Experiences with Secure Pipelines in Highly Regulated Environments

Jose Andre Morales, Jeffrey Hamed, Douglas Reynolds,
David Shepard, Luiz Antunes, Joseph Yankel, Hasan Yasar
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
{jamorales,jhamed,djreynolds,djshepard,lantunesjdyankel,hyasar}@sei.cmu.edu

ABSTRACT

In this experiential paper, we present observations from our collaborative efforts with multiple entities operating in highly regulated environments that enabled or disrupted the construction, use, and sustainment of secure CI/CD pipelines as part of a larger DevSecOps strategy. From these observations, we provide insights and recommendations to support enablers and avoid or minimize disruptions. Our insights reveal that along with noted established progress in the area of secure pipelines, there still exists a need to amend multiple cultural and technical barriers to fully realize secure pipelines in a highly regulated environment. Areas of improvement include streamlining security approvals, revising and updating polices to relevance with current technology, increasing automation in multiple pipeline relevant tasking, improving inquiries to better understand pipeline requirements at commencement, and ensuring appropriate sustained training of technical staff. Recommendations presented here address observed gap areas with the purpose of assisting further advancement of achieving formal and refined pipeline incorporation in a highly regulated environment.

CCS CONCEPTS

• **Software and its engineering → Software development methods**.

KEYWORDS

secure pipelines, continuous integration, continuous delivery, DevSecOps, highly regulated environments

ACM Reference Format:
Jose Andre Morales, Jeffrey Hamed, Douglas Reynolds,, David Shepard, Luiz Antunes, Joseph Yankel, Hasan Yasar. 2023. Experiences with Secure Pipelines in Highly Regulated Environments. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3600160.3605466

ARES 2023, August 29–September 01, 2023, Benevento, Italy

ACM ISBN 979-8-4007-0772-8/23/08.
https://doi.org/10.1145/3600160.3605466

1 INTRODUCTION

A CI/CD pipeline [30], a foundational component of a Development, Security, and Operations strategy (DevSecOps) [10, 19, 27], is a utilitarian automated software system composed of an ensemble of tools, serving the purpose of building, testing, and deploying various software components, resulting in a completed system ready for operational use in one or more target environments. A secure CI/CD pipeline (hereafter referred to as a pipeline) integrates various security elements into the pipeline itself and the system it is preparing for operational use. The security elements serve to ensure various security requirements are satisfied. A highly regulated environment (HRE) [16–18] often imposes multiple strict security requirements that must be satisfied for software systems to function within the HRE's digital environments. DevSecOps was first used by HREs to improve their secure software and system development processes, which up to that time was implemented primarily using a waterfall strategy [1, 12, 22]. Since that time, efforts to incorporate pipelines have become more pervasive in HREs. In this experiential paper, we reflect on our collaborations with several of these efforts to pinpoint the capabilities and advancements provided by pipelines that have enabled success along with the pitfalls and drawbacks which have either delayed or disrupted pipeline integration and sustained usage in an HRE. The noted observations classify in two broad categories: cultural and technical. From a cultural perspective, we primarily observed the continued practice of established traditional approaches in multiple aspects of software development in an HRE often delays or disallows constructing and using pipelines for these efforts. Furthermore, the need for various teams in an HRE, such as developers and security personnel, to function in isolation of each other results in disparate pipeline creation and sustainment, resulting in time periods of partial pipeline functionality. Other cultural hindrances we observed were out-of-date policy requirements and sub-optimal staff training and awareness efforts of latest technology trends. From a technical perspective, our primary observation was on the increased availability of tools often used in pipelines updated with built-in security features [2]. We also observed processes to implement certain pipeline tasks re-engineered to facilitate completion with lowered security requirements. Our observations also included pipeline delays and disruptions due to multiple factors, such as various time-consuming manually driven required processes, vague security guidance, limited communication capabilities, and presence of legacy software. The general positive observations show evidence of an overall increase in community recognition of the need to build software products with built-in security features and HREs' willingness to use them, while the negative observations illustrate the continued use by HREs of traditional processes further

hardens the sustainable establishment and usage of pipelines and, in some cases, facilitates extended time periods of exposed risks and potential compromise. The balance of this paper will detail all of our noted observations and their impacts on pipelines along with inferred insights and recommendations.

2 CULTURAL OBSERVATIONS

The cultural aspect to pipelines is, in general, an ability for an organization to adapt its software development process to new technologies. Often this organizational cultural shift can take time to become commonplace and needs to be steered from senior leadership. Based on our observations, in an HRE, a cultural shift is hindered by multiple factors stemming from: organizational policies, security requirements, business models, traditional mindsets, contrarian goals in hierarchical management, and isolated team and work environments. In spite of these hindrances, we observed that the software development community mindset has shifted positively with respect towards security, resulting in wider availability of tools with security features. Below we detail our HRE cultural observations.

Increased availability of tools built with security in mind. In the past, security was seen as a standalone field of software development applied to a pipeline as individual tools. We observed, in the current practice of pipeline construction, that security is built in to non-security focused tools and platforms, suggesting a community mindset shift, embracing the importance of the built-in security strategy [10]. Security capabilities such as vulnerability scanning, malware scanning, static and dynamic code analysis, and deployable artifact security scanning, amongst others, are now standard features in non-security focused tool categories, such as repositories, container building, code compilers, and various platforms. The benefit of this is pipelines can be constructed in a smaller amount of time due to a decrease of integrating security-specific tools. The current practice still exercises usage of standalone security tools but often, as we observed, in situations where the specific security need was not already integrated into an existing tool category, such as those mentioned above. The built-in security strategy provides security by default, which adds a base security layer without struggling to do so. In the particular case of platforms [24, 25, 29], which is typically a single system providing a set of tools fulfilling multiple non-security pipeline capabilities, we observed that even though some security tooling is included, as noted above, the platform is designed to simplify the process of adding custom tools to provide focused security when needed.

