OpenDXL in active response scenarios

Tarmo Randel
e-mail: tarmo.randel@ccdcoe.org

NATO CCD COE

November 2017

Abstract

Automating response to cyber security incidents is the trend which is - considering increasing amount of incidents organizations handle and ever-increasing attack surface - already becoming mainstream. In this paper I explore the options for using OpenDXL as central data bus in real life situation of mixed environments, legacy solutions and multiple vendors for connecting existing and future cyber security system components for coordinated information exchange and orchestrating incident response action. Presented in the DeepSec'17 Cnference in Vienna.

Keywords: Cybersecurity incident, active response, automation, open source, OpenDXL, MQTT, MISP, Phantom

1 Introduction

This paper is not presenting OpenDXL as yet another silver bullet to solve all the problems, rather explores options introduced by well known technology from the Internet of things world implemented in new environment - cybersecurity.

2 The need for rapid incident response

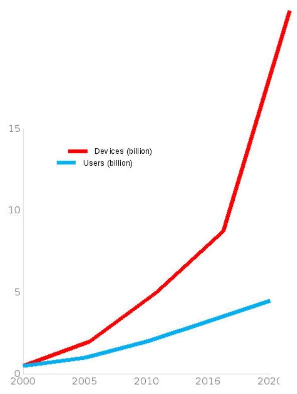

Fig 1. Growth of cyberspace

We are living in the interconnected world which is becoming more and more connected literally every second (Fig 1). We do not see such rapid increase in team numbers dealing with cyber security incidents and people, who are qualified to handle cyber security incidents.

In our world of limited resources and increasing number of connected, and potentially vulnerable devices, forces organizations and teams, responsible for cybersecurity issues, optimize their performance with all available means and methods. Only way it can happen is with increasing use of data analysis and process automation.

Relevant actors - security communities and industry - responds to this trend by developing products directed to maximizing efficiency of incident response workflow. Incident response workflow automation and orchestration are current trends and active response is becoming part of these products sooner or later (if not already there).

2.1 What is active response?

Active response, is usually considered to be immediate reaction to identified threat. It should be noted that "hack back" is not, and hopefully never will be[1], considered as viable response option in active response scenarios. It must be noted that there is no clear and commonly acknowledged definition available for either "active response" or "active defence" in cybersecurity perspective.

According to Christopher Ensey [2] there are six conditions to be met in order to have active protection in place:

1. Centralized event management
2. Analytics
3. Open APIs
4. Dynamic infrastructure.
5. The Human element
6. Complete visibility

All the six items are vital for automation and central orchestration.

2.2 Automation and orchestration

Active protection can of course be implemented in decentralized manner but central orchestration would be the recommended way to implement active response. One reason for that is that in decentralized approach the system has limited response options because of limited visibility and knowledge about response actions performed by other systems. As an example one could imagine two data paths for redundant system connectivity and case of attack where edge devices on both data path decide independently to cut the traffic in order to mitigate the attack. So in order to avoid undesired serious consequences centralized view and orchestration is vital.

There are already some options in the market to do automation and orchestration in centralized manner. Community version the Phantom is one of the options to explore in research to see how and if it can be used in active response scenarios.

2.3 Complexity of orchestrating connected systems

Orchestrating, or remotely controlling, the systems actually means that systems have to talk with other systems. It is quite common topic in incident response since information exchange has essential part in the everyday life of the incident response team.

For successful information exchange we need to make sure that the transmitted data is interpreted same way by all parties. Many years ago, when we did not have STIX and Yara as de facto standards for information exchange, teams had to implement parsers for every information provider. I have to admit - STIX is difficult to implement since it is quite extensive standard but information can be interpreted same way and you do not have to worry about the structure and format.

Additional component required for orchestration is interface to control the systems (devices). For this purpose all enterprise level devices or software have APIs for interaction. Via APIs we can exchange information with and control behaviour of the device or system with other devices and systems. Most of the devices or vendors have, of course, their own APIs.

For research purposes I attempt to create a system where all connected devices and systems interact via common data bus (OpenDXL). This way we can hopefully reduce complexity and avoid software version incompatibility issues which are common problems when trying to connect devices from multiple vendors. This is also a great jumping point to extend functionality of response workflow to automated active response.

