# DevSecOps Pipeline for Complex Software-Intensive Systems: Addressing Cybersecurity Challenges

**Carol WOODY**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Tim CHICK**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Aaron REFFETT**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Scott PAVETTI**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Richard LAUGHLIN**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Brent FRYE**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

**Mike BANDOR**
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-2612, USA

## ABSTRACT

A major challenge for cybersecurity comes from new technology approaches that focus primarily on the benefits of implementation rather than on defining the governance and risk management changes necessary to establish and enforce appropriate protections. This challenge is especially important for the adoption of technology that impacts critical infrastructure and shared services, such as voting and defense. Researchers examined the challenges and the effective cybersecurity options facing Department of Defense (DoD) programs delivering cyber-physical systems and adopting DevSecOps. These researchers found a lack of broad understanding about the level of management and governance responsibility needed to define and use the DevSecOps pipeline. Adopting DevSecOps is a socio-technical decision that links technology with operational process and practice. Researchers identified several areas that require cross-functional and organizational management attention to fit the pipeline for mission use and considerations to address for producing the system. This paper describes the case study and lessons learned to date.

When a program adopts DevSecOps, it creates and supports two major systems concurrently: (1) the product the program was assigned to produce, and (2) the pipeline the program uses to develop and operationalize the product. Both systems need effective built-in security. In addition, neither the product nor the pipeline can remain static, so the cybersecurity of each must change to ensure sufficiency. The product expands with added functionality, which includes added vulnerabilities that tools and developers must address. The pipeline should be continually refined and improved as new tools and techniques better enable the consistent throughput of new features and capabilities. The focus on functionality and throughput is not sufficient for either system because the threat landscape changes constantly with new attacker capabilities. As a result, the need for improved tools to avoid and remove vulnerabilities from the product become critical. These tools must also be patched since they are software and contain vulnerabilities. As more data about the product is collected through the pipeline, it is critical to tap this information to improve the product and pipeline. However, the pipeline is not a single entity. It is a collection of highly configurable pieces built independently and assembled to perform together.

The increased use of the DevSecOps pipeline to automate software assurance, cybersecurity, and safety compliance transfers the responsibilities for identifying and addressing pipeline and product risks to roles that were not involved in the past. For example, acquirers and maintainers of pipeline tools may now be responsible for the level of verification performed on the product and its associated effectiveness. If the criteria for tool selection remains focused only on cost, availability, and compliance, the expectations for this new responsibility could fall short of stakeholder expectations, especially if structuring the pipeline does not include stakeholder requirements. There is a lack of broad understanding about the level of management and governance responsibility needed to define and assure the responsible use of a DevSecOps pipeline. Our work is focused on bringing these under-addressed areas to light.

**Keywords**: DevSecOps, cybersecurity, risk management, software-intensive systems, tooling, pipeline.

## 1. WHY IS ADOPTION OF DEVSECOPS SO COMPLEX?

By definition, a system is "a regularly interacting or interdependent group of items forming a unified whole" [1]. Thus, DevSecOps is a system. DevSecOps also has the characteristics of a socio-technical system [2]. Because DevSecOps is composed of people, processes, and computer technology that are "designed to collect, process, store, and distribute information" [3], it is a computer information system. So, it is no different from any other IT system that supports a complex business or a critical mission. If we add to this definition that a DevSecOps pipeline is composed of independently developed, independently maintained, likely physically and logically distributed, task-dedicated, interoperable components, then we can affirm that a DevSecOps pipeline is a complex sociotechnical computer information system. When a program adopts DevSecOps, it creates and supports two major systems concurrently: (1) the product the program was assigned to produce and (2) the pipeline the program uses to develop and operationalize the product. Both of these systems need effective built-in security. Figure 1 depicts the software factory pipeline that is used for product development integrated with the pipeline tools and infrastructure; both pipelines must have integrated security to ensure good cybersecurity.