Legacy manual approval process of a pipeline and its components. We observed pipeline construction requires a risk analysis typically lead by a security adjudicator and that person's team. The analysis focuses on various security perspectives of the ensemble tools that are proposed to realize the pipeline. The goal of the analysis is gaining permission for the pipeline to be constructed and used on an entity's digital environment. The security analysis is an accepted, and in most cases required, practice. Our observations discovered the implementation of the analysis was, in many cases, a traditional manual style leading to potentially undiscovered vulnerabilities. The analysis was performed by using a list of requirements recorded in a document with a checkbox for each indicating acknowledgement that the requirement is satisfied. A team of security and engineering personnel, along with the adjudicator, posed each question and, relying on reports from the developer and some security tools, would determine if the requirement was satisfied and check the box. Many of these requirements were checked as satisfied simply by having a person, not necessarily a security person, state affirmation of satisfaction. There were few instances of verifying an answer given by these individuals; the rest were accepted at face value. This approach facilitates malicious actors to misinform and places both the pipeline and the completed system it is producing at risk of compromise.

High-level security requirements. Several entities provided high-level security requirements for the construction, usage, and sustainment of pipelines in the form of documents that are too generalized and cannot be validated in a codified manner. These general requirements lacked metrics and other precise forms of verification such as: variable settings, folder permissions, existence of specific files and folders, and specific user account settings amongst others. We observed these security requirements were interpreted in multiple ways and, as such, facilitated the manual approval process described above. The open interpretation led to affirmation of satisfaction for a given requirement based on beliefs and evidence that in many cases were part of the previously mentioned manual approval process and were not verified. In several cases, the observer considered the given evidence as insufficient to warrant a satisfaction of affirmation. This open interpretation of high-level guidance facilitates existence of risk in both a pipeline and the completed system it produced.

Allowed use of software with detected vulnerabilities. We have observed several instances of continued operation of both the pipeline and the completed system it produced when a vulnerability was detected in either one or both, facilitating a window of risk for a period of time lasting from hours to days. Entities allowed this to occur based on guidance granting priority of continuance functionality and operation over security. The guidance was followed by providing an immediate fix that was implemented with a workaround subverting the vulnerable piece of software in question. In some cases, the vulnerability was not applicable in the deployed environment or was mitigated by pipeline-external security controls. The workaround approach re-established a sense of security supporting continued use of a still vulnerable system for a period of time until an official security fix was released.

Required testing of security fixes. When a security fix was made available, its integration to a pipeline was often delayed days or weeks due to the entity's testing requirements of any and all software prior to deploying into the target environment. These testing requirements tended to be long and tedious with several manual steps and multiple levels of approvals by different teams. In some cases, a subset of teams would not approve for various reasons such as: incompatibilities with existing software, a need to acquire other approvals, and not whitelisted resulting in further extending the time window of opportunity for malicious compromise.

Incongruities between requirements and HRE abilities. Several instances were observed of an inability for an HRE to fulfill

pipeline requirements due to several factors including: limited access to tooling or infrastructure, security impositions, managerial disagreements, and entity-imposed policies. Whitelisted software often excluded needed or best-of-breed titles, resulting in concessions producing a sub-optimal pipeline. The process of whitelisting was lengthy and not fully automated, causing the process to extend beyond acceptable thresholds for completion of some phase of the pipeline. Managers of teams in an HRE were, in some cases, unwilling to approve use of required software in a pipeline due to non-conformance with their team's security or operational standards, infrastructure configurations, institutional policies, or previous sub-optimal experiences. In these cases, alternate software with reduced functionality were chosen, producing pipelines lacking features such as full automation and modularity.

Sub-optimal cross-functional teams. Often times, various teams in an HRE operated in an isolated manner. In this mode, communication between teams ranged from limited to non-existing. A cross-functional team (CFT) [14, 20, 27, 28] for a pipeline should consist of members from various teams across an HRE, where each individual contributes desired domain expertise. The siloed operating environment of an HRE hindered the creation of CFTs due to policies disallowing sharing of knowledge and technology between individual teams. This resulted in a pipeline effort occurring with a waterfall strategy. The result was a pipeline lacking capabilities in some manner. Adding or fixing the sub-optimal capabilities added delays to the completion of pipeline construction, usage, or sustainment. A further disruption was the inability to share pipeline details such as capability needs and security requirements due to policies disallowing sharing of information between teams either for security reasons, protection of intellectual property, or competing interests.

Permissive contractual agreements. Often times, contracts were articulated in such a manner to allow teams in an HRE to deliver a pipeline that did not fulfill all stated requirements, including security. Extensions to due dates could be re-forecast to future dates for generalized reasons with minimal or no justification. Further, a pipeline could be delivered in a form that is usable, but not reproducible, by others due to the exclusion of the pipeline's source code. The lack of reproducibility created a long-term dependency on the pipeline's development team to implement modifications such as bug fixes, version changes, and integration of new tooling.

