Automatic CIP Security via Pull Policy

Joakim Wiberg
Head of Technology
HMS Networks

David Smith
Cybersecurity Architect
Schneider Electric

Jack Visoky
Principal Engineer and Security Architect
Rockwell Automation

Presented at the ODVA
2023 Industry Conference & 22nd Annual Meeting
October 18, 2023
El Vendrell, Spain

Abstract

The CIP Security Pull Model Profile provides a major benefit of allowing a device to automatically discover a certificate enrolment server and request a certificate for secure communication. However, secure communication with CIP Security requires additional configuration beyond just the certificate. Additional value could be realized by defining a mechanism for a device to request not just a certificate, but also the associated configuration for enabling CIP Security. This configuration includes things like allowed cipher suites, trust anchors, certificate revocation lists, etc. One benefit of this ability would be the seamless application of device replacement, where a replaced device could automatically discover a security configuration server and request all of the configuration needed for CIP Security. Furthermore, this would also enable devices to work in network architectures where a configuration tool could not reach the device, like a NAT with the device on the private network and the configuration tool on the public network. This paper will explore use cases and requirements for a feature such as this, as well as potential technology choices.

Keywords

IIoT, Cybersecurity, CIP Security, Policy Configuration, Networking, Automatic Configuration

Introduction

CIP Security provides important information assurance properties that are needed to mitigate threats in many industrial applications using EtherNet/IP. However, prior to using CIP Security there must be a configuration step that takes place. The CIP specification has defined a common way in which this configuration is delivered to a device via CIP objects and services. This works well in many cases,

although there are some limitations. This is explored further in the use cases section, but briefly there are at least three situations where delivering CIP Security configuration via CIP objects and services presents a limitation:

- Network architectures where routing is not allowed like Network Address Translation (NAT)
- Software that only has CIP client capabilities and no server functionality
- Fully automatic device replacement with CIP Security enabled

In order to serve these use cases (and potentially others) a new mechanism for delivering CIP Security configuration is discussed here. Requirements for this mechanism are discussed in this paper, but essentially it needs to be able to serve CIP Security configuration via a document that is requested by a client. In this context, the term “document” refers to a packaged data format that exists in one coherent file and can be transmitted to the consumer independent of the transport mechanism. Of course, this scheme must provide authenticity assurances and be resistant to cyber-attacks.

Besides discussing use cases and requirements, this paper will also investigate potential technologies to realize this scheme for delivery of CIP Security configuration via a document format. A few technologies are weighed against how well they meet the requirements, and a recommendation for a technology to use is given, as well as recommended future work.

Use Cases

A consistent security configuration is required for devices to operate in a secure manner. Due to the need for clients and servers to have the same configuration, a mechanism for the clients and servers to request, or “pull” security configuration is useful to distribute this security configuration across multiple use cases that may even overlap. The existing Push Model is useful in that a centralized tool can configure all the devices in the network, but it fails to accommodate some important use cases, which are explored below. Note that the Push Model as currently defined can provide a certificate as well as the necessary CIP Security configuration via CIP services and attributes. However, the current Pull Model only provides a certificate; therefore, this document format is necessary to provide parity of Push and Pull models.

Deliver security config to private networks, e.g. through a NAT
In this use case, the PLC opens a client connection (not a CIP connection, rather some TCP or similar session) to the OT Configuration server and pulls its security configuration periodically. The routes through the router and the firewall can be simply configured and monitored.

