Open-Source Architecture for Multi-Party Update Verification for Data Acquisition Devices

Benjamin Newberg and Santiago Grijalva
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia, USA
bnewberg@gatech.edu, sgrijalva@ece.gatech.edu

Vincent Mooney
School of Electrical and Computer Engineering
School of Computer Science
Georgia Institute of Technology
Atlanta, Georgia, USA
mooney@ece.gatech.edu

Abstract— **Power grids are integral parts of modern daily life and are increasingly under cyberattack. Common software update processes for grid control devices rely on only one organization to provide verification. This paper proposes a multi-signature software update process to help better secure data acquisition devices from malicious actions by using standard cryptosystems such as TLS. A prototype system build on Linux and off-the-shelf hardware has shown successful update and attack prevention with our customized multi-party update software.**

I. Introduction

Cybersecurity attacks on power grids have become an increasingly vexing problem around the world. Power grids are critical parts of daily life and so need the utmost in protection.

A notable example of weak update security is the attack on the Ukrainian power grid in 2015 [1]. Attackers were able to login using stolen credentials and executed an irrevocable overwrite of the firmware on Moxa UC 7408-LX-Plus data acquisition devices – which are commonly used in power grid control systems – rendering the devices inoperable [1]. This action caused a massive power outage affecting over 200,000 customers [1].

The update of control device software has been identified as particularly vulnerable to attacks as demonstrated in the Ukrainian attack. Current software upgrade methods rely on verifying only one cryptographical signature. Use of only one authentication method has some flaws, the most notable of which is that only one organization needs to be compromised to insert malicious code into devices. This paper proposes a method to rectify such a vulnerability and increase the transparency of the code being used in power grid control systems. The use of propriety devices which do not provide software access makes it very difficult or impossible to conduct comprehensive security analysis. This paper utilizes an open-source, flexible model in response to that challenge to model critical data acquisition devices in the power grid. While we recognize that the power grid is a very large and complex infrastructure and that most of the control devices utilized are proprietary, the utilization of verifiable software (via provision of source code whether the code is proprietary or is open source) to address security issues is gaining momentum in other industries [9]. Linux is a common kernel to deploy in embedded systems such as used in Moxa data acquisition systems, specifically the Moxa UC 7408-LX-Plus [3]. Modeling such systems is important to develop cybersecurity solutions.

Moxa did not update the software on the UC 7408-LX-PLUS as it was discontinued as mentioned by US CERT [4]. Many utilities still use this device, and therefore a new method is needed to allow the update of legacy devices [4]. This paper demonstrates a system that improves the creation of secure updates to enhance the lifespan of mission critical control devices. A relay is used in this paper to better demonstrate the practical application of the proposed setup.

This work has been partially supported by the U.S. Department of Energy's Office of Cybersecurity, Energy Security, and Emergency Response (CESER) under Cybersecurity for Energy Delivery Systems (CEDS) Agreement Number DE-CR0000004 to the Georgia Tech Research Corporation

II. Background

a) Primitives

Many components are part of a cybersecurity system. Some of the most important are the cryptographic primitives used. Rivest-Shamir-Adleman (RSA) is a cryptosystem that uses asymmetric key cryptography [5]. Transport Layer Security (TLS) is an implementation of cryptosystems that is used to encrypt data in transit, and in the scenario investigated in this paper TLS is used as a machine-to-machine method [6]. TLS 1.3 is the latest version, and it is the only version used in this paper [6]. A Certificate Authority (CA) is critical to augment the TLS standard, as a CA provides the cryptographic signature to verify the authenticity of an entity, for instance a website a person is visiting by vouching for the authenticity of a cryptographic key used to communicate with the website [6].

b) Gentoo

Gentoo Linux is the operating system used to model the data acquisition device [8]. We choose Gentoo Linux as the base of our prototyping system and experiment because it provides flexibility in creating a model system via its ability to control the compilation of everything from source on the computer. Gentoo allows the user to control the features included in applications via USE flags [8]. Further, Gentoo allows the complete customization of kernel options. Gentoo introduces an ebuild system that allows a developer to write a bash type script that controls how the package is compiled and installed on the system. Two Gentoo ebuild features that the paper utilizes extensively are src_unpack and verify_sig [8].

