ADALOGICS

SEAPATH Security Audit 2024

Security Audit Report

In collaboration with the SEAPATH project

and Open Source Technology and Investment Fund

Adam Korczynski, David Korczynski

23rd July 2024

Contents

About Ada Logics

Ada Logics is a software security company founded in Oxford, UK, 2018 and is now based in London. We are a team of dedicated, pragmatic security engineers and security researchers that work hands-on with code auditing, security automation and security tooling.

We are committed open source contributors and we routinely contribute to state of the art security tooling in the fuzzing domain such as advanced fuzzing tools like Fuzz Introspector and continuous fuzzing with OSS-Fuzz. For example, we have contributed to fuzzing of hundreds of open source projects by way of OSS-Fuzz. We regularly perform security audits of open source software and make our reports publicly available with findings and fixes, and we have audited many of the most widely used cloud native applications.

Ada Logics contributes to solving the challenge of securing the software supply-chain. To this end, we develop the tooling and infrastructure needed for ensuring a secure software development lifecycle, and we deploy these tools to critical software packages. On the tooling and infrastructure side, we contribute to projects such as the OpenSSF Scorecard project as well as the Sigstore projects like SLSA and Cosign.

Ada Logics helps some of the most exposed organisations secure their software, analyse their code and increase security automation and assurance, and if you would like to consider working with us please reach out to us via our website. We write about our work on our blog. You can also follow Ada Logics on Linkedin, Twitter and Youtube.

Ada Logics ltd
71-75 Shelton Street,
WC2H 9JQ London,
United Kingdom

Executive summary

In January through May 2024, Ada Logics carried out a holistic security audit of the SEAPATH project (Software Enabled Automation Platform and Artifacts (THerein)). The Security Audit was sponsored by LF Energy and facilitated by Open Source Technology Improvement Fund, Inc. SEAPATH is an LF Energy hosted project intended for managing clusters of servers in energy substations. It packages hypervisors, virtualization and VM images into a platform for substations to increase their dependence on software and decrease their dependence on hardware. SEAPATH is fully open source and its core job is to package existing, mature open source software projects into a platform for substations.

The audit included the following exercises:

1. We formalized a threat model for SEAPATH focusing on scope and SEAPATHs responsibility in that scope.
2. We audited SEAPATH for existing security vulnerabilities that would allow an attacker to attack SEAPATH users.
3. We reviewed SEAPATHs security practices from a perspective of their sufficiency in making SEAPATH production-ready in the future.
4. We reviewed SEAPATHs choice of open source software and how SEAPATHs manages it. We did this primarily assess the health, risk and maturity of the projects for SEAPATHs use case. During the audit we both made notes of security practices of specific software projects that SEAPATH packages, and we also made specific improvements to demonstrate how these should be implemented in practice.
5. We reviewed SEAPATH against the ISA/IEC-62443 security standards for areas of non-compliance in both documentation practices and technical capabilities.

During the audit we saw that SEAPATH prioritises security. SEAPATH has invested ample work into its security practices and has both researched good practices to follow and is investing effort into implementing these practices. We found that SEAPATH has taken a holistic approach to security; For example, SEAPATH maintains an extensive testing suite which runs on actual, production-intended hardware and tests for many heuristics required for safe deployment. We also found that SEAPATH has invested in hardening the product to a high degree, and that security plays a role in multiple layers of the product.

As a result, for exercise 1-4 above, we also took a holistic approach in this audit in the context of how an adversary would attempt to compromise SEAPATH in either now or in the future. We considered both direct compromises as well as indirect maneuvres such as attacks against SEAPATHs supply-chain and its development lifecycle.

We found several minor issues related to SEAPATHs development lifecycle of which SEAPATH has addressed all. Our major concern in the SEAPATH platform is that it relies on memory-unsafe software

projects that do not employ fuzz testing. From experience, fuzz testing should be applied to memory-unsafe software and has historically been effective in finding hard-to-find bugs. From the threat modelling exercise, we consider it necessary for SEAPATH to ensure that it packages projects that are fuzz tested if they are implemented in memory-unsafe languages such as C and C++. As a proof of concept of what we consider an necessary fuzzing practice for SEAPATH, we added fuzz testing to a third-party dependency of SEAPATH - the Pacemaker project. SEAPATH uses Pacemaker as the underlying infrastructure for managing virtual machines on SEAPATH clusters. We wrote 4 fuzzers for Pacemaker and integrated Pacemaker into the OSS-Fuzz project such that the fuzzers run continuously. We recommend that this work is repeated and improved upon for Pacemaker and other SEAPATH dependencies that do not employ fuzz testing. This includes Libvirt and Ceph.

For the auditing against the ISA/IEC-62443 standards, we found several areas in which SEAPATH needs improvements to comply. Most of these are related to documentation and organizational requirements rather than technical shortfalls.

We can summarize the outcome of the audit as follow:

1. We formalized a threat model for scope and responsibility within its scope.
2. We made hardening suggestions related to configuration, authentication and testing.
3. We found several areas in which SEAPATH needs to maintain user documentation to comply with ISA/IEC-62443 cyber security standards. We have documented these in this report.
4. We found several areas in which SEAPATH does not follow best practices in its development lifecycle. SEAPATH has mitigated all of these.
5. We wrote four fuzzers for Pacemaker and integrated Pacemaker into OSS-Fuzz. The fuzzers found 7 issues during the audit and continue to test Pacemaker.
6. Reported a total of 32 issues and suggestions to improve SEAPATHs security.

Strategic recommendations

Here we include our strategic recommendations for the SEAPATH project.

Continue the threat modelling from this audit

A limitation in SEAPATHs security posture is that it lacks a threat model. We consider it an obstacle to SEAPATH in working with its security posture moving forward, if the project does not formalize a threat model and iteratively improve it over time. In this audit we started this process with the focus on scope and responsibility within the scope. We recommend that the SEAPATH project continues this exercise and includes both security-focused community members and non-security-focused community members. Non-security-focused community members will likely have deep technical

knowledge that can aid the threat modelling process. Maintaining a threat model is also important from the perspective of complying with the ISA/IEC-62443 standards which have multiple requirements to threat modelling.

Ensure memory-unsafe code is fuzz tested

SEAPATH relies on several projects implemented in memory-unsafe languages such a C and C++. We recommend that this code is fuzz tested in a continuous manner. During this audit, we prepared a proof-of-concept setup for the Pacemaker project. To avoid risk for users, the SEAPATH community should ensure that this work is replicated for memory-unsafe software used in the SEAPATH platform. Over time, coverage should improve as fewer crashes are found in covered code.

Project dashboard

Contact	Role	Organisation	Email
Adam Korczynski	Auditor	Ada Logics Ltd	adam@adalogics.com
David Korczynski	Auditor	Ada Logics Ltd	david@adalogics.com
Amir Montazery	Facilitator	OSTIF	amir@ostif.org
Helen Woeste	Facilitator	OSTIF	helen@ostif.org
Derek Zimmer	Facilitator	OSTIF	derek@ostif.org
Eloi Bail	SEAPATH TSC Chair	Savoir-faire Linux	eloi.bail@savoirfairelinux.com
Florent Carli	SEAPATH maintainer	RTE	florent.carli@rte-france.com
Bastien Desbos	SEAPATH TSC voting members representative	RTE	bastien.desbos@rte-france.com
Mathieu Dupré	SEAPATH maintainer	Savoir-faire Linux	mathieu.dupre@savoirfairelinux.com
Jan Hille	SEAPATH TSC voting members representative	Welotec	j.hille@welotec.com
Erwann Roussy	SEAPATH maintainer	Savoir-faire Linux	erwann.roussy@savoirfairelinux.com

ANSSI

SEAPATH follows the hardening guidance of ANSSI - the French national cyber security agency. ANSSI maintains a guide for secure and hardened Linux configuration which relies on the following principles:

1. Complexity reduction: Linux systems should avoid unnecessary complexity
2. Least privilege: Objects and entities should only have the permissions they need for their purpose.
3. Defence in depth: The system should implement multiple layers of defence.
4. Secure monitoring: Service activity should be logged, and the log should be retrievable.

The French government consumes ample free and open source software including Linux distributions. In 2013, the French government had switched 37,000 desktop machines in the French national police force to Linux, making it the largest public service open source software user in Europe. ANSSI's role as an organisation is to protect the French government's networks. ANSSI is an organisation of around 600 employees which makes it hard to directly secure the entire infrastructure of the French government. As a consequence, ANSSI has adopted a strategy of securing their software upstream when possible. Part of that strategy is to develop documentation and recommendations that promote that strategy. We consider the document "CONFIGURATION RECOMMENDATIONS OF A GNU/LINUX SYSTEM" to be part of that strategy, and as result, systems that follow its guidelines comply with the same requirements that are bestowed upon systems in the French government. Systems in the French government operate in a context of national security, and we consider SEAPATH to meet the requirements of operating in a national security context, if SEAPATHs Linux distributions follow ANSSI's "CONFIGURATION RECOMMENDATIONS OF A GNU/LINUX SYSTEM".
At the time of the audit, SEAPATHs work on complying with ANSSI's linux configuration guide is ongoing with some parts completed and others not completed yet. SEAPATH maintains a compliance matrix that detail which points have been completed, which have not been completed, which are not applicable to SEAPATH, which should be done by the user and which are partially done: https://wiki.lfenergy.org/display/SEAP/SEAPATH+and+cybersecurity. In addition, SEAPATH implements a test suite that dynamically tests that the configurations are applied correctly. The findings of each test run are published in a report here: https://github.com/SEAPATH/ci/tree/reports-PRmain/docs/reports/PR-main.