Ethernet LAN Diagram

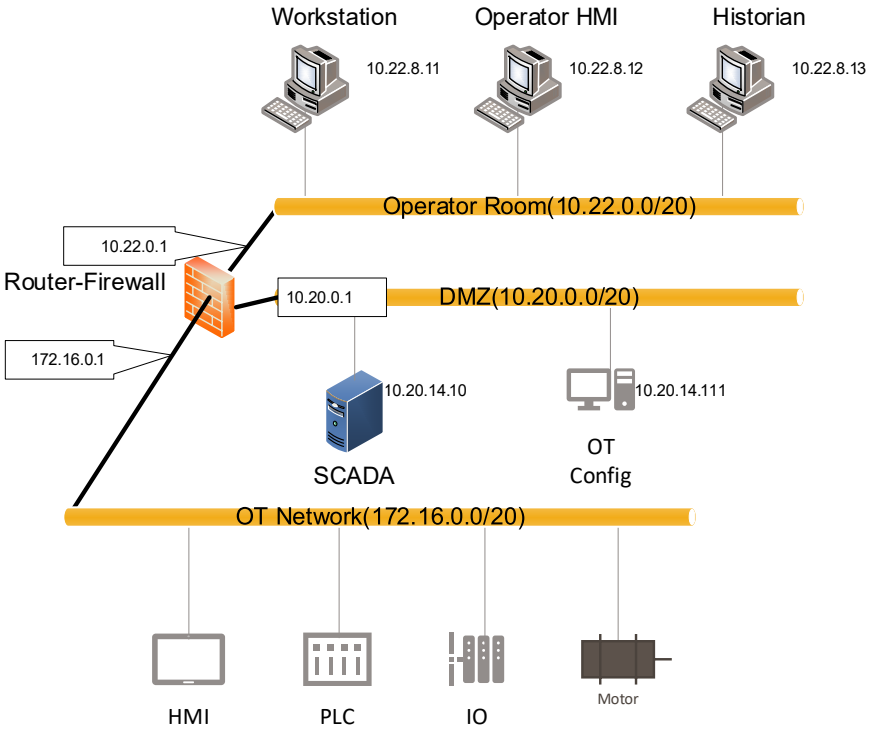

Figure 1

Network topologies that enforce network segregation through routers with Network Address Translation (NAT) make it difficult for Push Model connections to be made through the router. In Figure 1, any devices that connect to the DMZ will have a translated IP address of 10.20.14.1. The DMZ will not be allowed to make a connection through the router to the OT Network unless pre-configured routes are used. This requires substantial configuration and continuous maintenance as devices in the segregated network are added and removed. Preconfigured routes also weaken the security posture of the segregated network giving attackers a well-defined path to gain access to the network.

Deliver security config to "client-only" software

SCADA, HMI, Engineering, Configuration and Management tools, are often client only functionality. Unlike OT devices such as PLCs and IO, they have no ability to accommodate a Push Model for configuration. However, there is still a need for coherent security configuration for clients to be aligned with servers in the system. Much like the first use case where a client connection can be easily configured through routers and firewalls, the Pull Model provides simpler system configuration while ensuring a high level of security posture.

Fully automatic device replacement

In one last use case regarding device replacement, allowing a device being replaced to retrieve its own configuration is a straightforward way to facilitate device replacement. The specifics of how a device may come to be trusted on the network are outside the scope of this paper, but a simplified configuration to enable that process can be accomplished with a Pull Model. As an example, an electric motor driver is replaced. The motor driver searches for a well-known address using mDNS (multicast DNS packets to discover a service) or DNS-SD (a specific protocol for discovering network services) in order to find the OT Configuration service address. After the address is resolved, the motor driver can then pull its initial security configuration from the OT Configuration service. This configuration may be enough to allow the full bootstrapping of trust of the new motor driver without manual intervention.

Requirements

This section outlines some of the crucial requirements ensuring the effective delivery of a robust security configuration. Here, a diverse set of requirements is outlined that encompass various aspects, including technical specifications and practical use case considerations. By addressing these crucial prerequisites, a solid foundation can be established for implementing a reliable and tailored security configuration.

Document format

A comprehensive document format is essential for managing and communicating critical security parameters from the configuration server. This document format enables the inclusion of crucial information such as allowed cipher suites, trust anchors, and certificate revocation list together with the communication certificate.

Integrity and consistency are vital aspects when it comes to delivering the security related configuration. Bundling all the configuration options in one single document maintains the reliability and trustworthiness of the information. Both these attributes ensure that the content of the document remains accurate and unaltered. This allows for all of the configuration to be applied atomically, as it can be parsed and interpreted by the device and then applied at once. Furthermore, the document format abstracts the transport of configuration from the configuration itself, meaning that this could be delivered over other transports (e.g. MQTT) if that was necessary or beneficial in some use cases.

A well-defined document format containing allowed cipher suites, trust anchors, and certificate revocation lists, delivered along with the device’s identity certificate establishes a robust security configuration.

Authenticity

The authenticity of a document confirms its genuine origin and ensures that it has not been tampered with. It ensures that the document can be trusted as an accurate representation of its contents, i.e. the full security configuration delivered from the server.

