SOFWERX

Building a Zero Trust Network with Open Source and Community Version Software

Jim Ladd

May 13, 2020

Contents

Introduction

The traditional network security architecture partitions the realm into different networks and each network into zones. Each zone is contained by one or more firewalls that form a perimeter defense. Within each zone, a level of trust is granted and access to specific resources is permitted [1]. An alternative architectural approach that is gaining interest embraces a "Never trust, always verify" mantra. This philosophy does not automatically trust anything inside or outside the perimeter. The general term for this approach is called zero trust networks (ZTN) [2].

While zero trust networks are gaining interest in the commercial sector [3] and within the Department of Defense [4], there are currently no standards [5]. To acquire more knowledge about ZTNs, this paper describes a set of core characteristics of ZTNs. A cloud-based architecture is presented that implements these attributes using open source frameworks and the community version of commercial products.

Overview

The term "zero trust" was first mentioned by John Kindervag, an analyst at Forrester Research, in 2010 when the model was introduced [2] [6]. One of the more popular descriptions is given in a recent Gartner report that describes the zero trust network access (ZTNA) concept:

> The new model presents an approach in which a trust broker mediates connections between applications and users. ZTNA abstracts away and centralizes the security mechanisms so that the security engineers and staff can be responsible for them. ZTNA starts with a default deny posture of zero trust. It grants access based on identity, plus other attributes and context (such as time/date, geolocation and device posture), and adaptively offers the appropriate trust required at the time. The result is a more resilient environment with improved flexibility and better monitoring [7].

There are three subtle themes in this definition that deserve emphasis. All three should drive the specification and construction of a ZTN architecture. The first theme is that ZTNs begin from a *"posture of zero trust"* as the default behavior. This means that all forms of implied trust are invalid and organizations must rely on explicit and dynamic assessments from as many sources of data as possible before allowing a user to perform a transaction [8].

The second theme is that access is *"based on identity, plus other attributes and context (such as time/date, geolocation and device posture)".* Some sources consolidate the data that extends beyond a traditional "user" into a single object called agent or network agent [1] [9]. Other entities such as Google's BeyondCorp, a pioneer in ZTN, do not. BeyondCorp's initiative maintains user and device credentials as two separate but necessary and critical components of a ZTN solution [10]. When explaining the concepts of zero trust network, it may be helpful to include the agent model (i.e. user data + device data + other data at the instance of the request).

The final theme is *"adaptively offers the appropriate trust required at the time"*. To continuously deliver the appropriate access in a timely manner, the ZTN architecture must be able to quickly update its logic and, ideally, create new logic based on evolving conditions. A subset of the logic can be captured in a rule-based structure. These rules should be stored external to the engine that executes them and in a version control system with the following benefits:

- *Changes to policy can be tracked over time.*

- *Rationale for changing policy is tracked in the version control system.*
- *The expected current policy state can be validated against the actual enforcement mechanisms.* [1]

The ZTN architecture should also leverage machine learning technologies to increase adaptability by discovering validation logic that is difficult to be recognized by humans. Machine learning can identify risky user behavior by searching for contextual and suspicious patterns in the data [11].

Taking these three concepts further, a zero trust network (ZTN) should be built upon five fundamental assertions:

- *The network is always assumed to be hostile.*
- *External and internal threats exist on the network at all times.*
- *Network locality is not sufficient for deciding trust in a network.*
- *Every device, user, and network flow are authenticated and authorized.*
- *Policies must be dynamic and calculated from as many sources of data as possible.* [1]

In addition to the fundamental assertions, the ZTN architect should consider the following RFC-style priorities for building a solution:

- *All network flows MUST be authenticated before being processed.*
- *All network flows SHOULD be encrypted before being transmitted.*
- *Authentication and encryption MUST be performed by the endpoints in the network.*
- *All network flows MUST be enumerated so that access can be enforced by the system.*
- *The strongest authentication and encryption suites SHOULD be used within the network.*
- *Authentication SHOULD NOT rely on public PKI providers. Private PKI systems should be used instead.*
- *Devices SHOULD be regularly scanned, patched, and rotated.* [1]

The goal of this project is to design and construct a zero trust network using these priorities as a guide. The hope is to gain insights into the challenges and benefits of a working proof of concept.

Domain Analysis

The first step in the development process is to define the boundary for the system. The boundary defines what's inside and what's outside the realm of the solution. The boundary definition also identifies the interfaces that need to be refined throughout the development process. For this project, the ZTN is non-intrusive to the protected resource. The resource to be protected is a web service that provides a set of time-related services in a secure and no cost fashion. A diagram of the system boundary is shown below:

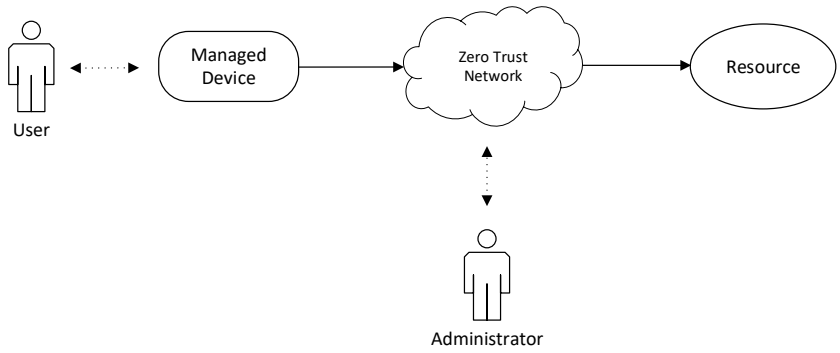

Figure 1 – System boundary

The major entities of our problem space include:

