOTS signal}/<policy
Accept: application/json
Content-type: application/json
{
   "target-ip": "string",
   "target-port": "string",
   "target-protocol": "string",
   "lifetime": "number",
   "attack-status": "string"
}
```

Figure 9: Efficacy Update

The 'attack-status' field is a mandatory attribute. The various possible values contained in the 'attack-status' field are explained below:

in-progress: DOTS client determines that it is still under attack.

terminated: Attack is successfully mitigated (e.g., attack traffic is dropped).

## 7. DOTS Data Channel

A DOTS client can use RESTful APIs to provision and manage filters on the DOTS server. TBD: The data channel is intended to be used for bulk data exchanges and requires a reliable transport, CoAP over TLS over TCP is used for data channel.

JSON [RFC7159] payloads is used to convey both filtering rules as well as data channel specific payload messages that convey request parameters and response information such as errors. All data channel





URIs defined in this document, and in subsequent documents, MUST NOT have a URI containing "/DOTS signal".

One of the possible arrangements for DOTS client to signal filtering

rules to the DOTS server via the DOTS relay is discussed below:

The DOTS conveys the black-list rules to the DOTS relay. The DOTS relay validates if the DOTS client is authorized to signal the black-list rules and if the client is authorized propagates the rules to the DOTS server. Likewise, the DOTS server validates if the DOTS relay is authorized to signal the black-list rules. To create or purge filters, the DOTS client sends HTTP requests to the DOTS relay. The DOTS relay acts as an proxy, validates the rules and proxies the requests containing the black-listed IP addresses to the DOTS server. When the DOTS relay receives the associated HTTP response from the DOTS server, it propagates the response back to the DOTS client. If an attack is detected by the DOTS relay then it can act as a DOTS client and signal the black-list rules to the DOTS server. The DOTS relay plays the role of both client and proxy.

### 7.1. Filtering Rules

The following APIs define means for a DOTS client to configure filtering rules on a DOTS server.

#### 7.1.1. Install Filtering Rules

An HTTP POST request is used to push filtering rules to a DOTS server (Figure 10).

```
POST {scheme}://{host}:{port}/.well-known/{version}/{URI suffix for filtering
Accept: application/json
Content-type: application/json
{
   "policy-id": "number",
   "traffic-protocol": "string",
   "source-protocol-port": "string",
   "destination-protocol-port": "string",
   "destination-ip": "string",
   "source-ip": "string",
   "lifetime": "number",
   "traffic-rate" : "number"
 }
```

Figure 10: POST to install filtering rules

The header fields are described below:

policy-id: Identifier of the policy represented using a number. This identifier MUST be unique for each policy bound to the DOTS client, i.e., the policy-id needs to be unique relative to the active policies with the DOTS server. This identifier must be generated by the client. This document does not make any assumption about how this identifier is generated. This is an mandatory attribute.

traffic-protocol: Valid protocol values include tcp, udp, sctp, and dccp. This is an mandatory attribute.

source-protocol-port: The source port number, port number range (using "-"). For TCP, UDP, SCTP, or DCCP: the source range of ports (e.g., 1024-65535). This is an optional attribute.

destination-protocol-port: The destination port number, port number range (using "-"). For TCP, UDP, SCTP, or DCCP: the destination range of ports (e.g., 443-443). This information is useful to avoid disturbing a group of customers when address sharing is in use [RFC6269]. This is an optional attribute.

destination-ip: The destination IP address, IP addresses separated by commas, or prefixes using "/" notation. This is an optional attribute.

source-ip: The source IP addresses, IP addresses separated by commas, or prefixes using "/" notation. This is an optional attribute.

lifetime: Lifetime of the rule in seconds. Upon the expiry of this lifetime, and if the request is not refreshed, this particular rule is removed. The rule can be refreshed by sending the same message again. The default lifetime of the rule is 60 minutes -- this value was chosen to be long enough so that refreshing is not typically a burden on the DOTS client, while expiring the rule where the client has unexpectedly quit in a timely manner. A lifetime of zero indicates indefinite lifetime for the rule. The server MUST always indicate the actual lifetime in the response. This is an optional attribute in the request.

traffic-rate: This is the allowed traffic rate in bytes per second indicated in IEEE floating point [IEEE.754.1985] format. The value 0 indicates all traffic for the particular flow to be discarded. This is a mandatory attribute.

The relative order of two rules is determined by comparing their respective policy identifiers. The rule with lower numeric policy

identifier value has higher precedence (and thus will match before) than the rule with higher numeric policy identifier value.

Figure 11 shows a POST request to block traffic from attacker IPv6 prefix 2001:db8:abcd:3f01::/64 to network resource using IPv6 address 2002:db8:6401::1 to operate a server on TCP port 443.