Lack of best practices and latest technologies. Several HREs were observed to be resistant to fully adopting the latest technological advancements and best practices, such as pipelines. This was due to traditional mindsets and established policies in engineering teams justifying their current approach of software development, often a waterfall or slight variation strategy as effective within a team's particular scope of work requirements. This mindset stemmed form an internally perceived limited ability to adopt new technologies or a lack of willingness to provide the required effort. Limited adoption ability was based, in some cases, on the use of required specialized digital technologies that only accepted software that was developed and tested in a specific manner. In other cases, several years of development success was the source of justification to avoid adopting new practices such as pipelines since it was viewed as a much larger adaptation process that could hinder more then benefit their traditional software development process.

Hindered progress toward a common goal. In a typical pipeline endeavour, multiple teams were required to participate. These teams, as previously discussed, are typically organized to work in siloed spaces within an HRE. We further observed these same teams were also siloed in their job functionalities. Each team is directed to only perform very detailed and specific tasks or services; this was especially prevalent in security teams. This produced an overall security team with a large number of specialized sub-teams with limited collaboration. The common goal of, for example, constructing a pipeline with full security approvals was difficult to accomplish due to approaching multiple teams where each perform a single function without understanding the overall design of the pipeline. The impact of this was incompatibilities between different teams' contributions to the pipeline, causing delays and additional re-engineering to complete.

Teams with focused domain expertise. We found in many HREs teams devoted to pipeline engineering possessed domain expertise only in that field with less understanding of security. The opposite was true for security teams, where those members lacked knowledge in pipeline development. This resulted in two phenomena: first, pipelines were developed with minimal thought to security, causing extensive delays and re-engineering to satisfy multiple security requirements and second, security requirements were imposed on pipelines, causing a reduction of functionality. The reduced functionality caused, in some cases, a pipeline effort to cease and fully restart.

Security requirements with transparency. We observed the common HRE security requirement of providing a detailed list of every software title and version including libraries and other auxiliary tools. This list, viewed as a software inventory list [7, 15, 21], created in clear text, was used for security analysis to determine if a particular software title or version was currently disallowed in a given digital environment. Controlling this list's distribution was not well governed and raised security concerns of potentially revealing all the software used in a given pipeline, which was itself viewed as an enhanced security risk.

Personnel retention hinders pipelines. We often observed pipeline efforts, especially in sustainment, being dependent on individual members of various teams in an HRE. When one of those members departed, the remaining team struggled to maintain an active pipeline and approved security, causing delays, disruptions, and potential security risks. The reason was, in these HREs, knowledge transfer amongst team members was not practiced often. Furthermore, standard operating procedure manuals were often not updated to address new technologies like pipelines. We also observed team members departing due to frustration from constant changes in technology usage options for a given project. In one example, a pipeline's hosting environment was changed multiple times in a short span, requiring team members to be trained in each. After a few changes, two members left and a group of five reduced to three. This caused pressures on the two lead engineers, and one ended up leaving, which culminated with the team manager's departure due to pressure from senior leadership for an inability to keep the team intact and meet deadlines. A different form of departures was

observed in a few HREs, where a team member promoting new technologies for pipelines was viewed as disruptive by others. The reason was that promoting new technology was viewed as a change to the status quo and the traditional way of performing various pipeline tasks. The resolution was to promote the individual to a higher position in a different team, thus preserving the status quo.

3 TECHNICAL OBSERVATIONS

The technical aspect to pipelines is, in general, establishing an infrastructure that primarily enables automation and modularity in all pipeline functionalities. With automation, a pipeline can fulfill code and system building, testing, validation, and security with minimized human intervention. This facilitates sustaining the velocity of multiple deliveries of completed systems as often as required. Modularity supports fixes, enhancements, and replacement of a pipeline's constituent parts with minimized disruption to ongoing pipeline operations. Modularity is most useful primarily during pipeline usage and sustainment. We observed the development of tooling used in pipelines are being designed with a security focus along with redesigned pipeline tasks implemented with lowered security requirements. We further observed in HREs the adoption of new technology infrastructure is hindered, primarily, by acquisition protocols requiring significant time periods to complete. Also observed were technical implementation barriers to pipeline functionalities imposed by security policies resulting in sub-optimal implementations of various tasks. Below we detail our HRE technical observations.

Tooling with a security focus. Several HREs have incorporated tooling for pipelines that are designed with built-in security. When in the past, scanning code stored in a repository for various security concerns required a handful of tools, now it is completed using one repository software title that brings multiple security scanning tools by default. A further improvement has been the redesign of pipeline tasks to require reduced privileges and access. A clear example dealt with deployable artifacts. These artifacts typically required a `root` system access to be built and executed, since then newer tools accomplish both without the need for `root` access [8]. HREs benefit from this by building pipelines faster with less software titles and in some cases expedited security analysis.

Legacy digital assets. Often observed in HREs was the continued use of legacy hardware and software. This was most prevalent in instances requiring specialized digital assets that are not easy to upgrade or replace. In other instances, the continuance was imposed by acquisition processes, which consumed significant time and investment to complete. Legacy digital assets often could not benefit from pipelines. In cases where benefit was possible, the pipeline was highly customized and not modular. Specifically, production testing was hindered by the inability to automate deployment to hardware, often resulting in a manual intervention causing delays. Also, security risks were an ongoing issue due to the difficulty of applying fixes or patches resulting from minimal support by the assets original creators.

