PRESENTS

Argo Security Audit

In collaboration with the Argo project maintainers and The Open Source Technology Improvement Fund

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 18th July 2022

This report is licensed under Creative Commons 4.0 (CC BY 4.0)
# Table of Contents

## Table of Contents
- Issues by project  
- Executive Summary  
- Project Information  
- Fuzzing  
- Issues Found  
  - CVEs  
  - All issues  
  - ADA-ARGO-SA-01: XSS in Argo CD Chat URL  
  - ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript  
  - ADA-ARGO-SA-03: XSS in Argo Workflows chat url  
  - ADA-ARGO-SA-04: XSS in Workflows workflow link  
  - ADA-ARGO-SA-05: XSS in Pod Logs link  
  - ADA-ARGO-SA-06: XSS in sensor logs link  
  - ADA-ARGO-SA-07: XSS in Event Source logs url  
  - ADA-ARGO-SA-08: XSS in Pod link url  
  - ADA-ARGO-SA-09: XSS through server side rendering  
  - ADA-ARGO-SA-10: Denial of service from malicious trigger parameters  
  - ADA-ARGO-SA-11: Insufficient logging when executing a trigger  
  - ADA-ARGO-SA-12: 8 Uses of deprecated API can be used to cause DoS in user-facing endpoints  
  - ADA-ARGO-SA-13: Insecure path traversal in Git Trigger Source can lead to arbitrary file read  
  - ADA-ARGO-SA-14: Insufficient logging if OIDC Provider is created with nil TLS config  
  - ADA-ARGO-SA-15: Path traversal leading to arbitrary YAML file read with malicious Helm chart  
  - ADA-ARGO-SA-16: Path traversal leading to reading of out of bounds manifest files  
  - ADA-ARGO-SA-17: DoS through large manifest files  
  - ADA-ARGO-SA-18: nil-pointer in 3rd-party library 2  
  - ADA-ARGO-SA-19: DoS through well-crafted Boolean Expression  
  - ADA-ARGO-SA-20: Unmaintained 3rd-party dependencies  
  - ADA-ARGO-SA-21: Unused function parameters  
  - ADA-ARGO-SA-22: Functions always returning nil  
  - ADA-ARGO-SA-23: Nil-pointer dereferences in sensor controller  
  - ADA-ARGO-SA-24: Use of math/rand in production code  
  - ADA-ARGO-SA-25: Improper use of InsecureSkipVerify  
  - ADA-ARGO-SA-26: Insecure production use of crypto/ssh.InsecureIgnoreHostKey  

- Follow-up on fixes for existing security issues

---

**Note: The table of contents includes issue numbers and titles for the issues found during the fuzzing process.**
Issues by project

Argo CD
ADA-ARGO-SA-01: XSS in Argo CD Chat URL
ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript
ADA-ARGO-SA-09: XSS through server side rendering
ADA-ARGO-SA-14: Insufficient logging if OIDC Provider is created with nil TLS config
ADA-ARGO-SA-15: Path traversal leading to arbitrary file read with malicious Helm chart
ADA-ARGO-SA-16: Path traversal leading to reading of out of bounds manifest files
ADA-ARGO-SA-17: DoS through large manifest files

Argo Workflows
ADA-ARGO-SA-03: XSS in Argo Workflows chat url
ADA-ARGO-SA-04: XSS in Workflows workflow link
ADA-ARGO-SA-05: XSS in Pod Logs link
ADA-ARGO-SA-06: XSS in sensor logs link
ADA-ARGO-SA-07: XSS in Event Source logs url
ADA-ARGO-SA-08: XSS in Pod link url

Argo Events
ADA-ARGO-SA-10: Denial of service from malicious trigger parameters
ADA-ARGO-SA-11: Insufficient logging when executing a trigger
ADA-ARGO-SA-12: 8 Uses of deprecated API can be used to cause DoS in user-facing endpoints
ADA-ARGO-SA-13: Insecure path traversal in Git Trigger Source can lead to arbitrary file read
ADA-ARGO-SA-18: nil-pointer in 3rd-party library 2
ADA-ARGO-SA-19: DoS through well-crafted Boolean Expression
ADA-ARGO-SA-20: Unmaintained 3rd-party dependencies
ADA-ARGO-SA-21: Unused function parameters
ADA-ARGO-SA-22: Functions always returning nil
ADA-ARGO-SA-23: Nil-pointer dereferences in sensor controller
ADA-ARGO-SA-24: Use of math/rand in production code
ADA-ARGO-SA-25: Improper use of InsecureSkipVerify
ADA-ARGO-SA-26: Insecure production use of crypto/ssh.InsecureIgnoreHostKey
Executive Summary

This report outlines the security audit carried out by Ada Logics of the software packages in the Argo project’s ecosystem. The audit was carried out in March and April of 2022 and Ada Logics also provided assistance following the core audit on managing and fixing the security issues discovered. The focus of this engagement was split across four tasks:

1. Review and validate the fixes by Argo from their 2021 external security review.
2. Audit the manifests in the Argo repository and handling of these.
3. Audit the UI code of Argo projects.
4. Audit the event triggers and event sources in Argo Events.

Ada Logics also developed 7 fuzzers into the existing Argo fuzzing suite. This was enabled by Argos independent fuzzing audit from earlier in 2022, where continuous fuzzing was integrated into several of the Argo projects.

This engagement was allocated 25 days of person effort, which was split amongst two security researchers from Ada Logics. Ada Logics and the Argo team met regularly during these 5 weeks to discuss findings and progress, and the findings were handed over to the Argo team that then triaged the issues and fixed the ones that required security advisories. At the time of release, all issues have been triaged by the Argo team, and all exploitable issues have been patched and published as security advisories on GitHub. 9 issues were assigned CVEs ranging from low to critical severity. 7 CVEs were found in ArgoCD and 2 in Argo Events.

In this document we will briefly outline the fuzzing efforts of the audit and then focus on listing the issues discovered with detailed technical details for each of the issues.

Repositories in scope

- https://github.com/argoproj/argo-events
- https://github.com/argoproj/argo-workflows
- https://github.com/argoproj/argo-cd
- https://github.com/argoproj/argo-rollouts/tree/master/ui/src

Results summarised

<table>
<thead>
<tr>
<th>26 security issues found</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 CVEs</td>
</tr>
<tr>
<td>- 1 critical severity</td>
</tr>
<tr>
<td>- 4 high severity</td>
</tr>
<tr>
<td>- 3 moderate severity</td>
</tr>
<tr>
<td>- 1 low severity</td>
</tr>
<tr>
<td>7 fuzzers added</td>
</tr>
<tr>
<td>Review of fixes of previous security audit</td>
</tr>
</tbody>
</table>
Project Information

Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>03/05/2022</td>
<td>First version shared with the Argo team</td>
</tr>
<tr>
<td>1.1</td>
<td>13/07/2022</td>
<td>Final version of report delivered.</td>
</tr>
</tbody>
</table>

Contacts

Ada Logics

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Role</th>
<th>Email address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam Korczynski</td>
<td>Security Engineer</td>
<td>Lead</td>
<td><a href="mailto:Adam@adalogics.com">Adam@adalogics.com</a></td>
</tr>
<tr>
<td>David Korczynski</td>
<td>Security Researcher</td>
<td>Support</td>
<td><a href="mailto:David@adalogics.com">David@adalogics.com</a></td>
</tr>
</tbody>
</table>

Open Source Technology Improvement Fund

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amir Montazery</td>
<td><a href="mailto:Amir@ostif.org">Amir@ostif.org</a></td>
</tr>
<tr>
<td>Derek Zimmer</td>
<td><a href="mailto:Derek@ostif.org">Derek@ostif.org</a></td>
</tr>
</tbody>
</table>

Argo Team

<table>
<thead>
<tr>
<th>Name</th>
<th>Email address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex Collins</td>
<td><a href="mailto:Alex_Collins@intuit.com">Alex_Collins@intuit.com</a></td>
</tr>
<tr>
<td>Derek Wang</td>
<td><a href="mailto:Derek_Wang@intuit.com">Derek_Wang@intuit.com</a></td>
</tr>
<tr>
<td>Hari Rongali</td>
<td><a href="mailto:Hari_Rongali@intuit.com">Hari_Rongali@intuit.com</a></td>
</tr>
<tr>
<td>Henrik Blixt</td>
<td><a href="mailto:Henrik_Blixt@intuit.com">Henrik_Blixt@intuit.com</a></td>
</tr>
<tr>
<td>Jann Fischer</td>
<td><a href="mailto:Jann@mistrust.net">Jann@mistrust.net</a></td>
</tr>
<tr>
<td>Jesse Suen</td>
<td><a href="mailto:Jesse@akuity.io">Jesse@akuity.io</a></td>
</tr>
</tbody>
</table>
Fuzzing

In addition to the agreed upon scope, Ada Logics identified several areas in the Argo Events code base that would benefit from having fuzzing coverage. Argos recent independent fuzzing audit\(^1\) showed that fuzzing helps Argo to automatically and continuously find bugs. In this audit, Ada Logics wrote 7 new fuzzers and added them to Argos OSS-Fuzz integration to allow them to run continuously. The fuzzers found a few issues which have been included in this report.