[1] Jody Westby, https://www.forbes.com/sites/jodywestby/2012/11/29/caution-active-response-to-cyber-attacks-has-high-risk/#1bf97d996d71

[2] http://www.securityinfowatch.com/article/12360289/6-steps-to-achieving-active-response

3 The Goal - active response scenarios with proof of concepts

For the purpose of exploring suitability of OpenDXL fabric as central data bus for active response we implement in proof of concept level following activity flow:

- Data flow from local Honeynet feed (or MISP) with possibly malicious IPs.
- Lookup traffic to IP from Netflow database, to identify possibly compromised hosts inside organizations network.

  Apply predefined activity on identified hosts, if there are any matches. Create security incident.
- Apply firewall rules as preventative measure, Create security event.
- Expire data, if applicable.

Implementing this workflow could give us quite good overview of the features and problems.

4 Introduction to the OpenDXL

OpenDXL client libraries were made public by McAfee in the beginning of year 2017 under Apache2.0 license, there are some implementations available in the Github (https://github.com/opendxl) but there is serious lack of documentation about the OpenDXL protocol itself and its implementation requirements.

Besides the lack of the documentation another blocking point would be the fact that OpenDXL is not so open as the name claims since the server side requires infrastructure containing at least two central McAfee components: TIE[3] and ePO[4].

While investigating options for implementing OpenDXL as central data exchange bus for the threat related technical information in order to share the data between various security systems and manage the operation of threat detection and mitigation infrastructure, this were identified:

- OpenDXL data exchange layer is based on known and mature MQTT protocol[5] widely used in IoT[6] solutions
- systems connected to the OpenDXL bus have to implement MQTT specific publish/subscribe logic for interaction
- Security of the data exchange layer is provided by industry standard cryptographic implementations

What about "Open" in the OpenDXL? That is good question since it is based on MQTT, which is publicly available standard, and McAfee has not published implementation guidelines or standards, just libraries to interact with it's implementation of MQTT broker. Good news is that McAfee has made it's implementation of MQTT broker publicly available as the Docker image[7] so proof of concepts and developing interfaces using OpenDXL fabric can be carried out without actually installing all the commercial components that make up OpenDXL in the McAfee world. In this paper I will not cover OpenDXL related commercial products properties in detail; however it is important to understand how they operate while investigating options for our desired implementation.

4.1 Architecture of the OpenDXL

OpenDXL architecture follows, as it is based on the MQTT, same principles as MQTT. Data bus is created when you have at least one broker where the clients can connect to and exchange data packets. You can have several brokers which brokers can behave either as selective (filtering) or forwarding proxies.

[3] McAfee Threat Intelligence Exchange, https://www.mcafee.com/us/products/threat-intelligence-exchange.aspx
[4] McAfee ePolicy Orchestrator, https://www.mcafee.com/us/products/epolicy-orchestrator.aspx
[5] see http://mqtt.org and https://github.com/mqtt/mqtt.github.io/wiki
[6] Internet of Things refers to the devices that interact using Internet, https://www.ibm.com/internet-of-things/resources/library/what-is-iot/
[7] see https://github.com/opendxl/opendxl-broker

Connected clients may subscribe to the data paths (topics) and process the data received from there however they see fit, clients can also publish data.

Messages are published to and read from topics in hierarchical format, for example power switch could controlled by sending message to topic b1/switches/sonoff13/POWER and switch publishes its state in topic b1/switches/sonoff13/STATE and temperature sensor publishes its reading in b1/sensors/ temperature/temp8 [8].

One can filter topics with wildcard characters, where ”+” is wildcard for single level and ”#” is wildcard for remaining levels.

Examples of filters:
b1/switches/+/STATE matches
b1/switches/sonoff13/STATE
b1/# matches
b1/switches/sonoff13/STATE and b1/sensors/temperature/temp8

4.2 OpenDXL security

As the OpenDXL is based on MQTT the security implementation is derived from MQTT as well. MQTT open standard describes need for implementations to provide mechanisms for:

1. Authentication of users and devices
2. Authorization of access to Server resources
3. Integrity of MQTT Control Packets and application data contained therein
4. Privacy of MQTT Control Packets and application data contained therein

Chapter covering providing security is non normative, current implementations however use industry proven verified and certified technologies in order to meet compliance requirements set by various industry security standards[9].

Since there is no official OpenDXL protocol specification published, we can only assume from the published client library implementations that OpenDXL is exploiting Level 3 security profile. Secrecy of communication and client/broker authentication is implemented in OpenDXL with transport layer security (TLS[10]).