Disruptions to automated builds. A pipeline will often release multiple versions of a completed system in short time periods. The building phase of the pipeline will gather all needed software artifacts to ensure the system executes as expected in a target environment. Gathering the artifacts is typically performed by automated pull requests of specific files from various repositories located on networks external to the HRE. Often, we observed HRE security policies greatly hindering the process of automated pull requests from HRE external locations. An HRE external repository had to be whitelisted in order to access and pull files from it into an HRE. The approval process to access HRE external repositories was lengthy and complicated, requiring significant time for completion and manual approvals. We did observe several HREs approving access to often used external repositories. In cases where a required external repository was not whitelisted, an HRE internal mirror repository was created as a clone of its external counterpart. Creation, testing, approval, and sustainment of a mirror repository was in itself a significant task. In other cases, where mirroring was not an option, we observed files from HRE external repositories acquired separately via a customized application and manually copied into an HRE system for a local build process. Another observed hindrance to automated builds occurred at the individual file level. When files entered the HRE, they were evaluated for security and disallowed if a violation such as not being whitelisted was detected. Sources of violations were many and included: variation in a file's hash value [3, 4], differing version numbers, and an unrecognized or unapproved file dependency [6, 13]. In some cases, an HRE would catalog a file with its version number and hash value and use this triplet for comparisons with incoming files. Some files were disallowed due to the maintainers of that file implementing changes without updating the version number, thus causing a mismatch with the HRE's record.

Using latest file versions. A commonly accepted practice in a pipeline's build phase is to use the latest version of a software title. In many cases, acquiring the latest version is the default setting for most tools used to pull these files from their respective repositories. A typical version of a completed system consists of a multitude of files, each with its own version. In an HRE, as we observed, whitelisting new versions was typically a lengthy process. An older version of a file was used while the most recent version underwent the whitelisting approval process. During the interim, multiple failures in the build phase would occur due to a latest version of some file requiring more recent versions of other files that were excluded from the build due to not yet being whitelisted. We also observed cases where, due to a multitude of reasons, a specific version of some file was required and could not be upgraded to a more recent version. In these cases, the build phases were often customized to ensure preservation of several file versions related to the file in question. This often produced security risks, leading to justifications allowing the build to complete. Often, versions of completed systems containing these security risks were not released. Production environments would continue using older build releases, in some cases for extended time periods, which sometimes created problems with HRE policies requiring its own custom solution.

Pipeline component interfaces. A pipeline consists, primarily, of a set of tools interacting with each other by exchanging data transfers in an automated manner. The data transfer is primarily the output of one tool serving as the input of the next tool in

the sequence. These inter-tool data transfers were facilitated with the use of interfaces [26, 31]. The interfaces were, in some cases, built in by default for use with software titles belonging to the same tool development organization. In many cases, interfaces were not available and had to be acquired or customized. HREs lacked guidance in how to implement such an interface and, more importantly, which security best practices should be followed in this specific case. The result was either the use of third party tools or custom written code. Both solutions worked well enough to perform the transfer and received all the requisite approvals but, in our observations, were not extensively tested for robustness and exception case handling. This created, in our opinion, opportunities for untested inputs to potentially cause an unhandled exception of the interface code, resulting in random pipeline behaviors.

HRE data transfers. When a pipeline component required a patch or update, often times the new files were not whitelisted by the HRE. The files in question were acquired via systems outside of the HRE. These files were then manually brought into the HRE via an approved physical medium such as one or more CDs [23] or DVDs [5]. This process is known as an offline data transfer, also commonly referred to as a *sneakernet* [9, 11]. Critical to sneakernet is access to the pre-approved physical mediums and devices that can read/write onto the medium. As technology has advanced, acquisition of this medium and read/write devices has become harder and more costly. The process of pulling needed files and writing to a medium would take from a few hours to a couple of days. Often, the process was repeated multiple times for several reasons including: missing files, incorrect versions, and formats that could not be processed in the HRE. All medium received in the HRE entered a security analysis to receive approval for its contents to be written into the system where the pipeline resided. This was often a lightweight process with the majority being approved. Once approved, the contents were transferred and entered a second round of security analysis to receive approval to merge with the existing pipeline. The additional resources required for sneakernetting motivated pipeline developers to build using what was accessible within the HRE. This led to pipelines using components that were often not the best choice for a given capability or the latest version for a specific software title.

Usage of open source repositories. Incorporating software titles from open source repositories is a universally accepted practice due to no monetary commitment needed for usage. We observed that HREs are no exception to this practice. Open source repositories are mostly community based, where members can contribute libraries, packages, and other artifacts for community use. Repository maintainers do review and verify the code that is submitted. However, due to the complexity of most code, not all problems can be identified. Popular and important repositories do have security audits, but those are community funded and do not occur as often as they should. HREs that pull files from these repositories create a potential security risk based on the inherent trust bestowed on whitelisted repositories for automated pulls and all others acquired via sneakernetting. A pipeline could incorporate files sourced from these repositories that have malicious intent but pass all HRE imposed security requirements. Open source repositories typically do not fund maintenance of the software titles posted there with the assumption that the original contributor will fulfill this task. Only software titles with a large user base may be funded to provide maintenance from donations and private interests. In many cases when a security vulnerability or some other issue arises, a fix is typically implemented by a community member. This lack of a fixed entity employed and paid to maintain software titles resulted in HREs using software titles patched by potentially unknown individuals, thus creating a potential security risk.