- ***User*** – The "current" user of the managed device. This person's credentials will be used in the attempts to gain access to the projected resource.
- ***Managed Device*** – This device is a computing platform such as a personal computer, tablet, smartphone, etc.
- ***Resource*** – The resource to be protected. Resources can be web sites, applications, web services, etc.
- ***Administrator*** – This role will have access to install and configure the different services of the ZTN.
- ***Zero Trust Network*** – This is the software that protects the resource based on the principles of zero trust networks.

The next step in the process is to decompose the problem space into the natural domains. A diagram of these domains is show below:

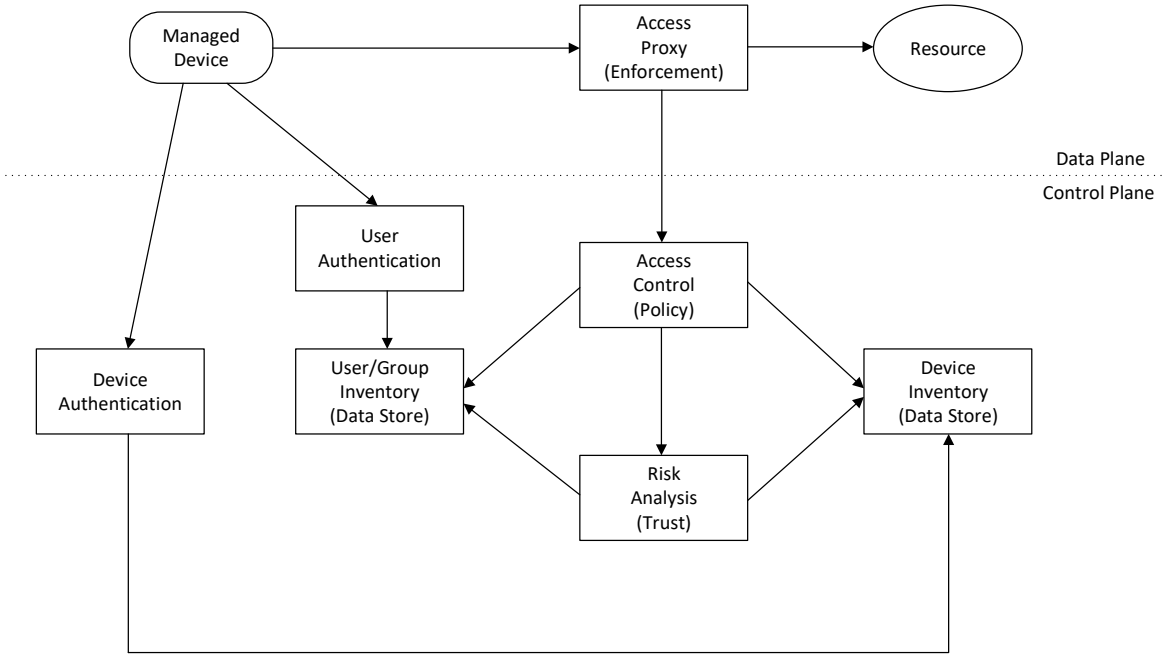

Figure 2 – Domains of a ZTN

The domains of the problem space include:

- ***Access Proxy*** – This domain is responsible for enforcing the decision of allowing access or not for the specified request. This service can be intrusive in that the code that implements the resource is modified or non-intrusive which requires no changes. For this project, the non-intrusive approach is taken.
- ***Access Control*** – This domain makes the final determination of whether access is granted or not by following security policies. The decision is based on various types of information from different sources. Ideally, the policies are dynamic and their implementations can be maintained via a version control system.
- ***Risk Analysis*** – The risk analysis domain performs risk analysis on specific requests. This domain may implement static rules in addition to rules that are dynamic and evolving based on user and device traffic patterns over time.
- ***User Authentication –*** This domain authenticates the user based on traditional methods that are approved by the enterprise.
- ***User/Group Inventory –*** The user/group inventory domain stores the user and group data along with their associations. This information is made available to the other domains for risk analysis and policy creation.
- ***Device Authentication –*** This domain authenticates the devices that are accessing resources in the network.
- ***Device Inventory –*** This data store maintains the information related to each managed device. This data is made available to the other domains for risk analysis and policy creation.