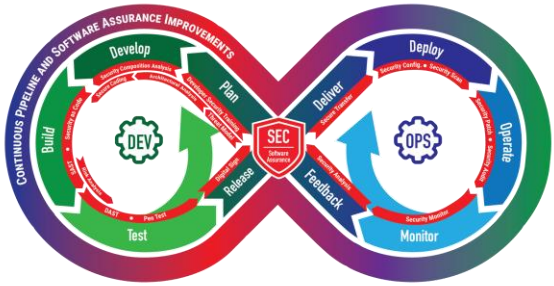


*Figure 1: Pipelines with Integrated Security*

In addition, neither the product nor the pipeline can remain static, so the cybersecurity of each must evolve to ensure sufficiency. The product expands its functionality, which includes adding vulnerabilities that tools and developers must address. The pipeline should continue to be refined and improved as new tools and techniques better enable consistent throughput of new features and capabilities. The focus on functionality and throughput is not sufficient for either system because the threat landscape is changing constantly with new attacker capabilities. As a result, the need for improved tools to avoid and remove vulnerabilities from the product becomes critical. These tools must also be patched since they are software and contain vulnerabilities. As more data about the product is collected through the pipeline, it is critical to tap this information to improve the product and pipeline. However, the pipeline is not a single entity. It is a collection of highly configurable pieces that were built independently and then assembled to perform together.

Traditionally many of these pipeline and development activities are performed without integration. This interdependence carries cybersecurity risk that is not widely recognized. For example, the development and operational environments may be air gapped in traditional development approaches, which allows the strict separation of roles and responsibilities. However, in a fully integrated pipeline, the same orchestration tool that builds the system for testing purposes could also be used for operational deployments. This approach can increase the risk of an unauthorized change to the production environment or the propagation of a vulnerability. A pipeline is not a system to be built or acquired. It is a personal and organizational mindset that defines processes for rapidly developing, fielding, and operating software and software-based systems. A pipeline should use automation where feasible to achieve the desired throughput of new features and capabilities. Multiple roles must perform various steps independently that, with the support of tools and infrastructure, can be integrated into a completed product. Pipeline capabilities must also be structured and maintained. Figure 2 provides a realistic perspective of what is involved and shows that there are two distinct branches that use the same tools, processes, and activities.

A pipeline is a means for building products that support an organization's mission. Details that define what the technology addresses are prepared by developing business cases and requirements. These cases and requirements are further refined to feed into the pipeline to establish the development cadence, as shown in Figure 2. Tools and infrastructure capabilities are selected that allow designers, architects, developers, testers, verifiers, users, and operators to work together to produce the products needed to meet the mission using the pipeline (following the right branch in Figure 2). In addition, a parallel group of participants implement and support the automation that allows product creators to build and facilitate management oversight (following the left branch in Figure 2). Each of these roles requires specialized technical expertise, and each branch relies on the same tools and processes structured through the pipeline. The pipeline must be structured to allow each participant to access what they need to perform their role, and the processes must be arranged so that the work flows through the pipeline and is handed off from one role to the next smoothly from planning to delivery. This automation is unique to each instance of the pipeline and reflects mandates such as the Risk Management Framework (RMF),[1] which provides for monitoring and controls for the governance and management of technology assets by the organization. Components of the pipeline are tailored for the specific products to be delivered by the pipeline. How the pipeline enforces control gates[2] between each step of the flow and how automation is used are uniquely structured to meet the compliance approval needs and control requirements that the pipeline will enforce. In our research to date, we found no standards or guidance for organizations identifying these unique needs. There is extensive information about the tools available for pipeline support from vendors and open source, but there is very little information about how the pipeline should be effectively managed. We found little to help organizations define the scope of management and governance needed to ensure that a pipeline is secure and that it produces products with the appropriate security built in.

[1] RMF is described in NIST Special Publication (SP) 800-37: https://csrc.nist.gov/publications/detail/sp/800-37/rev-2/.