Verify_sig is a specialized class that allows signature verification on the source tarball [8]. Using Gentoo allows great flexibility in designing a setup that can model virtually any software stack based on Linux in use in current power grids of today. This paper combines default Gentoo behavior with additions to better verify the authenticity of the source tarball downloaded before compilation. The default Gentoo behavior is to verify the source tarball against a hash that was created during the maintainer's upload of the ebuild. This model trusts the maintainer and that the source tarball was not malicious when the maintainer created the ebuild. The method presented here ensures that more people must sign off on the source tarball due to two detached signatures being needed, which improves security from a single point of failure as shown in the default Gentoo model.

c) OS Components

Modern operating systems are often broken into two different components: the *kernel space* and *userland*. The kernel space is at the lowest level and most trusted part of the operating system and handles hardware access, process management, along with other low-level tasks. Userland is where the desktop and applications that users interact with daily reside.

d) Containers

Container technology is lighter weight than a virtual machine because a container still runs on the host machine's kernel, but the userland stack is separate from the host [7]. Container technology is used to separate services for easier and more secure management; using containers follows best practices in modern data centers and reduces dependency conflicts. Many current server farms in the cloud use containers.

TABLE I
SOFTWARE PACKAGES

Software	Description
Podman	Container Management Software [7]
Moby	Container Management Software [20]
Nomad	Workload orchestrator [13]
Consul	Service mesh communication coordinator [14]
Vault	Management of the certificate authority [10]
Ceph	Storage backend that creates storage clusters [16]
Hockeypuck	Key server [18]
GitLab	Git server [21]
Traefik	Reverse proxy [15]
Chrony	Network time protocol server [11]
CoreDNS	DNS Server [19]
Gemato	Gentoo manifest creator/verifier [17]
Portage	Gentoo package manager [22]

Containers also make it easier to update software by allowing the update of the entire software stack by deploying a new container in place of the old one as persistent data is stored outside the container, which allows a container to be ephemeral. Container updating, however, is not the focus of this paper as it already is a well-discussed topic in software engineering. We choose Podman, a project of Red Hat, as the container management software for only the storage drivers [7]. Moby, another container manage-ment solution, is used for any service deployed by Nomad, due to compatibility issues with Nomad and Podman [20].

e) Software Used

Table I displays the software packages used. A service mesh as mentioned in the table, which is a way for applications to share information with each other, is managed by Consul.

III. ATTACK SCENARIO

In the attack scenario we use in this paper, a lone wolf attacker has access to the power grid control system. The attacker is a low-level engineer that can change firmware source code before it is installed on a device or launch an attack similar to the (assumed key steps of the) Ukrainian attack as seen in Figure 1. The attacker may use ssh to carry out the attack along with sftp to move the files that contain the exploit onto the Vendor server cluster. The low-level insider has these engineering permissions in order to carry out the duties of the position held in the company.

Figure 1. Diagram of a possible attack.

IV. PROPOSED MITIGATION: MULTI-PARTY UPDATE

A proposed mitigation implements a multi-party update verification process based on Gentoo Linux (or any other similarly provisioned software). Figure 2 shows our proposed multi-party verification based on the concept of multiple organizations providing cryptographic verification of the integrity and authenticity of the update.

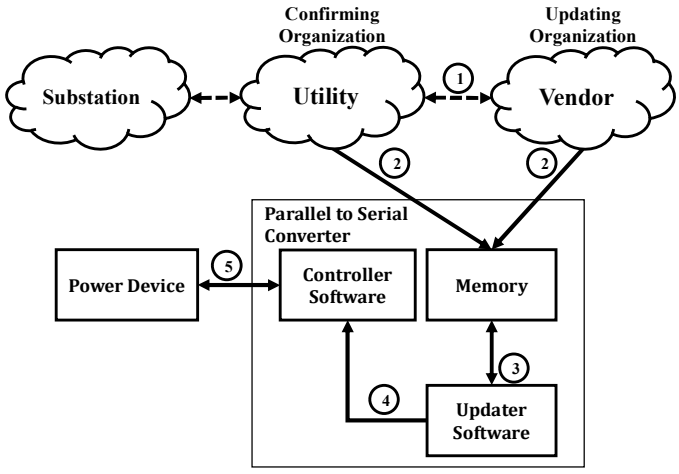

Figure 2. Diagram of update method.