To summarize, at this point, the SEAPATH project has taken a strategic decision to implement ANSSI's hardening guide. Meeting the criteria specified in ANSSI's above-mentioned Linux configuration guide has a positive impact on SEAPATHs security posture, and it makes it hardened enough to be consumed in an environment that is part of national security goals. Software projects are not required to adopt all recommendations of the ANSSI standards to be suited for production. Projects should carry out analyse their security context and implement the measures that are required in that given context.

Fuzzing

One of our observations during the audit was that several of SEAPATHs memory-unsafe third-party packages had not implemented fuzz testing into their testing suites. We consider this an impactful shortcoming, as fuzzing has been helpful in testing for classes of vulnerabilities specific to memory-unsafe languages including memory management and logical issues. Our high level assumption at the start of the audit was that these projects should undergo rigorous, continuous fuzz testing for a longer period of time before we consider them suitable for exposure to the national grid. To test our assumption, we wrote four fuzzers for the Pacemaker project and ran them on our own infrastructure to see if they caught any immediate issues. This found a few issues which we reported to the Pacemaker project. We then integrated Pacemaker into OSS-Fuzz such that the fuzzers can run continuously, and this found new issues which OSS-Fuzz disclosed directly to Pacemaker.

Pacemaker is a tool to manage remote cluster resources. SEAPATH uses Pacemaker in vm_manager which is a command-line tool to deploy, manage and delete servers in a SEAPATH cluster. It is operated by users with privileges to manage cluster resources, and security issues in vm manager and its underlying dependencies could result in an attacker escalating their privileges to a level where they can also manage cluster resources. As such, we consider it an important part of the SEAPATH ecosystem and a great place to apply fuzzing.

Our initial efforts found 3 issues in Pacemaker: Two memory corruptions and one logical bug. We reported these directly to the Pacemaker project privately. The Pacemaker maintainers fixed the three issues within a few days. We have included the findings from the fuzzers in the “Found issues”. All the issues found by the fuzzers we wrote are:

Issue #	Title	Severity	Fixed
ADA-SEAP-2024-8	Memory corruption in pacemaker	Moderate	Yes
ADA-SEAP-2024-9	Memory corruption in pacemaker	Moderate	Yes
ADA-SEAP-2024-10	Division by zero in Pacemaker	Moderate	Yes

Our work forms part of the fuzzing set up that we recommend that all critical third-party, memory-unsafe packages maintain. In this section we describe the full model of a comprehensive fuzz test suite. This model consists of three different parts:

1. High coverage: The fuzzers must have high coverage in the parts of the code that SEAPATH uses.
2. Continuity: The fuzzers must run regularly and continuously.
3. CI integration: The projects fuzzers should run in the CI on pull requests to catch bugs before they are merged in.

4. Findings must be triaged and fixed: When the fuzzers find an issue, their should be a response team with bandwidth to triage and potentially fix.

Supply-chain risk

SEAPATH maintains a significant supply-chain risk. To reiterate what we discussed earlier in the report, SEAPATHs purpose is to configure and package open source software with the current goal of using this in electric grid environments. In this section we discuss the impact on supply-chain security on SEAPATH and the areas we have looked at in detail in this audit.

In SEAPATHs case, the line between first-party software and third-party software is more blurry than other cases. To a certain degree, we understand that a goal of SEAPATH is to not develop its own runtime software and instead package it in a suitable way for national grid use cases. As such, third-party software is a core part of SEAPATH, and a solution as of now is to not implement any third-party software packages.

With that in mind, the security practices that SEAPATH maintains for itself must exist for its third-party components as well.

In our assessment of SEAPATHs third-party modules, we have focused on areas that could lead to security issues, rather than limiting our assessment to existing security issues. There are two reasons for this: First, some of SEAPATHs third-party modules are huge in code size and are far beyond the scope of this audit. Second, considering that SEAPATH is a project under development, we think it makes sense to look at factors that could lead to issues in the future as SEAPATH matures.

Our supply-chain assessment had two separate outcomes. First, we found specific areas where SEAPATHs supply-chain could be compromised to a degree that would affect SEAPATH users. We have listed these in the “Found issues” section and have also worked on remediating some of these. Second, we have made high-level conclusions about the long term risk of SEAPATHs supply-chain.

We draw on the SLSA projects supply-chain threat model which breaks the software development lifecycle up into three categories:

1. Source: The source code of the software package.
2. Build: The way and environment in which the software package is built.
3. Package: The way the software package is available to consumers.

The SDLC looks as such:

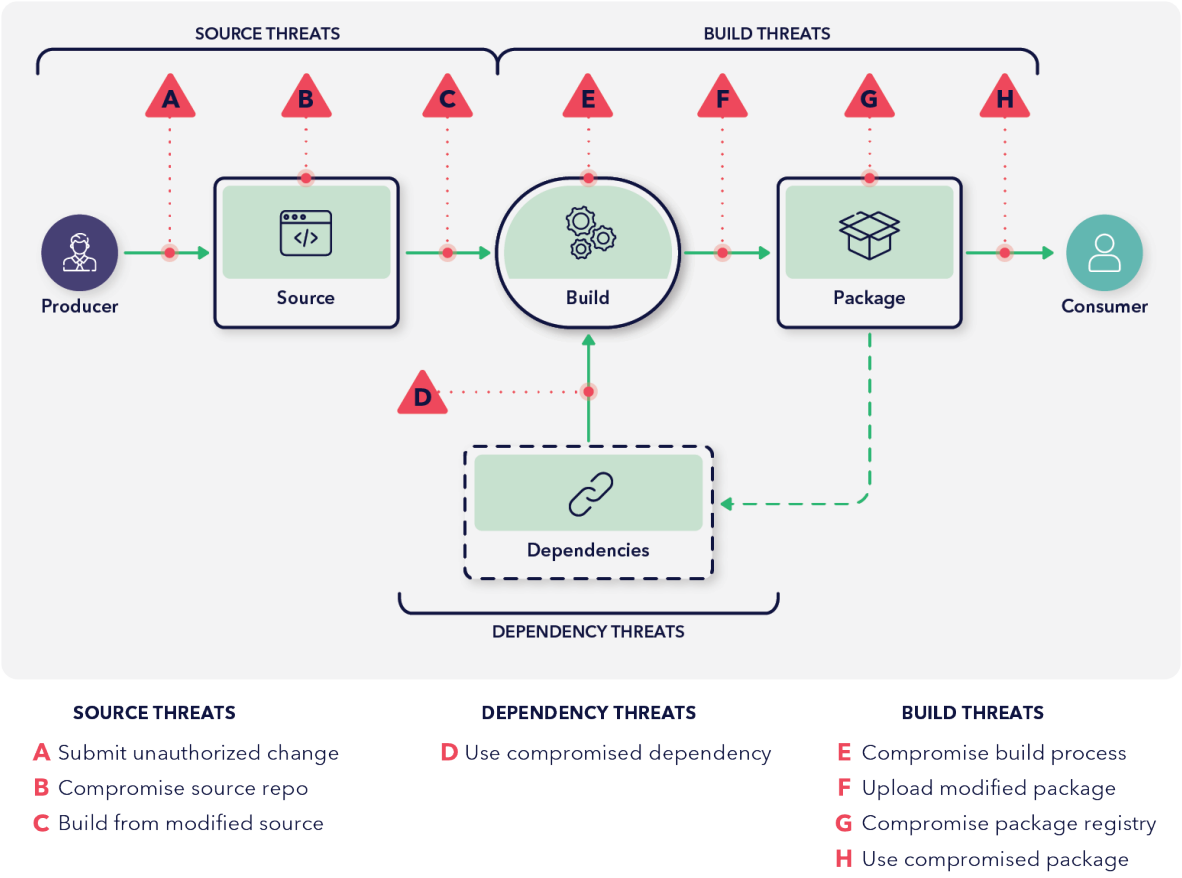

Figure 1: SDLC diagram

The SDLC has its supply-chain shown as “Dependencies” in the above diagram.

The software package maintains source code in a code repository such as GitHub. When the software package releases a new version, it first builds it in a build environment and then makes it available in a package repository from where consumers download it and invoke it.

Several things can go wrong in the process that can affect SEAPATH users, and SLSA’s examples of real world attacks describe attack vectors that also impact SEAPATHs third-party software: https://slsa.dev/spec/v1.0/threats-overview#real-world-examples. We rely on SLSAs threat model in our supply-chain assessment of SEAPATH.

SEAPATH Threat Model

An important prerequisite for reasoning about security in any system is a threat model. A threat model describes the security-relevant features of the system, its trust boundaries, the threats the system faces and who is able to impact the security of the system. When the audit commenced, we found that SEAPATH did not have a formal threat model, nor did the project have an aligned understanding of the scope of SEAPATHs security boundaries. Conducting a security assessment of a project without a threat model is challenging and will result in overlooking problems and risks. A threat model formalizes a higher level understanding of the system than its actual implementation, and a good and clear threat model will guide the projects maintainers to answer concrete questions about the security of the actual implementation. As such, before we can reason about SEAPATHs security, we need a reference to reason against. In addition, in SEAPATHs case, it must maintain a threat model to comply with the ISA-62443 security framework which includes specific requirements to the threat model.