The fuzzers were uploaded to the CNCF-Fuzzing repo: https://github.com/cncf/cncf-fuzzing/tree/main/projects/argo

### New fuzzers

<table>
<thead>
<tr>
<th>Fuzzer Name</th>
<th>Link</th>
</tr>
</thead>
</table>

---

\(^1\) https://github.com/argoproj/argoproj/blob/dd7cae43d81c5a11f21ff4ea0a4afadcae4799c7/docs/audit_fuzzer_adalogics_2022.pdf
Issues Found

In this section we iterate through all of the issues discovered throughout the audit. First, we list the issues in table form to give an overview of the results and then proceed to give a detailed description of each of the issues found.

CVEs

<table>
<thead>
<tr>
<th>Issue #</th>
<th>CVE</th>
<th>Github Advisory</th>
<th>Severity (CVSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA-ARGO-SA-02</td>
<td>CVE-2022-31035</td>
<td>GHSA-h4w9-6x78-8vrj</td>
<td>Critical (9.0)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-09</td>
<td>CVE-2022-31102</td>
<td>GHSA-pmjg-52h9-72qv</td>
<td>Low (2.6)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-12</td>
<td>CVE-2022-31054</td>
<td>GHSA-5q86-62xr-3r57</td>
<td>High (7.5)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-13</td>
<td>CVE-2022-25856</td>
<td>GHSA-qpgx-64h2-gc3c</td>
<td>High (7.5)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-15</td>
<td>CVE-2022-31036</td>
<td>GHSA-q4w5-4q2-98vm</td>
<td>Moderate (4.3)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-16</td>
<td>CVE-2022-24904</td>
<td>GHSA-6g5c-gp2x-q54h</td>
<td>Moderate (4.3)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-17</td>
<td>CVE-2022-31016</td>
<td>GHSA-jhgp-vf4w-rpwq</td>
<td>Moderate (6.5)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-24</td>
<td>CVE-2022-31034</td>
<td>GHSA-2m7h-86qq-fp4v</td>
<td>High (8.3)</td>
</tr>
<tr>
<td>ADA-ARGO-SA-25</td>
<td>CVE-2022-31105</td>
<td>GHSA-7943-82jg-wmw5</td>
<td>High (8.3)</td>
</tr>
</tbody>
</table>

All issues

<table>
<thead>
<tr>
<th>Issue #</th>
<th>CVE</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADA-ARGO-SA-01</td>
<td>ADA-ARGO-SA-01: XSS in Argo CD Chat URL</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-02</td>
<td>ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-03</td>
<td>XSS in Argo Workflows chat url</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-04</td>
<td>XSS in Workflows workflow link</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-05</td>
<td>XSS in Pod Logs link</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-06</td>
<td>XSS in sensor logs link</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-07</td>
<td>XSS in Event Source logs url</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-08</td>
<td>XSS in Pod link url</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-09</td>
<td>XSS through server side rendering</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-10</td>
<td>Denial of service from malicious trigger parameters</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-11</td>
<td>Insufficient logging when executing a trigger</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-12</td>
<td>8 Uses of deprecated API can be used to cause DoS in user-facing endpoints</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-13</td>
<td>Insecure path traversal in Git Trigger Source can lead to arbitrary file read</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-14</td>
<td>Insufficient logging if OIDC Provider is created with nil TLS config</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-15</td>
<td>Path traversal leading to arbitrary YAML file read with malicious Helm chart</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-16</td>
<td>Path traversal leading to reading of out of bounds manifest files</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-17</td>
<td>DoS through large manifest files</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-18</td>
<td>nil-pointer in 3rd-party library 2</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-19</td>
<td>DoS through well-crafted Boolean Expression</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-20</td>
<td>Unmaintained 3rd-party dependencies</td>
<td>No</td>
</tr>
<tr>
<td>ADA-ARGO-SA-21</td>
<td>Unused function parameters</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-22</td>
<td>Functions always returning nil</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-23</td>
<td>Nil-pointer dereferences in sensor controller</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-24</td>
<td>Use of math/rand in production code</td>
<td>Yes</td>
</tr>
<tr>
<td>ADA-ARGO-SA-25</td>
<td>Improper use of InsecureSkipVerify</td>
<td>Partially</td>
</tr>
<tr>
<td>ADA-ARGO-SA-26</td>
<td>Insecure production use of crypto/ssh.InsecureIgnoreHostKey</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Argo CD allows admin users to add a button in the bottom right corner of the page. The intended use case is to provide the means to link to a support channel such as Slack, a help page with FAQ’s or to a live chat with a support team. The button is susceptible to an XSS attack due to lack of sanitization of the URL. An attacker who controls the Config Map can specify javascript to be run when a user clicks the “help” button.

**Attack scenario**
1: An attacker controlling the ConfigMap places the payload in a Config Map in `data -> help.chatUrl`. The payload will run whenever the victim clicks on the chat button in the bottom right corner of the page. For the sake of demonstration in this scenario, the attacker sets the URL value to "javascript: alert(1337)"

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
    name: argocd-cm
    namespace: argocd
    labels:
        app.kubernetes.io/name: argocd-cm
        app.kubernetes.io/part-of: argocd
data:

    url: https://argo-cd-demo.argoproj.io

statusbadge.enabled: "true"
users.anonymous.enabled: "true"
users.session.duration: "24h"
passwordPattern: "^.{8,32}$"

ga.trackingid: "UA-12345-1"
```
2: The ConfigMap is applied to the cluster with `kubectl apply -n argocd -f argocd-cm.yaml`.

3: A user clicks the “Chat Now” button:

![Diagram showing the application and chat interface]

4: XSS has been achieved:
ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript

<table>
<thead>
<tr>
<th>Severity</th>
<th>Critical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>Medium</td>
</tr>
<tr>
<td>Target</td>
<td>Argo CD</td>
</tr>
<tr>
<td>Github advisory</td>
<td>GHSA-h4w9-6x78-8vrj</td>
</tr>
<tr>
<td>CVE</td>
<td>CVE-2022-31035</td>
</tr>
</tbody>
</table>

Argo CD allows users to add external links for Deployments: [https://argo-cd.readthedocs.io/en/stable/user-guide/external-url/#add-external-url](https://argo-cd.readthedocs.io/en/stable/user-guide/external-url/#add-external-url). This could be a link to monitoring pages or documentation for the related Deployment. The URL values are not sanitized. Therefore, an attacker who can specify external links can launch an XSS attack against other users of the dashboard.

**Attack scenario**

1. An attacker adds a payload as a URL for a Deployment. This is done through the `link.argocd.argoproj.io/external-link` value through the dashboard. For the sake of this demonstration, the attackers sets the URL value to `javascript: alert(1337)`:

![Diagram of Argo CD dashboard with a deployment named myapp and the external link set to javascript: alert(1337)]
2: After syncing, ArgoCD adds a clickable icon that is visible to everyone with access to the dashboard:

3: The victim accesses the dashboard and clicks the external URL link. At this point XSS has been achieved:
ADA-ARGO-SA-03: XSS in Argo Workflows chat url

<table>
<thead>
<tr>
<th>Severity</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Workflows</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed</td>
</tr>
</tbody>
</table>

Argo Workflows allows users to add a “Get Help” button to the dashboard. The intended use is to provide users with the means to link to a help resource such as a Slack channel, a live support or a help page with FAQs. The button is added by way of a Config Map, as demonstrated here:


An attacker who controls the Config Map can specify a malicious url value and launch an attack through a reflected XSS vector against other users of the dashboard.

**Attack scenario**

1: An attacker manages to specify the `url` value of the “Get Help” button in a config map. The config map applied to the cluster. The value of the URL is a payload that is invoked at the time of attack. For the sake of this example, an attacker sets the url value to `javascript:alert("XSS")`.

2: The “Get Help” button appears at the bottom of the page. The victim accesses the dashboard and at some point needs help. Instead of being redirected to a support page or a chat room, XSS is achieved.
No workflows

To create a new workflow, use the button above.

You can find workflows in the examples or templates in Workflow Template Catalog.
Argo Workflows allows users to add a button linking to any URL of their choosing on the Workflow page. This could be a link to an organization's logging facilities. The link is specified via a Config Map like is demonstrated here: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L56. The URL is not sanitized.

An attacker who controls the Config Map can specify a malicious `url` value and launch an attack through a reflected XSS vector against other users of the dashboard. If an attacker controls the Config Map, they have admin privileges.

**Attack scenario**

1. An attacker controls the Config Map and specifies a malicious `url` value for the button on the Workflow page. For the sake of demonstration, the attacker sets the value to `javascript:alert("XSS")`.

2. Once the Config Map is applied to the cluster, a button is shown on the Workflow page. The victim then accesses the dashboard, navigates to a workflow and clicks the button. At this time, XSS is achieved.
Argo Workflows allows users to add a link to any URL of their choosing on the pod logs page. This could be a link to the logging facilities of the given pod. Both the text of the link as well as the URL can be specified through a Config Map: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L60.

An attacker who controls the Config Map can specify a malicious url value and launch an attack through a reflected XSS vector against other users of the dashboard. If an attacker controls the Config Map, they have admin privileges.

**Attack scenario**

1: An attacker controls the Config Map and specifies a malicious url value for the link on the pod logs page. For the purpose of demonstration, the attacker sets the value in the ConfigMap to `javascript:alert("XSS")`.

2: The link is added to the “Logs” view of a pod. The victim later accesses the dashboard, navigates to the “Logs” view of a pod and clicks the link. At this point, XSS has been achieved:
Click here
Argo Workflows allows users to add a link to the logs view of a sensor. This can be a link to the logs of a given sensor. The text and URL of the link can be specified via a Config Map as is demonstrated here:
An attacker who controls the Config Map can specify a malicious url value and launch an attack through a reflected XSS vector against other users of the dashboard. If an attacker controls the Config Map, they have admin privileges.

**Attack scenario**

1: An attacker controls the Config Map and specifies a malicious url value for the link on the sensor logs page. For the purpose of demonstration, the attacker sets the value in the ConfigMap to javascript:alert("XSS").

2: The victim accesses the dashboard, navigates to the sensor log page and clicks the link. XSS has been achieved.
Click here
Argeo Workflows allows users to add a link to the logs view of an Event Source. This can be a link to the logs of a given Event Source. The text and URL of the link can be specified via a ConfigMap as is demonstrated here: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L66.

An attacker who controls the Config Map can specify a malicious URL value and launch an attack through a reflected XSS vector against other users of the dashboard. If an attacker controls the Config Map, they have admin privileges.

Attack scenario
1: An attacker controls the Config Map and specifies a malicious URL value for the link on the Event Source logs page. For the purpose of demonstration, the attacker sets the value in the ConfigMap to javascript:alert("XSS").
2: The victim accesses the dashboard, navigates to the logs page of a given Event Source and clicks the link. At this point, XSS has been achieved:
Argo Workflows allows users to add a button in the “Summary” view of a pod. This could be a link to the logging facility for the given pod. The text and URL of the button can be specified via a ConfigMap as is demonstrated here: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L60
An attacker who controls the Config Map can specify a malicious url value and launch an attack through a reflected XSS vector against other users of the dashboard. If the attacker controls the Config Map, they have admin privileges.

Attack scenario
1: An attacker controls the Config Map and specifies a malicious url value for the link on the Pod Summary page. For the purpose of demonstration, the attacker sets the value in the ConfigMap to javascript:alert("XSS").
2: The victim accesses the dashboard, navigates to the summary page of a given Pod and clicks the link. At this point, XSS has been achieved:
Argo CD utilizes the `html/template` templating engine in the backend in the callback handler for the for OAuth2 login flow. This is done here:


An attacker who can create a cookie in the victim’s browser with a carefully crafted payload can launch an attack through a stored XSS vector against the same victim. This value of the cookie would need to be encrypted with the server’s key, which makes this attack highly difficult.

To exemplify the attack, Ada Logics wrote a fuzzer that would control the `ReturnURL` of `implicitFlowtmpl`. The attack has been successfully completed once `returnURL` in the response is controlled.

**Attack scenario**

The fuzzer takes the following steps to control `returnURL`:

1. A string (“the payload”) that consists of a string concatenated with a `:` character concatenated with another string is created.
2. A cookie with the name of `github.com/argoproj/argo-cd/v2/common.StateCookieName` is then created with the value of the encrypted payload.
3. An HTTP request is created with 2 form values:
   1. “state”: Contains the first string (everything before `:`) in the created cookie.
   2. “code”: Contains an empty string.
   This would satisfy the checks in `verifyAppState`.
4. Next, an app is created and the fuzzer passes a response writer and the request to `app.HandleCallback()`.
5. Finally, the fuzzer checks the body of the response which will return a string like this:

```javascript
<script>
var hash = window.location.hash.substr(1);
var result = hash.split('&').reduce(function (result, item) {
    var parts = item.split('=');
    result
```
result[parts[0]] = parts[1];
return result;
}, {});
var idToken = result['id_token'];
var state = result['state'];
var returnURL = "{{{OUR CONTROLLED STRING}}}";
if (state != "" && returnURL == ") {
    window.location.href = window.location.href.split("#")[0] + "?state" + result['state'] + window.location.hash;
} else if (returnURL != ") {
    document.cookie = "{{{CookieName}}}" + idToken + "; path=/";
    window.location.href = returnURL;
}
</script>

At this point, an attacker is able to perform an XSS attack, if they can force the execution flow into the else if branch by ensuring that state is ". An example of an XSS attack would be possible if var returnURL = "javascript: alert(1337)".

If window.location.href with a malicious payload is invoked in the victim's browser, XSS is possible.

The fuzzer:

```go
package oidc
import {
    "encoding/hex"
    "github.com/argoproj/argo-cd/v2/common"
    "fmt"
    "net/url"
    "github.com/argoproj/argo-cd/v2/util"
    "golang.org/x/oauth2"
    gooidc "github.com/coreos/go-oidc"
    "github.com/argoproj/argo-cd/v2/util/settings"
    "net/http/httptest"
    "net/http"
    "strings"
    "github.com/argoproj/argo-cd/v2/util/crypto"
    fuzz "github.com/AdaLogics/go-fuzz-headers"
}

type fakeProvider struct {
}

func (p *fakeProvider) Endpoint() (*oauth2.Endpoint, error) {
    return &oauth2.Endpoint{}, nil
}

func (p *fakeProvider) ParseConfig() (*OIDCConfiguration, error) {
```
return nil, nil
}

func (p *fakeProvider) Verify(_, _ string) (*gooidc.IDToken, error) {
    return nil, nil
}

func Fuzz(data []byte) int {
    f := fuzz.NewConsumer(data)

    // the payload string
    var b strings.Builder
    stateData, err := f.GetString()
    if err != nil {
        return 0
    }
    if stateData == "" {
        return 0
    }
    if !strings.Contains(stateData, "<") {
        return 0
    }
    if !strings.Contains(stateData, "}") {
        return 0
    }
    b.WriteString(stateData)
    b.WriteString(":")
    // secondString will be the "returnURL" in the template
    secondString, err := f.GetString()
    if err != nil {
        return 0
    }
    b.WriteString(secondString)
    cookieData := b.String()
    // Sanity checks
    parts1 := strings.SplitN(cookieData, ":", 2)
    if len(parts1)!=2 && parts1[1]=="" {
        return 0
    }
    if parts1[0] != stateData {
        return 0
    }
    // End of sanity checks
    signature, err := util.MakeSignature(32)
    if err != nil {
        panic(err)
    }
}
cdSettings := &settings.ArgoCDSettings{ServerSignature: signature}

req := httptest.NewRequest("GET", "/", nil)
req.Form = url.Values{
    "state":             []string{stateData},
    "code":              []string{""},
}

// Set the cookie with the encrypted payload
key, err := cdSettings.GetServerEncryptionKey()
if err != nil {
    panic(err)
}
encrypted, err := crypto.Encrypt([]byte(cookieData), key)
if err != nil {
    return 0
}
req.AddCookie(&http.Cookie{Name: common.StateCookieName, Value: hex.EncodeToString(encrypted))}

// Everything has been set up now. Before we set up the app, we will check the encrypted cookie value. This is another sanity check.
c, err := req.Cookie(common.StateCookieName)
if err != nil {
    return 0
}
val, err := hex.DecodeString(c.Value)
if err != nil {
    return 0
}
val, err = crypto.Decrypt(val, key)
if err != nil {
    return 0
}

cookieVal := string(val)
parts := strings.SplitN(cookieVal, ":", 2)
if len(parts)!=2 && parts[1]=="" {
    return 0
}
if parts[0] != stateData {
    return 0
}
// End of sanity check

app, err := NewClientApp(cdSettings, "", ""
if err != nil {
    panic(err)
}
app.provider = &fakeProvider{}

w := httptest.NewRecorder()
app.HandleCallback(w, req)

fmt.Println(w.Body.String())

return 1

ADA-ARGO-SA-10: Denial of service from malicious trigger parameters

<table>
<thead>
<tr>
<th>Severity</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed</td>
</tr>
</tbody>
</table>

This issue was found and reported by the FuzzConstructPayload fuzzer that tests the github.com/argoproj/argo-events/sensors/triggers.ConstructPayload() API. It was found that a malicious TriggerParameter could make Argo Events consume more memory than is available and crash the application with an out-of-memory panic.

OSS-fuzz details

Link to report: [https://oss-fuzz.com/testcase-detail/4621594267353088](https://oss-fuzz.com/testcase-detail/4621594267353088)

Reproducing OSS-fuzz crashes: [https://google.github.io/oss-fuzz/advanced-topics/reproducing/](https://google.github.io/oss-fuzz/advanced-topics/reproducing/)

Below, the fuzzer has been modified to print out the parameters, and the parameters that were passed to make ConstructPayload consume excessive memory are below.

Fuzzer

