OPSWG WG
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

T. Reddy
McAfee
D. Wing
Citrix

July 8, 2019

MUD (D)TLS profiles for IoT devices
draft-reddy-opswg-mud-tls-00

Abstract

This memo extends Manufacturer Usage Description (MUD) to model DTLS and TLS usage. This allows a network element to notice abnormal DTLS or TLS usage which has been strong indicator of other software running on the endpoint, typically malware.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Encryption is necessary to protect the privacy of end users using IoT devices. In a network setting, TLS [RFC8446] and DTLS [I-D.ietf-tls-dtls13] are the dominant protocols to provide encryption for IoT device traffic. Unfortunately in conjunction with IoT applications rise of encryption, malware is also using encryption which thwarts network-based analysis such as deep packet inspection (DPI). Other mechanisms are needed to notice malware is running on the IoT device.

Malware frequently uses its own libraries for its activities, and those libraries are re-used much like any other software engineering project. Research [malware] indicates there are observable differences in how malware uses encryption compared with non-malware uses encryption. There are several interesting findings specific to DTLS and TLS which were found common to malware:

o Older and weaker cryptographic parameters (e.g., TLS_RSA_WITH_RC4_128_SHA).

o TLS SNI and server certificates are composed of subjects with characteristics of a domain generation algorithm (DGA) (e.g., www.33mhwt2j.net).

o Higher use of self-signed certificates compared with typical legitimate software.

o Discrepancies in the server name indication (SNI) TLS extension in the ClientHello message and the DNS names in the SubjectAltName(SAN) X.509 extension in the server certificate message.

o Discrepancies in the key exchange algorithm and the client public key length in comparison with legitimate flows. As a reminder, Client Key Exchange message has been removed from TLS 1.3.

o Lower diversity in TLS client advertised TLS extensions compared to legitimate clients.

If observable (D)TLS profile parameters are used, the following discusses the favorable impact on network security:

o Although IoT devices that have a single or small number of uses might have very broad communication patterns. In such a case, MUD rules using ACLs on its own is not suitable for these IoT devices but observable (D)TLS profile parameters can be used for such IoT devices to permit intended use and to block malicious behaviour of IoT devices.

o Several TLS deployments have been vulnerable to active Man-In-The-Middle (MITM) attacks because of lack of certificate validation. By observing (D)TLS profile parameters, a network element can detect when the TLS SNI mismatches the SubjectAltName and detect when the server's certificate is invalid, and alert those situations.

o IoT device can learn a new skill, and the new skill changes the way the IoT device communicates with other devices located in the local network and Internet. In other words, if IP addresses and domain names the IoT device connects to rapidly changes and MUD rules using ACLs cannot be rapidly updated, observable (D)TLS profile parameters can be used to permit intended use and to block malicious behaviour of IoT device.

This document extends MUD [RFC8520] to model observable (D)TLS profile parameters. Using these (D)TLS profile parameters, an active MUD-enforcing firewall can identify MUD non-compliant DTLS and TLS behavior that can indicate malware is running on the IoT device. This detection can prevent malware download, block access to malicious domains, enforce use of strong ciphers, stop data exfiltration, etc. In addition, organizations may have policies around acceptable ciphers and certificates on the websites the IoT devices connect to. Examples include no use of old and less secure versions of TLS, no use of self-signed certificates, deny-list or accept-list of Certificate Authorities, valid certificate expiration

time, etc. These policies can be enforced by observing the (D)TLS profile parameters. Enterprise firewall can use the IoT device's (D)TLS profile parameters to identify legitimate flows by observation of (D)TLS sessions, and can make inferences to permit legitimate flows and to block malicious flows. The proposed technique is also suitable in deployments where decryption techniques are not ideal due to privacy concerns, non-cooperating end-points and expense.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

"(D)TLS" is used for statements that apply to both Transport Layer Security [RFC8446] and Datagram Transport Layer Security [RFC6347]. Specific terms are used for any statement that applies to either protocol alone.

3. Overview of MUD (D)TLS profiles for IoT devices

In Enterprise networks, protection and detection are typically done both on end hosts and in the network. Host agents have deep visibility on the devices where they are installed, whereas the network has broader visibility. Installing host agents may not be a viable option on IoT devices, and network-based security can only be used to protect such IoT devices. (D)TLS profile parameters of IoT device can be used by middle-boxes to detect and block malware communication, while at the same time preserving the privacy of legitimate uses of encryption. Middle-boxes need not proxy (D)TLS but can passively observe the parameters of (D)TLS handshakes from IoT devices and gain good visibility into TLS 1.0 to 1.2 parameters and partial visibility into TLS 1.3 parameters. Malicious agents can try to use the (D)TLS profile parameters as legitimate agents to evade detection but it becomes a challenge to mimic the behavior of various IoT device types and IoT device models from several manufacturers. In other words, malware developers will have to develop malicious agents per IoT device type, manufacturer and model (which will be several thousands), infect the device with specific malware agent and will have keep up with the updates to (D)TLS profile parameters of IoT devices. Further, the malware command and control server certificates needs to be signed by the same certifying authorities trusted by the IoT devices.