Specifically, Fig. 2 shows a confirming organization, the electric Utility, and an updating organization called Vendor. Multi-party update verification allows multiple companies to verify the provenance and authenticity of an update before it is applied to a machine. This method is different from other

methods because currently most updates only utilize a single signature check.

In the first step (e.g., see the "1" circle in Figure 2), the multiple organizations involved (Fig. 2 has two such organizations) agree to update a device in the power grid. A first organization (e.g., an organization responsible for overall software update management such as Vendor in Figure 2) signs the source code and binary and then sends the signed update source code and binary to the next organization involved in supervising the update process. This second organization could – before either applying the update to their machines or their customer's machines – utilize software engineering methods to verify the binary matches the source code and/or the source code contains no malicious additions.

The step 2 circles in Figure 2 show both organizations (Utility and Vendor) sending signed updates to the energy device, e.g., a data acquisition device. Note that only one organization (the "Updating Organization" in Fig. 2) needs to send the full firmware update encrypted and signed; the rest of the entities involved (e.g., the "Confirming Organization" in Figure 2) may send signed cryptographic hash values to verify the complete firmware update.

Specifically, in step 3 in Fig. 2, the updater software (which cannot be altered in our model) regenerates the hash value of the firmware or software binary to be updated: only if the encrypted and signed hash values from all entities (Fig. 2 shows two entities, a Utility and a Vendor) match does the update proceed. If there are any errors, e.g., the signatures do not match or the hash values are not equal, an error message is generated and the update is not applied.

Step 4 in Figure 2 updates the controller software of the data acquisition device if all hash and signature checks pass. Step 5 shows the data acquisition device communicating with a power device in the grid.

Our proposed mitigation raises the cost of an attack because now instead of needing to break one network, an attacker would need to breach two networks in order to successfully carry out the attack, i.e., in our example the primary action required is to steal appropriate private keys in each network. In Fig. 1 and Fig. 2 the keys used are RSA 4096-bit keys, and the exchange of the keys is done via USB drives between the devices. The keys could also be preloaded at the factory or delivered through a keyserver. The ebuild script is written to include an extra security check (using the RSA keys) in the src_unpack section. This extra check is in addition to the enablement of Gentoo's verify_sig USE flag. If either of these checks fail, then the build process will be aborted. The Utility is air gapped to simulate the most secure type of system and prevent any interface from outside networks since it is not connected to the outside internet.

V. Prototype System

In order to model the above attack scenario and its proposed mitigation, a Dell desktop with Gentoo, henceforth called Utility device, is used along with a Dell laptop, henceforth called Vendor server cluster, acting as the server cluster that Utility uses. The desktop models a data acquisition device that is connected to Ethernet. The laptop is running four virtual machines to simulate the cluster. To route between these subnets, an Ubiquiti EdgeRouter 12 is utilized with IPv4 addressing as seen in Figure 3. Also, the connection between the Utility device, Vendor server cluster, and router is Ethernet.

Figure 3. Diagram of a network and assigned subnets to each organization.

Container technology (see Section 2) is utilized by the Vendor to run the different services needed to carry out the experiment.

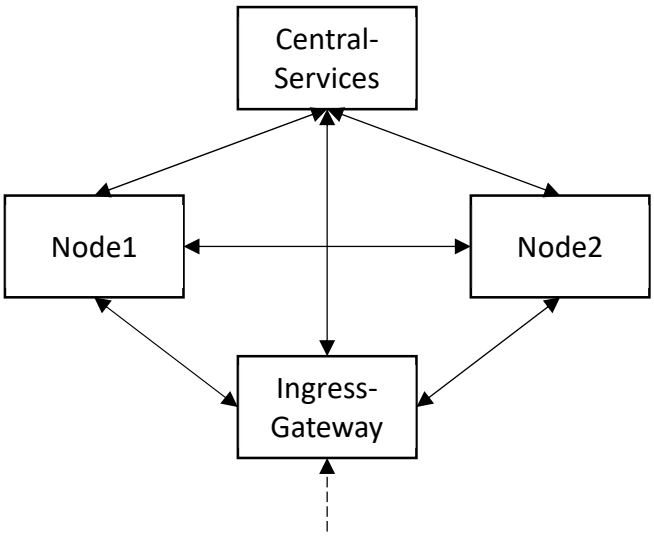

Figure 4. Server cluster layout. Dashed line is the incoming outside connections accessible from the 10.0.1.3 address. The solid lines represent the internal connections made either by the services used, cluster only ssh connections, or Consul Connect in the 192.168.122.0/24 subnet.