Default security settings. Some HREs were observed to use security software titles in a pipeline with default settings. These settings often served as a baseline that can be customized to a user's needs. Security engineers reviewed the baseline settings and determined if that satisfied one or more HRE-imposed security requirements. If yes, the software was used as is with no further effort to determine if customizing would be an enhanced benefit for the security of the pipeline or the system it was producing. In cases where security requirements were not very specific, it was often determined the baseline would suffice. This produced a pipeline with a potentially misleading security state due to lack of customization.

Lack of uniformity in pipelines. HREs often created pipelines customized to the end user's needs. In some cases, this was appropriate due to specialized requirements. In most cases, we observed end user needs could be satisfied by a standard pipeline architecture with a standard security and testing harness. In general, HREs would customize almost every pipeline to an end user's needs, including cases when pipeline reuse was, in our opinion, an option. This resulted in increased customization of sustainment tools needing to address particularities in individual pipelines. This further increased time resources to ensure security requirements were constantly satisfied due to the need to analyze almost each active pipeline instead of one standard pipeline in use by many end users.

Domain-specific production environment requirements. An observed source of many disruptions and delays in pipelines across several HREs dealt with production environments. These environments are where a completed system will operate and thus was required for a pipeline to access for testing and delivery. A common problem was a need for the HRE to receive environment access approval. This was a long and complex process, which included required manual approvals from multiple entities. Setting up the connection to the production environment to support pipeline automation was often difficult and complex, requiring special connectivity and configurations. The most disruptive scenario was when a production environment could only be accessed by specific authorized personnel external to the HRE. In these cases, the pipeline would deliver a completed system to a local storage that was passed on to the authorized official via a separate process that, in some cases, was done via sneakernet. The authorized official was also given a test suite to run in production. The authorized official would process and validate security requirements and then load the completed system and test suite in production, run the tests, and deliver the results to the HRE. This process often took days to complete.

4 INSIGHTS

Further consideration of the cultural and technical observations detailed above revealed insights of the current state of HRE's treatment of pipelines affecting construction, usage, and sustainment. The insights revealed both enablers and disruptors in the build, test, validate, and delivery phases of a pipeline. The insights imply that although progress has been made in the treatment of pipelines in an HRE, improvements continue to be needed. Detailed insights are listed below.

Built-in security given higher importance. The concept of built-in security has become prevalent in the development of pipeline tools. In the past, these security tools were standalone software titles included in pipelines, resulting in the need for a much larger set of tools to complete a pipeline construction. With built-in security, the number of tools needed to create a pipeline has noticeably reduced. The level of security in these pipelines has also greatly increased. Currently, pipelines can be built faster and with less tools and achieve a higher level of security than in past. This further simplifies sustainment and maintenance of active pipelines. HREs have become receptive to this concept and several have or are starting to approve, test, and include this new generation of tools in their pipelines. In addition to built-in security, several common pipeline tasks have been fundamentally redesigned to accomplish the same tasks with a lowered security requirement. In the past, these tasks often required a level of security that was viewed as potential risks for HREs. Requirements such as `root` or `admin` privileges, full read/write permissions to files and folders, and access to credentials were common in pipelines and often required a lengthy security review, including written justifications, manual assessments and analysis, and approvals by multiple individuals. This often required days to weeks before being completed. The result was to re-engineer and implement an alternate approach removing the risk but requiring customized solutions that triggered further security reviews. More recently, we observed releases of software implementing these common tasks in a re-designed form that has removed several of the potential risks by greatly reducing security requirements. This has eliminated the need for lengthy security reviews and customized solutions. We observed the HRE community's senior leadership is promoting continuation down the path of adopting improved security tools and processes for pipelines.

HRE policies hinder pipeline functionalities. HREs are governed by a multitude of policies, primarily in the areas of operational procedures and security. Most of these policies are time intensive, include many manual components involving multiple personnel, are not regularly updated, and, in some cases, prescribe vague high-level guidance facilitating adherence interpretation. Pipeline construction often required technology compromises due to conflicts between policy adherence and functionality requirements. This often resulted in a pipeline with reduced capabilities than desired. End user needs, in some cases, took precedence over a policy, allowing a pipeline to function with potential security risks. The amount of effort and personnel involved in assuring security policy adherence caused constant pipeline delays, typically months at a time. Guidance vagueness in some policies, including security, allowed a diverse set of implementations verified as satisfactory. The guidance only required adherence to a high-level generalized requirement such as: "Modularity will be used throughout," "All code must be stored securely," "Identified vulnerabilities will be resolved in X days." These exemplar generalized policies could be satisfied in a multitude of ways, and each could be very different in implementation. Using customized implementations to satisfy a policy occurred in multiple pipelines and caused increased effort and resources in long-term pipeline sustainment. Some policies were difficult to adhere to and required interpretation and approval by individual security adjudicators. This was due to the policy having been created at a time when different technology was in use and its relevance and applicability to current pipeline state of the art was not easily evident.

Partial initial planning facilitates avoidable problems. The preparation for initial pipeline endeavors was often planned by a team missing key domains, such as operators and security personnel. This was due to HRE policies directing teams to function in isolation. The planning often excluded considerations such as security needs, access to operational environments, required tooling, and access to files in HRE external locations, amongst others. As tasking advanced, the unconsidered topics became evident and required a solution in real time. This, of course, impacted the overall effort and caused a ripple effect of re-engineering other aspects of the pipeline, resulting in delays and new required rounds of reviews and approvals. Initial planning often focused on pipeline construction with less consideration in its usage and sustainment. Often these latter phases were given operational guidance when needed and in an ad-hoc nature. The result was an acceptably constructed pipeline requiring real-time customization to mend usage problems and further capability enhancements to achieve long-term sustainment along with the additional resources needed for analysis and approvals.