[2] Control gates provide human and automated review to determine when output is ready to move to the next phase [4, p. 22].

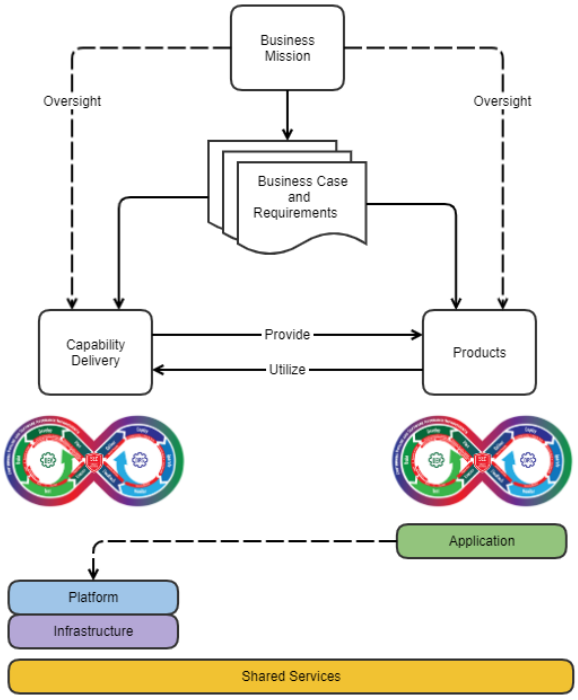


*Figure 2: Integrated Pipeline and Infrastructure*

Consider the current guidance published by the U.S. DoD:

"DoD organizations should define their own processes, choose proper activities, and then select tools suitable for their systems to build software factories and DevSecOps ecosystems." [4]

"The PM [program manager] shall ensure that software teams use iterative and incremental software development methodologies (such as Agile or Lean), and use modern technologies (e.g., DevSecOps pipelines) to achieve automated testing, continuous integration and continuous delivery of user capabilities, frequent user feedback/engagement (at every iteration if possible), security and authorization processes, and continuous runtime monitoring of operational software." [5]

Programs supporting large, complex, software-intensive systems struggle since current guidance fails to provide the necessary details to define and defend a proper balance among features, defensibility, and stability of the pipeline to achieve a program's mission and vision in a cost-effective way. Most guidance paints a picture of a desired technical outcome, but determining such an outcome requires a considerable amount of analysis and interpretation to determine what will work for a specific situation. This analysis and interpretation can result in increased contractor costs and schedule delays. Current guidance does not provide a basis for performing an analysis of alternatives (AoA) to the DevSecOps pipeline tools and processes.

We started by identifying the needed processes, activities, and tools, and then we began evaluating whether each of them was handled with the appropriate security.

## 2. TOOLS MANAGEMENT IS CRITICAL TO PIPELINE MANAGEMENT

Managing the pipeline has had little definition, but this is a critical area for cybersecurity risk. Table 1 lists the eight tool groups that must be structured to connect roles to capabilities in a pipeline. Each component connects with at least two other pipeline components. (See the #Coupling column in Table 1.) Since the pipeline is a blend of development, security, and operational capabilities, the tool groups are also a blend, reflecting interactions that did not exist in earlier structures of the acquisition and development lifecycle. Each tool type requires specific technical skills that must be drawn from the blended environments and work together in a different process flow.

The administrative resources that structure these mappings control what each participant can see and do. This control goes beyond the typical responsibility of authentication and authorization. The administration structures (1) the actions each tool group can perform and (2) how the interface works. For product development and pipeline administration, roles are defined that guide which tool groups can be used and which individuals are assigned to those roles. Management controls, such as separation of duties, are structured and monitored by these administrative resources.