As a disclaimer, we can see practical reasons for why the project has not considered its threat model. First and foremost, SEAPATH has not needed one until now; SEAPATH is still under heavy development and has a roadmap of security-optimizations to do before being production-ready: https://wiki.lfenergy.org/display/SEAP/SEAPATH+and+cybersecurity. Second, SEAPATH has not had a single official release, and as such, no production-users that depend on official releases can be affected by security issues. Finally, since SEAPATH is under heavy development, the exact target use case may lack and prevent an understanding of the security problems of the end user. Currently, SEAPATH adopters carry out their own threat modelling which they are unlikely to share for security reasons, and SEAPATH does not maintain a common threat model that cover all or most use cases.

In this section, we present the threat modelling we have done as part of this audit which is based on our own observations, our recommendations and feedback from the SEAPATH community. We strive to present the problems we faced in threat modelling SEAPATH, and we answer these problems with our own observations and recommendations. When highlighting the problems in threat modelling SEAPATH, we do not intend to put a negative light on the work that the SEAPATH community has been doing in terms of working on securing SEAPATH. Instead, the goal is to concretize the problems we have observed in reasoning

The first question we need to answer is: What is in scope of SEAPATHs security boundaries? And perhaps equally importantl, what is not? During the audit we did not get a clear answer to these two questions from the community. Here, we consider different answers to this question, but to do that, we will first take a look at what SEAPATH is and does.

Currently, components running workloads in electric grid substations rely on hardware-based implementations. This makes it expensive to replace and update and slow to develop. Substations consist of bays. Each bay is connected to a high voltage power line and has the purpose of monitoring, protecting

and controlling the power line. Currently, bays are put together by multiple different physical components. SEAPATH allows users to replace these physical components with software applications, and it does that by implementing a platform with a hypervisor and virtualization that can handle substation workloads in terms of performance, availability and security. SEAPATH builds this platform fully on open source software and relies mainly on the following open-source projects:

1. Yocto Project: For configuring SEAPATHs components.
2. Pacemaker: For managing clusters of hypervisors.
3. Ceph: For data replication.
4. KVM: For virtualization.
5. Libvirt: For virtualization.
6. Open vSwitch: For a fully virtualized switch architecture.
7. FAI: For building images.
8. Debian: for building images.

We can illustrate the intended use case of SEAPATH below, with the traditional substation bay architecture on the left, and the SEAPATH-managed bay architecture on the right:

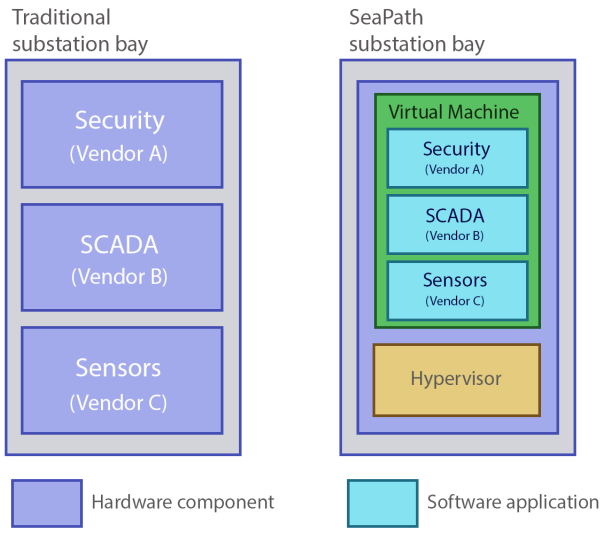

Figure 2: SEAPATH in substation bay

Security scope

From our observation, SEAPATHs main function in delivering this platform is to configure production-grade open-source projects. As such, there are different ways to look at what SEAPATH is and what is not SEAPATH. A fundamental question is whether SEAPATH is only the configuration layer, or if SEAPATH is the output that the configuration layer produces. The vast majority of code in SEAPATH are

configuration files, but the artifact that SEAPATH users consume includes software implemented and maintained by third parties. As such, one option is to consider only the configuration files within scope of SEAPATHs security boundary, while another option is to include the software packages that SEAPATH relies on. There is no distinction between the usage between these two options. In both cases, user build their Linux distributions based on the SEAPATH configuration, deploy the output to their servers and run them in the substation. The question here is, whether SEAPATHs threat model includes the third-party software and in that case which. To specify the distinction, let's consider a scenario where any of the software that SEAPATH relies on has a security vulnerability that affects users, what are the implications of each of the two options? In such a case, several things need to happen:

1. The vulnerability needs to be found.
2. The security response team of the third-party software must be aware of the vulnerability.
3. The security response team of the third-party software must fix the vulnerability.
4. The third-party project maintainers must issue a release containing the patch.
5. SEAPATH users must be aware of the release containing the patch.
6. SEAPATH users must update the version of the third-party software.

When using SEAPATH with Yocto, users can complete steps 2-4 without interaction with upstream maintainers; Users can do it themselves or engage a third-party vendor to carry out the work. All six steps must complete for SEAPATH debian users. Currently, SEAPATH Debian does not define an SLA or user contract, and as such does not have a formal and legal requirement to follow the above process.

With both options from above, all of these six steps must be completed for the SEAPATH user to not be vulnerable to a given security vulnerability.

Let's consider the risk to SEAPATH users related to finding, fixing and patching security issues and SEAPATH users updating on their end. It is a risk if none of these steps happen. For example, if vulnerabilities exist in the third-party software, but that they are not found. It is a risk if any of these steps fail and the entire process above fails. This could for example be the case if a vulnerability is found, fixed and released, but the SEAPATH user fails to update to the latest release. The less efficient any of these steps are, the higher the risk is that users can be compromised by vulnerabilities. For example, if the third-party project takes too long to release security fixes after fixing them, the SEAPATH user might be compromised by another, malicious threat actor who is more efficient. As such, this process must happen, and it must be efficient for all runtime-dependent software in the SEAPATH stack. SEAPATH Yocto users are less at risk from delays by third-party software maintainers, since Yocto users can complete steps 2-4 from the above list.

Our thoughts on scope

Here we present our observations on the scope of the security of third-party software in SEAPATH. The primary goal of SEAPATH is to package existing open-source software in such a way that users can use virtualisation on remote hardware and deploy Linux distributions on top of that virtualisation. In our understanding, an inherent element of this goal is that SEAPATH should not maintain this third-party software, nor should it implement technical features that are necessary at runtime. In fact - and this is again according to our understanding of the goals of SEAPATH - SEAPATH does not need to solve its problems in a specific way; It is sufficient to combine independent software in such a way that it achieves its goal of enabling management of remote servers. As part of this inherent goal is that the software that SEAPATH packages must also handle its own security properly with best practices, and we consider it out of scope for SEAPATH to actively maintain the security of the software it packages. What we consider important for SEAPATHs security posture is that the third-party projects maintain their security sufficiently to decrease their risk for SEAPATH users. With the above example of finding, mitigating and fixing security issues and releasing patches, it is important to SEAPATH that the third-party software it packages does this efficiently. As such, SEAPATHs critical third-party software packages should follow similar security practices as SEAPATH itself follows.

SEAPATH threat actors

A threat actor is a person or group of people that can impact the security of SEAPATH. Because a threat actor can impact SEAPATHs security does not mean that they will do so negatively. The goal of enumerating SEAPATHs threat actors is to draw a line between who is trusted and who is not trusted. In our threat model enumeration, we do not consider impersonated threat actors; For example, trusted users can be targets of social engineering campaigns which can lead to attackers stealing the victims credentials and obtaining the same privileges as the victim. In the case of a sudo user, an untrusted user can thereby achieve sudo privileges if they launch a successful campaign. In our threat actor enumeration, threat actors are not impersonated.

SEAPATH Artifact builder

This is a user whose task is to build SEAPATH artifacts. From internal discussions with the SEAPATH team, we understand that SEAPATH intends to distribute source code releases corresponding to tags. As such, this threat actor's job is to fetch the latest source code release and build new artifacts that they will either deploy to the substation themselves or have another team member deploy it.

Building artifacts is not the highest obtainable privilege in a SEAPATH use case. In fact, the user does not require permissions to a SEAPATH cluster to do their job. A question to SEAPATH is whether this user should be able to carry out tasks with higher privileges in the cluster. For example, should SEAPATH have hardening mechanisms against malicious activity in the cluster by this user, or does SEAPATH

trust this threat actor?