The Vendor server cluster is used to create a four virtual machine server cluster as shown in Figure 4. The first virtual machine is called central-services, and it runs in a virtual machine the servers for Consul, Nomad, and Vault along with Podman containers for the storage backend, Ceph. Node1 and Node2 are two other virtual machines that run Consul and Nomad as nodes onto which jobs can deploy. Node1 and Node 2 also run Podman with Ceph to create a 60GB storage pool. The final virtual machine is called ingress-gateway, and it maintains the Domain Name Service (DNS) server to advertise the domains created for this experiment with the IP for ingress-gateway, the Network Time Protocol (NTP) server for the Utility device, and the Traefik reverse proxy to terminate the TLS connections and route each request to the appropriate services. The domains created, but only valid within the experiment subnets, are the following: (i) registry.gmlab, the container registry storage; (ii) git.gmlab, the self-hosted GitLab

location; (iii) keys.gmlab, Hockeypuck key server; and, finally, (iv) serving.gmlab, which serves the static content of the source tarball and detached signatures as needed during the ebuild process. Each of these domains provides a way to access the needed service inside the cluster.

Three HashiCorp products are used to create, administer, and secure the server cluster that runs the services needed. The products are used under their free and open-source terms. The products are the following: Vault, Nomad, and Consul. When discussing these applications, server means the virtual machine controlling what the worker nodes and what the clients are doing.

Vault is an application that allows the management of certificates, access tokens, and other security functions [10]. Consul allows for service mesh communication coordination [14]. Nomad is for job orchestration across the nodes to determine where a service is going to run [13].

Nomad is a workload orchestrator that can deploy jobs across multiple nodes, thus helping to reduce the complexity of deciding where each job should be run and improving redundancy as multiple copies can be ran at once. However, this experiment is running on limited RAM and so only one copy of each application is deployed. Nomad describes a job as something that encompasses multiple task groups, where each task is a container or other application desired to be run on a server cluster. Groups help organize tasks based on whether tasks need to be on the same server or not.

Consul helps manage service meshes to ensure services inside it can connect to each other. Consul is used to route for instance the key server to its database master as best practices dictate for microservices. Consul Connect, a feature of Consul, is also used to register each service with Traefik for routing from the outside into the cluster. Also, Consul Connect creates a sidecar proxy using Envoy for each service which leaves the routing between services being managed by Consul, which further simplified networking operations. Consul runs on all the virtual machines to manage the networking.

The services deployed via Nomad are the Hockeypuck key server, a GitLab instance, Ceph controller for Nomad, Ceph nodes for Nomad, a static web content server to serve the source tarball and its detached signatures, and a private Docker container registry to handle any custom images that must be made and deployed. The Alpine Linux userland tag, if an option from Docker Hub, is always chosen. Custom images are created for the static web content served via the Apache web server container, so that the need for persistent storage is eliminated and the Hockeypuck server container is custom created from Hockeypuck's source to better work in the experiment environment and run on Alpine.

Ceph controller is used to control the integration of Ceph and Nomad. The Ceph nodes are present on all Nomad nodes to handle connecting the containers back into the Ceph backend.

The git repository for ebuilds is hosted on GitLab as it is the best way to push ebuild updates out to the data acquisition device. GitLab is chosen due to its community edition being MIT licensed, and an easier way to setup a git server with a nice GUI and many features to use to manage the ebuild repository for the ebuild used in this experiment and provide the server from which to sync the ebuild. Because git is used, the top commit signature is checked during the cloning process by portage on Gentoo.

Hockeypuck is an open-source project that can be used to store key material for the signing applications and monitor key revocation status, which is why it is needed [18]. Gentoo uses the key server to check key revocation status when verifying the commit signature on the ebuild git repository or manifest to notice if the ebuild info was tampered by unauthorized users either on the server or in transit.

Traefik is used as the reverse proxy to provide a consistent entry point for all the services no matter on which nodes they are deployed [15]. Traefik also handles TLS termination, which eliminates the need to manage TLS certificates separate for each service, thus simplifying operation.

Chrony is selected as the NTP server due to it being the default of Red Hat Enterprise Linux, which models a more realistic corporate network [11].

CoreDNS is chosen for the DNS server as it provided a simple configuration via Corefiles and is a light statically compiled Go binary [19].