```
POST https://www.example.com/.well-known/v1/filter
Accept: application/json
Content-type: application/json
 {
   "policy-id": 123321333242,
   "traffic-protocol": "tcp",
   "source-protocol-port": "0-65535",
   "destination-protocol-port": "443",
   "destination-ip": "2001:db8:abcd:3f01::/64",
   "source-ip": "2002:db8:6401::1",
   "lifetime": 1800,
   "traffic-rate": 0
 }
```

Figure 11: POST to Install Black-list Rules

### 7.1.2. Remove Filtering Rules

An HTTP DELETE request is used to delete filtering rules from a DOTS server (Figure 12).

```
DELETE {scheme}://{host}:{port}/.well-known/{URI suffix for filtering}
Accept: application/json
Content-type: application/json
 {
   "policy-id": "number"
 }
```

Figure 12: DELETE to remove the rules

### 7.1.3. Retrieving Installed Filtering Rules

An HTTP GET request is used to retrieve filtering rules from a DOTS

server.

Figure 13 shows an example to retrieve all the black-lists rules programmed by the DOTS client while Figure 14 shows an example to retrieve specific black-list rules programmed by the DOTS client.





```
GET {scheme}://{host}:{port}/.well-known/{URI suffix for filtering}
```

Figure 13: GET to retrieve the rules (1)

```
GET {scheme}://{host}:{port}/.well-known/{URI suffix for filtering}
Accept: application/json
Content-type: application/json
{
   "policy-id": "number"
}
```

Figure 14: GET to retrieve the rules (2)

TODO: show response

## 8. IANA Considerations

TODO

## 9. Security Considerations

Authenticated encryption MUST be used for data confidentiality and message integrity. (D)TLS based on client certificate MUST be used for mutual authentication. The interaction between the DOTS agents requires Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS) with a ciphersuite offering confidentiality protection and the guidance given in [RFC7525] MUST be followed to avoid attacks on (D)TLS.

If TCP is used between DOTS agents, attacker may be able to inject RST packets, bogus application segments, etc., regardless of whether TLS authentication is used. Because the application data is TLS protected, this will not result in the application receiving bogus

data, but it will constitute a DoS on the connection. This attack can be countered by using TCP-AO [RFC5925]. If TCP-AO is used, then any bogus packets injected by an attacker will be rejected by the TCP-AO integrity check and therefore will never reach the TLS layer.

Special care should be taken in order to ensure that the activation of the proposed mechanism won't have an impact on the stability of the network (including connectivity and services delivered over that network).

Involved functional elements in the cooperation system must establish exchange instructions and notification over a secure and authenticated channel. Adequate filters can be enforced to avoid that nodes outside a trusted domain can inject request such as deleting filtering rules. Nevertheless, attacks can be initiated





from within the trusted domain if an entity has been corrupted. Adequate means to monitor trusted nodes should also be enabled.

## 10. Acknowledgements

Thanks to Christian Jacquenet, Roland Dobbins, Andrew Mortensen, Roman D. Danyliw, and Gilbert Clark for the discussion and comments.

## 11. References

### 11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <http://www.rfc-editor.org/info/rfc5925>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

### 11.2. Informative References

[I-D.ietf-dots-requirements] Mortensen, A., Moskowitz, R., and T. Reddy, "DDoS Open Threat Signaling Requirements", draft-ietf-dots-requirements-00 (work in progress), October 2015.





[I-D.ietf-tls-cached-info] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-22 (work in progress), January 2016.

[I-D.ietf-tls-falsestart] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", draft-ietf-tls-falsestart-01 (work in progress), November 2015.

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", August 1985.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <http://www.rfc-editor.org/info/rfc791>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <http://www.rfc-editor.org/info/rfc4732>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <http://www.rfc-editor.org/info/rfc5077>.

[RFC5575] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, DOI 10.17487/RFC5575, August 2009, <http://www.rfc-editor.org/info/rfc5575>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <http://www.rfc-editor.org/info/rfc6269>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.





[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <http://www.rfc-editor.org/info/rfc6555>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <http://www.rfc-editor.org/info/rfc6724>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <http://www.rfc-editor.org/info/rfc7413>.

## Appendix A. BGP

BGP defines a mechanism as described in [RFC5575] that can be used to automate inter-domain coordination of traffic filtering, such as what is required in order to mitigate DDoS attacks. However, support for BGP in an access network does not guarantee that traffic filtering will always be honored. Since a DOTS client will not receive an acknowledgment for the filtering request, the DOTS client should monitor and apply similar rules in its own network in cases where the DOTS server is unable to enforce the filtering rules. In addition, enforcement of filtering rules of BGP on Internet routers are usually governed by the maximum number of data elements the routers can hold as well as the number of events they are able to process in a given unit of time.

## Authors' Addresses

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

Email: tireddy@cisco.com





Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

Prashanth Patil
Cisco Systems, Inc.

Email: praspati@cisco.com

Mike Geller
Cisco Systems, Inc.
3250
Florida  33309
USA

Email: mgeller@cisco.com

Mohamed Boucadair
Orange
Rennes  35000
France

Email: mohamed.boucadair@orange.com

Robert Moskowitz
HTT Consulting
Oak Park, MI  42837
United States

Email: rgm@htt-consult.com