```go
package triggers

import {
    "fmt"
    "github.com/argoproj/argo-events/pkg/apis/sensor/v1alpha1"
    fuzz "github.com/AdaLogics/go-fuzz-headers"
}

func Fuzz(data []byte) int {
    f := fuzz.NewConsumer(data)
    events := make(map[string]*v1alpha1.Event)
    err := f.FuzzMap(&events)
    if err != nil {
        return 0
    }
    parameters := make([]v1alpha1.TriggerParameter, 0)
    err = f.FuzzMap(&parameters)
    if err != nil {
        return 0
    }
    fmt.Printf("events: 
%+v
", events)
    fmt.Printf("parameters: 
%+v\n", parameters)
```
... = ConstructPayload(events, parameters)
    return 1
}

**Parameters**

- **events:**
  ```json
  map[:{
    "context": {
      "id": "",
      "source": "",
      "specversion": "",
      "type": "",
      "datacontenttype": "",
      "subject": "",
      "time": "0001-01-01 00:00:00 +0000 UTC"},
    "data": ""
  }]
  ```

- **parameters:**
  ```json
  [{
    "Src": &TriggerParameterSource{
      "DependencyName": 
      "ContextKey": 
      "ContextTemplate": 
      "DataKey": 
      "DataTemplate": 
      "Value": *,
    },
    "Dest": 4444444444444444444444444444444444444444444444444444,
    "Operation": 4444444444444444444444444444444444444444444444444444
  }]
  ```
ADA-ARGO-SA-11: Insufficient logging when executing a trigger

<table>
<thead>
<tr>
<th>Severity</th>
<th>Informational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed</td>
</tr>
</tbody>
</table>

In the event an attacker is able to send a payload that crashes Argo Events during the execution of a trigger, the logs will not be helpful in tracing a possible attack.

The issue is present across multiple trigger types, but to demonstrate the issue, the Apache Openwhisk trigger is used as an example:

```go
// Execute executes the trigger
func (t *TriggerImpl) Execute(ctx context.Context, events map[string]*v1alpha1.Event, resource interface{}) (interface{}, error) {
    var payload []byte
    var err error

    openwhisktrigger, ok := resource.(*v1alpha1.OpenWhiskTrigger)
    if !ok {
        return nil, errors.New("failed to interpret the OpenWhisk trigger resource")
    }

    if openwhisktrigger.Payload != nil {
        payload, err = triggers.ConstructPayload(events, 
            openwhisktrigger.Payload)
        if err != nil {
            return nil, err
        }
    }

    ...
```

The parameters to `github.com/argoproj/argo-events/sensors/triggers.ConstructPayload()` are not logged on line 10 in the above code snippet. This will make mitigation in case of an attack difficult.

The issue was found in the following triggers:
1. Apache Openwhisk
2. AWS-Lambda
3. Azure Event Hubs
This issue can be chained with "ADA-ARGO-SA-10: Denial of service from malicious trigger parameters" which allows an attacker to both crash Argo Events and avoid the malicious input to be available in the logs.

We recommend maintaining consistent logging styles in the execution of all triggers. That should include:

At the top of each conditional branch, if execution can continue. For example, in the Apache Openwhisk trigger above, logging should be added right before the invocation of ConstructPayload() including the payload.

The Custom Trigger performs best of the triggers listed above, in that it logs the payload after it has been created:

```go
if trigger.Payload != nil {
    payload, err = triggers.ConstructPayload(events, trigger.Payload)
    if err != nil {
        return nil, err
    }
    ct.Logger.Debugw(
        "payload for the trigger execution",
        zap.Any("payload", string(payload)))
}
```

ADA-ARGO-SA-10 is a good example of why this approach is still insufficient. When it comes to logging in this user-exposed part of the code base, we recommend assuming that every invocation of an external API could crash Argo Events. If that was to happen, the logs should reveal which API was invoked to cause the crash, and which parameters were passed to it.
ADA-ARGO-SA-12: 8 Uses of deprecated API can be used to cause DoS in user-facing endpoints

<table>
<thead>
<tr>
<th>Severity</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>Medium</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
</tbody>
</table>

Several HandleRoute endpoints make use of the deprecated `ioutil.ReadAll()`. `ioutil.ReadAll()` reads all the data into memory. As such, an attacker who sends a large request to the Argo Events server will be able to crash it and cause denial of service.

Eventsources susceptible to an out-of-memory denial-of-service attack:
1. AWS SNS
2. Bitbucket
3. Bitbucket Server
4. Gitlab
5. Slack
6. Storagegrid
7. Webhook

Note that the Stripe Event Source uses `ioutil.ReadAll()` but limits the size of the request body:
[https://github.com/argoproj/argo-events/blob/master/eventsources/sources/stripe/start.go#L77](https://github.com/argoproj/argo-events/blob/master/eventsources/sources/stripe/start.go#L77)

Since `io/ioutil` has ceased maintenance we recommend discontinuing all use of this package.
ADA-ARGO-SA-13: Insecure path traversal in Git Trigger Source can lead to arbitrary file read

<table>
<thead>
<tr>
<th>Severity</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Github advisory</td>
<td>GHSA-qpgx-64h2-9c3c</td>
</tr>
<tr>
<td>Fix</td>
<td><a href="https://github.com/argoproj/argo-events/pull/1965">https://github.com/argoproj/argo-events/pull/1965</a></td>
</tr>
</tbody>
</table>

A path traversal issue was found in the (g *GitArtifactReader).Read() API. Read() calls into (g *GitArtifactReader).readFromRepository() that opens and reads the file that contains the trigger resource definition:

https://github.com/argoproj/argo-events/blob/master/sensors/artifacts/git.go

```go
func (g *GitArtifactReader) readFromRepository(r *git.Repository, dir string) {
    ...
    if err := w.Pull(pullOpts); err != nil && err != git.NoErrAlreadyUpToDate {
        return nil, fmt.Errorf("failed to pull latest updates. err: %v", err)
    }
    return ioutil.ReadFile(fmt.Sprintf("%s/%s", dir, g.artifact.FilePath))
}
```

No checks are made on this file at read time, which could lead an attacker to read files anywhere on the system. This could be achieved in at least three ways:

**Symbolic link in Git repository**
An attacker controls a Git repository that the victim uses in a Git Trigger Source. The attacker adds a file to the Git repository that is a symbolic link to a file containing sensitive information on the victims machine.

Argo then clones the repository onto the victims machine, and the symbolic link is followed during file read on the marked line above. An attacker could now read the file containing sensitive information.

**Race condition**
An attacker who has limited access to the file system may be able to read arbitrary files by leveraging a race condition. The attacker could replace the git-temp directory created by argo with a symbolic link to the directory containing the file to be read. This could be done anytime between the time it is created in (g *GitArtifactReader).Read() and the file is read in the return statement of (g *GitArtifactReader).readFromRepository(r *git.Repository, dir string).

**Malicious manifest**
An attacker controls a manifest for a Git Trigger Source that the victim creates.

The manifest has a `filePath` to a sensitive file anywhere on the victims machine, for example:

```yaml
triggers:
  - template:
      name: workflow-trigger
      k8s:
        operation: create
        source:
          git:
            url: "git@github.com:argoproj/argo-workflows.git"
            cloneDirectory: "/git/argoproj"
            sshKeyPath: "/secret/key"
            namespace: argo-events
        filePath: "/path/to/sensitive/file"
      branch: "master"
```

**Recommendations**

**Disallow symbolic links**
Check whether the file at GitArtifactReader.artificat.FilePath is a symbolic link before it is is opened and read in (g *GitArtifactReader).readFromRepository(). Fail if it is.

**Sanitize** GitArtifactReader.artifact.FilePath
This is includes checks for unsafe path patterns, such as:
- Check whether the string begins with "/".
- Disallow ".", "\", "~" in path.
- Other checks to ensure that only the files from the Git repository can be read
ADA-ARGO-SA-14: Insufficient logging if OIDC Provider is created with nil TLS config

<table>
<thead>
<tr>
<th>Severity</th>
<th>Informational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo CD</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed. Tracked in <a href="https://github.com/argoproj/argo-cd/issues/9957">https://github.com/argoproj/argo-cd/issues/9957</a></td>
</tr>
</tbody>
</table>

During our review of the ui parts of the Argo ecosystem, an issue was found in `oidc.NewClientApp()` ([https://github.com/argoproj/argo-cd/blob/master/util/oidc/oidc.go#L82](https://github.com/argoproj/argo-cd/blob/master/util/oidc/oidc.go#L82)), where the `ClientApp.client.Transport.TLSClientConfig` can be nil:

```go
...  tlsConfig := settings.OIDCTLSConfig()
  a.client = &http.Client{
      Transport: &http.Transport{
        TLSClientConfig: tlsConfig,
        Proxy:           http.ProxyFromEnvironment,
        Dial: (&net.Dialer{
            Timeout:  30 * time.Second,
            KeepAlive: 30 * time.Second,
        }).Dial,
        TLSHandshakeTimeout: 10 * time.Second,
        ExpectContinueTimeout: 1 * time.Second,
      }
  }
```

The `tlsConfig` variable is created from `(a *ArgoCDSettings).OIDCTLSConfig()` which returns the result of `(a *ArgoCDSettings).TLSConfig()`. `(a *ArgoCDSettings).TLSConfig()` can return `nil`, and if it does, we recommend that this is logged.
ADA-ARGO-SA-15: Path traversal leading to arbitrary YAML file read with malicious Helm chart

<table>
<thead>
<tr>
<th>Severity</th>
<th>Moderate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo CD</td>
</tr>
<tr>
<td>Github advisory</td>
<td>GHSA-q4w5-4gq2-98vm</td>
</tr>
<tr>
<td>CVE</td>
<td>CVE-2022-31036</td>
</tr>
</tbody>
</table>

A path traversal attack was found in func (s *Service).GetAppDetails() due to insecure reading of the files in a supplied Helm chart. The issue is similar to another issue that was reported earlier this year.

GetAppDetails() checks the application source and populates the details of the application supplied in the query. If a request is being made to provide the details of a Helm application source, GetAppDetails() will call into populateHelmDetails():