Nomad job descriptions are created that described how to run the container server, the key server, and GitLab. The job descriptions also created sidecar proxies to connect the service into Consul. Figure 4 displays the layout of the service cluster. The jobs are deployed onto Node1 and Node2 by Nomad through its allocation algorithm. It is not necessary to know which of the nodes contains the services as Consul knows how to route requests to each service and Traefik routes based on Consul's knowledge.

To show both a solution to ensure reliability and to deal with the way workload orchestration is done by Nomad, where the job may change nodes each time it is run so static storage on a single node is not possible, Ceph is chosen as a storage backend because it copies the data across multiple nodes and allows integration with Nomad in order for the data to be accessed from any of the worker nodes [16]. Ceph is an open-source project under the Ceph Foundation under the Linux Foundation, and it is used to create clusters of nodes to use as storage for many applications, but in this paper used to be the persistent storage for the container registry, GitLab instance and Hockeypuck key server.

Our own Certificate Authority (CA) is developed and then deployed to the different machines via a USB drive or secure file transfer protocol (SFTP). Another option is the preloading of certificates from the factory. The crucial step of building our own certificate authority is needed to ensure that we maintain our own CA root of trust and give us greater flexibility in managing our network. The CA is needed to ensure TLS connections used for the container registry, the GitLab instance, the key server, and the static content server. The data acquisition device in the Utility will be the device receiving the update.

Portage, the Gentoo package manager, manages the update process and uses the ebuild script from the git repository and source tarball served by the Apache web server on the Vendor server cluster. RSA keys are utilized for the signature checking

to validate the validity of both the ebuild and the source tarballs. Two components are needed for a successful installation: (i) the ebuild to describe dependencies, compilation, and installation process along with (ii) the source tarball that actually contains the program's source to be compiled. Two detached signatures are also shipped with the source tarball. The source tarball is validated by both the Vendor and the Utility signatures before being installed. The ebuild is verified by both the Vendor and Utility's signature. The verification happens through two existing verification methods but combined in a novel way for git repositories. Standard Gentoo security has an optional check for the top commit for a git repository to be signed by a trusted developer. However, this method has two major flaws. The first is a lone wolf could be that developer and thus compromise the ebuild and corresponding manifests, and so now the attacker would have the ability to install any software on the machine during installation instead of the expected software. The second flaw is that if a trusted developer commits to the repository, then this trusted developer has to be assured that all previous commits are not malicious, which is a daunting task. The method used in this paper eliminates the first flaw while providing strong mitigations for the second flaw that can be used in addition to other methods to detect malicious code changes or security issues in software. The method used in this paper still uses the git signature, and in this experiment that is the Utility. The method also takes from Gentoo's rsync security mechanism of verifying all the hashes of the manifest files through a signed top level manifest file by implementing that in a git repository, which is non-standard Gentoo feature relating to a git repository. The Gentoo software component that carries it out is called gemato [17]. Gemato is also used to create the signed manifest. The first step to creating the manifest tree is to run the ebuild manifest command which creates manifests in each application ebuild's folder containing the hashes of the ebuilds and any data that is downloaded like the source tarball for instance. Gemato takes these manifests and creates a manifest tree that contains hashes of manifests in lower folders, so the manifest at the top of the repository just has the hashes of the manifests in the next folder. Only the top manifest is signed. This setup reduces the burden of management while indirectly protecting all the data with cryptographical security. This manifest is signed by the Vendor. These two methods combine increase the complexity of the attack by now two different verification methods being used and two signatures. It also increases the number of eyes on (people looking at) the code as the Utility must now review the ebuilds before committing them to their repository to be pushed out to their devices. This extra layer of security is extraordinarily important because the compromise of the ebuild hosting server makes any other checks moot due to the ebuild containing the source tarball checks in the first place along with the accompanying manifest files containing the hashes of the source tarball and associated detached signatures.

The Utility device has no containers in this operation, and the applications needed are shown in Figure 5. Git is used by portage to download the ebuild repository as it is hosted on the GitLab server mentioned earlier. The exchange of both the source tarball, its detached signatures, and ebuild is done via TLS, which is also be verified by CAs, which further enhances the authenticity, confidentiality, and integrity of the information. The src_unpack step uses verify_sig functions to verify both the Vendor and Utility's signature. Figure 5 depicts the IP address and main software used on the Utility device.