OpenDXL implements TLS 1.2 protocol[11] in order to provide three basic components of security - confidentiality, availability and integrity. As the MQTT protocol itself is not trust symmetrical since there are no mechanisms for the client to authenticate the broker (server), OpenDXL solves that problem by requiring broker certificate in client TLS implementations to be present and verifies it. Testing if this mechanisms for preventing communication between compromised clients and servers is functional is beyond the scope of this paper.

It should be noted that having TLS as an requirement can make OpenDXL implementation tricky if the infrastructure and processes of your organization do not provide and support certificate authority service.

Setting up TLS security for server and clients involves following steps:

1. Create the CA or initiate subchain in existing CA. There are many options: Certification Authority for Microsoft Server software[12], open source CA's software (EJBCA[13]) or CA as an service. For testing or really small deployments one can also create local CA fit openssl CLI tools[14]

[8] see http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718106
[9] NIST Cyber Security Framework, PCI-DSS, FIPS-140-2 and NSA Suite B
[10] https://datatracker.ietf.org/wg/tls/charter/
[11] http://tools.ietf.org/html/rfc5246
[12] https://msdn.microsoft.com/en-us/library/cc875810.aspx
[13] https://www.ejbca.org
[14] https://opendxl.github.io/opendxl-client-python/pydoc/certcreation.html

2. Import CA certificate into ePO[15]
3. Export broker certificate(s) from ePO[16]

Selected method can be inconvenient to implement and maintain and can introduce unplanned expenses, since it pretty much relies on the proper setup of own certificate authority (CA) or purchasing service from third parties.

4.3 Identified problems of OpenDXL

There are two obvious problems with the OpenDXL:

- Unregulated use of main concept of OpenDXL fabric: MQTT topic.
- Feature of MQTT: you have to be on the bus in order to act/react. If information is passed to the bus while device was not listening then information is last. This problem can be overcome with logger/historian type of element as either part of broker or as separate service.

We note this as topics to address in later research projects.

4.4 Server side of the OpenDXL

In order to successfully deploy OpenDXL in your infrastructure, following steps must be completed:

1. McAfee ePolicy Orchestrator must be installed and configured (at least) with these plugins:
   DXL Client
   DXL Client Management
   DXL Broker Management
   TIE Server Management
2. McAfee TIE must be installed and configured
3. CA must be available

As mentioned earlier in the paper providing proper certificate authority as a service inside the organization can be tricky and/or expensive in many terms and setting CA up properly is essential since compromised client can provide attacker unrestricted access to the OpenDXL fabric.

Good news is that starting from October 2017 standalone version of OpenDXL broker is available as Docker image (https://github.com/opendxl/opendxl-broker) greatly reducing complexity of developing and testing software integrations.

5 Existing OpenDXL bindings

5.1 OpenDXL TIE bindings

McAfee Threat Intelligence Exchange (TIE) is commercial component of we will not go into too detail but TIE provides us good opportunity to explore internals of OpenDXL connected system.

TIE is basically central repository for reputation. MQTT style OpenDXL topics for TIE is good example of how the MQTT path is built and used by the application so it is the reason why it is included in this paper.

Service type:
/mcafee/service/tie

Topics (fragment):

[15] https://opendxl.github.io/opendxl-client-python/pydoc/epocaimport.html
[16] https://opendxl.github.io/opendxl-client-python/pydoc/epobrokercertsexport.html

/.../
/mcafee/service/tie/cert/reputation/get
/mcafee/service/tie/cert/reputation/set
/.../
/mcafee/service/tie/file/reputation/get
/mcafee/service/tie/file/reputation/set
/.../
/mcafee/service/tie/file/url/reputation/add
/.../

Service type:
/mcafee/service/tie/management

Topics:
/mcafee/service/tie/management/ca/get
/mcafee/service/tie/management/csr

5.2 Other bindings

Other known McAfee OpenDXL bindings are with MAR, ATD, and possibly others. We will not cover these components in detail.

6 Client side of the OpenDXL

In order to explore capabilities of OpenDXL and evaluating its suitability for accomplishing the goal (OpenDXL in active response) two client side solutions were implemented in proof of concept (POC) level:

- OpenDXL data bus monitoring and logging
- OpenDXL enabled MISP[17] adapter for bi-directional interaction with TIE

6.1 OpenDXL data bus logger

The OpenDXL the data bus logger, which can be deployed in Windows hosts without Python installation, was created in order to study interaction of components and better understand the data flows.

Logger can be compiled to Windows PE and project is available from https://github.com/zyxtarmo/OpenDXL-logger. User must generate client certificates and can build binary by following instructions in the README.md file.