The pipeline flow should move the following processes security as part of each: plan, develop, build, test, release, deliver, deploy, operate, monitor, and feedback. Unfortunately, limited information is available about how this works. Security considerations can be in the control gates that monitor and control the pipeline flow. To build security into the product, each process step must include actions that incorporate security as outlined in Table 2. However, neither of these actions address security for the pipeline's capabilities. The responsibility for pipeline security must be integrated into the roles and responsibilities of those that administer and support these capabilities, similar to how IT infrastructure is supported. To perform their roles, pipeline administrators should perform the similar processes and use similar tools, but they are applied to different content (i.e., use a pipeline tool instead of product code).

*Table 2: Security for DevSecOps Processes [4]*

| Process Type | Process | Security Activities |
|---|---|---|
| Dev | Plan | Threat Model |
| | Code | Secure Coding |
| | Build | SAST, Security as Code |
| | Test | DAST, Pen Test |
| | Release | Digital Sign |
| Ops | Deliver | Secure Transfer |
| | Deploy | Security Configuration and Scan |
| | Operate | Security Patch and Audit |
| | Monitor | Security Monitor |
| | Feedback | Security Analysis |

We determined that there is a range of processes that can be allocated to various roles. (See Table 3.) Each process focuses on a different component of the pipeline, but all processes are needed to keep the pipeline functioning effectively.

*Table 1: DevOps Tooling*

| Tool Group | #Coupling | Interface |
|---|---|---|
| Issue Tracking System | 7 | create, modify, delete, read issues where an issue has some schema definition |
| Code Review System | 2 | create review, start review, add source files to review, add comments to review, create issue from review item, resolve issue from review item, close review |
| Monitoring System | 7 | write message; write metric; display metric; create, modify, delete, read alarm threshold on metric; notify on alarm; show dashboard; process message; extract metric |
| Integration and Test Environment | 3 | deploy system, tear down system, execute tests, collect test results |
| Documentation System | 3 | create, modify, delete document where a document has some schema definition |
| Build System | 6 | execute build; create, modify, delete, read build definition where a build is a collection of steps executed to create artifacts that can be executed |
| Source Control System | 6 | create, modify, delete, read repository; write source files to repository; modify source lines in repository; read repository |
| Communication System | 4 | create, modify, delete, read channel; read and write comment to channel where a channel is an interactive conversation of text between human users with machine users making contributions |

*Table 3: Operational Process*

| Operational Process | Component | Role |
|---|---|---|
| Add Hardware | Host System | infra |
| Code Software | Source Control System<br>Issue Tracking System<br>IdAM<br>Communication System<br>Code Review System | dev |
| Configure Infrastructure | Host System | infra |
| Decommission Hardware | Host System | infra |
| Deploy Application | Any | ops |
| Disaster Recovery | Any | all |
| Install Software | Any | admin |
| Manage Incidents | Monitoring System | admin |
| Manage Users | IdAM System | admin |
| Monitor Infrastructure | Monitoring System | infra |
| Operate Solutions | Any | ops |
| Patch Infrastructure | Host System | infra |
| Patch Software | Any | admin |
| Perform Backup | Any | admin |
| Review Logs | Monitoring System | ops |
| Test Applications | Any | dev |

Increasingly, infrastructure services and development tool types are the target of attacks. Many of these capabilities are supported by third-party software, including open source software, which come to the organization through the supply chain. Successful software security analysis builds on knowledge about how systems were compromised and which mitigations were successfully deployed. Such attacks on development tools are examples of what can go wrong. Common Attack Pattern Enumeration and Classification (CAPEC)[3] provides a comprehensive dictionary of known attack patterns used by adversaries to exploit known weaknesses in cyber-enabled capabilities. CAPEC lists attack patterns by Mechanisms of Attack or Domains of Attack. SQL-Injection attacks appear in the Inject Unexpected Items mechanisms category. The Common Weakness Enumeration (CWE)[4] entry for SQL-Injection includes recommended mitigations.