Architecture

Once the domain analysis is completed, the high level architecture of the solution space can be constructed. Each of the domains will be realized by either commercially available solutions, open source frameworks, custom code, or a combination of these options. The diagram of the high level architecture is shown below:

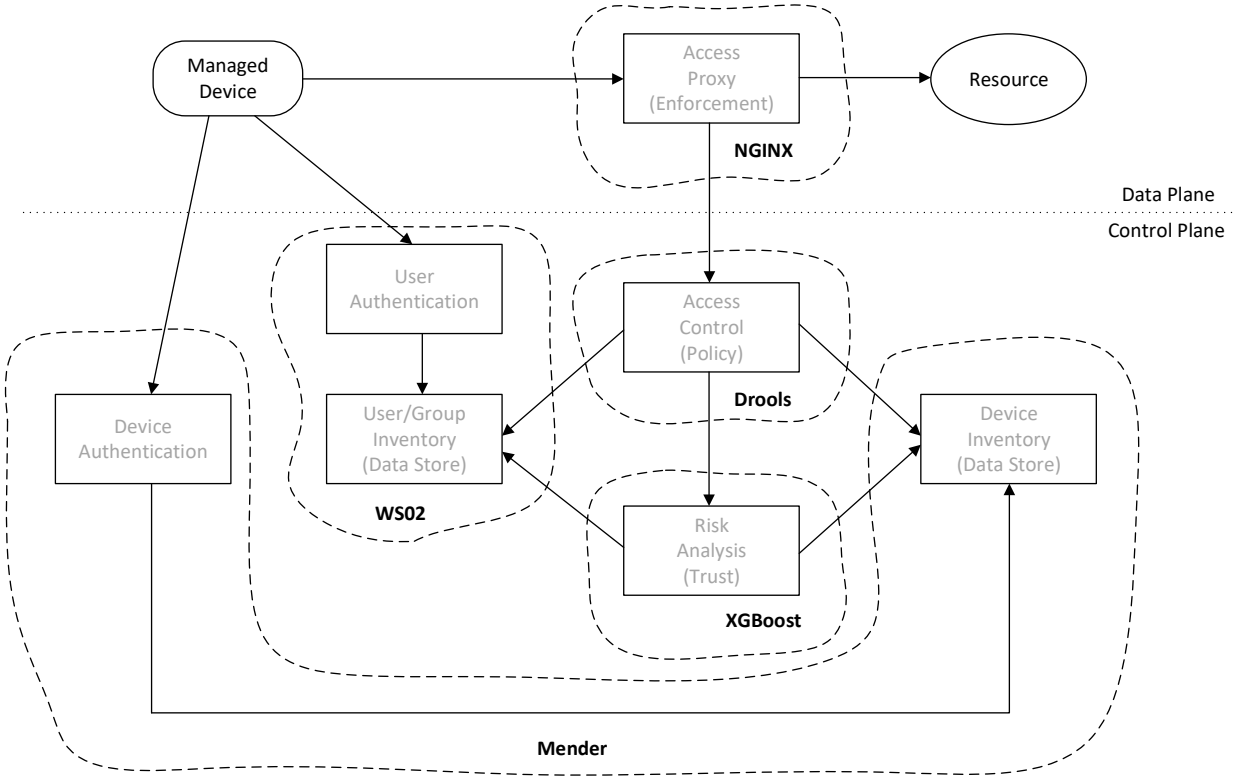

Figure 3 - High level architecture

The major components of the high level architecture:

- ***Managed Device –*** For this project, a Raspberry Pi single board computer running the Raspbian operating system will be used as the managed device.
- ***Resource –*** The web service for the World Time API (http://worldtimeapi.org/api/ip) is used as the unprotected resource. Access to this web service will be guarded by the ZTN.
- ***NGINX*** – This open source software is a proxy engine that intercepts the requests from the managed device and queries the Access Control domain to allow access or not. NGINX provides the enforcement of the Access Proxy domain.
- ***Drools –*** Drools is an open source Business Rules Management System (BPMS) that provides web authoring and runtime execution of rules. This software provides the policy capture and execution for the Access Control domain.
- ***XGBoost –*** This software is an optimized gradient boosting library that implements machine learning algorithms. It performs the trust determination for the Risk Analysis domain.

- ***Mender –*** This software provides system level and application updates to target platforms. For this project, it will be used for the Device Authentication and Device Inventory domains.
- ***WS02 –*** This software provides central authentication and authorization for web and mobile applications. For this project, WSO2 is responsible for the User Authentication and User/Group Inventory domains.

One of the early architecture decisions was to use web services technologies. All of the major components are implemented as micro-services with TLS interfaces. Another decision was to host the services on Amazon Web Services (AWS). With the exception of the Raspberry Pi acting as the Managed Device and the target system (worldtimeapi.org), all of the system components were hosted on AWS. A diagram of the detailed architecture is shown below:

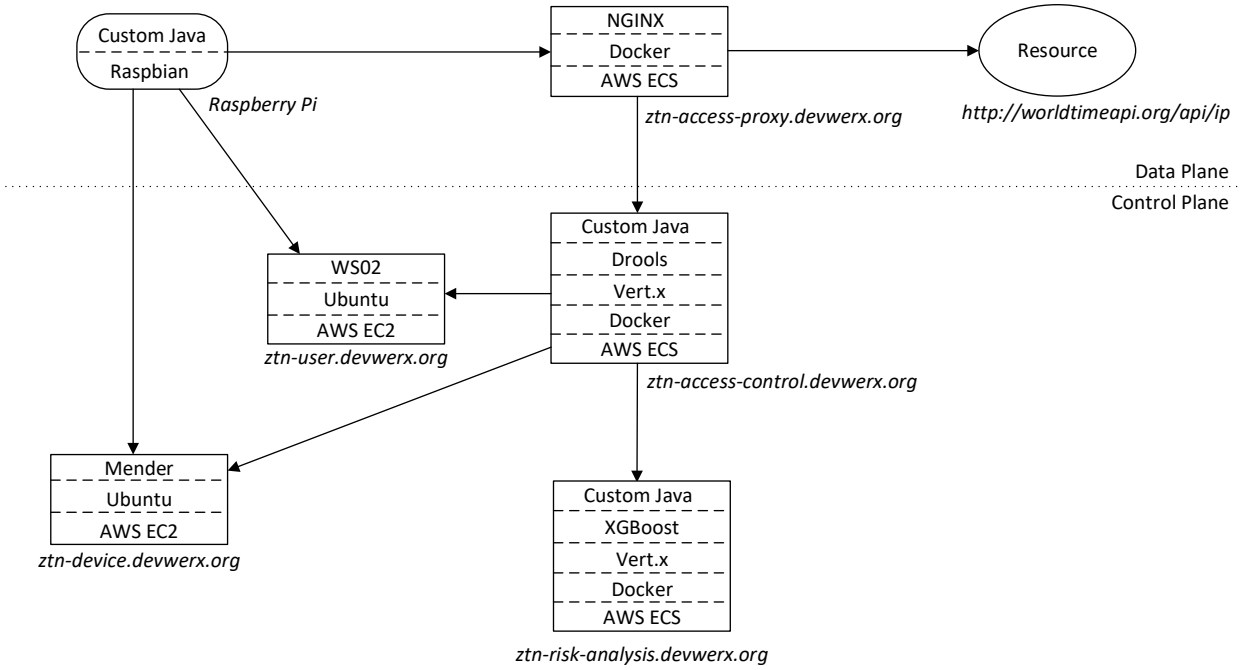

Figure 4 - Detailed architecture

The remaining components of the detailed architecture include:

- ***Raspbian –*** This is the operating system used on the Raspberry Pi platform.
- ***Docker –*** This is a container technology that allows software to be easily deployed and scaled.
- ***AWS ECS –*** The Elastic Container Service is an Amazon service that allows Docker images to be executed with little configuration.
- ***Ubuntu –*** This is a popular version of Linux and is one of the operating systems available on AWS's computing platform.
- ***AWS EC2 –*** The Elastic Compute Cloud is one of Amazon's services for virtual machines.
- ***Vert.x*** – This is a high performance micro-service framework provided by the Eclipse open source project.

Now that all of the services have been identified and described, the next step is to present how their coordinated interactions achieve the desired behavior of the system. The following is an event sequence diagram that shows the synchronous, secure requests between the different micro-services.

