Argo CD Secrets

Nicolas Iooss
nicolas.iooss@ledger.fr

Ledger Donjon

Abstract. Argo CD is a tool designed to manage Continuous Deployment pipelines between code repositories and Kubernetes clusters. As it is granted important privileges (it runs by default as cluster administrator), it is important to ensure it is deployed securely. It heavily relies on standard Kubernetes objects and store sensitive values in Kubernetes Secrets. What happens when the content of these Secrets is compromised? This question is all the more important when a security incident happens.

This article presents how Argo CD uses its Kubernetes Secrets and provides some recommendations to help ensure the security.

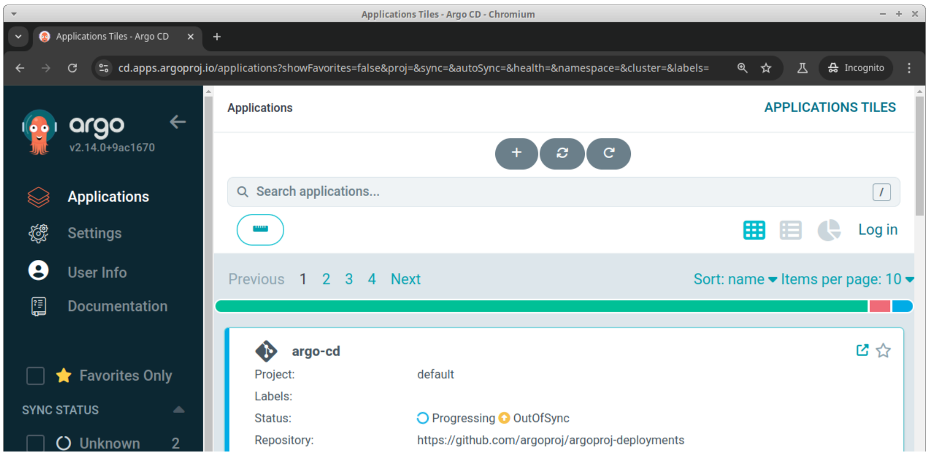
1 Introduction

Managing applications deployed in Kubernetes clusters can be very complex. Several projects and tools were created to tackle these challenges. Nowadays, it is common to hear companies use "CI/CD pipelines" (Continuous Integration and Continuous Delivery [1]) to ease deploying and managing some applications in production environments. This leverages some components which bridge the source code hosting platform (such as GitHub [2] or GitLab [3]) with the places where applications run, like Kubernetes clusters.[4] Many new terms emerged over time: "Infrastructure as Code", "DevOps", "GitOps", etc.

This article focuses on a specific project commonly used in such contexts: Argo CD.[5]

From developers' perspective, Argo CD provides a lightweight way to deploy applications and to monitor their health: most of its configuration happens in files in git repositories or in Kubernetes resources; the web interface provides a summary of the state of the application, as well as

[1] https://en.wikipedia.org/wiki/CI/CD
[2] https://github.com/
[3] https://about.gitlab.com/
[4] https://kubernetes.io/
[5] https://argo-cd.readthedocs.io/en/stable/


Applications Tiles - Argo CD - Chromium
Applications Tiles - Argo CD
cd.apps.argoproj.io/applications?showFavorites=false&proj=&sync=&autoSync=&health=&namespace=&cluster=&labels=
Incognito
argo
v2.14.0+9ac1670
Applications
Settings
User Info
Documentation
Favorites Only
SYNC STATUS
Unknown 2
Applications
APPLICATIONS TILES
Search applications...
Log in
Previous 1 2 3 4 Next
Sort: name Items per page: 10

argo-cd

Project: default

Labels:

Status: Progressing OutOfSync

Repository: https://github.com/argoproj/argoproj-deployments

Fig. 1. Argo CD demonstration website, `https://cd.apps.argoproj.io/`

a sneak peek at the generated logs and events; the command-line tool enables to easily automate some tasks.

What about its security? It seems to be taken very seriously:

— The documentation includes a page titled "Security"[6] with great details about how security-relevant topics are handled (authentication, authorization, logging, etc.).

— A security policy has been published on GitHub repository `argoproj/argo-cd`.[7] It includes information about supported versions, fixed vulnerabilities and a bug bounty program.

— Argo CD is mostly written in Go, a language helping protect against software memory safety issues.[8]

— The release artifacts are cryptographically signed and attested. The documentation explains how to check signatures and attestations.[9]

— There have been at least two security audits with public reports, in 2021 by Trails of Bits [2] and in 2022 by Ada Logics [6]. Before them, Soluble published five security issues in a blog post in 2020 [4]. In

[6] `https://argo-cd.readthedocs.io/en/release-2.13/operator-manual/security/`

[7] `https://github.com/argoproj/argo-cd/security`

[8] According to the NSA [1], "Some examples of memory safe languages are [. . .], Go.". Even though it was claimed that data races could break the memory safety guarantees (in `https://blog.stalkr.net/2015/04/golang-data-races-to-break-memory-safety.html` and `https://blog.stalkr.net/2022/01/universal-go-exploit-using-data-races.html`), Go is widely considered as memory-safe.