One of the operational activities needed to address vulnerabilities is "patch software." (See Table 3, entry is marked in green.) To perform this activity for the pipeline, first ensure that only authorized resources can perform the process, and then identify the controls needed to monitor performance of the process.

[3] https://capec.mitre.org/

[4] https://cwe.mitre.org/

Structure the actions that take place and who performs them. The remainder of this section is our initial attempt to assemble this information.

Patching software is viewed from the perspective of software being patched outside the software that the organization produces. Patching application software that is a product of the organization is presumed to follow the procedures and practices of the development team, and it is pushed through the normal DevSecOps cycle and put into production. The process described in this paper follows the patching of software development tools from a vendor that does not deliver updates via an automatic system and the patch itself contains a remediation for some vulnerability. In the scenario below, we consider the vendor to be an untrusted source.

1) **Determination**. The administrator determines from some mechanism that a development tool requires a security patch.
2) **Triage**. The administrator triages, categorizes, and prioritizes the update before deploying it in the organization.
3) **Acquisition**. The administrator acquires the patch from the vendor.
4) **Security**. The administrator determines the authenticity of the patch.
5) **Test Deployment**. The administrator deploys the patch to a test system and performs tests.
6) **Production Deployment**. The administrator deploys the patch to the production system.
7) **Monitoring**. Operations personnel monitor the status of the product system.

**1. Determination**
*Methods*: Manual Checking, Subscription Notification

*Input*: CVE Notification

*Output*: Vendor, CVE, Systems Affected, Change Management (NIST 800-171)

Determining if a patch is needed for software in a system can be done using a few different methods. The most basic approach is for operations personnel to regularly check a published database of vulnerabilities. Vulnerability publishing sources include a risk score and detailed information about the vulnerability that is useful to keep during triage. There might be other reasons for updating or patching software, such as to acquire new features or to satisfy version constraints on other software in the system. Vendors may also notify their customers of a vulnerability through other means, such as by sending email or listing it on their website. Once a vulnerability is identified, the information gathered is carried over into triaging the issue.

**2. Triage**
*Methods*: Risk Assessment, Impact Assessment

*Input*: Vendor, CVE, Systems Affected, Security Policy, Change Management (NIST 800-171)

*Output*: Risk Matrix

Once an update is discovered that affects a system, it is triaged, categorized, and prioritized for being deployed in the organization. Through a risk assessment (as suggested in NIST 800-171), the course of action is determined, which can range from doing nothing to deploying the update immediately. Impact analysis determines the extent to which the vulnerability affects the system and how much work might be involved to update it. The output is a risk matrix that prioritizes the updates needed during operations work.

**3. Acquisition**
*Methods*: Manual Acquisition

*Input*: Risk Matrix, Vendor, Security Policy, Change Management (NIST 800-171)

*Output*: Software Patch

Acquiring the software in this scenario most likely requires the administrator to download the update via the web, but other methods are possible. To mitigate the risk of downloading a patch from an untrusted source, actions to consider include using a secure connection, isolating the patch after downloading, or obtaining the patch on a network separate from the target environment. The actions determined in this step are a consequence of the organization's security policies and the risk assessment performed during triage.

**4. Security**
Methods: Malware Scan, Authenticity Check

Input: Security Policy, Patch, Scan Tool, Authenticity Check Tool, Change Management (NIST 800-171)

Output: Go/No-Go

A software patch may need additional scrutiny to check its authenticity and ensure it doesn't contain malware. Whether the patch came from a trusted or untrusted source, performing both of these checks helps prevent unwanted software from being injected into a system. Once these checks are performed and they succeed to a satisfactory level (and in this case, it would be all or nothing), then the outcome would be Go, and the next step (Test Deployment) can begin. If a check fails, then the action would be No-Go, and notifying the vendor might be warranted.