Logger stores collected data in the 'logs' subfolder in the text files with name pattern 'YYYY-mm-dd.log'[18].

6.2 OpenDXL connected MISP

One of the publicly available implementations was connecting the MISP adapter with the McAfee ATD[19]. One of our goals would also be successful interaction with MISP via OpenDXL fabric.

Goals:

1. Check new hashes on arrival from MISP via MISP Python API
2. Lookup hashes from third parties when new indicators arrives in MISP. We will be grabbing MISP messages with topic misp_json from ZeroMQ for that.

Steps:

[17] Malware Information Sharing Platform, https://circl.lu/services/misp-malware-information-sharing-platform/
[18] YYYY is year with century, mm is month with leading zero and dd day with leading zero
[19] https://www.mcafee.com/in/products/advanced-threat-defense.aspx

1. Install and configure MISP (can be time consuming so use ready-made images/containers/recipes if available)
2. Install ZeroMQ libraries and redis, if you have not yet, and configure MISP to publish real-time data to ZeroMQ[20]
3. Install python bindings for MISP ([21])
4. Install proof of concept package of bidirectional communication available for download from https://github.com/zyxtarmo/OpenDXL-TIE-MISP

Proof of concept code with desired capabilities was created and can be downloaded from https://github.com/zyxtarmo/OpenDXL-MISP. Components:

1. opendxl2misp.py: OpenDXL subscriber, subscribes to channel ”X” with callback function to interact with the MISP
2. misp2opendxl.py: OpenDXL publisher, pushes the data from MISP to the topic ”Y”

Both components have been implemented as 'daemons' for Linux based systems, one must refer to included documentation if daemon must be run in the Windows based environment. Configuring the components is pretty simple, just follow the documentation in README and 'dxlclient.config'.

6.3 McAfee components and PowerShell

From existing examples to learn from there is small project involving endpoint components by McAfee endpoint technical specialist Colby Burkett. By itself it is a very nice example showing how easy it is to create specialized solution by binding together different components - in current case these are McAfee components - for getting new executables run by users from connected endpoint systems with help of PowerShell[22].

Code itself is available for download from https://github.com/ColbyBurkett/OpenDXL-new-exe-launch-pull.

6.4 MQTT orchestration tools for managing existing systems via OpenDXL

Since the OpenDXL is based on MQTT then quite logical option would be to use same tools as by IoT solutions for managing the workflow. IBMs Node-RED is quite popular tool for visual programming of the Internet connected devices and there is one project by Martin de Jongh from McAfee exploring this option. Such approach requires minimal programming effort and execution flows can be designed visually by someone who does not need to be expert in Python programming.

Proof of concept code is available from https://github.com/Mdejongh74/OpenDXL-NodeRed.

6.5 Firewall/IDS and NetFlow connection

Quite attractive option would be connecting IDS/IPS and firewall and MISP to OpenDXL data bus. In order to accomplish this as proof of concept we would require Python interfaces for all components:

- Netflow: https://pypi.python.org/pypi/pynfdump
- iptables: https://pypi.python.org/pypi/python-iptables
- MISP interface (see previous section)
- OpenDXL interface (see previous sections)

Proof of concept code was created and can be downloaded from https://github.com/zyxtarmo/OpenDXL-NET. Components:

[20] https://www.circl.lu/doc/misp/misp-zmq/, you have to do: 'pip install redis' as well since this is not in the linked documentation
[21] https://github.com/CIRCL/PyMISP
[22] https://msdn.microsoft.com/en-us/powershell/scripting/core-powershell/running-remote-commands

1. opendxl2netflow.py: OpenDXL subscriber, subscribes to channel "Z" with callback function to interact with the Netflow
2. opendxl2iptables.py: OpenDXL subscriber, subscribes to channel "W" with callback function to interact with the Netflow

Both components have been implemented as 'daemons' for Linux based systems, one must refer to included documentation if daemon must be run in the Windows based environment. Configuring the components is pretty simple, just follow the documentation in README and 'dxlclient.config'.

6.6 Designing execution flow with standard business solutions

Exploring possibilities of using standardized (ISO/IEC 19510:2013[23]) methodology (Business Process Modeling Notation) and tools (Business Process Execution Language) for describing vendor-neutral design of the execution flow so that future implementations involving machine learning (IMS learning design (IMS LD)) would be supported without modifications to the existing solution should be covered in separate research paper.

7 Achieving the goal

First iteration of research on current topic covers event based reactive response. Second iteration implements orchestrated response using previously built solutions, which differs fundamentally in approach on how the OpenDXL databus is used.

