How to secure your CI/CD Pipeline and why it's important

Omid Mircoram, mircoram@kth.se

DD2482 Automated Software Testing and DevOps 2023

I certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

1 Introduction

As the adoption of devops practices grow in the tech industry, development become faster than ever. But with the introduction of technology such as continuous integration and continuous deployment (CI/CD) pipelines, they also introduce new risks and security threats. This essay will discuss some of the risk which are present for every CI/CD pipeline and some suggestions on how to handle them. This paper also aims to discuss a example of failure to combat these threats and its consequences. Lastly, a state of the art technology of machine learning integration will be discussed.

2 CI/CD Pipeline Breach

Before a devops engineer can start securing the pipeline, they need to understand what types of threats there are. A CI/CD pipeline breach falls under the category software supply chain attacks, which is a very broad topic. This type of attack is considered to be one of the most dangerous attacks out of all the security threats present for the software supply chain. There are a myriad of ways the pipeline can be breached, which consequently means that there are a lot of security measures which needs to be taken.

2.1 Breaching the Pipeline

There are 10 main risks published by Cider Security which are the most common attack points. Cider Security also provides their recommendation to counteract them [1]. This essay will discuss the first, second, third and sixth. The first threat is called "insufficient flow control mechanism" (CICD-SEC-1). One of the main characteristics of a pipeline is the speed of which it functions. A developer should be able to write code and push it into production without having to worry about testing it. The pipeline does all this through automation. Insufficient flow control refers to the lack of proper testing and reviews before a code can be pushed into production. The weakness allows the attacker to push harmful code to the pipeline and have it pass all the tests.

The next common risk is called "inadequate identity and access management"(CICD-SEC-2). This becomes an issue the more collaborators a project has. The larger a project grows, the more people want access to push code to the pipeline. As such, it can be hard to keep track of who has access to what, and it's especially hard when there are systems in place which allow users to grant themselves access to accounts such as their personal emails.

The third common vulnerability type is known as "dependency chain abuse"(CICD-SEC-3). This refers to how developers use decencies during both their build as well as their development. There are four main types of attacks for this type of vulnerability;

1. dependency confusion,
2. dependency hijacking,
3. typosquatting,
4. brandjacking

Dependency confusion is the act of uploading infected packages to public repositories with the same name as internal ones in order to attempt the developer to fetch the packet from them instead of the private one. Hijacking requires a larger attack. The attacker has to get control of a public repository, which is owned by the maintainers of a package. Then they upload a new infected version of the package, so that all developers who fetch this new version get infected. Typosquatting and brandjacking is very similar to decency confusion. Squatting requires the attacker to create a public repository with a slightly misspelled version of a commonly used package and brandjacking is the publication of a infected package, but it follows the conventions of a brand package, hoping that developers associates the infected package with the known brand.

Finally, the sixth common threat is called "insufficient credential hygiene" (CICD-SEC-6). This refers to an attackers ability to obtain the credentials used within the pipeline. This can happen in various ways. For starters, not rotating the credentials frequently or updating them with a clear pattern increases this risk since the keys used in a pipeline is exposed to both internal actors such as developers as well as external contractors. This makes it easy to eventually leak. Another problem which is more usual for inexperienced developers is pushing the credentials to a branch or repository available in the supply chain. This exposed the credentials to everyone who can access the repository or its commits. Since even after deletion from the existing newest version of the branch, it still appears in the commit history until manually removed.

2.2 Preventive Actions

In order to counteract these vulnerabilities, there are a few measure that can be taken without changing the workflow in a significant way. For CICD-SEC-1, limiting auto-merges in the pipeline, especially from third parties. Third parties in general should not be allowed to push their code to production without a code review. Another measure which limits the use of the pipeline to a select, approved group of individuals is only allowing code to be pushed through the pipeline if it is pushed from an already approved account.

Since the main cause of CICD-SEC-2 is a lack of control over the access, continuously mapping the access is one way to mitigate this risk. For all accounts with access, the level of access as well as the actual used level of access should be stored. There should be a "per need" level of access, meaning individuals should only have access to the parts they need. Lastly, removing the self-granting access system can also reduce this risk.

CICD-SEC-3 issues are rooted in accidents. In order to counteract this, a few policies can be upheld. Firstly, all packaged pulled should be checked for verification and if available signature verification. This can help reduce the wrongly imported packaged. To counteract the hijacking, make sure to not pull the latest version of the packages. Instead, default to version which have been tested by others and are known to work. One example of this can be to have a team dedicated to dependency testing and have them create a so call bill of materials where all dependencies which are required are tested and a version is given.

TO address the credential management, implementing a policy which ensures that each system and pipeline only has access to the credentials they require is one step which reduces the leakage change. As mentioned before, rotating the credentials is also important in this context. Implementing these together is a common technique since this helps ensure the least privileged principle. Furthermore, scanning the repositories both the current and previous commits for credentials is a good way to ensure that no accidental leaks occur.