Let's consider a case where the SEAPATH artifact builder has turned evil; They have received compensation from a foreign agency or have been motivated by political beliefs to hurt users of the substation to which SEAPATH is deployed. This threat actor could clone the latest release tag and modify it before building the artifacts which would result in tampered artifacts. Now, if the organization managing the substation does not have a way to verify that the artifact has not been tampered with, then essentially the SEAPATH artifact builder threat actor could tamper the artifact in such a way, that they have other permissions such as crashing the server or running their own code on it. At a high level, this type of attack from the SEAPATH Artifact Builder could look like this:

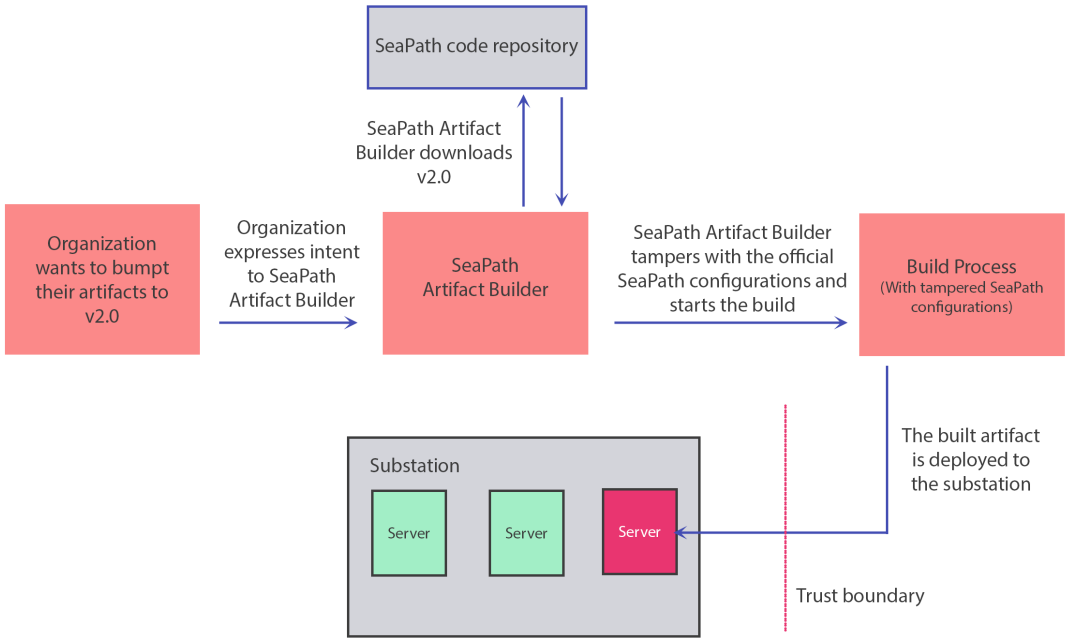

Figure 3: SEAPATH artifact builder

Ultimately it is up to the SEAPATH project to decide whether the SEAPATH Artifact Builder should have permissions to manage the server. If this is not the case, then SEAPATH should enforce a check at the trust boundary in the above diagram before deploying artifacts to the substation. At the first step of the diagram, the organization has the intent to deploy the official v2.0 release. At the trust boundary, the organization should be able to verify that the built artifact is the official v2.0 release. During this audit, we have not found such a check in SEAPATH. We recommend implementing this.

SEAPATH Cluster Admin

Users need an admin to manage the resources of the cluster. This includes deploying new servers and updating or decommissioning existing servers. We consider this user the highest privileged user of a

standard SEAPATH deployment, and we consider this user fully trusted in the context of SEAPATH. If a feature or bug in SEAPATH allows only this threat actor to damage the substation servers, then it is not a security issue in SEAPATH, since this threat actor should have these permissions. In fact, it might be a breach of security, if this user cannot manage clsuter resources, for example if they are prevented by an attacker from taking down servers in the substation.

We consider a trust boundary to exist between this user and non-SEAPATH-managed servers in the substation. The SEAPATH Cluster Admin should not be able to escalate privileges from SEAPATH-managed servers to other hardware or software in the substation. That being said, we don't consider it within the scope of SEAPATH to harden this trust boundary. This is in scope of other devices in the substation to be hardened against. Below, we demonstrate where the trust boundary exists:

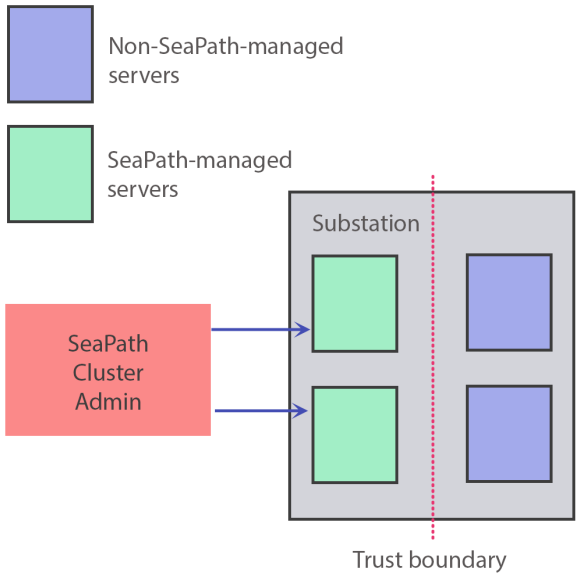

Figure 4: SEAPATH cluster admin

Local attacker

A local attacker is one who has physical or logical access to the substation but not to the SEAPATH servers. This could be anyone that fits that criteria, but for simplicity we exemplify this with substation maintenance staff. Substations need maintenance of other hardware and equipment than the SEAPATH servers. This includes inspections and cleaning, condition and performance evaluations, compliance inspections and document and record keeping. As such, staff members that should not be able to influence the security of SEAPATH servers are in close physical proximity to the SEAPATH servers in the substation. These staff members can attempt to transcend their role and compromise the SEAPATH servers by way of physical access. This includes both physically damaging the equipment and accessing the software running in the servers by way of physical ports. We have not found that SEAPATH specifies

the trust boundaries of this threat actor, so here we include our thoughts on where they should be defined. We welcome the SEAPATH project to disagree and steer away from these. Note that the substation maintenance staff threat actor covers other threat actors with similar work characteristics and even malicious intruders. These characteristics are that they are physically or virtually present present on the substation but have no reason in their job function to manage SEAPATH servers on the substation. From SEAPATHs perspective, it is not important whether the threat actor has obtained their privilege maliciously or legitimately. We consider five scenarios relevant for this threat actor:

1. Threat actor has neither physical nor logical access to a SEAPATH server
2. Threat actor has physical but not logical access to a SEAPATH server
3. Threat actor has logical access but not physical access to a SEAPATH server. They have not been authenticated.
4. Threat actor has physical and logical access to a SEAPATH server. They have not been authenticated.
5. Threat actor has physical and logical access to a SEAPATH server. They have been authenticated and can access the file system of the server.

Scenario 1: Threat actor has neither physical nor logical access to a SEAPATH server
The threat actor can neither touch the SEAPATH server nor reach it via a local WiFi. At this point, the threat actor has the lowest privilege position it can have in an attempt to compromise the SEAPATH server. Any damage the attacker can do to the user's SEAPATH deployment is outside of the scope of SEAPATHs threat model, since they are limited to indirect physical damage. SEAPATHs software boundaries do not need to be hardened against attacks from the threat actor at this permission level. The trust boundary for this position is physical in the form of a wall or a locked cabinet.

Scenario 2: Threat actor has physical but not logical access to a SEAPATH server
The threat actor can physically touch the SEAPATH server but has no way to access the underlying software. In this position, the threat actor cannot reach the authentication interface. Any damage that the threat actor can cause to SEAPATH at this point is outside of the scope of SEAPATH, since it is limited to direct physical damage.

Scenario 3: Threat actor has logical access but not physical access to a SEAPATH server. They have not been authenticated
The threat actor has access to all authentication interfaces via a local WiFi network. In this position we assume that the substation staff should not be able to authenticate. As such, SEAPATH should withstand pre-authenticated attacks and authentication bypasses. For example, the threat actor should not be able to hog all connections, so that legitimate users cannot log in. Or, the threat actor should not be able to crash the server before being authenticated. In this position, the threat actor faces the first trust boundary which is the authentication:

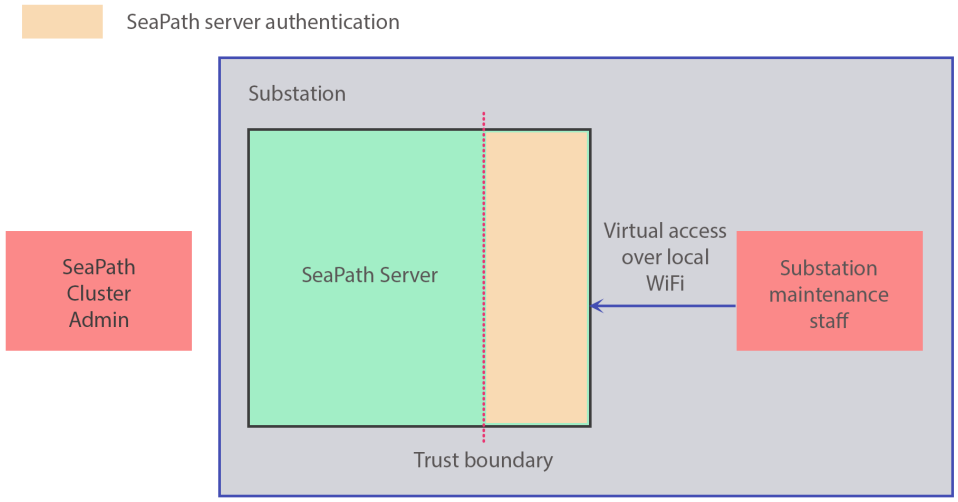

Figure 5: SEAPATH local attacker

Scenario 4: Threat actor physical and logical access to a SEAPATH server. They have not been authenticated
This is similar to scenario 3, except that the substation staff member also has physical access to the machine. In this position, the threat actor is able to launch pre-authentication attacks, authentication-bypass attacks and physical attacks on the hardware. Physical attacks on the hardware are not in scope of SEAPATHs threat model. SEAPATHs threat model must withstand pre-authenticated attacks and authentication bypasses.