Status quo mindset delays technology improvements. A traditional mindset persists in many teams within several HREs. This mindset enforces a continuation of the current software development process and is hesitant to consider incorporating new technologies. A key contributor to this is a lack of understanding how a new pipeline technology works and its potential benefits to the team. Another contributor is a lack of motivation by teams due to HREs not providing incentives and rewards when newly integrated pipeline technology is considered a success and instead enforce reprimands when due dates are not kept. Some teams actively boast the benefits of their current software development procedure and focus on the perceived downside of any new pipeline technology. Many cases occurred where team members requested transfers or left the HRE due to their lack of desire in adapting to a new pipeline development process that incorporated new technology products and processes. Few HREs offered training to facilitate the adaptation, thus leaving personnel to learn on their own or in small groups. This created frustrations and walkouts, causing delays in various pipeline endeavors. Some teams expressed their members were dealing with "technology burnout," referring to the constant churning of pipeline tools and process in short time periods. With each churn, team members had to adapt to new technology right after reaching a comfortable level of competence with the last technology adoption. After multiple churns and rounds of repeated

technology adaptation within a span of a couple of years, teams seemingly lost their ability to identify a standard software pipeline development process and were criticized by upper management for lack of job fulfillment. This led to increased transfer requests and resignations.

Lacking a standardized pipeline baseline increases unneeded resource usage. In HREs, almost every new pipeline endeavor started with a blank slate. In several cases, the pipeline requirements could be satisfied by cloning a previously existing pipeline implementation. Many HREs lack a standard initial pipeline design and the result was, in most cases, a customized pipeline for each need of each end user. Having so many custom pipelines required specialized treatment in their usage and sustainment. This specialized treatment required increased resources to complete. Using different software titles in customized pipelines required continued access and security approvals for a much larger catalog of whitelisted titles. This, in turn, required increased number of security reviews when new versions or patches were released and caused almost constant pauses in pipelines due to time requirements to attain security approvals. Due to the large number of software titles in a catalog required for all the customized pipelines, HREs received multiple daily vulnerability discoveries across several titles, with each requiring hours to weeks to amend. In several HREs, vulnerability amendments lasted days due to teams being unable to streamline processing, with some cases being overwhelmed by the sheer volume of pending vulnerability resolutions.

Manual oversight of security requirements. HREs are governed by a large corpus of security requirements. A subset of these requirements is written in a very generalized fashion. This generalization disallowed automation of evidence gathering to verify requirement satisfaction. The majority of these requirements involved analysis, evidence gathering, and approval by multiple individuals requiring an extended time period to complete and, in many cases, re-engineer a pipeline. This generalization also facilitated an interpretation of the requirement to determine what specifically needs to be satisfied. Often, the identified specification differed in various teams and the implementation used to justify satisfaction also varied. This resulted with security in general being implemented in different forms across multiple pipelines in an HRE. Some of the security requirements were originally developed with different technologies in mind and, in spite of this, are still currently enforced. The relevance of these requirements to a pipeline endeavour was not well understood and both development and security teams would subjectively interpret and implement a best effort to justify satisfaction. This resulted in requirements satisfied on paper but lacked assurance if the implementation truly satisfied the requirement or not. Some security requirements necessitated extra pipeline engineering that did not advance the operational goals of the pipeline. This resulted in a pipeline consisting of certain capabilities for the sole purpose of satisfying a security requirement and did not contribute to needed functionality.

Pipeline is not DevSecOps. Several HREs have claimed success in adopting a DSO strategy when, in reality, only pipelines have been realized. Using pipelines as a standalone solution results in an integration with traditional development approaches, most often a waterfall strategy. Further, when asked, a good number of team members across HREs understood DevSecOps to be just running pipelines. These same team members were also never provided formal training in DevSecOps to realize the difference. Using pipelines as part of some other software development strategy leads to unaddressed and needed changes in culture and process, the other main components of DSO. This results in an overall sub-optimal implementation failing to take full advantage of pipelines in broader software development strategies.

5 RECOMMENDATIONS

In general, a committed increase in adherence to the principles and guidance of DevSecOps, supported by senior leadership, would further enhance and improve the current state of pipelines in HREs. Within this continued commitment, there are several areas that could improve with consideration of the recommendations that follow.

Cross-functional teams. The policies enforcing siloed working environment of teams in HREs should be amended to allow cross-functional teams to be assembled with appropriate representation of all required expertise domains. This would provide full awareness of a pipeline from all relevant perspectives facilitating every phase in its lifecycle and assist in avoiding unforeseen issues as well as reducing unnecessary efforts. Further, once a team is assembled, it should persist, at a minimum, for the duration of a pipeline's construction and initial usage and sustainment. In order to accomplish this, a baseline of metrics and validation tests should be established that, when fully satisfied, indicates a pipeline is functioning at a point of stability. The point of stability implies that routine tasks such as upgrades and enhancements can be achieved with the expected amount of resource consumption. Other capabilities such as pipeline scalability and robustness are also functioning as expected. Once pipeline stability is established for usage and sustainment, the CFT could be queued for release.