The validation of the authenticity of a document is crucial and something that can be done using different mechanisms. In all cases there needs to be some provisioning done before the validation of the authenticity takes place. One common approach is the use of digital signatures. Digital signatures utilize cryptographic techniques to bind a unique identifier to the document. By verifying the digital signature, the device can authenticate the document and trust that it came from the expected sender and was not modified by an unauthorized party. However, in this case the device would need to be pre-provisioned with a key from the server generating the document.

In all cases the pre-provisioning requires that some trust provisioning with the device takes place before the authenticity of the document can be verified. Once the pre-provisioning of trust occurs this could allow for a fully automatic and seamless device replacement. To establish the initial trust, the process of a trust

on first use (TOFU) approach would be required. Upon the first attempt to connect, to a configuration server the device is initially provisioned with cryptographic keys. These keys then facilitate validation of the authenticity of the signed configuration document, providing the replaced device with security configuration.

Confidentiality

In general, the security configuration isn't secret and doesn't need to be protected by encryption. However, the pre-shared keys are one attribute within the security configuration that needs to be protected and kept secret. Installations that use pre-shared keys are normally smaller installations with a limited number of nodes. In those cases, it's less likely that the Pull Model will be used, as a centralized configuration server may not be present.

If pre-shared keys are to be delivered via the Pull Model, at a minimum they need to be encrypted in some way. Alternatively, the whole document could be encrypted. This would protect the pre-shared keys in the case when they are used. Since only the pre-shared keys need to be protected and pre-shared keys are not the normal use-case, it is an unnecessary burden to always encrypt the whole document.

There are however merits to allow for encryption of the whole document if additional configuration attributes that require protection, are added in the future. For this reason, it's preferable to make it an option for the end-user to protect the whole document with encryption.

Versioning

End nodes need to be aware of the specific version of the policy they are applying in order to ensure proper functionality and compatibility. This is crucial when implementing updates or modifications to policies while maintaining backward compatibility. There are two primary methods to accomplish version tracking; using a counter or using timestamps.

The counter-based approach involves assigning a numerical value to each new version of the policy document. Every time a new version is created, the counter increments by one. End nodes can then reference this counter value to determine the policy version they should apply.

Alternatively, timestamps can be employed to track policy versions. Each time a new version of the policy document is created, a timestamp indicating the date and time of the update is assigned. End nodes can compare their own timestamp with the latest timestamp in the document to determine which version to apply. This method provides more value, allowing for precise version identification when the configuration change in the document was done. However, it does require that consumers of the policy document have time set in a synchronous manner with the configuration server.

It's worth noting that choosing the counter-based approach initially does not preclude the utilization of timestamps in the future. Timestamps can be introduced for additional functionalities as needed in the future. For example, timestamps can be used to track the last modification time of the document or for auditing purposes, while the counter remains dedicated to policy versioning.

Automatic discovery

In the CIP Security Pull Model Profile, the initial setup involves locating an Enrollment over Secure Transport (EST) server through mDNS/DNS-SD and subsequently requesting identity and trust information using the EST protocol. The certificate and trust provisioning process in the Pull Model comprises two primary steps. Firstly, the EST server is discovered using mDNS/DNS-SD. Secondly, certificates are retrieved from the EST server.

Likewise, the new Pull Policy approach requires the discovery of a server to access the policy document. Since mDNS/DNS-SD is already established in the CIP Security specification, it is preferable to continue utilizing these mechanisms. By reusing the same mechanisms and protocols, unnecessary burden on end nodes is eliminated. Generally, this requirement is meant to ensure that any technology chosen would not prevent the continued use of mDNS/DNS-SD.

Configuring retry

During the initial discovery process of the Pull Policy server, the end node will continuously attempt to find the server and request the policy document. However, it is crucial to avoid overwhelming the network with excessive traffic when the server is offline or temporarily unable to provide the policy document.

To address this issue, it is necessary to implement a mechanism that reduces network traffic generated by the end node's attempts to discover the server or obtain a policy document. This mechanism should include a sensible default value, which can potentially be modified by end users through configuration options. Additionally, incorporating a back-off algorithm could be a beneficial consideration.

Trigger a reconfiguration

Over time, it is expected that the configuration policy will undergo changes, necessitating the deployment of a new policy document on the end-nodes. To ensure proper functioning, this new policy deployment must be synchronized across the entire system or, at the very least, among all interconnected end-nodes.