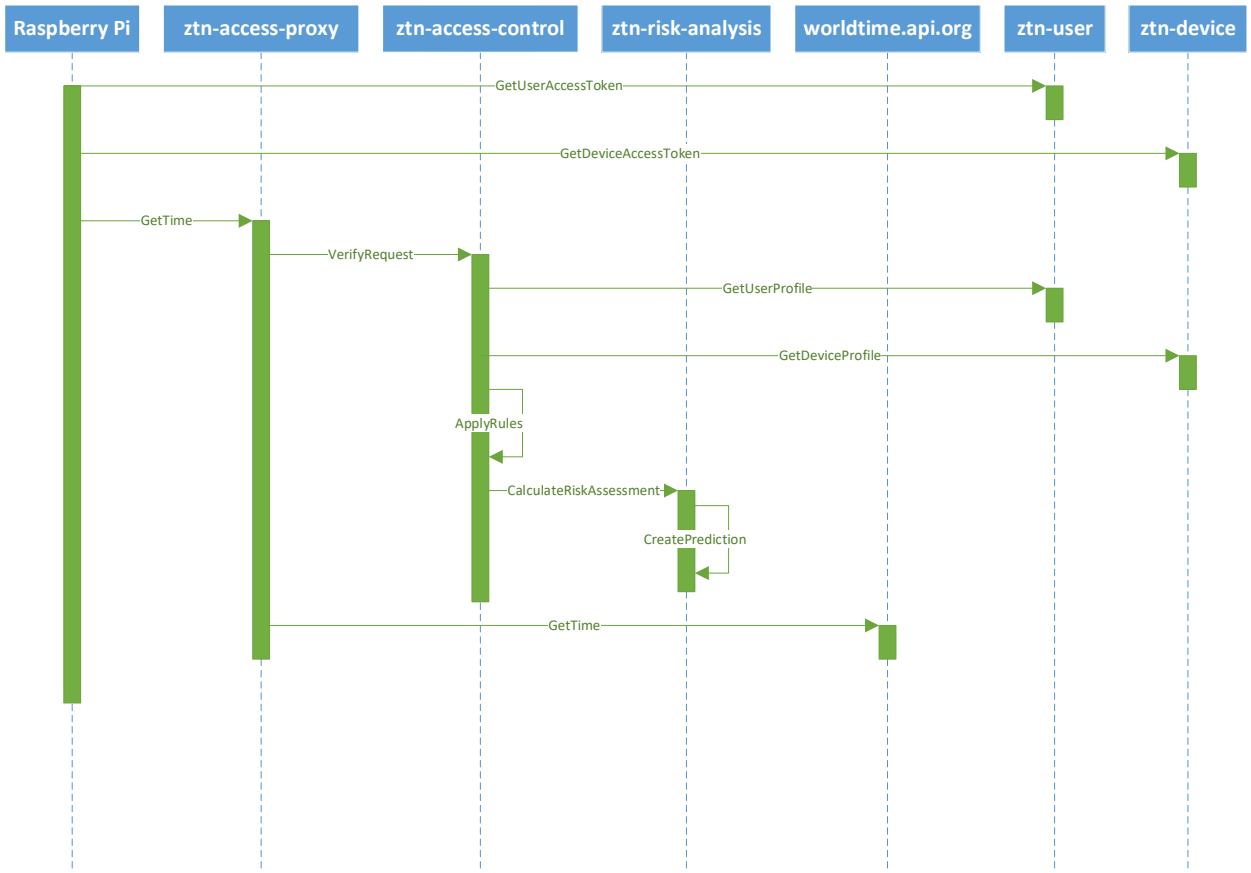

Figure 5 – Event sequence

The process of a successful request by the client consists of the following steps:

1. The custom Java client program on the ***Raspberry Pi*** uses the OAuth2 protocol to request an access token for the user data from the ***ztn-user*** service.
2. The custom Java program then uses the OAuth2 protocol to request an access token for the device data from the ***ztn-device*** service.
3. The custom Java client program sends a request for the time along with the two access tokens to the ***ztn-access-proxy*** service.
4. The ***ztn-access-proxy*** service receives the request and forwards it to the ***ztn-access-control*** service.
5. The ***ztn-access-control*** service uses the user access token to retrieve the user profile data from the ***ztn-user*** service.
6. The ***ztn-access-control*** service uses the device access token to retrieve the device profile data from the ***ztn-device*** service.

7. The ***ztn-access-control*** service applies the Drools rules to the data. If any of the rules return a false result, a "Not Authorize" status code is returned to the ***ztn-access-proxy*** service. If all of the rules return a true result, the processing continues to the next step.
8. The ***ztn-access-control*** service sends a request along with the user and device profiles to the ***ztn-risk-analysis*** service.
9. The ***ztn-risk-analysis*** service applies the machine learning algorithm to the profiles and calculates a risk prediction that is in the range between 0-100. This value is returned in the response message.
10. The ***ztn-access-control*** service applies another rule to the prediction value. If the prediction is 75 or greater, a success status code is returned. If it is less than 75, a failure status code is returned.
11. The ***ztn-access-proxy*** service examines the returned status code and, if approved, forwards the request to the ***worldtime.api.org*** service. If the request was not approved, a "not authorized" status code is returned to the originator of the request (i.e. the custom Java code on the ***Raspberry Pi***).
12. The ***worldtime.api.org*** service receives the request and returns a response with the JSON-based time data.
13. The ***ztn-access-proxy*** service receives the response and returns it to the custom Java program on the ***Raspberry Pi***.

User and Device Data

A data driven system is only as effective as the data it has access to. For this version of a zero trust network, the depth and breath of the data is limited to 1) the user and device inventory systems (i.e. WSO2 and Mender respectively) and 2) the dimension of time. Other candidate dimensions include geolocations of the requests, sequence patterns of the requests, concurrency of requests from other managed devices authorized with the user.

The user data that is maintained in the WS02 system is accessible via an OAuth2-based interface. This information is represented in a JSON-based structure. An example of a data record for a specific user is shown below.


```
{
    "emails": [
        "jim.ladd@sofwerx.org"
    ],
    "profileUrl": "www.sofwerx.org",
    "meta": {
        "created": "2020-01-21T17:34:41.151Z",
        "location": "https://localhost:9443/scim2/Users/f8c79fc3-10b4-
4ce3-83fa-c8f8fece5e2f",
        "lastModified": "2020-01-23T21:00:40.069Z",
        "version": "jladd",
        "resourceType": "User"
    },
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "roles": [
        {
```

```
            "type": "default",
            "value": "ViewMe,Internal/everyone"
        }
    ],
    "name": {
        "givenName": "Jim",
        "familyName": "Ladd"
    },
    "groups": [
        {
            "display": "ViewMe",
            "value": "a2c4013d-6157-4d04-8c1a-d6b12dcf5b29",
            "$ref": "https://localhost:9443/scim2/Groups/a2c4013d-6157-
4d04-8c1a-d6b12dcf5b29"
        }
    ],
    "id": "f8c79fc3-10b4-4ce3-83fa-c8f8fece5e2f",
    "userName": "jladd",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "organization": "Sofwerx"
    },
    "phoneNumbers": [
        {
            "type": "mobile",
            "value": "5551112222"
        }
    ]
}
```