Scheduling accounts for reviews and approvals. Calendar scheduling of pipeline endeavours must take into account required time periods for security reviews and approvals along with other time-intensive tasks, such as acquisition needs in order to establish a realistic forecast of completion. To accomplish this, planners need to fully understand, at a minimum, the software and hardware that is required to realize the requested pipeline functionalities, and for each, determine what is already available for use within the HRE and what needs to be acquired from HRE-external sources. Informed decisions from experience will be useful in this exercise, such as time needed to whitelist a software title and approve access or mirror an HRE-external repository. Further assistance here would be the existence of well-defined actionable guidelines detailing the prescribed process one must follow for each of the identified tasks.

Policy revisions. Continual reviews and revisions of policies in an HRE should occur to sustain relevancy with the current state of technology in use and pipeline development trends. Each policy should be evaluated to determine if it is relevant, actionable, specific, and automatable. A relevant policy applies to current technology and development trends within an HRE. If relevancy is not established, the policy should be considered for removal or replacement. An actionable policy describes a set of detailed atomic steps that can be carried out by human or machine. A specified policy is a set

of very precise articulated requirements and steps to validate satisfaction of that requirement. A specified policy is not generalized and does not facilitate assumptions or interpretations by the reader. An automatable policy is written with technology in mind facilitating the codification of implementation and evidence gathering to justify its satisfaction. An automatable policy can be expressed as a well-articulated set of directives to carry out in a digital asset. Examples of policy guidance adhering to the above are: "Microsoft Defender activation set to true," "all outgoing email traffic passes through SMTPS port 587," "folder 'absolute folder path' should be set to hidden." Note, this form of guidance can be very technical in nature and should reference a single artifact like a variable, file, or folder that can be queried via automated methods.

Standardized pipeline configuration. HREs should establish a standard baseline pipeline that adheres to all current security requirements. An instance of this pipeline should serve as a starting point for all initial pipeline endeavours. HRE personnel should leverage experience to determine the best design for this baseline; ideally it represents a configuration repeatedly used in a majority of completed pipeline efforts. HREs could consider going a step further and require end users to adhere to the baseline and not vice versa. Adherence would assist in reducing customization and their additional sustainment resource needs. Any identified customization work should be viewed as modifications upon the baseline. Maximally preserving the baseline configuration should be a stated goal as this will ease the resources needed for all pipeline phases.

Reoccurring state-of-the-art pipeline technology training. To sustain awareness of trends and team skill sets, HREs should sustain a regular continuing training of their staff with the latest pipeline technologies, tools, trends, and processes. Further, HREs should have regularly invited speakers presenting the aforementioned technologies. This would assist in setting a comfort level and willingness by teams to incorporate new technology into their pipeline development process since those topics will not be completely unfamiliar to individual members. This would further assist in setting a ground truth for these topics, which helps avoid misplaced assumptions. Justification for negating the introduction of a new pipeline technology could occur by leveraging misplaced assumptions. It is important in both training and invited talks that the advantages of incorporating a new technology or process within an HRE be clearly detailed.

Centralized knowledge transfer process. Given the seemingly regular trend of team members departing, a policy of documenting all pipeline-relative knowledge in a centralized location should be established. Even though this is a principle in DevSecOps, we observed it is not often practiced. Further, entire teams and not individuals or pairs should conduct regular source code reviews and testing of all aspects of a pipeline. The goal is full team awareness of any technical intricacies employed as part of a pipeline endeavour. In this manner, a pipeline avoids being dependent on an individual team member but instead is able to be advanced by all members of the team. When a team member does choose to depart, a review of all pipelines that individual contributed to should occur to ensure all details have been properly recorded for team benefit. Intricacies include: specialized commands and configurations, execution steps, environment settings, and other similar digital atomic actions that are not commonly performed in pipeline endeavours.

6 CONCLUSIONS AND FUTURE WORK

Pipelines are a fundamental component of a DevSecOps strategy and are widely used in all sectors, including HREs. The evolutionary path of integrating pipelines into an HRE's software development process has advanced at a moderate pace with several lessons learned. In this experiential paper, we put forth insights and recommendations based on our observations across multiple collaborations with diverse HREs advancing along this path. Our insights revealed an increase in the availability and integration of pipeline-relative tools with built-in security in HREs. Also noted is the adoption of lowered security requirement processes for common pipeline tasks. A culture of traditional mindsets hinders changes to current pipeline development in HREs but can be partially resolved through ongoing training and awareness of latest trends and their HRE-relevant benefits. Increased, and potentially avoidable, use of time and resources has occurred in all aspects of a pipeline endeavour. This can be amended through policy revisions, standardized pipeline designs, and full awareness of all needed tasks and their resource requirements at the start of any pipeline endeavour. In general, HREs are advancing through a learning curve and are establishing the best ways to incorporate and leverage pipelines into their technical tasking. Continued improvements as discussed in this experiential paper, with a focus on increased automation, policy revisions, streamlined requirement satisfaction processes and sustained senior leadership support will further formalize and refine the use of pipelines in HREs. Future work will delve into specific applications of pipelines to achieve capabilities often only relevant in HREs and the current obstacles needing amendment and advances being achieved. Further, the current body of work will be enhanced with available data driven analysis of pipeline impact to an HRE based on rendered business logic analytics and project relevant statistics using established metrics in these domains.