**5. Test Deployment**
*Methods*: Orchestration, Monitoring, Testing

*Input*: Patch, Deployment Mechanism, Test Criteria, Change Management (NIST 800-171)

*Output*: Go/No-Go, Installation Instructions, Change Management (NIST 800-171) Approval

This step involves installing the patch to a test system, where tests can be performed while the system is monitored for faults. This step requires a test system that duplicates the system being patched. In a cloud environment, this is more easily attained from a cloud service consumer perspective since the service side and the operational slide of the cloud are largely separated. The feasibility of a patch should be determined through repeatable, and appropriately rigorous, definition and procedure. Testing should ensure that the patch (1) installs correctly and without disruption of other co-located software (i.e., dependency version conflict) and (2) runs correctly once installed. Patches that fail tests or that cause compatibility issues elsewhere in the system should be rolled back from the test system. In these cases, the

outcome is No-Go, which implies that there is a rollback procedure or policy in place to recover the state of the test system or to destroy it. Once the tests are satisfied, we can move to the next step (Production Deployment). Update the change management system to record Approval for compliance with NIST 800-171.

### 6. Production Deployment

*Methods*: Orchestration, Monitoring

*Input*: Patch, Installation Instructions

*Output*: Success/Rollback

*Weaknesses*: inadequate testing causes rollback

Deployment to the production system can be done in many different ways, ranging from manual distribution of individual system components to full distribution from an orchestration system. It may be necessary to catalog the change in a change management process as required by NIST 800-171. Careful patching of the system to ensure its compatibility with the production system is needed, and if everything proceeds without changing the working state of the production system, then a successful patch can proceed. If errors are encountered, the system can be rolled back to recover the last working state of the system.

### 7. Monitoring

*Methods*: Monitoring Tools

*Input*: System Affected, Monitoring Configuration

*Output*: Monitoring Alerts

*Weaknesses*: inadequate monitoring allows bad patch to go undetected

Part of DevOps and DevSecOps is monitoring the system's performance, security, and usage metrics. An inventory system that is configured to monitor software versions of system components can inform system operators about the rollout of the patch. However, monitoring in DevOps and DevSecOps isn't just about monitoring the deployed application for health or usage data; it is also useful for tracking and quantifying system attributes, such as the system's software or firmware versions. Once deployed, the patched system is monitored for unexpected behaviors and if any are detected, a Monitoring Alert is issued to identify the issue and provide details about it. If the vendor is an untrusted source, determine how to address the following potential weaknesses:

- The patch source might have been tampered with.
- A scanning and authenticity check might not catch carefully crafted malware.
- The patch might cause a related activity to fail. (For example, changes to a tool whose output is merged with other data could break the merge.)
- The test system doesn't reflect the production system, causing incorrect test results.
- Inadequate testing might require a rollback. (Since operational rollbacks are not on the initial operational list from Table 3, decide who can authorize them and how they are done.)
- Inadequate configuration monitoring allows a bad patch to deploy undetected.

## 3. CONCLUSION

Through our initial exploration of the tools, processes, and activities needed for consideration of pipeline and product security, we identified additional analysis needed for each piece of the pipeline to determine how it should be applied. The information we assembled to date only touches on one of these many activities, but it enables us to begin to reason about potential security weaknesses and undesirable outcomes. Using this information, we can evaluate the controls in the pipeline to verify their sufficiency.

## 10. REFERENCES


[1] Merriam-Webster dictionary definition of "system" https://www.merriam-webster.com/dictionary/system.

[2] SEBoK Glossary. https://www.sebokwiki.org/wiki/Sociotechnical_System_(glossary)

[3] Wikipedia entry for *Information System*. https://en.wikipedia.org/wiki/Information_system

[4] DoD CIO, DoD Enterprise DevSecOps Reference Design. V1, August 2020. p. 15. https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583

[5] Memorandum, Software Acquisition Pathway Interim Policy and Procedures. January 3, 2020. p. 3. https://www.acq.osd.mil/ae/assets/docs/USA002825-19%20Signed%20Memo%20(Software).pdf





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.





DM20-0682