4. (D)TLS profile YANG module

This document specifies a YANG module for representing (D)TLS profile. The (D)TLS profile YANG module provides a method for firewall to observe the (D)TLS profile parameters in the (D)TLS handshake to permit intended use and to block malicious behavior. This module uses the common YANG types defined in [RFC6991] , rules defined in [RFC8519] and cryptographic types defined in [I-D.ietf-netconf-crypto-types].

The (D)TLS profile parameters include the following:

- (D)TLS versions supported by the IoT device
- List of supported symmetric encryption algorithms
- List of supported compression methods
- List of extension types
- List of client key exchange algorithms and the client public key lengths in versions prior to (D)TLS 1.3

o List of trust anchor certificates used by the IoT device. Note that server certificate is encrypted in (D)TLS 1.3 and the middle-box without acting as (D)TLS proxy cannot validate the server certificate.

o List of DHE or ECDHE groups supported by the client

o List signature algorithms the client can validate in X.509 server certificates

o List of SPKI pin sets pre-configured on the client to validate self-signed server certificates or raw public keys

o If SNI mismatch is allowed or not, and if SNI mismatch is allowed, the server names for which SNI mismatch is allowed.

If the (D)TLS profile parameters are not observed in a (D)TLS session from the IoT device, the default behaviour is to block the (D)TLS session.

4.1. Tree Structure

This document augments the "ietf-mud" MUD YANG module defined in [RFC8520] for signaling the IoT device (D)TLS profile. This document

defines the YANG module "reddy-opsawg-mud-tls-profile", which has the following tree structure:

```
module: reddy-opsawg-mud-tls-profile
  augment /mud:mud/mud:from-device-policy:
    +--rw client-profile
       +--rw tls-profiles* [protocol-version supported_versions]
          +--rw protocol-version          uint16
          +--rw supported_versions        boolean
          +--rw encryption-algorithms*    encryption-algorithm
          +--rw compression-methods*      compression-method
          +--rw extension-types*          extension-type
          +--rw acceptlist-ta-certs*      ct:trust-anchor-cert-cms
          +--rw SPKI-pin-sets*            SPKI-pin-set
          +--rw SPKI-hash-algorithm       ct:hash-algorithm-t
```

```
         +--rw supported-groups*        supported-group
         +--rw signature-algorithms*    signature-algorithm
         +--rw client-public-keys
         |  +--rw key-exchange-algorithms*      key-exchange-algorithm
         |  +--rw client-public-key-lengths*    client-public-key-length
         +--rw SNI-mismatch-allowed?    boolean
         +--rw server-name*             inet:domain-name
         +--rw actions
            +--rw forwarding    identityref
```