Scenario 5: Threat actor has physical and logical access to a SEAPATH server. They have been authenticated and can access the file system of the server
In this position, the threat actor has made an effort to compromise a local SEAPATH server in the substation and now has permission to access it. They could have used a number of techniques to get themselves in this position including some that are outside of the scope of SEAPATHs threat model. For example, the threat actor could have stolen credentials of another user or accessed an open connection left open for a minute by another user. At this point, we consider the threat actor to have permissions to modify the server including its filesystem. We consider any direct damage to the server a compromise of SEAPATHs threat model even at this point. As such, SEAPATH should withstand attacks like:

Spinning up local processes that exhaust resources and deny the server of service.
Installing malware. With the ability to modify the file system, the threat actor could attempt to download malware and install it.
Preventing other users from accessing the server. For example, the threat actor could attempt to disable remote connections such that other users could not control or recover the server.

Threat actors with the current privilege level are likely to attempt to escalate their position by targeting other users of the cluster that manage the machine. This includes users with privileges to change the current and other SEAPATH-managed servers:

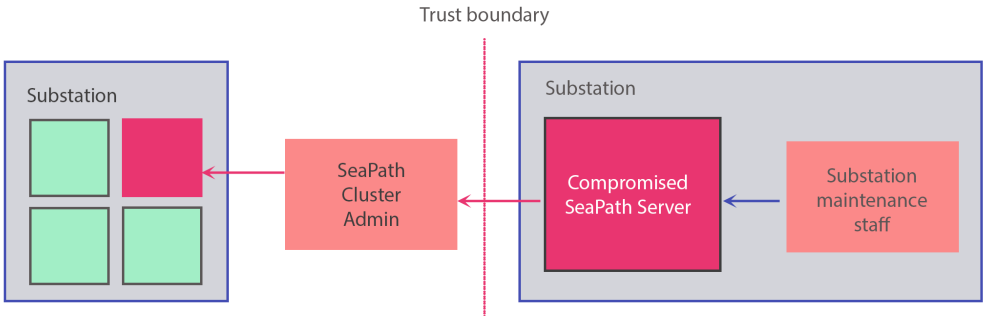

Figure 6: SEAPATH local attacker escalation

Nation-state-backed actors

Because SEAPATH is intended to be used for critical infrastructure, we expect it will undergo pressure from well-funded agencies. In essence, nation-state-backed threat actors are fully untrusted users that do not have any privileges in a given SEAPATH deployment. They do have the characteristic that they have more funds behind them which allow them to carry out targeted attacks. The difference between this threat actor and non-nation-state-backed threat actors is that this threat actor is able - and likely - to continue a campaign against SEAPATH, both in case they do have ways to compromise SEAPATH users and in case they do not. Additionally, this threat actor is highly skilled in many different areas and can combine high levels of technical skills to coordinate targeted campaigns. Recently, a CISA advisory included information about recent attempts to compromise critical infrastructure operational technology: https://www.cisa.gov/news-events/alerts/2024/05/01/cisa-and-partners-release-fact-sheet-defending-ot-operations-against-ongoing-pro-russia-hacktivist. Once SEAPATH is widely deployed in production in substations, we expect the same and similar groups will start to probe SEAPATH for weaknesses.

SEAPATH maintainers

A SEAPATH maintainer is a maintainer of a SEAPATH-hosted GitHub repository. We have not found any attempted compromise by any maintainers during this audit, nor do we suspect anyone in particular, however, a maintainer can turn evil and deliberately attempt to hurt SEAPATH users. This could happen for a number of reasons such as political beliefs, threats, personal grudges, or they could receive monetary rewards from the nation-state-backed threat actor directly or through a proxy, and SEAPATHs threat model should account for the threat from an actual evil maintainer. There is one fundamental question SEAPATHs threat model should answer in this regard which is how much damage an evil maintainer should be able to do. For example, if a maintainer wants to hurt users, which systemic

mechanisms should be in place to protect users from this maintainer? In our opinion, one single maintainer should not be able to directly impact the security of SEAPATH users without scrutiny of other maintainers. However, there is a scenario where more maintainers than a single one have turned evil, and they could support each other in a coordinated attack. SEAPATHs threat model should define acceptable mitigation against such cases too.

SEAPATH contributors

SEAPATH is an open-source project which anyone with a GitHub account can contribute to. The person behind a given GitHub account can easily remain anonymous, and as such SEAPATH accepts contributions from anonymous users. Design-wise, this is part of the idea behind SEAPATH with the project being open-source. Contributors may have malicious intent towards certain SEAPATH users and attempt to compromise SEAPATH to lower the effort necessary or entirely enable compromises of their targets.

Found issues

In this section we present the issues we found during the audit. These findings are split into two different categories: 1) Findings in breach of SEAPATHs threat model and 2) Findings related to complying with the ISA/IEC-62443 standards for industrial automation and control systems (IACS). The findings in breach of SEAPATHs threat model cover any potential attack vector discussed in the threat model and vary in criticality, severity and exploitability. The findings related to ISA/IEC-62443 standards are all informational and we consider these good practices but not critical or urgent to fix. Some of these findings require ample work to implement properly, and the timeline for fixing them depends on when the SEAPATH project aims to comply with the ISA/IEC 62443 standards. Therefore, we consider all findings related to ISA/IEC 62443 informational.

#	ID	Title	Severity	Fixed
1	ADA-SEAP-2024-1	Pygments version vulnerable to three CVEs	Moderate	No
2	ADA-SEAP-2024-2	Dependencies are pinned by tags	Informational	No
3	ADA-SEAP-2024-3	Over-permitted Github Tokens in SEAPATH workflows	Informational	Yes
4	ADA-SEAP-2024-4	SEAPATH maintainers can merge without approval from other maintainers	Moderate	Yes
5	ADA-SEAP-2024-5	Third-party project allows maintainers to merge code without maintainer approvals	Low	No
6	ADA-SEAP-2024-6	Third-party project: Changes to code approved code in pull requests do not require re-approval	Low	No
7	ADA-SEAP-2024-7	SEAPATH: Changes to code approved code in pull requests do not require re-approval	Moderate	Yes
8	ADA-SEAP-2024-8	Memory corruption in third-party project	Moderate	Yes
9	ADA-SEAP-2024-9	Memory corruption in third-party project	Moderate	Yes
10	ADA-SEAP-2024-10	Division by zero in Pacemaker	Moderate	Yes

#	ID	Title	Severity	Fixed
11	ADA-SEAP-2024-11	Missing test coverage	Informational	No
12	ADA-SEAP-2024-12	Add documentation on how admins can prevent users from seeing other users' processes	Informational	No
13	ADA-SEAP-2024-13	Add two-factor authentication	Informational	No
14	ADA-SEAP-2024-14	Make ANSSI compliance clear in test suite	Informational	No

ISA/IEC-62443 issues

#	ID	Title
19	ADA-SEAP-2024-19	Missing Role Overview
20	ADA-SEAP-2024-20	Missing Security Training for Maintainers
21	ADA-SEAP-2024-21	Lack Scoping components of SEAPATH that must conform to ISA-62443
22	ADA-SEAP-2024-22	Lack of Documentation of Intended Controls for Source Code Protection
23	ADA-SEAP-2024-23	SEAPATH must ensure all security-related issues are fixed in releases
24	ADA-SEAP-2024-24	Verifying SEAPATHs compliance before cutting release
25	ADA-SEAP-2024-25	Iterative improvements to security-related processes
26	ADA-SEAP-2024-26	Lack of documentation about the system in which SEAPATH is used and/or deployed
27	ADA-SEAP-2024-27	Lack of threat model
28	ADA-SEAP-2024-28	Lack of security requirements documentation
29	ADA-SEAP-2024-29	Lack of Secure design principles and Security Review processes
30	ADA-SEAP-2024-30	Local access timeout
31	ADA-SEAP-2024-31	Add pre-authentication banner for SSH and local access
32	ADA-SEAP-2024-32	Add documentation or ability to whitelist IP addresses

Pygments version vulnerable to three CVEs

Severity	Low
Status	Reported
id	ADA-SEAP-2024-1
Component	SEAPATH module documentation

SEAPATHs module documentation uses a vulnerable version of the Pygment library which is vulnerable to three known vulnerabilities. Pygments >= 1.5, < 2.7.4 contains the following three CVE's:

1. CVE-2021-20270
2. CVE-2022-40896
3. CVE-2021-27291

SEAPATH uses Pygments 2.4.0 or greater:

https://github.com/SEAPATH/ansible/blob/2bbf1da1935a6c061d8b1f9af53cf25b000b2eca/module_documentation_sc
L7

```
1 jinja2
2 PyYAML
3 rstcheck
4 sphinx
5 sphinx-notfound-page
6 Pygments >= 2.4.0
7 ansible >= 2.9,<2.10
```

The impact of this is low since the module documentation does not receive untrusted input at runtime. The only attack vector is from one code contributor to another. All three CVE's are denial of service vulnerabilities, and as such, the end impact is that one code contributor can cause denial of service of another code contributor's development environment. While this theoretically can impact users, it is unlikely to do so in practice, since common sense will prevent such an attack. The vulnerable part of SEAPATH is used to generate HTML documentation about the ansible modules in the library subdirectory (https://github.com/SEAPATH/ansible/tree/main/library). An attacker could commit code to the SEAPATH library subdirectory that would trigger the vulnerabilities when another person would generate the documentation.