Following that point, access to non-humans also has to be mapped and managed properly [2]. This means that the communications between the pipeline and outside services or servers should be blocked unless actively needed. This is also beneficial if infected code does infiltrate the pipeline, since it cannot communicate with outside sources which are not authorized by the pipeline.

Aside from these preventive measures which are directly connected to the CICD-SECs mentioned above, there are other actions which can be taken in order to strengthen the pipeline. Firstly, monitoring the entire pipeline for suspicious behaviour such as attempts to accessing third party services or changes to privileges can help detect attempts of breaching the pipeline [2]. Logs are usually used after an event has occurred to figure out what went wrong but with appropriate monitoring, they can help stop an active attempt.

Another method of securing the pipeline is setting traps. The traps are called "honey tokens" and are credential that look legitimate but are not functional. Instead, when they are used they trigger an alarm for the pipeline owners. This can be done at a larger scale, where whole systems are implemented. These are called "honey pots" [3]. These are used for a different purpose. Honey pots are used to gather knowledge about the nature of the attack and the tactics used by the attacker while honey tokens help identify the attacker.

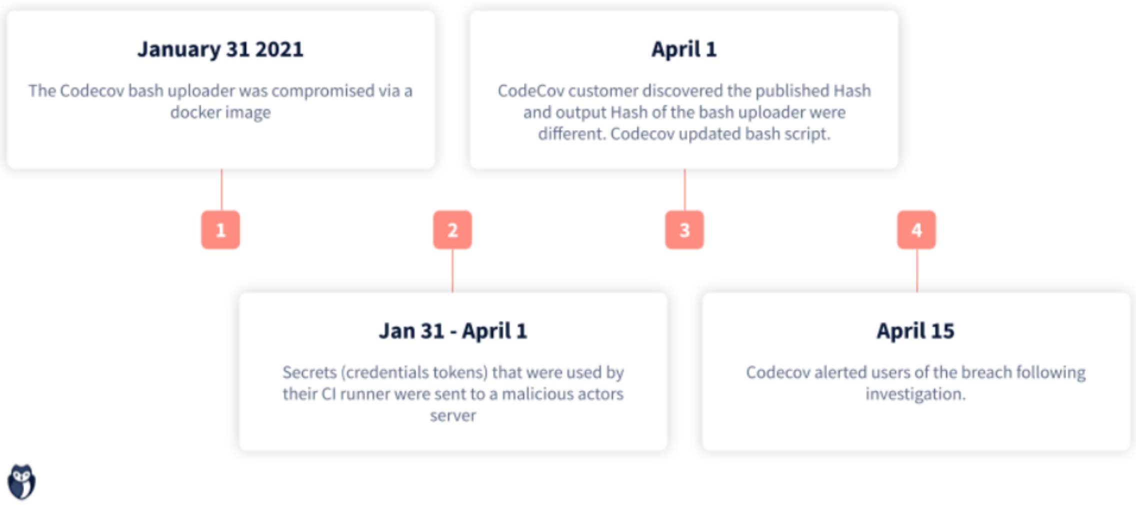
January 31 2021
The Codecov bash uploader was compromised via a docker image
April 1
CodeCov customer discovered the published Hash and output Hash of the bash uploader were different. Codecov updated bash script.
1
2
3
4
Jan 31 - April 1
Secrets (credentials tokens) that were used by their CI runner were sent to a malicious actors server
April 15
Codecov alerted users of the breach following investigation.

Figure 1: Timeline by gitguardian

3 The CodeCov Breach

One of the most popular examples of a pipeline breach is the incident at CodeCov 2021. This breach had a series of the vulnerabilities presented by Cider Security. According to Carolynn van Arsdale at ReversingLabs, the involved CICD-SEC vulnerabilities are 5,6,8,9 and 10 [4]. The attack was successful because CodeCov had failed to secure their credentials and some of them could be extracted from a docker image. This allowed the attackers to modify a script used in their pipeline. They added a line which sent all credentials that passed through the script to the attackers. The attack was only discovered once a customer of CodeCov noted that the bash script had a different hash than what it usually produces. This led to an investigation and the attack was resolved.

There are a few examples of actions which has been discussed previously that would have helped prevent this attack. Firstly. the breach started with the attackers getting access to credentials from an image. This is in clear violation of CICD-SEC-6, which specifically point to docker images as a point of weakness. Rotating credentials in this case might not have been useful, since the new docker image would expose it none the less, but at the very least this would have created another hurdle for the attacker. Furthermore, the attackers added a line of code to a file which communicated with an external server which also should have set of some warning inside the pipeline.