4.2. YANG Module

```
module reddy-opsawg-mud-tls-profile {
   yang-version 1.1;
   namespace "urn:ietf:params:xml:ns:yang:reddy-opsawg-mud-tls-profile";
   prefix mud-tls-profile;


   import ietf-crypto-types {
     prefix ct;
     reference "draft-ietf-netconf-crypto-types-01:
                Common YANG Data Types for Cryptography";
   }

   import ietf-inet-types {
     prefix inet;
     reference "Section 4 of RFC 6991";
   }

   import ietf-mud {
     prefix mud;
     reference "RFC 8520";
   }
```



```
   import ietf-access-control-list {
     prefix ietf-acl;
     reference
       "RFC 8519: YANG Data Model for Network Access
                  Control Lists (ACLs)";
   }

   organization
```

```
     "IETF Operations and Management Area Working Group Working Group";
   contact
      "Editor:  Konda, Tirumaleswar Reddy
               <mailto:TirumaleswarReddy_Konda@McAfee.com>";

   description
     "This module contains YANG definition for configuring
      aliases for resources and filtering rules using DOTS
      data channel.

      Copyright (c) 2019 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject
      to the license terms contained in, the Simplified BSD License
      set forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (http://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC XXXX; see
      the RFC itself for full legal notices.";

   revision 2019-06-12 {
     description
       "Initial revision.";
   }

   typedef compression-method {
     type uint8;
     description "Compression method.";
   }

   typedef extension-type {
     type uint16;
     description "Extension type.";
   }

   typedef encryption-algorithm {
     type uint16;
```



```
     description "Encryption algorithms.";
```

```
   }

   typedef supported-group {
     type uint16;
     description "supported DHE or ECDHE group.";
   }

   typedef SPKI-pin-set {
     type binary;
     description "Subject Public Key Info pin set.";
   }

   typedef signature-algorithm {
     type uint16;
     description "Signature algorithm";
   }

   typedef key-exchange-algorithm {
     type uint8;
     description "key exchange algorithm";
   }
   typedef client-public-key-length {
     type uint8;
     description "client public key length";
   }

   augment "/mud:mud/mud:from-device-policy" {
     container client-profile {
       list tls-profiles {
         key "protocol-version supported_versions";
         description
          "(D)TLS version profiles supported by the client";
         leaf protocol-version {
           type uint16;
           description "Legacy protocol version";
         }
         leaf supported_versions {
           type boolean;
           description "supported versions extension for TLS 1.3";
         }
         leaf-list encryption-algorithms {
           type encryption-algorithm;
           description "Encryption algorithms";
         }
         leaf-list compression-methods {
           type compression-method;
            description "Compression methods";
```

```
         }
         leaf-list extension-types {
           type extension-type;
           description "Extension Types";
         }
         leaf-list acceptlist-ta-certs {
           type ct:trust-anchor-cert-cms;
           description
             "A list of trust anchor certificates used by the client";
         }
         leaf-list SPKI-pin-sets {
            type SPKI-pin-set;
            description
             "A list of SPKI pin sets pre-configured on the client
              to validate self-signed server certificate or
              raw public key";
         }
         leaf SPKI-hash-algorithm {
           type ct:hash-algorithm-t;
           description
             "cryptographic hash algorithm used to generate the SPKI pinset";
         }
         leaf-list supported-groups {
            type supported-group;
            description
             "A list of DHE or ECDHE groups supported by the client";
         }
         leaf-list signature-algorithms {
            type signature-algorithm;
            description
             "A list signature algorithms the client can validate
              in X.509 certificates.";
         }
         container client-public-keys {
           when "../supported_versions = 'false'";
           leaf-list key-exchange-algorithms {
             type key-exchange-algorithm;
             description
             "Key exchange algorithms supported by the client";
           }
           leaf-list client-public-key-lengths {
             type client-public-key-length;
             description
             "client public key lengths";
           }
         }
```

```
        leaf SNI-mismatch-allowed {
          type boolean;
```



```
          default "false";
          description
           "If set to 'false', SNI mismatch is not allowed.";
        }
        leaf-list server-name {
           when "../SNI-mismatch-allowed = 'true'";
           type inet:domain-name;
           description
           "Server names (FQDN) for which SNI mismatch is allowed.";
        }
        container actions {
          description
          "Definitions of action for this profile.";
          leaf forwarding {
            type identityref {
              base ietf-acl:forwarding-action;
            }
            mandatory true;
            description
            "Specifies the forwarding action for the (D)TLS profile.";
            reference
            "RFC 8519: YANG Data Model for Network Access
                          Control Lists (ACLs)";
          }
        }
      }
    }
  }
}
```

5. (D)TLS 1.3 handshake

In (D)TLS 1.3, full (D)TLS handshake inspection is not possible since all (D)TLS handshake messages excluding the ClientHello message are encrypted. (D)TLS 1.3 has introduced new extensions in the handshake record layers called Encrypted Extensions. Using these extensions handshake messages will be encrypted and network devices (such as a firewall) are incapable deciphering the handshake, thus cannot view the server certificate. However, a few parameters in the ServerHello are still visible such as the chosen cipher. Note that Client Key Exchange message has been removed from (D)TLS 1.3.

5.1. Encrypted SNI

To increase privacy, encrypted SNI [I-D.ietf-tls-sni-encryption] prevents passive observation of the TLS Server Name Indication and to effectively provide privacy protection, SNI encryption needs to be used in conjunction with DNS encryption (e.g., DNS-over-(D)TLS or

DNS- over-HTTPS). Firewall inspecting the (D)TLS 1.3 handshake cannot decrypt encrypted SNI. If an IoT device is configured to use public DNS-over-(D)TLS or DNS- over-HTTPS servers, the policy enforcement point is moved to that public server, which cannot enforce the MUD policy based on domain names (Section 8 of [RFC8520]). Thus the use of a public DNS-over-(D)TLS or DNS- over-HTTPS server is incompatible with MUD. A local DNS server is necessary to allow MUD policy enforcement on the local network ([I-D.ietf-doh-resolver-associated-doh] and [I-D.reddy-dprive-bootstrap-dns-server]).

5.2. Full (D)TLS 1.3 handshake inspection

Middle-box needs to act as a (D)TLS 1.3 proxy to observe the parameters of (D)TLS handshakes from IoT devices and gain good visibility into TLS 1.3 parameters. The following steps explain the mechanism to automatically bootstrap IoT devices with local network's CA certificates and to enable the middle-box to act as a (D)TLS 1.3 proxy.