Given the requirement for synchronization among the affected end-nodes, a triggering mechanism becomes necessary for initiating this reconfiguration. In many cases, it is advantageous for the reconfiguration trigger to be executed through a CIP command. This approach allows for the synchronization of reconfiguration with the control operation of the system, including the secure closure and re-establishment of IO connections. However, there are also advantages to triggering the reconfiguration through the policy server especially if it could be done through a non-CIP mechanism, as the policy server would not need to know about CIP at all.

Alternatively, it is possible for each end-node configured via the policy document to periodically check for policy updates. In many cases this will provide an acceptable level of synchronization, as changes across the system need not be synchronized atomically. Many security changes will result in connections being dropped and re-made, which will provide some amount of downtime. If this downtime extends by a few seconds that is likely acceptable, as these types of changes are likely not being applied during production.

Ease of use with CIP Configuration

Even if a technology was able to easily meet all of the aforementioned requirements, it would be useless if it could not be used to encode CIP configuration. This effort is focused on deploying CIP Security configuration as document-based policy in a secure manner, so it is very important that the document is able to encode CIP configuration. Some technologies may be optimized for other types of information and therefore are not able to easily encode CIP configuration, in which case they would not be suitable. Configuring CIP Security involves several things, from simple setting of Boolean attributes to transmission of certificates for use as trust anchors. Although not strictly necessary for the minimum configuration, it is also helpful to be able to encode a request for the execution of a certain CIP service, as these might be necessary for the policy to be fully realized (for example, the policy may include running the Object_Cleanup service of the CIP Security object to clean up any unused certificates after a new policy is applied).

Suitable for embedded computing environments

Given that many of the CIP stacks execute on an embedded software platform, any technology chosen for pull policy must be able to execute within an embedded environment. Although this might not completely disqualify some possible solutions, it will likely make some less suitable. Technologies that rely on large files or require software agents to run would be less suitable for use in an embedded environment.

Human Readable

It is ideal to choose a technology that allows for the policy document to be human readable. This would provide a quick interpretation of a given policy document, and possibly even allow for manual editing in a simple system. Auditing would also be easier, as a human or machine could interpret the document without any additional decoding needed.

Technology

There are several possibilities for the technology used for the configuration policy formatting. Given that the configuration policy is to be encoded and delivered in a document, the transport mechanism is not very important; a document can be transmitted over various communication protocols. However, the technology used to format this document is important to define. Various technologies offer trade-offs. A discussion of a few possibilities follows.

AutomationML

Automation Markup Language or AutomationML, is a data modeling language created specifically for industrial automation. AutomationML is object oriented and provides some powerful features like object inheritance. This is realized using role classes and interface classes. This provides a high degree of flexibility in modeling various data concepts, and it has been used successfully in many applications, such as in the Process Industry with CAEX via IEC 62424. However, with these features comes complexity, and given that CIP objects do not directly support object inheritance, AutomationML may not be the best fit for CIP configuration policy, as not all of the functionality can or will be realized. AutomationML relies on XML for the data encoding. XML is a well-used format, although it is not generally viewed as a compact format, especially in the context of embedded devices. However, AutomationML is already deployed in various use cases across the industrial automation space, often as a unifying format to ensure different engineering tools are able to use the same data. Although this is a very important use case it is quite different from the use case of distributing CIP Security configuration policy.

Custom JSON encoded document

A custom encoding could be defined using JSON to represent CIP objects, attributes and services. The downside of this is that it requires some upfront work to define this. However, it would likely be very well suited for CIP Security configuration policy, as it would be defined specifically for that purpose. JSON is a very popular data exchange format that is compact and has wide support in parsers, some being quite lightweight and optimized for embedded environments. JSON already has signing and encryption defined via the JOSE standard; that could be applied for authenticity and confidentiality. Furthermore, a JSON schema could be defined to better document the configuration policy format and to help possible future enhancement efforts. For all these reasons this is an attractive option if the upfront development cost is palatable.

Other Policy Languages

There are a number of “policy languages” that are already in use for various purposes. Although it is not practical to analyze all of these languages, it is possible to speak about them generically. All of these were created for a use other than with CIP Security policy, so they will of course not be well optimized for

that. It may be possible in some cases for them to still be used, but that is not the purpose of these languages and there will likely be limitations in using them for CIP Security policy.