```go
func (s *Service) GetAppDetails(ctx context.Context, q *apiclient.RepoServerAppDetailsQuery) (*apiclient.RepoAppDetailsResponse, error) {
    res := &apiclient.RepoAppDetailsResponse{}

    cacheFn := s.createGetAppDetailsCacheHandler(res, q)
    operation := func(repoRoot, commitSHA, revision string, ctxSrc operationContextSrc) error {
        opContext, err := ctxSrc()
        if err != nil {
            return err
        }

        appSourceType, err := GetAppSourceType(ctx, q.Source, opContext.appPath, q.AppName, q.EnabledSourceTypes)
        if err != nil {
            return err
        }

        res.Type = string(appSourceType)

        switch appSourceType {
        case v1alpha1.ApplicationSourceTypeHelm:
            if err := populateHelmAppDetails(res, opContext.appPath, repoRoot, q); err != nil {
                return err
            }
        }
        return nil
    }

    return operation(repoRoot, commitSHA, revision, cacheFn)
}
```

2 [https://github.com/argoproj/argo-cd/blob/master/reposerver/repository/repository.go#L1352](https://github.com/argoproj/argo-cd/blob/master/reposerver/repository/repository.go#L1352)
In `populateHelmDetails()`, Argo CD will read the `values.yaml` file by calling into `loadFileIntoIfExists()`:

```go
func populateHelmAppDetails(res *apiclient.RepoAppDetailsResponse, appPath string, q *apiclient.RepoServerAppDetailsQuery) error {
    var selectedValueFiles []string
    if q.Source.Helm != nil {
        selectedValueFiles = q.Source.Helm.ValueFiles
    }
    availableValueFiles, err := findHelmValueFilesInPath(appPath)
    if err != nil {
        return err
    }
    res.Helm = &apiclient.HelmAppSpec{ValueFiles: availableValueFiles}
    var version string
    var passCredentials bool
    if q.Source.Helm != nil {
        if q.Source.Helm.Version != "" {
            version = q.Source.Helm.Version
        }
        passCredentials = q.Source.Helm.PassCredentials
    }
    if err != nil {
        return err
    }
    defer h.Dispose()
    err = h.Init()
    if err != nil {
        return err
    }
    if err := loadFileIntoIfExists(filepath.Join(appPath, "values.yaml"), &res.Helm.Values); err != nil {
        return err
    }
}
```

The second parameter to `loadFileIntoIfExists()` is a pointer to a string in the response that is being returned by `GetAppDetails()`. `loadFileIntoIfExists()` will read the `values.yaml` file of the Helm chart and store its contents in the response. However, if the `values.yaml` file is a symbolic link to any file of the system, that file will be read and stored in the response.
Because of that, Ada Logics was able to take the following steps to read an arbitrary file on the system:

1. A file containing sensitive information was created at `/etc/passwd`. The contents of that file was:

   ```yaml
   name: "Some Name"
   
   section:
     password: "1234"
   ```

2. A directory was created with a valid `Chart.yaml` file. In the same directory a symlink named `values.yaml` linking to `/etc/passwd` was created. For the sake of this example, the path of this mocked Helm chart directory was `/tmp/test-dir/app-path`.

3. A demo service was created by way of this test utility: https://github.com/argoproj/argo-cd/blob/master/reposerver/repository/repository_test.go#L98

4. A `RepoServerAppDetailsQuery` was created with the type "Helm" and with the `Source.Path` to be `.app-path`.

5. `GetAppDetails` was invoked, and the response was observed. The following response was produced:

   ```json
   {
     "type": "Helm",
     "helm": {
       "valueFiles": [
         "values.yaml"
       ],
       "parameters": [
         {
           "name": "name",
           "value": "Some Name"
         },
         {
           "name": "section.password",
           "value": "1234"
         }
       ]
     }
   }
   ```
ADA-ARGO-SA-16: Path traversal leading to reading of out of bounds manifest files

### Severity
Moderate

### Difficulty
High

### Target
Argo CD

### Github advisory
GHSA-6gcg-hp2x-q54h

### CVE
CVE-2022-24904

This issue was reported as low severity in the previous audit of 2021 with ID TOB-ARGO-018. The behavior went unchanged because at the time git resources were considered trusted. Evolving use cases mean this should now be addressed.

Improper handling of symbolic links in files read in `github.com/argoproj/argo-cd/v2/reposerver/repository.findManifests()` may lead to arbitrary file read through symbolic link.

The vulnerability can be triggered by invoking

With an application source type `v1alpha1.ApplicationSourceTypeDirectory` to allow execution to proceed into
`github.com/argoproj/argo-cd/v2/reposerver/repository.findManifests()`:

```go
func GenerateManifests(ctx context.Context, appPath, repoRoot, revision string, q *apiclient.ManifestRequest, isLocal bool, gitCredsStore git.CredsStore, opts ...GenerateManifestOpt) (*apiclient.ManifestResponse, error) {
    opt := newGenerateManifestOpt(opts...)
    var targetObjs []unstructured.Unstructured
    var dest *v1alpha1.ApplicationDestination

    resourceTracking := argo.NewResourceTracking()
    appSourceType, err := GetAppSourceType(ctx, q.ApplicationSource, appPath, q.AppName, q.EnabledSourceTypes)
    if err != nil {
        return nil, err
    }
    repoURL := ""
    if q.Repo != nil {
        if q.Repo.Repo != nil {
            repoURL = q.Repo.Repo
        }
    }
    env := newEnv(q, revision)

    switch appSourceType {
    case v1alpha1.ApplicationSourceTypeHelm:
        targetObjs, err = helmTemplate(appPath, repoRoot, env, q, isLocal)
    }...
```
case v1alpha1.ApplicationSourceTypeKustomize:
kustomizeBinary := ""
if q.KustomizeOptions != nil {
    kustomizeBinary = q.KustomizeOptions.BinaryPath
}
k := kustomize.NewKustomizeApp(appPath,
q.Repo.GetGitCreds(gitCredsStore), repoURL, kustomizeBinary)
targetObjs, _, err = k.Build(q.ApplicationSource.Kustomize,
q.KustomizeOptions, env)
case v1alpha1.ApplicationSourceTypePlugin:
    if q.ApplicationSource.Plugin != nil &&
    q.ApplicationSource.Plugin.Name != ""
        targetObjs, err = runConfigManagementPlugin(appPath,
repoRoot, env, q, q.Repo.GetGitCreds(gitCredsStore))
    else {
        targetObjs, err = runConfigManagementPluginSidecars(ctx,
appPath, repoRoot, env, q, q.Repo.GetGitCreds(gitCredsStore), opt.cmpTarDoneCh)
        if err != nil {
            err = fmt.Errorf("plugin sidecar failed. %s",
err.Error())
        }
    }
case v1alpha1.ApplicationSourceTypeDirectory:
    var directory *v1alpha1.ApplicationSourceDirectory
    if directory = q.ApplicationSource.Directory; directory
    == nil {
        directory = &v1alpha1.ApplicationSourceDirectory{}
    }
    targetObjs, err = findManifests(appPath, repoRoot, env, *directory,
q.EnabledSourceTypes)
    }
    if err != nil {
        return nil, err
    }

In github.com/argoproj/argo-cd/v2/reposerver/repository.findManifests(), the
files in the provided appPath are read. Files that do not have the .jsonnet file extension are
read by way of the 3rd-party dependency, github.com/TomOnTime/utfutil:

```go
func findManifests(appPath string, repoRoot string, env *v1alpha1.Env, directory v1alpha1.ApplicationSourceDirectory, enabledManifestGeneration map[string]bool) ([][]*unstructured.Unstructured, error) {
    var objs [][]*unstructured.Unstructured
    err := filepath.Walk(appPath, func(path string, f os.FileInfo, err error) error {
        if err != nil {
            return err
        }
        if f.IsDir() {
            if path != appPath && !directory.Recurse {
```
```go
return filepath.SkipDir
} else {
    return nil
}

if !manifestFile.MatchString(f.Name()) {
    return nil
}

relPath, err := filepath.Rel(appPath, path)
if err != nil {
    return err
}
if directory.Exclude != "" && glob.Match(directory.Exclude, relPath) {
    return nil
}
if directory.Include != "" && !glob.Match(directory.Include, relPath) {
    return nil
}
if strings.HasSuffix(f.Name(), ".jsonnet") {
    ...
} else {
    out, err := utfutil.ReadFile(path, utfutil.UTF8)
    if err != nil {
        return err
    }
}
```

The contents of the read file are then Unmarshalled and returned for further processing in `github.com/argoproj/argo-cd/v2/reposerver/repository.GenerateManifests()` where they are finally returned in the ManifestResponse:

```go
res := apiclient.ManifestResponse{
    Manifests:  manifests,
    SourceType: string(appSourceType),
}
if dest != nil {
    res.Namespace = dest.Namespace
    res.Server = dest.Server
}
return &res, nil
```

However, `utfutil.ReadFile()` follows symbolic links, and a symbolic link with a valid file name can be used to read arbitrary files on the system and return the contents in the response. As such, an attacker could make a request to
github.com/argoproj/argocd/v2/reposerver/repository.GenerateManifests()

with the appPath parameter specified to be a path containing a symbolic link to a file containing sensitive information anywhere on the system to read the file contents in the response. The attack would be difficult to launch due to a requirement of having a level of privileges to place a malicious manifest on the machine as well as the high amount of processing of read files. Furthermore, the attacker would have to know where sensitive files are stored, however, the attacker could also use this method to perform reconnaissance to locate sensitive files.
ADA-ARGO-SA-17: DoS through large manifest files

<table>
<thead>
<tr>
<th>Severity</th>
<th>Moderate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo CD</td>
</tr>
<tr>
<td>Github advisory</td>
<td>GHSA-jhqp-vf4w-rpwq</td>
</tr>
<tr>
<td>CVE</td>
<td>CVE-2022-31016</td>
</tr>
</tbody>
</table>

In [github.com/argoproj/argo-cd/v2/reposerver/repository.findManifests()](https://github.com/argoproj/argo-cd/v2/reposerver/repository/findManifests()), files are read via the 3rd-party API [github.com/TomOnTime/utfutil.ReadFile()](https://github.com/TomOnTime/utfutil/ReadFile()). This API is a thin wrapper around [io/ioutil.ReadFile()](https://golang.org/pkg/io/ioutil/ReadFile()) which will read the entire file into memory. Therefore, a large file being read by way of [github.com/TomOnTime/utfutil.ReadFile()](https://github.com/TomOnTime/utfutil/ReadFile()) can exhaust the memory and cause denial of service.

An attacker could exploit this by placing a large file in a repository and getting that repository onto the target machine. A request could then be made to [github.com/argoproj/argo-cd/v2/reposerver/repository.GenerateManifests()](https://github.com/argoproj/argo-cd/v2/reposerver/repository/GenerateManifests()) to generate manifests from the repository containing the large file. If the `appSourceType` is `v1alpha1.ApplicationSourceTypeDirectory`, execution would proceed to [github.com/argoproj/argo-cd/v2/reposerver/repository/findManifests()](https://github.com/argoproj/argo-cd/v2/reposerver/repository/findManifests()):

```go
def GenerateManifests(ctx context.Context, appPath, repoRoot, revision string, q *apiclient.ManifestRequest, isLocal bool, gitCredsStore git.CredsStore, opts ...GenerateManifestOpt) (*apiclient.ManifestResponse, error) {
    opt := newGenerateManifestOpt(opts...)
    var targetObjs []unstructured.Unstructured
    var dest *v1alpha1.ApplicationDestination

    resourceTracking := argo.NewResourceTracking()
    appSourceType, err := GetAppSourceType(ctx, q.ApplicationSource, appPath, q.AppName, q.EnabledSourceTypes)
    if err != nil {
        return nil, err
    }
    repoURL := ""
    if q.Repo != nil {
        repoURL = q.Repo.Repo
    }
    env := newEnv(q, revision)

    switch appSourceType {
    case v1alpha1.ApplicationSourceTypeHelm:
        targetObjs, err = helmTemplate(appPath, repoRoot, env, q, isLocal)
    case v1alpha1.ApplicationSourceTypeKustomize:
        kustomizeBinary := ""
    }
```
if q.KustomizeOptions != nil {
    kustomizeBinary = q.KustomizeOptions.BinaryPath
}
k := kustomize.NewKustomizeApp(appPath, q.Repo.GetGitCreds(gitCredsStore), repoURL, kustomizeBinary)
case v1alpha1.ApplicationSourceTypePlugin:
    if q.ApplicationSource.Plugin != nil &&
        q.ApplicationSource.Plugin.Name != "" {
        targetObjs, err = runConfigManagementPlugin(appPath, repoRoot, env, q, q.Repo.GetGitCreds(gitCredsStore))
    } else {
        targetObjs, err = runConfigManagementPluginSidecars(ctx, appPath, repoRoot, env, q, q.Repo.GetGitCreds(gitCredsStore), opt.cmpTarDoneCh)
        if err != nil {
            err = fmt.Errorf("plugin sidecar failed. %s", err.Error())
        }
    }
case v1alpha1.ApplicationSourceTypeDirectory:
    var directory *v1alpha1.ApplicationSourceDirectory
    if directory = q.ApplicationSource.Directory; directory == nil {
        directory = &v1alpha1.ApplicationSourceDirectory{}
    }
targetObjs, err = findManifests(appPath, repoRoot, env, *directory, q.EnabledSourceTypeApps)
}

Here, the large manifest file would be read into memory. At that moment Argo CD would crash, and the attacker would have successfully launched a denial of service attack.
ADA-ARGO-SA-18: nil-pointer in 3rd-party library 2

<table>
<thead>
<tr>
<th>Severity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed</td>
</tr>
</tbody>
</table>

github.com/argoproj/argo-events/eventbus/jetstream/sensor.NewJetstreamTriggerConn() uses github.com/Knetic/govaluate to evaluate dependency expressions. This is the line that creates an expression:

https://github.com/argoproj/argo-events/blob/master/eventbus/jetstream/sensor/trigger_conn.go

```go
collection.evaluatableExpression, err = govaluate.NewEvaluableExpression(strings.ReplaceAll(dependencyExpression, "-", \"\-\"))
```

A well-crafted dependencyExpression can cause Argo-Events to crash.

To reproduce:

```go
package sensor

import (
    "strings"
    "github.com/Knetic/govaluate"
)

func Reproduce() int {
    input := []byte{0x2, 0x2, 0x5c}_, _, _ = govaluate.NewEvaluableExpression(strings.ReplaceAll(string(input), ",", \\"\-\"))
```

Running the above code will result in the following stacktrace:

```
panic: runtime error: index out of range [3] with length 3
goroutine 17 [running, locked to thread]:
github.com/Knetic/govaluate.(*lexerStream).readCharacter(...)github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/lexerStream.go:28
github.com/Knetic/govaluate.readUntilFalse(0x10c0002c6f30, 0x0, 0x1, 0x1, 0x2030d68)
github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/parsing.go:319 +0x39e
github.com/Knetic/govaluate.readTokenUntilFalse(0x2, 0x0)
```
The issue is also present here:

- [https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go#L45](https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go#L45)
- [https://github.com/argoproj/argo-events/blob/master/sensors/listener.go#L131](https://github.com/argoproj/argo-events/blob/master/sensors/listener.go#L131)
ADA-ARGO-SA-19: DoS through well-crafted Boolean Expression

<table>
<thead>
<tr>
<th>Severity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td><a href="https://github.com/argoproj/argo-events/pull/1958">https://github.com/argoproj/argo-events/pull/1958</a></td>
</tr>
</tbody>
</table>

Argo Events can be manipulated to spend excessive time on handling a single BoolExpression. An attacker can exploit this by passing a well-crafted dependency expression to the Argo Events.

The following minimized testcase shows the issue. To reproduce, place this file in argo-events/common and run it. Argo Events will spend 25 seconds on running the reproducer.

The caught panic in catchPanics() is an issue that has also been reported in this report.

```go
package common

import (
    "fmt"
    "runtime"
    "runtime/debug"
    "strings"
)

func Reproduce() int {
    input := []byte{0x28, 0x28, 0x28, 0x29, 0x58, 0x26, 0x26, 0x58, 0x73, 0x26, 0x26, 0x5a, 0x58, 0x58, 0x35, 0x26, 0x26, 0x5a, 0x58, 0x58, 0x26, 0x26, 0x78, 0x73, 0x26, 0x26, 0x26, 0x5a, 0x58, 0x58, 0x73, 0x26, 0x26, 0x26, 0x58, 0x73, 0x26, 0x26, 0x26, 0x26, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0x54, 0xa}
    defer catchPanics()
    expr, err := NewBoolExpression(string(input))
    if err == nil {
        fmt.Println("Calling getExpression...")
        fmt.Printf("expr: %+v\n", expr)
        expr.GetExpression()
    }
    return 1
}