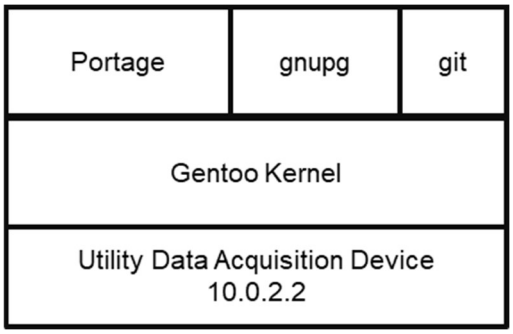

Figure 5. Software layout at the Utility.

A high-level overview of where each step happens is displayed in Figure 6.

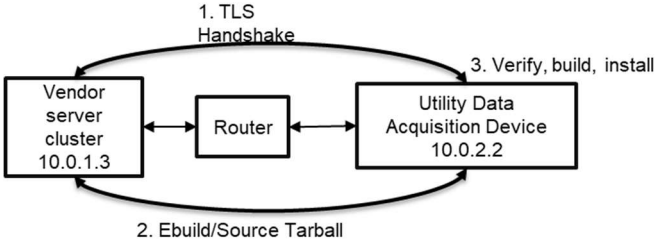

Figure 6. Basic experimental setup.

VI. Experiment

We test our prototype system by using twelve different update scenarios to test the various aspects. A picture of the experimental setup is shown in Figure 7.

Each experiment is run after changing the git repository as needed or redeploying the web server with the changed source tarball or/and detached signatures. Table 2 illustrates the experiment and what part is changed for each experiment and the results.

Man-in-the-middle (MITM) is a tactic where the attacker intercepts the communication before two parties and changes information or impersonates the other party.

The first scenario is a normal scenario where all the security checks pass, and the update succeeds on the Utility terminal.

The next two scenarios test scenarios where either the Utility's signature or the Vendor's signature is missing on the source tarball. The installation fails at the downloading step as the detached signatures are missing.

The 4th scenario tests CA validation. The attacker creates a fake CA and tries to push an update. Portage on the data acquisition device fails because the ebuild will not even download due to an https connection failure.