An example of a language like this is Rego, which is used with Open Policy Agent (OPA). This language is well optimized for access control, and although it is flexible, it would be challenging to tailor it to work with CIP object/attribute configuration.

Encoded CIP Services

Another possibility is to simply use the encoded CIP services within a document format. CIP already defines a format for calling services as a transmission protocol. This “on-the-wire” formatting could simply be encoded into a document and used there. This is quite straightforward from the standpoint of capturing the necessary configuration. However, it would not be human readable. Furthermore, it would not be straightforward to encode information that might not be part of a CIP object, such as the revision of the policy document. Signing and encryption would also need to be defined for this option. Although there are many ways to sign and encrypt a file, there is no standard way that this would be done since it is a custom file.

Technology Comparison

Building on the discussion above, some scoring can be done of these different choices against the requirements. An arbitrary value of 0 – 10 is used for how well each requirement is met. A score of 10 means the requirement is met in an ideal way, 0 means it is not met at all.

	AutomationML	Custom JSON	Existing Policy Language like Rego	Encoded CIP Commands	Explanation
Document Format	10	10	10	10	All options provide data in a document format
Authenticity	10	10	10	8	AutomationML, JSON, and most existing policy languages already have mechanisms for applying a digital signature. A custom file of encoded CIP commands would need to define a mechanism or choose from one of the many file signing formats.
Confidentiality	10	10	10	8	Essentially the same scoring and same reasoning as for authenticity
Versioning	10	10	10	8	Once again, the same reasoning holds; existing languages can use existing versioning
Automatic Discovery	n/a	n/a	n/a	n/a	None of these technologies provide this, it would need to be added through another means like DNS-SD
Configuration Retry	9	9	9	7	AutomationML, JSON and existing policy languages can easily encode this via a name-value pair, or encoded CIP commands as those don’t have a seamless way to do this
Trigger a Reconfiguration	9	9	9	7	Same reasoning as for Configuration Retry

Suitable for an embedded environment	6	10	2	10	JSON is a very lightweight technology and would be specifically tailored to this use case, therefore it is highly efficient. Many of the existing languages are not well suited to an embedded space. AutomationML is used in some embedded applications, but is feature rich and built on XML, which is not very lightweight. CIP commands are already used in the embedded space, so this option is also very well suited.
Human Readable	8	9	9	3	JSON and many existing languages are very human readable, AutomationML is a bit more complex but still fits here. CIP commands however are not generally human readable.
Optimized for CIP	6	9	1	10	A custom JSON for CIP Security policy is well suited to delivering CIP, and of course CIP commands are perfectly suited to this task. AutomationML is not, and many existing policy languages are not possible to use for this purpose.
Totals	**78**	**86**	**70**	**71**	

Conclusions

This paper explored some of the important use cases for automatic pull policy as a new approach to delivering CIP Security configuration, discussed requirements, and evaluated some technology options. The ability to pull all of the CIP Security configuration is important for enabling use cases which will be important in the future, including securing devices within a private NAT network and client-only software. It is likely that client-only software will grow in usage as more IIoT applications are realized in practice, such as connecting software agents that harvest data or applications that reside on mobile devices. Furthermore, the ability to replace a CIP device and deliver all of the CIP Security configuration seamlessly will enable better workflows and further ease of use. Although it is possible to realize this through a number of technologies, analysis shows that defining a JSON schema for encoding CIP Security configuration is the best technology choice. Follow on work from this paper will include defining a specification enhancement to Volume 8 of the CIP specification that defines a mechanism for this, likely using JSON. Other follow on work includes an investigation into whether or not this might be suitable for delivering other CIP configuration via a policy document. It is likely that the same requirements would hold, although certain applications like CIP Motion or CIP Safety might have additional needs that were not explored in this paper.

References

[1] DNS-SD https://datatracker.ietf.org/doc/html/rfc6763
[2] mDNS https://datatracker.ietf.org/doc/html/rfc6762
[3] AutomationML https://www.automationml.org/
[4] OPA and Rego https://www.openpolicyagent.org/docs/latest/policy-language/
[5] JSON https://www.json.org/json-en.html
[6] ODVA, Inc. The CIP Networks Library, Volume 8: CIP Security™, PUB00299

****