[9] `https://argo-cd.readthedocs.io/en/release-2.13/operator-manual/signed-release-assets/`

2022 Trend Micro also published a blog post in May 2022 [7] where it explained how admin's initial password was stored.

Moreover, its demo website[10] (figure 1) provides anonymous read-only access. This emphasizes granting read-only access to an Argo CD instance should not enable attackers to read sensitive data or modify the deployed applications. For example, the hash of the admin password is stored in a Kubernetes Secret[11] named `argocd-secret`. The demo website displays the content of this Secret[12]:

```
data:
  admin.password: ++++++++
  admin.passwordMtime: ++++++++
  server.secretkey: ++++++++
  tls.crt: ++++++++
  tls.key: ++++++++
kind: Secret
```

The sensitive values were redacted in the web interface and there is no way to edit the Secret. If there were, this would be considered as a vulnerability in Argo CD (and should be reported to Argo CD's security team).

This philosophy goes beyond the interface. Indeed, contrary to most mainstream web application frameworks, Argo CD does not use a database to store its persistent data. It instead only relies on Kubernetes objects, using the standard ones and some Custom Resources of its own. This has the consequence that every persistent sensitive piece of information used by Argo CD has to be stored in a Kubernetes Secret. This includes configuration values in the previously described Secret `argocd-secret`, credentials to access other managed clusters,[13] the initial Redis credentials,[14] etc.

Despite such a strong security posture, Argo CD can be configured in ways creating vulnerabilities. In a blog post published in December

[10] `https://cd.apps.argoproj.io/`

[11] `https://kubernetes.io/docs/concepts/configuration/secret/`

[12] `https://cd.apps.argoproj.io/applications/argocd/argo-cd?view=tree&resource=&node=%2FSecret%2Fargocd%2Fargocd-secret%2F0`

[13] function `CreateCluster` is creating a Secret to store the configuration of the managed cluster, in `https://github.com/argoproj/argo-cd/blob/v2.13.1/util/db/cluster.go#L109`. There also is a Secret containing a persistent service account token in each managed cluster, created if needed by function `getOrCreateServiceAccountTokenSecret` in `https://github.com/argoproj/argo-cd/blob/v2.13.1/util/clusterauth/clusterauth.go#L258-L332`.

[14] command `argocd admin redis-initial-password` creates this secret in `https://github.com/argoproj/argo-cd/blob/v2.13.1/cmd/argocd/commands/admin/redis_initial_password.go#L48-L72`.

2024 [5], I studied two examples where Argo CD was deployed in ways which unexpectedly enabled privilege escalation and authentication bypass. In the second example, an attacker started their attack on Argo CD through accessing its Secrets. This kind of lateral movement attack curiously seems to be missing from the public state of the art related to Kubernetes cluster security. This article presents this example once again, highlighting the importance of Kubernetes Secrets.

2 Deployment Use-Case

Argo CD stores all its settings in Kubernetes resources such as Kubernetes ConfigMaps and Secrets. The Secrets can be synchronized with other secret management systems like AWS Secrets Manager,[15] HashiCorp Vault,[16] etc. A possible way to do this consists in deploying External Secrets Operator (ESO) [17] in a cluster. Such a configuration appears to be quite common, according to presentations given at public conferences, like one given at KubeCon + CloudNativeCon Europe 2024 [3].

The security policy around the secret management system is sometimes not fine-grained enough. For the studied use-case, let's consider an AWS account having two EKS [18] clusters for different purposes: "Cluster A" and "Cluster B". The administrator followed the official documentation to configure their AWS account.[19] Each ESO service account on Kubernetes is associated with an AWS IAM [20] role with the permission to read all secrets (action `secretsmanager:GetSecretValue` on resource `arn:aws:secretsmanager:eu-west-3:111122223333:secret:*`). Here is the associated IAM Policy, created after reading the documentation:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds"
```

[15] `https://aws.amazon.com/secrets-manager/`
[16] `https://www.vaultproject.io/`
[17] `https://external-secrets.io/`
[18] Amazon Elastic Kubernetes Service `https://aws.amazon.com/eks/`
[19] `https://external-secrets.io/v0.10.6/provider/aws-secrets-manager/`
[20] AWS Identity and Access Management `https://aws.amazon.com/de/iam/`