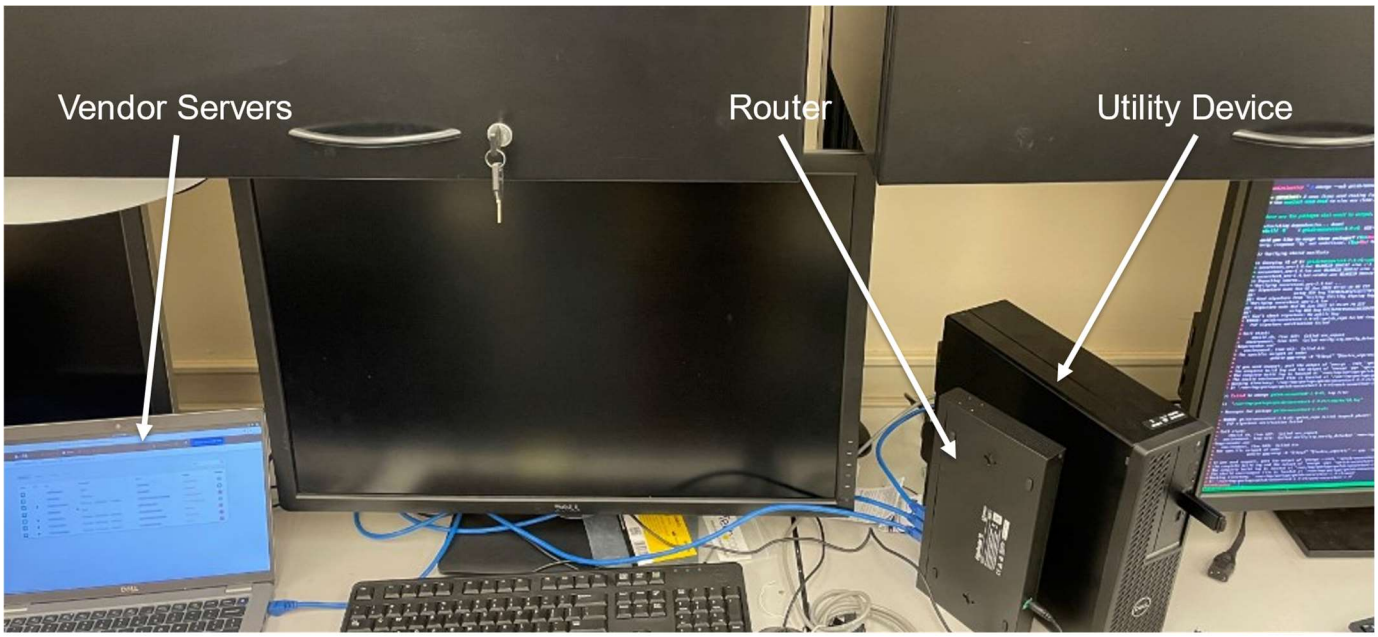

Figure 7. Photo of hardware used in experiment.

The 5th scenario tests an ebuild with an unknown Manifest signature. The result is failure during validation of the ebuild when portage downloads it.

In the 6th scenario, an ebuild with no Manifest signature is pushed to git, but portage warns the user of that fact.

In the 7th scenario, an ebuild with an unknown git signature is tested, but fails due to the signature not being known.

In the 8th scenario, an ebuild with a missing git (i.e., Utility) signature is tested, but failed due to the missing signature. The 5th through 8th scenarios both ended with the bad ebuilds being deleted off the system once the script notices that the signatures are not trusted.

In the 9th scenario, the Utility's signature is good, but where Vendor should be is an unknown signature. It fails at checking the signature hash and size.

The 10th scenario is the reverse of the 9th scenario, where Vendor is a good signature, but where Utility's signature should be is an unknown signature, so it still fails at the same place.

The 11th scenario is an attacker trying to launch a man in the middle attack by impersonating the Utility server cluster, which fails because the attacker will not have the CA private key, so the attacker will not be able to impersonate the server.

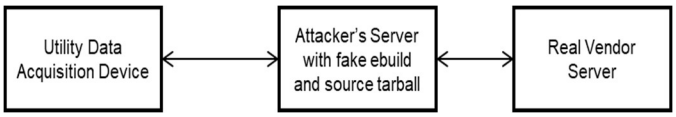

Figure 8. Twelfth Scenario Layout.

The 12th scenario is the most dangerous, where the attacker steals the CA's private key, and then forges a certificate. The attacker uses that to setup the attacker in the middle as shown in Figure 8. This attacker still fails as the tampered ebuild during the hash check of the downloaded files, because the attacker does not have access to any of the other signing infrastructure. The assumption is that the attacker cannot carry out a buffer overflow or other type of software attack during the malicious TLS negotiation phase or hashing functions of the proposed update solution. These scenario checks both demonstrate the proposed features of this paper and prove that the mitigations achieve the stated goals.

VII. Discussion and Conclusion

This paper proposes a solution to deal with a critical problem in modern infrastructure, which is cybersecurity on aging and vulnerable industrial control devices. To validate that the solution works as proposed, twelve scenarios is used in experiments to test the different facets of the solution and to demonstrate each part provides a layer of security that if broken another layer stops the attack. A multi-layered approach to security is needed as at least one layer may be broken as seen on a more regular basis with many companies across a range of industries having to deal with cyberattacks.

Because Gentoo can have the source tarball separated from the ebuild as demonstrated in this paper, two separate websites are needed to have two-signature protection. While this setup raises complexity, it can increase security by, along with two signatures, having the sources of data being in two separate networks. However, the servers are on the same subnet in this paper because of lack of time. By separating out the signatures among different organizations, it raises the bar to attack multiple organizations in a coordinated manner, which is far more complicated for an attacker and thus helps to better secure the grid. We also propose a modeling technique which should help to expand cybersecurity research as it provides a flexible