func catchPanics() {
    if r := recover(); r != nil {
        var err string
```
```
switch r.(type) {
    case string:
        err = r.(string)
    case runtime.Error:
        err = r.(runtime.Error).Error()
    case error:
        err = r.(error).Error()
}
if strings.Contains(string(debug.Stack()),
    "github.com/Knetic/govaluate.(*lexerStream).readCharacter(...)"
) {
    return
} else {
    panic(err)
}
```

Attack scenario
This issue can be exploited in several ways. One is to directly cause Argo Events to spend excessive CPU cycles on a single request causing a Denial of Service. Another is to use this vector in combination with other vulnerabilities. For example, in some circumstances an attacker can use a vulnerability like this chained with an exploitable race condition to slow down Argo Events in order to increase the chance of winning a race.

The Bool Minifier is not used in the Argo ecosystem, but its APIs are exported which makes it easy for other projects to add this vulnerable code to their projects.

Recommendation
Consider whether the Bool Minifier
(https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go) should cease maintenance, given that Argo itself doesn’t use it.

This issue has been fixed by removing the affected code.
ADA-ARGO-SA-20: Unmaintained 3rd-party dependencies

<table>
<thead>
<tr>
<th>Severity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>Low</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td>Not fixed</td>
</tr>
</tbody>
</table>

Two 3rd-party libraries used by Argo Events are unmaintained:

1. gogo/protobuf: No longer maintained as per https://github.com/gogo/protobuf/issues/691
The following functions accept parameters that are not used in the functions body:

<table>
<thead>
<tr>
<th>Function signature</th>
<th>File</th>
<th>Unused parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func (e *expr) evaluatePostfix(vars []string, set term, postfix []string) bool</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go">https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go</a></td>
<td>vars</td>
</tr>
<tr>
<td><code>func (r *reconciler) reconcile(ctx context.Context, eventSource *v1alpha1.EventSource) error</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/controllers/eventsource/controller.go">https://github.com/argoproj/argo-events/blob/master/controllers/eventsource/controller.go</a></td>
<td>ctx</td>
</tr>
<tr>
<td><code>func schema_argo_events_pkg_apis_common_Amount(ref common.ReferenceCallback) common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go</a></td>
<td>ref</td>
</tr>
<tr>
<td><code>func schema_argo_events_pkg_apis_common_Int64OrString(ref common.ReferenceCallback) common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go</a></td>
<td>ref</td>
</tr>
<tr>
<td>func</td>
<td>schema_argo_events_pkg_apis_common_S3Bucket(ref common.ReferenceCallback)</td>
<td>common.OpenAPIDefinition</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>func</td>
<td>schema_argo_events_pkg_apis_common_S3Filter(ref common.ReferenceCallback)</td>
<td>common.OpenAPIDefinition</td>
</tr>
<tr>
<td>func</td>
<td>schema_pkg_apis_eventsource_v1alpha1_AMQPConsumeConfig(ref common.ReferenceCallback)</td>
<td>common.OpenAPIDefinition</td>
</tr>
<tr>
<td>func</td>
<td>schema_pkg_apis_eventsource_v1alpha1_CatchupConfiguration(ref common.ReferenceCallback)</td>
<td>common.OpenAPIDefinition</td>
</tr>
<tr>
<td>func</td>
<td>schema_pkg_apis_eventsource_v1alpha1_EventsSourceFilter(ref common.ReferenceCallback)</td>
<td>common.OpenAPIDefinition</td>
</tr>
<tr>
<td>func</td>
<td>Description</td>
<td>URL</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----</td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_eventsource_v1alpha1_OwnedRepositories</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_eventsource_v1alpha1_Selector</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_eventsource_v1alpha1_StorageGridFilter</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_eventsource_v1alpha1_WatchPathConfig</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_ConditionsResetByTime</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_DataFilter</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_EventDependencyTransformer</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_FileArtifact</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_GitRemoteConfig</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td><code>schema_pkg_apis_sensor_v1alpha1_LogTrigger</code>&lt;br&gt;<code>common.OpenAPIDefinition</code></td>
<td><a href="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go">https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go</a></td>
</tr>
<tr>
<td>func</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Reference</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td><code>schema_pkg_apis_sensor_v1alpha1_PayloadField</code></td>
<td>(ref common.ReferenceCallback)</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
<tr>
<td><code>func schema_pkg_apis_sensor_v1alpha1_RateLimit(ref common.ReferenceCallback)</code></td>
<td>common.OpenAPIDefinition</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
<tr>
<td><code>func schema_pkg_apis_sensor_v1alpha1_StatusPolicy(ref common.ReferenceCallback)</code></td>
<td>common.OpenAPIDefinition</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
<tr>
<td><code>func schema_pkg_apis_sensor_v1alpha1_TimeFilter(ref common.ReferenceCallback)</code></td>
<td>common.OpenAPIDefinition</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
<tr>
<td><code>func schema_pkg_apis_sensor_v1alpha1_TriggerParameterSource(ref common.ReferenceCallback)</code></td>
<td>common.OpenAPIDefinition</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
<tr>
<td><code>func schema_pkg_apis_sensor_v1alpha1_URLArtifact(ref common.ReferenceCallback)</code></td>
<td>common.OpenAPIDefinition</td>
<td><img src="https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go" alt="ROBLOX" /></td>
</tr>
</tbody>
</table>
ADA-ARGO-SA-22: Functions always returning nil

<table>
<thead>
<tr>
<th>Severity</th>
<th>Informational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
</tbody>
</table>

Recommendation: Do not return a value if it is never used.

<table>
<thead>
<tr>
<th>Function signature</th>
<th>File</th>
</tr>
</thead>
</table>
### ADA-ARGO-SA-23: Nil-pointer dereferences in sensor controller

<table>
<thead>
<tr>
<th>Severity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>CVE</td>
<td>CVE-2022-31034</td>
</tr>
</tbody>
</table>

The following nil dereferences have been found in the sensor controller:

```go
github.com/argoproj/argo-events/controllers/sensor.Validate()
https://github.com/argoproj/argo-events/blob/master/controllers/sensor/validate.go#L36
```

#### Parameter 1

```go
func ValidateSensor(s *v1alpha1.Sensor, b *eventbusv1alpha1.EventBus) error {
  if err := validateDependencies(s.Spec.Dependencies, b); err != nil {
    s.Status.MarkDependenciesNotProvided("InvalidDependencies",
    err.Error())
    return err
  }
  s.Status.MarkDependenciesProvided()
  err := validateTriggers(s.Spec.Triggers)
  if err != nil {
    s.Status.MarkTriggersNotProvided("InvalidTriggers", err.Error())
    return err
  }
  s.Status.MarkTriggersProvided()
  return nil
}
```

#### Parameter 2