7.1 Event based reactive response

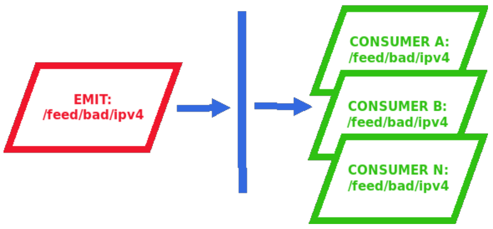
EMIT: /feed/bad/ipv4
CONSUMER A: /feed/bad/ipv4
CONSUMER B: /feed/bad/ipv4
CONSUMER N: /feed/bad/ipv4

Fig 2. Message flow

Emitting and consuming event messages with data type related hierarchical topics.

7.2 Orchestrated response

Referring to Enseys conditions to be met we'll go over all six conditions in the 'light of' OpenDXL.

Centralized event management requires direct link to SIEM or incident handling workflow orchestration tool to be established. Current OpenDXL bindings do not have such connector.

Analytics must be related to SIEM

Open APIs depend on devices and software, many exist, relatively easy to

Dynamic infrastructure. The Human element Complete visibility

Emitting and consuming event messages with vendor+product type hierarchical topics.
/mcafee/service/tie/cert/reputation/get
/mcafee/service/tie/cert/reputation/set
/mcafee/service/tie/file/reputation/get
/mcafee/service/tie/file/reputation/set
/mcafee/service/tie/file/url/reputation/add

8 Conclusions and future work

McAfee has opened many new options by open sourcing OpenDXL client libraries for the organizations using McAfee server side solutions, and possibly for others. These options can be taken advantage fo in various levels of complexity - it could either be quick hack and one time solution to burning issue or part of detailed and finely polished business process.

As a part of this small research paper properties of the OpenDXL were explored with proof of concept level implementation of simple active response workflow. Probably the biggest blocking point was lack

[23] https://www.iso.org/standard/62652.html

of proper documentation, and there are known limitations in the OpenDXL because of the MQTT as underlying protocol. This paper will most probably be extended with exploring linking existing orchestration platforms to already developed components.

9 References

1. Oasis open standard MQTT ver3.1.1, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

2. Collection of OpenDXL related software projects, https://github.com/opendxl

3. OpenDXL client configuration module, https://opendxl.github.io/opendxl-client-python/pydoc/dxlclient.client_config.html

4. ATD connection to MISP via OpenDXL, https://github.com/mohl1/OpenDXL-ATD-MISP

5. Towards a Common Graphical Language for Learning Flows: Transforming BPEL to IMS Learning Design Level A Representations, https://www.researchgate.net/publication/221423553_Towards_a_Common_Graphical_Language_for_Learning_Flows_Transforming_BPEL_to_IMS_Learning_Design_Level_A_Representations

6. Real-time bi-directional data sharing (bridging) between McAfee Data Exchange Layer and 3rd party data networks, https://github.com/Mdejongh74/OpenDXL-NodeRed

10 Link Collection

1. Threat Intelligence Exchange Getting Started Guide https://community.mcafee.com/docs/DOC-6454

2. OpenDXL-new-exe-launch-pull https://github.com/ColbyBurkett/OpenDXL-new-exe-launch-pull

3. automated threat intelligence collection with McAfee ATD, OpenDXL and MISP. https://github.com/zyxtarmo/Ope
ATD-MISP

4. PyMISP https://github.com/MISP/PyMISP/tree/master/examples

5. ePO broker cert export https://opendxl.github.io/opendxl-client-python/pydoc/epobrokercertsexport.html

6. ePO CA import https://opendxl.github.io/opendxl-client-python/pydoc/epocaimport.html

7. https://github.com/opendxl/opendxl-client-python

8. OpenDXL + NodeRED https://github.com/Mdejongh74/OpenDXL-NodeRed

9. MISP manual https://www.circl.lu/doc/misp/

10. Apache NiFi as an Orchestration Engine https://www.intersysconsulting.com/blog/apache-nifi-as-an-orchestration-engine/

11. Robots play Poker using Watson: Solution Architecture & Design https://www.ibm.com/blogs/watson/2016/09/robots-play-poker-using-watson-solution-architecture-design/

12. Apache ODE (Orchestration Director Engine) software executes business processes written following the WS-BPEL standard. https://ode.apache.org

13. Apache Synapse Enterprise Service Bus (ESB). http://synapse.apache.org