Device data is maintained in the Mender system and is accessible via an OAuth2-based interface. This information is represented in a JSON-based structure. An example of a data record for a specific device is shown below.


```
{
  "id": "5e00d50b91e094000149b486",
  "attributes": [
    {
      "name": "device_type",
      "value": "raspberrypi3"
    },
    {
      "name": "os",
      "value": "Raspbian GNU/Linux 9 (stretch)"
    },
    {
      "name": "ipv4_wlan0",
      "value": "192.168.12.140/22"
    },
    {
      "name": "artifact_name",
      "value": "mender-demo-artifact-2.2.1"
    },
    {
```

```
      "name": "mender_bootloader_integration",
      "value": "unknown"
    },
    {
      "name": "hostname",
      "value": "raspberrypi"
    },
    {
      "name": "mac_wlan0",
      "value": "b8:27:eb:5a:af:8a"
    },
    {
      "name": "cpu_model",
      "value": "ARMv7 Processor rev 4 (v7l)"
    },
    {
      "name": "rootfs_type",
      "value": "ext4"
    },
    {
      "name": "kernel",
      "value": "Linux version 4.14.98-v7+ (dom@dom-XPS-13-9370) (gcc
version 4.9.3 (crosstool-NG crosstool-ng-1.22.0-88-g8460611)) #1200 SMP
Tue Feb 12 20:27:48 GMT 2019"
    },
    {
      "name": "mem_total_kB",
      "value": "949448"
    },
    {
      "name": "mac_eth0",
      "value": "b8:27:eb:0f:fa:df"
    },
    {
      "name": "network_interfaces",
      "value": [
        "eth0",
        "wlan0"
      ]
    },
    {
      "name": "ipv6_wlan0",
      "value": "fe80::e3c1:6907:f87d:2a78/64"
    },
    {
      "name": "mender_client_version",
      "value": "2.1.2"
    }
  ],
  "updated_ts": "2019-12-23T14:58:32.794Z"
}
```

Rules Engine

Network security systems are under a constant barrage of changing threats and attacks. To be continuously successful, they need to evolve and adapt with minimal time and effort expenditures. Architectures of robust systems often rely on rules-based technologies to concentrate the desire logic into granular rule definitions that can be created, tested, maintained, and deployed independently. Ideally, these rules are processed by a high performance engine. For this version of a zero trust network, the Drools framework was selected.

Drools is a business rule management system (BRMS) with a forward and backward chaining inference based rules engine. The desired logic is captured in "if-then" statements called rules. A rudimentary system would check each rule, firing that rule if necessary, then moving on to the next rule (and looping back to the first rule when finished). For even moderate sized rules, this approach performs far too slow. To increase performance, the Drools engine is based on the Rete algorithm first made public in 1974 [12].

A Rete-based system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The Rete algorithm is designed to sacrifice memory for increased speed. In most cases, the speed increase over basic implementations is several orders of magnitude [13].

Examples of the Drools rules used in this project include the following three. The first rule checks if the user is a member of a specific group:

```
package org.sofwerx.ztn.access_control;

import org.sofwerx.ztn.model.user.UserProfile;

rule "Is Group Valid"
when
  $up : UserProfile( hasGroupWithDisplay( "ViewMe") )
then
  $up.setValidGroup(true);
End
```

The second rule checks if the URL in the user profile has a specific value:

```
package org.sofwerx.ztn.access_control;

import org.sofwerx.ztn.model.user.UserProfile;

rule "Is Profile Url Valid"
when
  $up : UserProfile( getProfileUrl() == "www.sofwerx.org")
then
  $up.setValidProfileUrl(true);
End
```

The last rule checks if the Ethernet MAC value equals a specific value:

```
package org.sofwerx.ztn.access_control;

import org.sofwerx.ztn.model.device.DeviceProfile;

rule "Is Device Ethernet MAC Valid"
when
  $dp : DeviceProfile( isAttributeMatch( "mac_eth0", "b8:27:eb:0f:fa:df" )
)
then
  $dp.setValidEthernetMAC(true);
End
```

These rules have a syntax similar to Java and are written in a text file. These files can be placed under version control. They are deployed by simply copying them to a specific path on the host environment where they are accessed by the rules engine. Likewise, they can be decommissioned by removing them from the path on the host.

Machine Learning

In addition to the explicit rules of the Drools engine, a zero trust network can benefit from the use of other technologies that discover other verification criteria. By leveraging types of technologies like machine learning, the ZTN can incorporate filtering patterns before humans have recognized and codified them. For this project, the XGBoost framework was selected to provide verification criteria discovery.

The XGBoost (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. This algorithm was developed for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. The XGBoost framework is also used in AWS's SageMaker service [14].

Like many artificial intelligent techniques, XGBoost requires a set of training data to build a model. This model is then applied to other data sets in order to achieve a result. When using machine learning with security networks, system monitoring, etc., most companies capture log files from the systems under observation [15]. From these log files, a subset is used for training and building the model. Our project didn't have access to valid and diverse log files. Since the goal of the project was to build an archetype of a ZTN architecture and not to generate models for production use, we pressed the easy button and built our own training set.

The hand built training file consisted, among other data fields, the time that the inbound request was recorded. The data was constructed so that even minutes resulted in a successful request and odd minutes resulted in a failed request. A model was built on this training data and used in the demonstration.

Demonstration

To demonstrate the ZTN proof of concept, a Java program was constructed and deployed on the Raspberry Pi that was acting as the Managed Device. This Java client retrieves the device access token

from the Mender system, the user access token from the WSO2 system and then sends a request to the access proxy along with the tokens. It receives a response that has 1) a HTTP status code of 200 (OK) and the JSON data of the date and time or 2) a HTTP status code of 401 (Unauthorized). The code will then wait 20 seconds before sending another request. This process will repeat for a total of 10 requests.