In the current case, the three CVE's are denial of service issues, and as such they have limited objectives to an attacker. They could have an impact if a user was generating documentation on a production

machine and the attacker could drain the machines resources. That being said, the issues are a testament to a lack of hardening against security issues in the module documentation generator, and if vulnerabilities with a higher impact were discovered in the future in Pygments or other dependencies to the module generator, SEAPATH would by default be vulnerable to them.

To mitigate this for the future, we recommend installing an automatic dependency updater across the entire SEAPATH ecosystem.

Dependencies are pinned by tags

Severity	Informational
Status	Reported
id	ADA-SEAP-2024-2
Component	SEAPATH Ansible workflows

SEAPATH follows a configuration practice where dependencies in GitHub workflows and Dockerfiles can be referenced with tags. This is generally not recommended practice, since tags are non-immutable, meaning that the content that the tag refers to can change. Referencing dependencies by tags is not a security vulnerability, i.e. there is no direct way to exploit this, but it is an unnecessary risk that can be easily mitigated.

We have seen dependencies pinned by tags in a limited manner across the SEAPATH ecosystem, and many of the cases are to trusted vendors. An example is of the GitHub hosted checkout action:

https://github.com/SEAPATH/ansible/blob/bebb93510f7b5ea80f15bae92a502ea963a07a94/.github/workflows/ansible-lint-debian-weekly.yml#L20

```
1  name: Ansible Lint weekly Debian
2
3  env:
4    WORK_DIR: /tmp/seapath_ci_${{ github.run_id }}_${{ github.run_attempt
       }}_${{ github.sha }}
5
6  on:
7    schedule:
8      - cron: '30 22 * * 6'
9    workflow_call:
10   workflow_dispatch:
11
12 jobs:
13   ansible-lint:
14     runs-on: self-hosted
15     steps:
16       - uses: actions/checkout@v2
17         with:
18           ref: debian-main
19       - name: Initialize sources
20         run: mkdir ${{ env.WORK_DIR }}; cd ${{ env.WORK_DIR }};
21              git clone -q --depth 1 -b main https://github.com/seapath/
                    ci ci;
```

```
22              echo "Sources downloaded successfully";
23              ci/ansible-lint.sh init;
24
25        - name: Lint
26          run: cd ${{ env.WORK_DIR }};
27               ci/ansible-lint.sh lint;
28
29        - name: Clean
30          if: always()
31          run: rm -rf $WORK_DIR;
```

This is an official GitHub action which is maintained at https://github.com/actions/checkout and is a high-cost target to compromise for an attacker compared to single-maintainer projects with fewer resources and less bandwidth.
As such, the risk to SEAPATH is not whether it uses any easily compromisable repositories at this point during the audit, but rather that it does not prevent tampering of dependencies as part of its development lifecycle. A malicious threat actor could introduce a non-vulnerable, legitimate dependency to SEAPATH that they control and introduce a vulnerability to it later. As such, the threat actor could introduce a vulnerability to SEAPATHs Dockerfiles and workflow files without SEAPATH noticing it.

We recommend adding CI tests that reject pull requests that introduce dependencies to Dockerfiles and workflow files that are not pinned by a hash.

Over-permitted Github Tokens in SEAPATH workflows

Severity	Informational
Status	Fixed
id	ADA-SEAP-2024-3
Component	SEAPATH module documentation

SEAPATH defines neither job-level nor top-level permissions in:

- https://github.com/seapath/build_debian_iso/blob/a1ddc82ba365e4cff34bc825a36328a87452ce48/.github/workflows/shellcheck-weekly.yml
- https://github.com/seapath/build_debian_iso/blob/9282acf82c60fa9c650dcb39062388df8d47f577/.github/workflows/shellcheck.yml

If SEAPATHs has set its Github Token to “permissive”, this could allow for attacks on SEAPATHs CI infrastructure.

SEAPATH maintainers can merge without approval from other maintainers

Severity	Moderate
Status	Fixed
id	ADA-SEAP-2024-4
Component	SEAPATH SDLC

SEAPATHs meta-SEAPATH repository allows maintainers to merge pull requests without approval from other maintainers. This includes the ability to merge their own pull requests.

This lowers the barrier for maintainers to turn evil and merge malicious code into the meta-seapath repo.

Reference: https://github.com/seapath/meta-seapath/pull/127

Third-party project allows maintainers to merge code without maintainer approvals

Severity	Low
Status	Reported
id	ADA-SEAP-2024-5
Component	SEAPATH dependencies SDLC

The Pacemaker project allows maintainers to merge pull requests without approval from other maintainers. This includes the ability to merge their own pull requests. This has the risk for SEAPATH in case of an evil-maintainer attack, where a maintainer deliberately wants to hurt SEAPATH users and merges in code to do that. In such a case, there is no defence mechanism to stop that, such as a second-maintainer review or a single-maintainer review from a maintainer that did not make a pull request.

In context of SEAPATH being used in a national electricity grid, gaining trust as a contributor and even achieving maintainer status is a reasonably low cost to a state-funded threat actor, if the potential outcome is to add malicious code to Pacemaker in such a way that SEAPATH users can be compromised. The threat actor could work on the project for years to gain trust and understand the system and its practices and then carefully add malicious code over time or by way of a single code commit. Without another reviewer as a gatekeeper, the bar to committing malicious code would be low.

Alternatively, a malicious threat actor could attempt to convince a Pacemaker maintainer to turn against SEAPATH users and deliberately merge in malicious code. Open source software has previously been weaponized in such a manner during war, and we recommend that SEAPATH hardens its dependencies in the same way.

As a disclaimer, we are making this recommendation from the perspective of SEAPATHs security posture. This is an observation based on SEAPATHs threat model only.

References:
- https://github.com/ClusterLabs/pacemaker/pull/3433
- https://github.com/ClusterLabs/pacemaker/pull/3431
- https://github.com/ClusterLabs/pacemaker/pull/3427

Third-party project: Changes to code approved code in pull requests do not require re-approval

Severity	Low
Status	Reported
id	ADA-SEAP-2024-6
Component	SEAPATH dependencies SDLC

The Pacemaker project allows contributors to make changes to pull requests after they are approved without requiring re-approval before merging.

This allows for a threat actor to sneak in malicious code into pull requests after a legitimate and non-malicious pull request has been approved. A sequence of steps for this type of attack would look as such:

A malicious threat actor makes a pull request containing legitimate code contributions.
A maintainer approves the pull request.
The threat actor makes a commit containing malicious code.
The maintainer does not notice the most recent commit and merges the pull request that contains malicious code.

References:
- https://github.com/ClusterLabs/pacemaker/pull/3419
- https://github.com/ClusterLabs/pacemaker/pull/3395

SEAPATH: Changes to code approved code in pull requests do not require re-approval

Severity	Moderate
Status	Fixed
id	ADA-SEAP-2024-7
Component	SEAPATH SDLC

The SEAPATH meta-SEAPATH repository allows contributors to make changes to pull requests after they are approved without requiring re-approval before merging.

This allows for a threat actor to sneak in malicious code into pull requests after a legitimate and non-malicious pull request has been approved. A sequence of steps for this type of attack would look as such:

A malicious threat actor makes a pull request containing legitimate code contributions.
A maintainer approves the pull request.
The threat actor makes a commit containing malicious code.
The maintainer does not notice the most recent commit and merges the pull request that contains malicious code.

References:
- https://github.com/seapath/meta-seapath/pull/159
- https://github.com/seapath/meta-seapath/pull/98

Memory corruption in third-party project

Severity	Moderate
Status	Reported
id	ADA-SEAP-2024-8
Component	SEAPATH cluster management dependency

This vulnerability was found by the fuzzers we wrote for the Pacemaker project.

SEAPATHs third-party dependency, Pacemaker, is vulnerable to a heap-based buffer overflow in `pcmk__time_format_hr` on line 1961 below:

https://github.com/ClusterLabs/pacemaker/blob/9d0f30818975ba4b949454a8e0b0c2e76fb02a0a/lib/common/iso860
L1969

```
1943      while ((format[scanned_pos]) != '\0') {
1944          mark_s = strchr(&format[scanned_pos], '%');
1945          if (mark_s) {
1946              int fmt_len = 1;
1947
1948              fmt_pos = mark_s - format;
1949              while ((format[fmt_pos+fmt_len] != '\0') &&
1950                  (format[fmt_pos+fmt_len] >= '0') &&
1951                  (format[fmt_pos+fmt_len] <= '9')) {
1952                  fmt_len++;
1953              }
1954              scanned_pos = fmt_pos + fmt_len + 1;
1955              if (format[fmt_pos+fmt_len] == 'N') {
1956                  nano_digits = atoi(&format[fmt_pos+1]);
1957                  nano_digits = (nano_digits > 6)?6:nano_digits;
1958                  nano_digits = (nano_digits < 0)?0:nano_digits;
1959                  sprintf(&nanofmt_s[1], ".%ds", nano_digits);
1960              } else {
1961                  if (format[scanned_pos] != '\0') {
1962                      continue;
1963                  }
1964                  fmt_pos = scanned_pos; /* print till end */
1965              }
1966          } else {
1967              scanned_pos = strlen(format);
1968              fmt_pos = scanned_pos; /* print till end */
1969          }
```

The issue was fixed by the Pacemaker project in https://github.com/ClusterLabs/pacemaker/commit/d8de867b68777ee2ed284ff0e6257c0bf83b82cf.