TABLE II: Experiment Setup and Results

Experiment	Utility Signature	Vendor Signature	Certificate Authority	ebuild Manifest Signature (Vendor)	ebuild Git Signature (Utility)	MITM	Result (at which point did it fail)
1st	Correct	Correct	Correct	Correct	Correct	Not tried	Update installed successfully
2nd	Missing	Correct	Correct	Correct	Correct	Not tried	Fails to download detached signature
3rd	Correct	Missing	Correct	Correct	Correct	Not tried	Fails to download detached signature
4th	Tampered	Tampered	Fake by attacker	Correct	Correct	Tried	Warning of unknown CA so nothing downloaded
5th	Correct	Correct	Correct	Unknown	Correct	Not tried	Warned no signature, and then deleted the downloaded copy
6th	Correct	Correct	Correct	Missing	Correct	Not tried	Warned signature missing, and then deleted the downloaded copy
7th	Correct	Correct	Correct	Correct	Unknown	Not tried	Warned signature not valid, and then deleted the downloaded copy
8th	Correct	Correct	Correct	Correct	Missing	Not tried	Warned no signature present, and then deleted the downloaded copy
9th	Correct	Unknown	Correct	Correct	Correct	Not tried	Manifest flags the signature as incorrect checksum value of file
10th	Unknown	Correct	Correct	Correct	Correct	Not tried	Manifest flags the signature as incorrect checksum
11th	Tampered	Tampered	Removed	Correct	Correct	Tried	Fails since SSL connection cannot succeed
12th	Tampered	Tampered	Forged certificate using real CA's signing key	Correct	Correct	Tried	Fails at hash check for source

method for practically any researcher to build models of intelligent energy devices and test new cybersecurity techniques using commodity software and open-source tooling. Further, commonly used cloud orchestration technology was demonstrated, which shows how cloud technology can be used to help secure power grids.

REFERENCES

[1] J. Styczyinski, N. Beach-Westmoreland, "When the Lights Went Out: A Comprehensive Review of the 2015 Attacks on the Ukrainian Critical Infrastructure", Booz Allen Hamilton, McLean, Virginia, USA, 2019.

[2] UC-7400 Hardware User's Manual, 6th ed., Moxa Inc., Brea, CA, 92823, 2009.

[3] UC-7408 Series, Moxa Inc., Brea, CA, 92823, 2010.

[4] "ICS Advisory (ICSA-16-152-01): Moxa UC 7408-LX-Plus Firmware Overwrite Vulnerability." U.S. CISA. https://us-cert.cisa.gov/ics/advisories/ICSA-16-152-01 (accessed Nov. 20, 2021).

[5] A. Menezes, P. Oorschot and S. Vanstone, Handbook of Applied Cryptography, 5th Edition, CRC Press, 1996.

[6] The Transport Layer Security (TLS) Protocol Version 1.3, IETF RFC 8446, Internet Engineering Task Force, 2018.

[7] S. McCarty. "A Practical Introduction to Container Terminology." Red Hat Developer. https://developers. redhat.com/ blog/2018/02/22/container-terminology-practical-introduction (accessed Nov. 22, 2021).

[8] "Gentoo Linux." Gentoo Authors. https://www.gentoo.org/ (accessed Nov. 27, 2021).

[9] Intuit Developer Team. "Security benefits of open source software." Intuit Developer. https://blogs.intuit.com/blog/ 2020/10/13/security-benefits-of-open-source-software/ (accessed Nov. 27,2021).

[10] "Vault." HashiCorp. https://www.vaultproject.io/ (accessed Jan. 13, 2022).

[11] "Chapter 30. Using Chrony." Red Hat Customer Portal. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_basic_system_settings/using-chrony_configuring-basic-system-settings (accessed Nov. 27,2021).

[12] "Apache HTTP Server Project." Apache Software Foundation. https://httpd.apache.org/ (accessed Nov.27, 2021).

[13] "Nomad." HashiCorp. https://www.nomadproject.io/ (accessed Jan. 13, 2022).

[14] "Consul." HashiCorp. https://www.consul.io/ (accessed Jan. 13, 2022).

[15] "Traefik Proxy." Traefiklabs. https://www.traefik.io /traefik (accessed Jan. 13, 2022).

[16] "Ceph." RedHat. https://www.ceph.io/ (accessed Jan. 13, 2022).

[17] "Gemato." GitHub. https://github.com/mgorny/gemato (accessed Jan. 13, 2022).

[18] "Hockeypuck." Casey Marshall. https://hockeypuck.io (accessed Jan. 13, 2022).

[19] "CoreDNS." Linux Foundation. https://coredns.io (accessed Jan. 13, 2022).

[20] "Mobyproject." Moby Project. https://mobyproject.org (accessed Jan. 15, 2022).

[21] "GitLab." GitLab. https://about.gitlab.com/ (accessed Jan. 15, 2022).

[22] "Portage." Gentoo Linux Wiki. https://wiki.gentoo.org/wiki/Portage (accessed Jan. 15, 2022).