As expected, the request would be successful during even minutes and would be rejected during odd minutes. The log statements from the Java client is shown below:

```
04/21/2020 15:02:31.883 [main] DEBUG - Sending request...
04/21/2020 15:02:32.628 [main] DEBUG - Response Code (Apache): 200
04/21/2020 15:02:32.631 [main] DEBUG - Response body =
{"week_number":17,"utc_offset":"-04:00","utc_datetime":"2020-04-
21T19:02:32.565205+00:00","unixtime":1587495752,"timezone":"America/New_York"
,"raw_offset":-18000,"dst_until":"2020-11-
01T06:00:00+00:00","dst_offset":3600,"dst_from":"2020-03-
08T07:00:00+00:00","dst":true,"day_of_year":112,"day_of_week":2,"datetime":"2
020-04-21T15:02:32.565205-
04:00","client_ip":"52.55.105.58","abbreviation":"EDT"}
04/21/2020 15:02:32.632 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:02:53.103 [main] DEBUG - Sending request...
04/21/2020 15:02:53.633 [main] DEBUG - Response Code (Apache): 200
04/21/2020 15:02:53.634 [main] DEBUG - Response body =
{"week_number":17,"utc_offset":"-04:00","utc_datetime":"2020-04-
21T19:02:53.580681+00:00","unixtime":1587495773,"timezone":"America/New_York"
,"raw_offset":-18000,"dst_until":"2020-11-
01T06:00:00+00:00","dst_offset":3600,"dst_from":"2020-03-
08T07:00:00+00:00","dst":true,"day_of_year":112,"day_of_week":2,"datetime":"2
020-04-21T15:02:53.580681-
04:00","client_ip":"52.55.105.58","abbreviation":"EDT"}
04/21/2020 15:02:53.634 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:03:14.366 [main] DEBUG - Sending request...
04/21/2020 15:03:14.707 [main] DEBUG - Response Code (Apache): 401
04/21/2020 15:03:14.707 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:03:35.139 [main] DEBUG - Sending request...
04/21/2020 15:03:35.503 [main] DEBUG - Response Code (Apache): 401
04/21/2020 15:03:35.504 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:03:55.923 [main] DEBUG - Sending request...
04/21/2020 15:03:56.227 [main] DEBUG - Response Code (Apache): 401
04/21/2020 15:03:56.227 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:04:16.659 [main] DEBUG - Sending request...
04/21/2020 15:04:17.135 [main] DEBUG - Response Code (Apache): 200
04/21/2020 15:04:17.136 [main] DEBUG - Response body =
{"week_number":17,"utc_offset":"-04:00","utc_datetime":"2020-04-
21T19:04:17.082404+00:00","unixtime":1587495857,"timezone":"America/New_York"
,"raw_offset":-18000,"dst_until":"2020-11-
01T06:00:00+00:00","dst_offset":3600,"dst_from":"2020-03-
08T07:00:00+00:00","dst":true,"day_of_year":112,"day_of_week":2,"datetime":"2
```

```
020-04-21T15:04:17.082404-
04:00","client_ip":"52.55.105.58","abbreviation":"EDT"}
04/21/2020 15:04:17.136 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:04:37.564 [main] DEBUG - Sending request...
04/21/2020 15:04:38.658 [main] DEBUG - Response Code (Apache): 200
04/21/2020 15:04:38.659 [main] DEBUG - Response body =
{"week_number":17,"utc_offset":"-04:00","utc_datetime":"2020-04-
21T19:04:38.592313+00:00","unixtime":1587495878,"timezone":"America/New_York"
,"raw_offset":-18000,"dst_until":"2020-11-
01T06:00:00+00:00","dst_offset":3600,"dst_from":"2020-03-
08T07:00:00+00:00","dst":true,"day_of_year":112,"day_of_week":2,"datetime":"2
020-04-21T15:04:38.592313-
04:00","client_ip":"72.185.253.160","abbreviation":"EDT"}
04/21/2020 15:04:38.659 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:04:59.094 [main] DEBUG - Sending request...
04/21/2020 15:04:59.652 [main] DEBUG - Response Code (Apache): 200
04/21/2020 15:04:59.652 [main] DEBUG - Response body =
{"week_number":17,"utc_offset":"-04:00","utc_datetime":"2020-04-
21T19:04:59.598582+00:00","unixtime":1587495899,"timezone":"America/New_York"
,"raw_offset":-18000,"dst_until":"2020-11-
01T06:00:00+00:00","dst_offset":3600,"dst_from":"2020-03-
08T07:00:00+00:00","dst":true,"day_of_year":112,"day_of_week":2,"datetime":"2
020-04-21T15:04:59.598582-
04:00","client_ip":"52.55.105.58","abbreviation":"EDT"}
04/21/2020 15:04:59.652 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:05:20.156 [main] DEBUG - Sending request...
04/21/2020 15:05:20.583 [main] DEBUG - Response Code (Apache): 401
04/21/2020 15:05:20.583 [main] DEBUG - Waiting for 20 seconds.

04/21/2020 15:05:41.184 [main] DEBUG - Sending request...
04/21/2020 15:05:41.501 [main] DEBUG - Response Code (Apache): 401
04/21/2020 15:05:41.502 [main] DEBUG - Waiting for 20 seconds.

Process finished with exit code 0
```

Looking Back and Moving Forward

One of the insights gleaned from this effort was that the selection of any Common Off The Shelf (COTS) package is critical. Any system that is a candidate for a role in a ZTN architecture must be 1) secure in its own right and 2) have a secure API to access its internal data. The first package chosen for the user authentication domain was difficult to install and was built on a complex architecture. The internal data was not readily accessible. After much frustration, an alternative was chosen (i.e. WSO2) which proved to be much easy to integrate into the architecture.