Memory corruption in third-party project

Severity	Moderate
Status	Reported
id	ADA-SEAP-2024-9
Component	SEAPATH cluster management dependency

This vulnerability was found by the fuzzers we wrote for the Pacemaker project.

SEAPATHs third-party dependency, Pacemaker, is vulnerable to a heap-based buffer overflow in `parse_date` on line 891 below:

https://github.com/ClusterLabs/pacemaker/blob/9d0f30818975ba4b949454a8e0b0c2e76fb02a0a/lib/common/iso8601.c#L891-L900

```
891     while ((format[scanned_pos]) != '\0') {
892         mark_s = strchr(&format[scanned_pos], '%');
893         if (mark_s) {
894             int fmt_len = 1;
895 
896             fmt_pos = mark_s - format;
897             while ((format[fmt_pos+fmt_len] != '\0') &&
898                 (format[fmt_pos+fmt_len] >= '0') &&
899                 (format[fmt_pos+fmt_len] <= '9')) {
900                 fmt_len++;
901             }
902             scanned_pos = fmt_pos + fmt_len + 1;
903             if (format[fmt_pos+fmt_len] == 'N') {
904                 nano_digits = atoi(&format[fmt_pos+1]);
905                 nano_digits = (nano_digits > 6)?6:nano_digits;
906                 nano_digits = (nano_digits < 0)?0:nano_digits;
907                 sprintf(&nanofmt_s[1], ".%ds", nano_digits);
908             } else {
909                 if (format[scanned_pos] != '\0') {
910                     continue;
911                 }
912                 fmt_pos = scanned_pos; /* print till end */
913             }
914         } else {
915             scanned_pos = strlen(format);
916             fmt_pos = scanned_pos; /* print till end */
917         }
```

The issue was fixed by the Pacemaker project in https://github.com/ClusterLabs/pacemaker/commit/e70c5412ca1ab56370aa3ad4c84a110a3bdeb131.

Division by zero in third-party project

Severity	Moderate
Status	Reported
id	ADA-SEAP-2024-10
Component	SEAPATH cluster management dependency

Pacemaker is prone to a division by zero when parsing an ISO8601 numeric value from a char pointer into an int.

The vulnerable API is for example used when parsing an interval:

https://github.com/ClusterLabs/pacemaker/blob/605895685d1d234a871a912f77477ee1c8216dcc/lib/common/utils.c#L525-L551

```
525 crm_parse_interval_spec(const char *input)
526 {
527     long long msec = -1;
528 
529     errno = 0;
530     if (input == NULL) {
531         return 0;
532 
533     } else if (input[0] == 'P') {
534         crm_time_t *period_s = crm_time_parse_duration(input);
535 
536         if (period_s) {
537             msec = 1000 * crm_time_get_seconds(period_s);
538             crm_time_free(period_s);
539         }
540 
541     } else {
542         msec = crm_get_msec(input);
543     }
544 
545     if (msec < 0) {
546         crm_warn("Using 0 instead of '%s'", input);
547         errno = EINVAL;
548         return 0;
549     }
550     return (msec >= G_MAXUINT)? G_MAXUINT : (guint) msec;
551 }
```

Here, `crm_parse_interval_spec` calls into `crm_time_parse_duration` on line 534. `crm_time_parse_duration` in turn calls into `parse_int` on line 1139:

https://github.com/ClusterLabs/pacemaker/blob/9d0f30818975ba4b949454a8e0b0c2e76fb02a0a/lib/common/iso8601.c#L1100-L1139

```
1099 crm_time_t *
1100 crm_time_parse_duration(const char *period_s)
1101 {
1102     gboolean is_time = FALSE;
1103     crm_time_t *diff = NULL;
1104
1105     if (pcmk__str_empty(period_s)) {
1106         crm_err("No ISO 8601 time duration given");
1107         goto invalid;
1108     }
1109     if (period_s[0] != 'P') {
1110         crm_err("'%s' is not a valid ISO 8601 time duration "
1111                 "because it does not start with a 'P'", period_s);
1112         goto invalid;
1113     }
1114     if ((period_s[1] == '\0') || isspace(period_s[1])) {
1115         crm_err("'%s' is not a valid ISO 8601 time duration "
1116                 "because nothing follows 'P'", period_s);
1117         goto invalid;
1118     }
1119
1120     diff = crm_time_new_undefined();
1121     diff->duration = TRUE;
1122
1123     for (const char *current = period_s + 1;
1124          current[0] && (current[0] != '/') && !isspace(current[0]);
1125          ++current) {
1126
1127         int an_int = 0, rc;
1128
1129         if (current[0] == 'T') {
1130             /* A 'T' separates year/month/day from hour/minute/seconds.
                    We don't
1131              * require it strictly, but just use it to differentiate
                    month from
1132              * minutes.
1133              */
1134             is_time = TRUE;
1135             continue;
1136         }
1137
1138         // An integer must be next
1139         rc = parse_int(current, 10, 0, &an_int);
```

`parse_int` is vulnerable to a floating point exception on line 1064:

https://github.com/ClusterLabs/pacemaker/blob/9d0f30818975ba4b949454a8e0b0c2e76fb02a0a/lib/common/iso8601.c#L1033-L1071

```
1033 static int
1034 parse_int(const char *str, int field_width, int upper_bound, int *
         result)
1035 {
1036     int lpc = 0;
1037     int offset = 0;
1038     int intermediate = 0;
1039     gboolean fraction = FALSE;
1040     gboolean negate = FALSE;
1041 
1042     *result = 0;
1043     if (*str == '\0') {
1044         return 0;
1045     }
1046 
1047     if (str[offset] == 'T') {
1048         offset++;
1049     }
1050 
1051     if (str[offset] == '.' || str[offset] == ',') {
1052         fraction = TRUE;
1053         field_width = -1;
1054         offset++;
1055     } else if (str[offset] == '-') {
1056         negate = TRUE;
1057         offset++;
1058     } else if (str[offset] == '+' || str[offset] == ':') {
1059         offset++;
1060     }
1061 
1062     for (; (fraction || lpc < field_width) && isdigit((int)str[offset])
         ; lpc++) {
1063         if (fraction) {
1064             intermediate = (str[offset] - '0') / (10 ^ lpc);
1065         } else {
1066             *result *= 10;
1067             intermediate = str[offset] - '0';
1068         }
1069         *result += intermediate;
1070         offset++;
1071     }
```

Missing test coverage

Severity	Informational
Status	Reported
id	ADA-SEAP-2024-11
Component	SEAPATH tests

SEAPATH lacks test coverage of several implemented, security-relevant configurations. We found multiple of such that are recommended Linux hardening practices as well as some that comply with the ISA/IEC 62443 standards. Below we include the configuration points that we have found need test coverage:

Disabled root SSH login
By default, SEAPATH disables remote root access to servers via SSH. This is recommended hardening. While it *is* implemented, SEAPATH lacks test coverage of this feature.

Max allowed failed SSH login attempts
SEAPATH allows a maximum of 3 failed SSH login attempts. It is recommended practice to set a limit, as it mitigates a number of easy attacks on SEAPATHs servers. SEAPATH does not have a test for ensuring that this configuration works as intended.

Add documentation on how admins can prevent users from seeing other users' processes

Severity	Informational
Status	Reported
id	ADA-SEAP-2024-12
Component	SEAPATH Documentation

As a hardening recommendation, SEAPATH administrators should be able to prevent users in their cluster from seeing other users' processes on machines for example by way of `ps aux`. Users may be able to obtain information about what other users are doing on a SEAPATH server including higher privileged users. We have not seen any reason for lower-privileged users to view other users' processes, and we have not seen this implemented in SEAPATH. We understand from the SEAPATH team that this is a measure that users should configure when needed, and we recommend adding a section in the SEAPATH Wiki about how to do this.

Add two-factor authentication

Severity	Informational
Status	Reported
id	ADA-SEAP-2024-13
Component	SEAPATH Documentation

As a hardening recommendation, SEAPATH could enable an out-of-the-box configuration that includes two-factor-authentication. 2FA adds an extra layer of security that we consider necessary for production deployment in substations.

SEAPATH could maintain playbooks with this enabled, or it could maintain documentation on how to enable it in SEAPATH. We consider both to be useful for users.

Make ANSSI compliance clear in test suite

Severity	Informational
Status	Reported
id	ADA-SEAP-2024-14
Component	SEAPATH Documentation

Earlier in the report we covered SEAPATHs goals related to complying with ANSSI standards for cyber security. While reviewing SEAPATHs test suite, we found it unclear which tests cover which parts of ANSSI compliance. We were not able to see in the test suite which areas of compliance SEAPATH is lacking and which it covers. To ensure compliance with ANSSI standards, we recommend organizing the test suite with references to ANSSI requirements and in such a way that it is clear which areas SEAPATH is lacking compliance.

Members of the SEAPATH team are working on exploring OpenSCAP which has policies that support ANSSI BP028 compliance, however this is still ongoing work.