```
      ],
      "Resource": [
        "arn:aws:secretsmanager:eu-west-3:111122223333:secret:*"
      ]
    }
  ]
}
```

To make things more precise, this example considers that Argo CD is installed in Cluster A, its Kubernetes Secret `argocd-secret` is synchronized with AWS Secrets Manager, and an attacker managed to compromise Cluster B (figure 2). This means that the attacker can impersonate the External Secrets Operator deployed in Cluster B to read all secrets stored in the shared AWS Secrets Manager.

In such a scenario, can the attacker move to Cluster A?

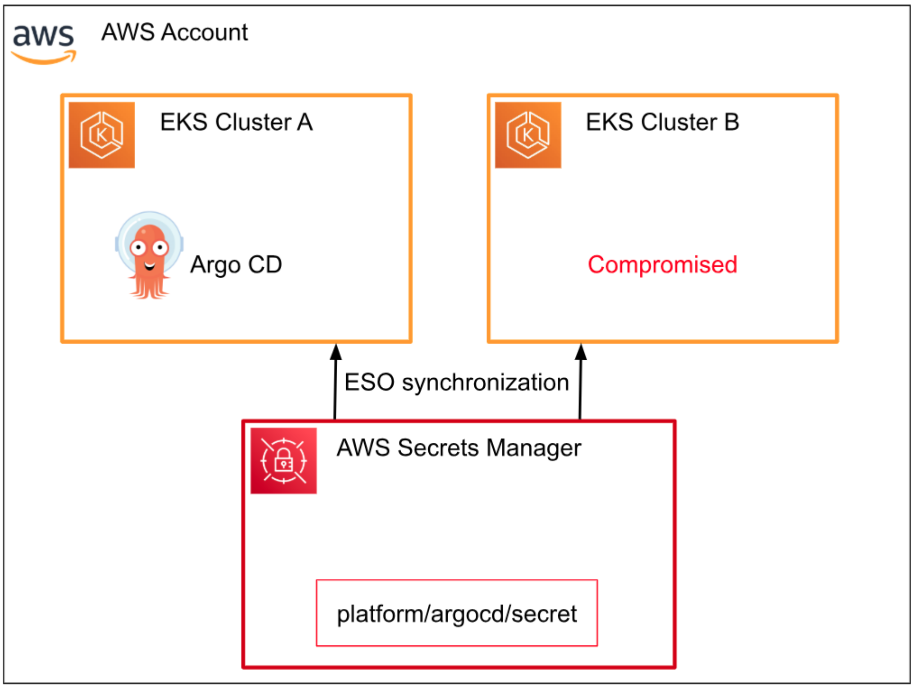


Fig. 2. Clusters using External Secrets Operator in the same AWS account

3 Lateral Movement Attack

The attacker can configure their AWS command-line to use the authentication token from Cluster B's ESO.[21] They can then read the AWS secret associated with `argocd-secret` with a command such as `aws secretsmanager get-secret-value --secret-id platform/argocd/secret` (the name of the AWS secret could be guessed or obtained through other means):

```
{
  "ARN": "arn:aws:secretsmanager:eu-west-3:[...]",
  "Name": "platform/argocd/secret",
  "SecretString": "{\"admin.password\":\"$2a$10$4sRnwtvm9XMbSAPDZyImTeh3
↪  7MREP6yDnRfellIQamT/cuMn5Jgm.\",\"server.secretkey\":\"JA+Lqmv/d7TbM
↪  8yrOEIT+cRIsJAGxAxrqo6ghgOK9MQ=\"}",
```

The `SecretString` contains the password hash of the admin user, in the field `admin.password`. The attacker could attempt to guess the password or to crack it through brute-force methods. However, such attacks would likely fail due to the password being randomly generated (in this example, the password was generated by Argo CD and its value was `AFZTfDcfHySb2Skv`)."

Moreover, if the AWS IAM policy used by Cluster B's ESO included `secretsmanager:PutSecretValue` (which is required for ESO feature `PushSecret`), the attacker would be able to modify the password hash. This would enable them to impersonate the admin user.

In the general case, knowing the hash in `admin.password` does not help the attacker much. But the Kubernetes Secret contains another field, `server.secretkey`. What is it used for?

In Argo CD's source code, `server.secretkey` is called the "server signature key". It is used to sign and verify a session token with HMAC-SHA256 in `argo-cd:util/session/sessionmanager.go`:

```go
func (mgr *SessionManager) signClaims(claims jwt.Claims) (string, error) {
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    // ...
    return token.SignedString(settings.ServerSignature)
}
```

[21] for example by configuring relevant environment variables such as `AWS_ROLE_ARN`, `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_SESSION_NAME` as documented by AWS in `https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html`

Knowing the key should be enough to forge a token to impersonate the user admin. There are some caveats to take care of (function GetSubjectAccountAndCapability requires the subject claim to actually be admin:login; function Parse requires the token to have a non-empty ID in claim jti; the issuer has to be argocd). Here is some Python code which forges a token valid for 24 hours, solving the difficulties:

```python
import base64
import json
import hmac
import time

def b64url_encode(data: bytes) -> bytes:
    return base64.urlsafe_b64encode(data).rstrip(b"=")

def forge_jwt(key: str, audience: str = "argocd") -> str:
    now = int(time.time())
    header = json.dumps({
        "alg": "HS256",
        "typ": "JWT",
    }).encode("ascii")
    claims = json.dumps({
        "iss": "argocd",
        "aud": audience,
        "iat": now,
        "nbf": now,
        "exp": now + 24 * 3600,
        "sub": "admin:login",
        "jti": "01234567-89ab-cdef-0123-456789abcdef",
    }).encode("ascii")
    signed = b64url_encode(header) + b"." + b64url_encode(claims)
    signature = hmac.digest(key.encode("ascii"), signed, "sha256")
    token = signed + b"." + b64url_encode(signature)
    return token.decode("ascii")

print(forge_jwt("JA+Lqmv/d7TbM8yrOEIT+cRIsJAGxAxrqo6hghOK9MQ="))
```

In a web browser, defining cookie argocd.token with the produced token is enough to bypass the login screen and successfully authenticate as admin.

Even though the obtained administrator privileges enable many actions in Argo CD, it does not enable reading Kubernetes Secrets or impersonating service accounts. It is nevertheless possible to deploy new Argo CD applications (if the underlying Kubernetes cluster enables it, which is usually true). The attacker can then deploy their own Helm chart with a Kubernetes Job[22] running commands with cluster administration privileges.

4 Recommendations to Mitigate the Attack

First, the Kubernetes Secret `argocd-secret` is very sensitive, as anyone who knows it can impersonate any local user in Argo CD, including `admin`. If it is synchronized with ESO, access to the Secrets Manager should be properly defined to prevent unauthorized access.

Second, as documented in Argo CD's documentation, the initial admin password should be modified, and the new one should be robust enough so that gaining access to the password hash does not enable an attacker to guess it. Moreover, when administrators are authenticated through some SSO (Single Sign On), disabling the local admin account successfully prevents the described attack (usually, session tokens of SSO users are not signed by `server.secretkey`).

Third, it reduces the impact of the attack to run Argo CD without cluster administration privileges and to apply some well-known best practices to harden the Kubernetes cluster. This includes configuring it to reject creating privileged pods in some namespaces, reducing the privileges of service accounts and configuring fine-grained AWS IAM policies for external resources.

Finally, if a security incident response analysis finds out the attacker managed to read `argocd-secret`, it makes sense to consider the attacker gained cluster administration privileges and to act accordingly. This may include reviewing the logs looking for more lateral movements, searching for some backdoors left by the attacker, revoking all access tokens, regenerating all passwords as well as `server.secretkey`, etc.

5 Conclusion

This article presents a kind of lateral movement attack in a context where Kubernetes clusters share the same external storage for their Secrets.

Even though Argo CD employs state-of-the-art security practices, this illustrates the importance of considering how it has been deployed when assessing its security.

[22] `https://kubernetes.io/docs/concepts/workloads/controllers/job/`

References

1. National Security Agency. NSA Releases Guidance on How to Protect Against Software Memory Safety Issues. `https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/`, November 2022.
2. Dominik Czarnota, David Pokora, and Mike Martel. Trail of Bits. Argo Security Assessment. `https://github.com/argoproj/argoproj/blob/2db4cda94956307ee080f51759aa6fcbda841f28/docs/argo_security_final_report.pdf`, March 2021.
3. Mads Høgstedt Danquah and Jeppe Lund Andersen. The LEGO Group. Keeping the Bricks Flowing: The LEGO Group's Approach to Platform Engineering for Manufacturing. `https://youtu.be/SmeekXGYuFU`, March 2024.
4. Matt Hamilton. Soluble. Argo CVEs. `https://web.archive.org/web/20220330042723/https://www.soluble.ai/blog/argo-cves-2020`, April 2020.
5. Nicolas Iooss. Ledger Donjon. Argo CD Security Misconfiguration Adventures. `https://www.ledger.com/argo-cd-security-misconfiguration-adventures`, December 2024.
6. Adam Korczynski and David Korczynski. Ada Logics. Argo Security Assessment. `https://github.com/argoproj/argoproj/blob/2db4cda94956307ee080f51759aa6fcbda841f28/docs/argo_security_audit_2022.pdf`, July 2022.
7. Magno Logan. Trend Micro. Abusing Argo CD, Helm, and Artifact Hub: An Analysis of Supply Chain Attacks in Cloud-Native Applications. `https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/abusing-argo-cd-helm-and-artifact-hub-an-analysis-of-supply-chain-attacks-in-cloud-native-applications`, May 2022.