One area of improvement that was ***not*** unexpected was performance. To paraphrase an old car racing saying, "Security is expensive, how secure can you afford?". Injecting any additional processing into a transaction will increase the execution time. The development of this project followed the manta of "First, make it work. Second, make it fast. Third, make it pretty." Since the scope was limited to the initial proof of concept (PoC), only the first step was taken. Now that the PoC is working, performance

tuning and refactoring can be considered. The first step in THAT process should be answering the question "How fast is fast enough?".

User and device related data are critical to the ZTN paradigm. In this project, the user data was retrieved from the User Inventory domain (i.e. WS02) and maintained in its native format. Likewise, the device data was extracted from the Device Inventory domain (i.e. Mender) and kept in its native format. An area of improvement is to create a canonical model of the data, possibly a Network Agent object, that is independent of the source systems. The data would be translated from native format to the canonical structure when retrieved from the source system. The rules within the Access Control domain could then be constructed to operate on the canonical data resulting in a more robust solution.

Currently there is no standard for a ZTN architecture. However, a proposal is in draft mode [5]. A potential future project is to compare the architecture presented here with the coming standard and to close any gaps.

While the solution described in this paper was deployed on the Amazon Web Services (AWS) platform, none of the AWS specific security features were utilized. This decision was intentional. The desire to have a vendor independent solution took priority over developing the very best solution. Additional effort can be spent incorporating AWS (or other cloud vendors) network security features like Virtual Private Cloud and Security Groups into the overall architecture.

One of the newer paradigms in software development is serverless cloud computing. In this model, containers such as Docker and virtual machines like Elastic Compute Cloud (EC2) are no longer required. Source code is uploaded to a cloud service and is executed when needed. The development is simplified, time to market is reduced, and operating costs are reduced. Research at the time of writing found no ZTN that was implemented on a serverless platform. This seems like a very interesting direction for future work.

Summary

While the core ZTN technology has been proven over the last few years, it is just now being considered by most enterprises [3]. The objective of this project was to learn about Zero Trust Networks and to do so by building a working proof of concept with the appropriate technologies. Several key characteristics and features for ZTN architectures were identified and pursued. An architecture was conceived that integrated both new and proven technologies into a robust and scalable solution. Lastly, additional research opportunities to further the understanding and progress of ZTNs were discussed.

Acknowledgements

I wish to thank the following team members who contributed to this project:

Kaitlyn Bub – Engineer at Space and Naval Warfare Systems Center

Robinson Simon – Intern at SOFWERX

Jason Young – Senior Software Engineer at SOFWERX

References

[1] E. Gilman and B. Doug, Zero Trust Networks, O'Reilly Media, 2017.

[2] paloalto networks, "What is Zero Trust?," [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture. [Accessed May 2020].

[3] Cybersecurity Insiders, "2020 Zero Trust Progress Report," Pulse Secure, 2020.

[4] K. DelBene, M. Medin and R. Murray, "The Road to Zero Trust (Security)," Defense Innovation Board, a Federal Advisory Committe, 2019.

[5] S. Rose, O. Borchert, S. Mitchell and S. Connelly, "Zero Trust Architecture (2nd Draft)," National Institute of Standards and Technology, Gaithersburg, 2020.

[6] CloudFlare, "Zero Trust Security | What's a Zero Trust Network?," [Online]. Available: https://www.cloudflare.com/learning/security/glossary/what-is-zero-trust/. [Accessed May 2020].

[7] S. Riley, N. MacDonald and L. Orans, "Market Guide for Zero Trust Network Access," Gartner, 2019.

[8] Ping Identity, "What Is Zero Trust?," 5 June 2019. [Online]. Available: https://www.pingidentity.com/en/company/blog/posts/2019/what-is-zero-trust.html.

[9] Fyde, "Fyde: Zero Trust architecture and components," [Online]. Available: https://www.fyde.com/resources/fyde-zero-trust-architecture-and-components. [Accessed May 2020].

[10] R. Ward and B. Beyer, "Beyond Corp - A New Approach to Enterprise Security," December 2014. [Online]. Available: www.usenix.org.

[11] L. Columbus, "Three Ways Machine Learning Is Revolutionizing Zero Trust Security," May 2018. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2018/05/11/three-ways-machine-learning-is-revolutionizing-zero-trust-security/#12fb469f81ed.

[12] JBoss Community Documentation, "Hybrid Reasoning," [Online]. Available: https://docs.jboss.org/drools/release/6.4.0.Final/drools-docs/html/ch05.html. [Accessed May 2020].

[13] Sandia National Laboratories, "The Rete Algorithm," November 2008. [Online]. Available: https://jess.sandia.gov/docs/71/rete.html.

[14] Amazon Web Service, 2020. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html.

[15] G. Hill, "AI, machine learning and your access network," Network World, 20 February 2018. [Online]. Available: https://www.networkworld.com/article/3256013/ai-machine-learning-and-your-access-network.html.