ISA/IEC 62443 issues

In this part we enumerate the findings related to ISA-62443 from the audit. These issues relate to complying with ISA-62443 rather than weaknesses in the SEAPATH system. As such, the SEAPATH project must first decide if and when it wants to dedicate bandwidth to complying, before these issues should be fixed. A goal of this exercise was to explore the amount of work needed to comply with the ISA-62443 framework and not to achieve compliance during the audit. Several of the issues include the phrasing: “SEAPATH must…”. This is meant in the context of complying with the ISA/IEC 62443 standards, namely that “to comply, SEAPATH must…”.

SEAPATH must maintain documentation about its development processes and update it such that it is consistent with how development and maintenance is carried out in practice.

This part of SEAPATHs ISA-62443-4-1 compliance documentation must include:

Missing Role Overview

id	ADA-SEAP-2024-19
ISA/IEC-62443 spec	62443-4-1-5.4

SEAPATH should maintain a nn overview of its organisational roles, their responsibilities and person(s) in charge of the requirements in the ISA/IEC-62443 standards.

Missing Security Training for Maintainers

id	ADA-SEAP-2024-20
ISA/IEC-62443 spec	62443-4-1-5.5

Documentation on the security training that maintainers in charge of security-relevant procedures receives. Maintainers must know of the security-relevant impact of their area of responsibility and must be able to demonstrate that by way of formal documentation. Maintainers can gain security-relevant knowledge by taking courses, attending seminars or conferences, by demonstrating experience or by acquiring certifications in security.

Lack Scoping components of SEAPATH that must conform to ISA-62443

id	ADA-SEAP-2024-21
ISA/IEC-62443 spec	62443-4-1-5.6

SEAPATH has not scoped which parts should conform to ISA-62443 and which should not. In addition, SEAPATH must scope the level of compliance to which SEAPATH should follow. To this end, SEAPATH must also justify the components that do not require compliance. For example if a component is exclusively used by an admin, SEAPATH may not require to achieve the highest level of compliance.

Lack of Documentation of Intended Controls for Source Code Protection

id	ADA-SEAP-2024-22
ISA/IEC-62443 spec	62443-4-1-5.8

ISA-62443-1 requires that products implement a process that describes how the product is protected during development, production/building and delivery/release. We did not find this documentation. In SEAPATHs case this is mostly relevant to source code protection, since the project has not had its first official release.

SEAPATH must ensure all security-related issues are fixed in releases

id	ADA-SEAP-2024-23
ISA/IEC-62443 spec	62443-4-1-5.12

SEAPATH must ensure that all releases have fixed all known-security issues in all components included in a given release. We recommend adding a requirement to SEAPATHs release documentation to ensure this in the future.

Verifying SEAPATHs compliance before cutting release

id	ADA-SEAP-2024-24
ISA/IEC-62443 spec	62443-4-1-5.13

Before issuing new releases, SEAPATH must verify that it has followed all required and applicable security requirements of ISA-62443. SEAPATH must document the way it has fulfilled these security requirements.

Iterative improvements to security-related processes

id	ADA-SEAP-2024-25
ISA/IEC-62443 spec	62443-4-1-5.14

SEAPATH must improve its software development lifecycle based on the latest trends in the security of its field. This includes considering the latest threats to itself and similar systems as well as vulnerabilities in similar systems that are deployed in production. 5.14 specifies that SEAPATH must maintain a process to consider its software development lifecycle against these inputs. We recommend implementing a working group that regularly discusses the latest security trends and threats and considers a) whether they apply to SEAPATH and b) how they could be mitigated in SEAPATH.

Lack of documentation about the system in which SEAPATH is used and/or deployed

id	ADA-SEAP-2024-26
ISA/IEC-62443 spec	62443-4-1-6.2

SEAPATH must document the security context in which it is deployed. This includes both the security context that SEAPATH expects in order to be secure as well as the security context that it must fit

into. For example, since SEAPATH is deployed to remote servers, it could specify the physical and virtual security perimeter that these servers require in order to be deployed securely. Another example is the necessary security perimeter around building the SEAPATH images for production use cases; From discussions with the maintainers, we understand that the current plan with regards to future SEAPATH releases is to release source code corresponding to tags in the SEAPATH source repository. As such, users are required to build their own binaries and images. SEAPATH should document a safe environment in which these should be built to avoid compromise at build time. In addition, the build environment should be provisioned solely for building SEAPATH at build time; Other activities such as downloading files, hosting services and even reading emails or browsing the Internet could compromise the build environment and thereby tamper with the build output.

This documentation should also include assumptions about the impact that compromise of SEAPATH could have on the environment. Consider a compromise of a SEAPATH server with privileges to cause denial of service of the server, the impact of such an attack would depend on the type of function that SEAPATH runs in a substation. If SEAPATH in this case was used to host a circuit breaker, the attacker could prevent the substation from either opening or closing it and thereby either prevent flow of electricity or keep electricity flowing even though the SEAPATH user needs to close it.

Lack of threat model

id	ADA-SEAP-2024-27
ISA/IEC-62443 spec	62443-4-1-6.3, 62443-4-1-9.4

SEAPATH must maintain a threat model for all components that are in scope of ISA-62443. From our observations, we consider the following characteristics applicable to a compliant threat model for SEAPATH:

- The intended dataflow through SEAPATH.
- Trust boundaries.
- Remotely accessible ports both virtual and physical.
- Potential and known attack vectors on both SEAPATH and the hardware on which it can run.
- Potential and known threats along with severity. For example, what would be the severity of an attacker achieving remote code execution on a server running SEAPATH?
- Previous security-related issues.
- Enumeration of external dependencies that are linked to the application.
- Security boundaries of physical and virtual components.

The threat model could include a definition of the physical and logical boundaries of the system to satisfy 62443-6.5. This should also include a technical attack surface anumeration to comply with 62443-9.4.1.b.

This threat model should be reviewed at least once a year for released components.

Lack of security requirements documentation

id	ADA-SEAP-2024-28
ISA/IEC-62443 spec	62443-4-1-6.4

SEAPATH must maintain documentation relating to securely installing, operating and maintaining SEAPATH. This is often called a “security best practices” document in which the adopter can look up all ways to use SEAPATH in the most secure way. See the Istio security best practices for reference: https://istio.io/latest/docs/ops/best-practices/security/. These should be reviewed periodically such that they align with the threat model to satisfy 6.6. Developers, testers, users and security advisors should take part in the review process.

Lack of Secure design principles and Security Review processes

id	ADA-SEAP-2024-29
ISA/IEC-62443 spec	62443-4-1-7.2, 62443-4-1-7.3, 62443-4-1-7.4, 62443-4-1-7.5, 62443-4-1-8.3

SEAPATH must document the high-level principles it should follow in its design. This could for example be:
Least privilege: SEAPATH could aim to set up permissions in such a way that they only fulfil the needed requirement. In turn, the design could outline where and how that would be necessary.
Attack surface reduction: SEAPATH could make it a goal to reduce attack surface in certain areas. The security design could then consider where that should apply, for example areas of the Linux kernel that are particularly prone to contain vulnerabilities and are not needed by SEAPATH.

SEAPATH must document its interfaces and boundaries to its surrounding environment. This includes:
- Which parts are accessible to other components such as components in the substation.
- Enumeration of all entrypoints to SEAPATH including admin-only interfaces and interfaces to the hardware to which they are deployed.
- Enumeration of which entrypoints exist at trust boundaries. We have discussed high-level trust boundaries in the threat modelling exercise in this audit. SEAPATH should do the same at an implementation level.
- The required privileges to use each interface. We understand SEAPATH may implement a role-based access system in the future, and the roles could also be included in this documentation.
- The way SEAPATH should be hardened against threats at its interfaces.

SEAPATHs security design should be reviewed regularly for updates, changes to the design or recent attacks and vulnerabilities that it should take into account.

SEAPATH should document the defense mechanisms that exist at each layer in its stack. In addition, SEAPATH should implement multi-layer defense, for example validation of packets and the network interface, authentication by way of SSH and logging at the OS-level.

Local access timeout

id	ADA-SEAP-2024-30
ISA/IEC-62443 spec	62443-4-2-6.7

SEAPATH has set a timeout period of 300 seconds that will terminate an SSH session in case of activity. This is good hardening practice and is required by ISA/IEC 62443-4-2-6.7 which also stipulates that local access also should terminate the session in case of inactivity. SEAPATH should verify that this exists for both remote access and local access to servers and add tests for both scenarios.

Add pre-authentication banner for SSH and local access

id	ADA-SEAP-2024-31
ISA/IEC-62443 spec	62443-4-2-5.14

All users in a SEAPATH deployment should see a pre-authentication warning that states that they are about to enter restricted area that only authorized personel has access to. This is so that people impersonating other users are aware that they are crossing a trust boundary and can suffer legal implications of unauthorized access. SEAPATH does not implement this and should do so and ensure test coverage to comply with ISA/IEC 62443-4-2-5.14.

Add documentation or ability to whitelist IP addresses

id	ADA-SEAP-2024-32
ISA/IEC-62443 spec	62443-4-2-15.3

To comply with ISA/IEC 62443-4-2-15.3, SEAPATH should allow users to whitelist IP addresses that can access SEAPATH servers remotely. In addition, all attempts to access SEAPATH servers from other IP addresses should be explicitly logged. SEAPATH can comply by either providing this feature with a pre-made configuration. SEAPATH supports the ability to upload iptables, and SEAPATH could instead maintain documentation on whitelisting IP addresses by way of iptables and applying this to their SEAPATH deployment.