- Bootstrapping Remote Secure Key Infrastructures (BRSKI) discussed in [I-D.ietf-anima-bootstrapping-keyinfra] provides a solution for secure automated bootstrap of devices. BRSKI specifies means to provision credentials on devices to be used to operationally access networks. In addition, BRSKI provides an automated mechanism for the bootstrap distribution of CA certificates from the Enrollment over Secure Transport (EST) [RFC7030] server. The IoT device can use BRSKI to automatically bootstrap the IoT device using the IoT manufacturer provisioned X.509 certificate, in combination with a registrar provided by the local network and IoT device manufacturer's authorizing service (MASA).

1. The IoT device authenticates to the local network using the IoT manufacturer provisioned X.509 certificate. The IoT device can request and get a voucher from the MASA service via the registrar. The voucher is signed by the MASA service and includes the local network's CA public key.

2. The IoT device validates the signed voucher using the manufacturer installed trust anchor associated with the MASA, stores the CA's public key and validates the provisional TLS connection to the registrar.

3. The IoT device requests the full EST distribution of current CA certificates (Section 5.9.1 in [I-D.ietf-anima-bootstrapping-keyinfra]) from the registrar operating as a BRSKI-EST server. The IoT device stores the CA certificates as Explicit Trust Anchor database entries. The

IoT device uses the Explicit Trust Anchor database to validate the server certificate.

4. The middle-box uses the "supported_versions" TLS extension (defined in TLS 1.3 to negotiate the supported TLS versions between client and server) to determine the TLS version. During the (D)TLS handshake, If (D)TLS version 1.3 is used, the middle-box ((D)TLS proxy) modifies the certificate provided by the server and signs it with the private key from the local CA certificate. The middle-box has visibility into further exchanges between the IoT device and server which enables it to inspect the (D)TLS 1.3 handshake, enforce the MUD (D)TLS profile and can inspect subsequent network traffic.

5. The IoT device uses the Explicit Trust Anchor database to validate the server certificate.

The proposed technique empowers the middle-box to reject (D)TLS 1.3 sessions that violate the MUD (D)TLS profile.

6. MUD File Example

This example below contains (D)TLS profile parameters for a IoT device. JSON encoding of YANG modelled data [RFC7951] is used to illustrate the example.

```
   {
      "ietf-mud:mud": {
        "mud-version": 1,
         "mud-url": "https://example.com/IoTDevice",
         "last-update": "2019-18-06T03:56:40.105+10:00",
         "cache-validity": 100,
         "is-supported": true,
         "systeminfo": "IoT device name",
         "reddy-opsawg-mud-tls-profile:from-device-policy": {
           "client-profile": {
             "tls-version-profile" : [
              {
                    "protocol-version" : 771,
                    "supported_versions_ext" : "FALSE",
                    "encryption-algorithms" : [31354, 4865, 4866, 4867],
                    "extension-types" : [10],
                    "supported-groups" : [29],
                    "actions": {
```

```
                     "forwarding": "accept"
                   }
             }
           ]
         }
       }
     }
   }
```

7. Security Considerations

Security considerations in [RFC8520] need to be taken into consideration.

8. IANA Considerations

This document requests IANA to register the following URIs in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

      URI: urn:ietf:params:xml:ns:yang:reddy-opsawg-mud-tls-profile
      Registrant Contact: The IESG.
      XML: N/A; the requested URI is an XML namespace.

9. Acknowledgments

TODO

10. References

10.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K. and H. Wang, "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-10 (work in progress), July 2019.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The

Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-31 (work in progress), March 2019.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

10.2. Informative References

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason,

S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-22 (work in progress), June 2019.

[I-D.ietf-doh-resolver-associated-doh]
Hoffman, P., "Associating a DoH Server with a Resolver", draft-ietf-doh-resolver-associated-doh-03 (work in progress), March 2019.

[I-D.ietf-tls-sni-encryption]
Huitema, C. and E. Rescorla, "Issues and Requirements for SNI Encryption in TLS", draft-ietf-tls-sni-encryption-04 (work in progress), November 2018.

[I-D.reddy-dprive-bootstrap-dns-server]
K, R., Wing, D., Richardson, M., and M. Boucadair, "A Bootstrapping Procedure to Discover and Authenticate DNS-over-(D)TLS and DNS-over-HTTPS Servers", draft-reddy-dprive-bootstrap-dns-server-04 (work in progress), June 2019.

[malware] Anderson, B., Paul, S., and D. McGrew, "Deciphering Malware's use of TLS (without Decryption)", July 2016, <https://arxiv.org/abs/1607.01639>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <https://www.rfc-editor.org/info/rfc7478>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/info/rfc8520>.

Authors' Addresses

Tirumaleswar Reddy
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore, Karnataka 560071
India

Email: kondtir@gmail.com

Dan Wing
Citrix Systems, Inc.
4988 Great America Pkwy
Santa Clara, CA 95054
USA

Email: danwing@gmail.com