ACKNOWLEDGMENTS

Copyright 2023 ACM. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM23-0468

REFERENCES

[1] Adetokunbo AA Adenowo and Basirat A Adenowo. 2013. Software engineering methodologies: a review of the waterfall model and object-oriented approach. *International Journal of Scientific & Engineering Research* 4, 7 (2013), 427–434.
[2] Shanai Ardi, David Byers, and Nahid Shahmehri. 2006. Towards a structured unified process for software security. In *Proceedings of the 2006 international workshop on Software engineering for secure systems*. 3–10.
[3] Lianhua Chi and Xingquan Zhu. 2017. Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (CSUR)* 50, 1 (2017), 1–36.
[4] Edward G Coffman Jr and James Eve. 1970. File structures using hashing functions. *Commun. ACM* 13, 7 (1970), 427–432.
[5] Paul B De Laat. 1999. Systemic innovation and the virtues of going virtual: The case of the digital video disc. *Technology Analysis & Strategic Management* 11, 2 (1999), 159–180.

[6] Alexandre Decan, Tom Mens, Maëlick Claes, and Philippe Grosjean. 2016. When GitHub meets CRAN: An analysis of inter-repository package dependency problems. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 493–504.
[7] William Enck and Laurie Williams. 2022. Top five challenges in software supply chain security: Observations from 30 industry and government organizations. *IEEE Security & Privacy* 20, 2 (2022), 96–100.
[8] Marcel Gagné. 2001. Linux system administration: a user's guide. *Linux Journal* 2001, 92 (2001), 1.
[9] JB Hiller. 1992. SneakerNet: Getting a grip on the world's largest network. *Computer Security Journal* 8 (1992), 43–43.
[10] Tony Hsiang-Chih Hsu. 2018. *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*. Packt Publishing Ltd.
[11] Andika Candra Jaya, Cutifa Safitri, and Rila Mandala. 2020. Sneakernet: A Technological Overview and Improvement. In *2020 IEEE International Conference on Sustainable Engineering and Creative Computing (ICSECC)*. 287–291. https://doi.org/10.1109/ICSECC51444.2020.9557509
[12] Mitch Kramer. 2018. Best practices in systems development lifecycle: An analyses based on the waterfall model. *Review of Business & Finance Studies* 9, 1 (2018), 77–84.
[13] Nathan LaBelle and Eugene Wallingford. 2004. Inter-package dependency networks in open-source software. *arXiv preprint cs/0411096* (2004).
[14] Edward F McDonough III. 2000. Investigation of factors contributing to the success of cross-functional teams. *Journal of Product Innovation Management: An International Publication of the Product Development & Management Association* 17, 3 (2000), 221–235.
[15] Gary McGraw. 2018. The new killer app for security: Software inventory. *Computer* 51, 02 (2018), 60–62.
[16] Jose Andre Morales, Thomas P Scanlon, Aaron Volkmann, Joseph Yankel, and Hasan Yasar. 2020. Security impacts of sub-optimal devsecops implementations in a highly regulated environment. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–8.
[17] Jose Andre Morales, Hasan Yasar, and Aaron Volkman. 2018. Implementing DevOps practices in highly regulated environments. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*. 1–9.
[18] Jose Andre Morales, Hasan Yasar, and Aaron Volkmann. 2018. Weaving security into DevOps practices in highly regulated environments. *International Journal of Systems and Software Security and Protection (IJSSSP)* 9, 1 (2018), 18–46.
[19] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: a multivocal literature review. In *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*. Springer, 17–29.
[20] Rennie Naidoo and Nicolaas Möller. 2022. Building Software Applications Securely With DevSecOps: A Socio-Technical Perspective. In *ECCWS 2022 21st European Conference on Cyber Warfare and Security*. Academic Conferences and publishing limited.
[21] Kai A Olsen, Per Sætre, and Anders Thorstenson. 1997. A procedure-oriented generic bill of materials. *Computers & industrial engineering* 32, 1 (1997), 29–45.
[22] Kai Petersen, Claes Wohlin, and Dejan Baca. 2009. The waterfall model in large-scale development. In *Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009. Proceedings 10*. Springer, 386–400.
[23] Ken C Pohlmann. 1992. *The compact disc handbook*. Vol. 5. AR Editions, Inc.
[24] Roshan Namal Rajapakse, Mansooreh Zahedi, and Muhammad Ali Babar. 2022. Collaborative Application Security Testing for DevSecOps: An Empirical Analysis of Challenges, Best Practices and Tool Support. *arXiv preprint arXiv:2211.06953* (2022).
[25] Xiaohan Sun, Yunchang Cheng, Xiaojie Qu, and Hang Li. 2021. Design and Implementation of Security Test Pipeline Based on DevSecOps. In *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Vol. 4. IEEE, 532–535.
[26] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2019. Continuous architecting with microservices and devops: A systematic mapping study. In *Cloud Computing and Services Science: 8th International Conference, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018, Revised Selected Papers 8*. Springer, 126–151.
[27] Nora Tomas, Jingyue Li, and Huang Huang. 2019. An empirical study on culture, automation, measurement, and sharing of devsecops. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 1–8.
[28] Sheila Simsarian Webber. 2002. Leadership and trust facilitating cross-functional team success. *Journal of management development* (2002).
[29] Yu Wu, Jessica Kropczynski, Patrick C Shih, and John M Carroll. 2014. Exploring the ecosystem of software developers on GitHub and other platforms. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*. 265–268.
[30] Fiorella Zampetti, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. 2021. Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 471–482.
[31] Uwe Zdun, Erik Wittern, and Philipp Leitner. 2019. Emerging trends, challenges, and experiences in devops and microservice Apis. *IEEE Software* 37, 1 (2019), 87–91.