4 Machine Learning in the Pipeline

Alongside the rise of DevOps, machine learning is also another popular trend which has gained some traction recently. These two have also merged into another topic called MLOps, which refers to the integration of machine learning with DevOps. There are many different examples of this, for instance quality assurance [5], TinyML which aims to integrate machine learning into embedded systems and many more subjects [6]. This paper will focus on the area of integration between ML and the CI/CD pipeline. Using a CI/CD pipeline for ML development can help keep the model updated and maintained properly [7]. Using a CI/CD pipeline for ML development can feed the model new, updated data at a much faster rate than regular ML development. Testing the model is also done as part of the CI/CD pipeline, which helps ensure that the new delivery will not malfunction with the new data [7]. This type of development is called MLOps level 2 [8]. An example of how an automated CI/CD MLOps pipeline can be configures is shown in figure 2.

Machine learning can also be used by regular DevOps teams to improve their pipelines. Integrating machine learning into a regular CI/CD pipelines can have a series of benefits for both the security as well as the efficiency. For instance, feeding data from devops tools such as git commits or from Jira can help detect abnormal code as well as develop tests for quality assurance [7]. Furthermore, it can help manage errors along the pipeline which could have gone undetected with regular system testing because of the capacities of machine learning models. It can also help detect and classify errors within the pipeline, which can help reduce the workload for developers should a so called alarm storm trigger.

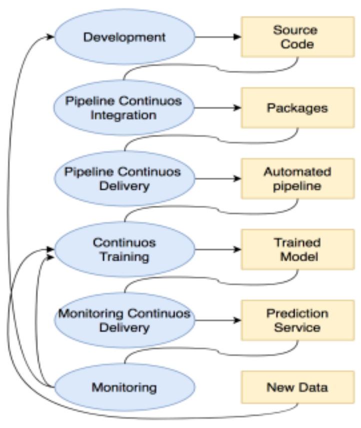

Figure 2: Figure from [8]

5 Conclusion

With all these topics discussed, it is clear that re-enforcing a CI/CD pipeline takes a lot of work and understanding of both the development process as well as the operations. It is important for a devops engineer to understand what parts of the pipeline can be considered a vulnerability and how to properly address these issues. This essay discussed CICD-SEC 1,2,3 and 6 which are often present in CI/CD pipelines, and also presented a myriad of actions which can be taken to address them. To address CICD-SEC-1, properly managing third parties and their integration to the pipeline and code will help combat the risk of an attack. Blocking third parties is virtually impossible for even mid-size projects and options such as approved accounts should be considered instead. Mapping all with access to the pipeline is a good idea to combat the second security threat. It also provides a better understanding of what goes in the pipeline in the first place and can help remove unnecessary access. Having systems in place for third party package inclusion will help reduce the risk of a dependency attack and could also reduce the headache of version handling between different packages. Lastly, the sixth security thread can be combated through constant monitoring for leakages of credentials in the pipeline as well as a rotation system of said credentials.

This essay also presented a case where a company failed to secure their pipeline and looked at the consequences. CodeCov breach had many different attack points, but was rooted in their failure to address the sixth security threat. As such, they lost customers, credibility and had to work on restructuring their pipeline. Other companies also had to look over their side to ensure that the attack did not spread into their systems.

To summarize, the most important thing a devops engineer has to be able to do is understand the threats present for the pipeline and be able to properly address them. Furthermore they have to understand that no one system will help prevent all attack types, which is why the understand of how the pipeline is attack is important.

6 Reflection

While this paper presents some ways to handle issues connected to the pipeline, a devops engineer has to be able to do a lot more than these things. Securing a pipeline is only part of the work. Arguably, it's more important to be able to set up a pipeline properly, and maybe even avoid some of the problems from the get go. Furthermore, the issues presented in this essay are mostly threats which can be handled through technical applications. But other issues which are not discussed has to be handled through education of other developers. A CI/CD pipeline is a shared responsibility.

References

[1] D. Krivelevich and O. Gil, “Top 10 CI/CD Security Risks,” 2022.

[2] D. MCcdaniel, “How To Secure Your CI/CD Pipeline,” 2023.

[3] Fortinet, “What Are Honey Tokens?,” 2023.

[4] C. van Arsdale, “5 CI/CD breaches analyzed: Why you need to update your software security approach,” 2022.

[5] A. Chatterjee, B. S. Ahmed, E. Hallin, and A. Engman, “Quality assurance in mlops setting: An industrial perspective,” 2022.

[6] S. Hymel, C. Banbury, D. Situnayake, A. Elium, C. Ward, M. Kelcey, M. Baaijens, M. Majchrzycki, J. Plunkett, D. Tischler, A. Grande, L. Moreau, D. Maslov, A. Beavis, J. Jongboom, and V. J. Reddi, “Edge Impulse: An MLOps Platform for Tiny Machine Learning,” 2023.

[7] D. S. Battina, “Improving La Redoute’s CI/CD Pipeline and DevOps Processes by Applying Machine Learning Techniques,” 2021.

[8] S. Garg, P. Pundir, G. Rathee, P. Gupta, S. Garg, and S. Ahlawat, “On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps,” in *2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 25–28, 2021.