```go
func ValidateSensor(s *v1alpha1.Sensor, b *eventbusv1alpha1.EventBus)
```

If this parameter is `nil`, `ValidateSensor` will pass it to `validateDependencies()` where a nil-pointer dereference will happen on the marked line below:
func validateDependencies(eventDependencies []v1alpha1.EventDependency, b *eventbusv1alpha1.EventBus) error {
    if len(eventDependencies) < 1 {
        return errors.New("no event dependencies found")
    }

    comboKeys := make(map[string]bool)
    for _, dep := range eventDependencies {
        if dep.Name == "" {
            return errors.New("event dependency must define a name")
        }
        if dep.EventSourceName == "" {
            return errors.New("event dependency must define the EventSourceName")
        }
        if dep.EventName == "" {
            return errors.New("event dependency must define the EventName")
        }
        comboKey := fmt.Sprintf("%s-$$$-%s", dep.EventSourceName, dep.EventName)
        if _, existing := comboKeys[comboKey]; existing {
            return errors.Errorf("Event '%s' from EventSource '%s' is referenced for more than one dependency in this Sensor object", dep.EventName, dep.EventSourceName)
        }
        comboKeys[comboKey] = true
    }

    if err := validateEventFilter(dep.Filters); err != nil {
        return err
    }

    if err := validateLogicalOperator(dep.FiltersLogicalOperator); err != nil {
        return err
    }

    return nil
}

ValidateSensor is only used in the sensor controller itself and nil-parameters will not be passed to it by Argo Events itself, but since it is exported, it may be used in contexts where nil-parameters can be passed.

Recommendation: Check for nil in beginning of function body.
ADA-ARGO-SA-24: Use of `math/rand` in production code

<table>
<thead>
<tr>
<th>Severity</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>Moderate</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Github advisory</td>
<td>GHSA-2m7h-86qq-fp4v</td>
</tr>
<tr>
<td>Fix</td>
<td><a href="https://github.com/argoproj/argo-events/pull/1959">https://github.com/argoproj/argo-events/pull/1959</a></td>
</tr>
</tbody>
</table>

A number of the Event Sources were found to use the weak random number generator. As per the official Go documentation, this package is not safe "unsuitable for security-sensitive work".³

The package was found in the following files:

- https://github.com/argoproj/argo-events/blob/master/eventsources/sources/bitbucket/start.go
- https://github.com/argoproj/argo-events/blob/master/eventsources/eventing.go
- https://github.com/argoproj/argo-events/blob/master/eventbus/stan/sensor/sensor_stan.go

³ https://pkg.go.dev/math/rand
### ADA-ARGO-SA-25: Improper use of InsecureSkipVerify

<table>
<thead>
<tr>
<th><strong>Severity</strong></th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Difficulty</strong></td>
<td>High</td>
</tr>
<tr>
<td><strong>Target</strong></td>
<td>Argo Events &amp; Argo CD</td>
</tr>
<tr>
<td><strong>Fix</strong></td>
<td>Partially (Fixed in ArgoCD but not Argo Events)</td>
</tr>
<tr>
<td><strong>Github advisory</strong></td>
<td>GHSA-7943-82jg-wmw5</td>
</tr>
<tr>
<td><strong>CVE</strong></td>
<td>CVE-2022-31105</td>
</tr>
</tbody>
</table>

ArgoCD and Argo Events use the `InsecureSkipVerify` option of `crypto/tls.Config` in production. Argo Events does not follow the recommendation offered in the official documentation on the `InsecureSkipVerify` option:

"In this mode, TLS is susceptible to machine-in-the-middle attacks unless custom verification is used. This should be used only for testing or in combination with `VerifyConnection` or `VerifyPeerCertificate`.

[https://pkg.go.dev/crypto/tls](https://pkg.go.dev/crypto/tls)

The following production areas of Argo Events are affected in Argo Events:

<table>
<thead>
<tr>
<th>API</th>
<th>File</th>
</tr>
</thead>
</table>

These do not impose an immediate security risk to Argo Events. The Jetstream connection is only used internally to connect to a JetStream EventBus configured with self-signed certificates. The remaining three allow users to skip verification if they choose to.

It is recommended that the Argo Events documentation explicitly warns users that skipping verification has security implications.
ADA-ARGO-SA-26: Insecure production use of crypto/ssh.InsecureIgnoreHostKey

<table>
<thead>
<tr>
<th>Severity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficulty</td>
<td>High</td>
</tr>
<tr>
<td>Target</td>
<td>Argo Events</td>
</tr>
<tr>
<td>Fix</td>
<td><a href="https://github.com/argoproj/argo-events/pull/1982">https://github.com/argoproj/argo-events/pull/1982</a></td>
</tr>
</tbody>
</table>

The git artifact uses the crypto/ssh.InsecureIgnoreHostKey API which is unsuited for production use cases.

See Golang docs for more: https://pkg.go.dev/golang.org/x/crypto/ssh#InsecureIgnoreHostKey
Follow-up on fixes for existing security issues

In this section we go through the results of this goal which was to review the fixes for the issues identified in the previous audit. The Argo team had fixed the findings, but the fixes had not been verified by a third party yet. The goal of this task was to validate all of the fixes and assess if the fix is accurate, incomplete and whether derivatives of the original issue are present.

In this section we list the results from assessing the previous fixes.

The issues in this section are enumerated by their ID from the previous audit. TOB-ARGO-TM is the prefix used for issue found as part of the threat modelling. TOB-ARGO is the prefix used for issues found by the auditing. The threat model and security audit can be found here: [https://github.com/argoproj/argoproj/tree/master/docs](https://github.com/argoproj/argoproj/tree/master/docs)

<table>
<thead>
<tr>
<th>Issue ID</th>
<th>Fixed</th>
<th>Relevant links</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOB-ARGO-TM1</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/6742">https://github.com/argoproj/argo-cd/pull/6742</a></td>
</tr>
<tr>
<td>TOB-ARGO-TM2</td>
<td>Fixed</td>
<td>Partially fixed</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/issues/9968">https://github.com/argoproj/argo-cd/issues/9968</a></td>
</tr>
<tr>
<td>TOB-ARGO-TM3</td>
<td>Fixed</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/issues/1364">https://github.com/argoproj/argo-cd/issues/1364</a></td>
</tr>
<tr>
<td>TOB-ARGO-TM4</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/5477">https://github.com/argoproj/argo-cd/pull/5477</a></td>
</tr>
<tr>
<td>Comments</td>
<td>Documentation should be updated here: <a href="https://github.com/argoproj/argo-cd/blob/a809469d9af10c626449bfcb8b9a09a9d2dc9065/util/session/sessionmanager.go#L153">https://github.com/argoproj/argo-cd/blob/a809469d9af10c626449bfcb8b9a09a9d2dc9065/util/session/sessionmanager.go#L153</a> - To include information about the admin user.</td>
<td></td>
</tr>
<tr>
<td>TOB-ARGO-TM5</td>
<td>Fixed</td>
<td>Will not fix</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td>n/a</td>
</tr>
<tr>
<td>TOB-ARGO-TM6</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/7071">https://github.com/argoproj/argo-cd/pull/7071</a></td>
</tr>
</tbody>
</table>
**TOB-ARGO-TM7**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/7094">https://github.com/argoproj/argo-cd/pull/7094</a></td>
</tr>
</tbody>
</table>

**Comments**
Guidance or links to guidance on configuring the proxy server might be helpful for some users.

**TOB-ARGO-TM8**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
</table>

**Comments**
Consider setting the `UserSessionDuration` explicitly in case `timeutil.ParseDuration()` returns an error: [https://github.com/argoproj/argo-cd/blob/fdceed6051f2659adff206498a9417695e23e74c/util/settings/settings.go#L1274](https://github.com/argoproj/argo-cd/blob/fdceed6051f2659adff206498a9417695e23e74c/util/settings/settings.go#L1274)

```go
if userSessionDurationStr, ok := argoCDCM.Data[userSessionDurationKey]; ok {
    if val, err := timeutil.ParseDuration(userSessionDurationStr); err != nil {
        log.Warnf("Failed to parse '%s' key: %v", userSessionDurationKey, err)
        settings.UserSessionDuration = time.Hour * 24
    } else {
        settings.UserSessionDuration = *val
    }
} else {
    settings.UserSessionDuration = time.Hour * 24
}
```

Otherwise, if execution proceeds to both if statements, `settings.UserSessionDuration` will be 0s.
This is being tracked in [https://github.com/argoproj/argo-cd/issues/9962](https://github.com/argoproj/argo-cd/issues/9962)

**TOB-ARGO-TM9**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Will not fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**TOB-ARGO-TM10**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Will not fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td>n/a</td>
</tr>
</tbody>
</table>
### TOB-ARGO-TM11
- **Fixed**: Yes

### TOB-ARGO-TM12
- **Fixed**: Yes
- **Relevant links**: [https://github.com/argoproj/argo-workflows/pull/5211](https://github.com/argoproj/argo-workflows/pull/5211)

### TOB-ARGO-TM13
- Does not exist.

### TOB-ARGO-TM14
- **Fixed**: No
- **Relevant links**: n/a

### TOB-ARGO-TM15
- **Fixed**: Yes
- **Relevant links**: [https://github.com/argoproj/argo-workflows/pull/5212](https://github.com/argoproj/argo-workflows/pull/5212)

### TOB-ARGO-TM16
- **Fixed**: Yes
- **Relevant links**: [https://github.com/argoproj/argo-workflows/pull/6515](https://github.com/argoproj/argo-workflows/pull/6515)

### TOB-ARGO-TM17
- **Fixed**: Yes
- **Relevant links**: [https://github.com/argoproj/argo-workflows/commit/3411407f1d39af720b1189261d1b233079940b0d](https://github.com/argoproj/argo-workflows/commit/3411407f1d39af720b1189261d1b233079940b0d)

### Comments
 Guidance or links to guidance on configuring the proxy server might be helpful for some users.

### TOB-ARGO-TM18
- **Fixed**: Wontfix
- **Relevant links**: [https://github.com/argoproj/argo-workflows/issues/5215](https://github.com/argoproj/argo-workflows/issues/5215)

### TOB-ARGO-TM19
- **Fixed**: Fixed
- **Relevant links**: [https://github.com/argoproj/argo-events/pull/1585](https://github.com/argoproj/argo-events/pull/1585)
### Comments
We recommend that Argo makes their work to improve logging an ongoing effort.

#### TOB-ARGO-TM20
- **Fixed**: Fixed
- **Relevant links**: [https://github.com/argoproj/argo-events/issues/1088](https://github.com/argoproj/argo-events/issues/1088)

#### TOB-ARGO-TM21
- **Fixed**: Yes
- **Relevant links**: [https://github.com/argoproj/argo-events/pull/1771](https://github.com/argoproj/argo-events/pull/1771)

### Comments
This fix is a bit comprehensive, and we recommend adding tests to showcase that events are encrypted. Ada Logics found no unit tests that verify explicitly that events are encrypted.

#### TOB-ARGO-TM22
- **Fixed**: Fixed
- **Relevant links**: [https://github.com/argoproj/argo-events/pull/1318](https://github.com/argoproj/argo-events/pull/1318)

#### TOB-ARGO-001
- **Fixed**: Fixed
- **Relevant links**: [https://github.com/argoproj/argo-cd/pull/8157](https://github.com/argoproj/argo-cd/pull/8157)

### Comments
During the review of the fixes, Redis 7.0.0 was released which fixes security issues, and it is recommended to upgrade immediately.

#### TOB-ARGO-002
- **Fixed**: Fixed

### Comments
We recommend avoiding storing credentials in environment variables.

#### TOB-ARGO-003
- **Fixed**: Fixed
- **Relevant links**: [https://github.com/argoproj/argo-cd/pull/6742](https://github.com/argoproj/argo-cd/pull/6742)

#### TOB-ARGO-004
- **Fixed**: Fixed
<table>
<thead>
<tr>
<th>Relevant links</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOB-ARGO-005</strong></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/5786">https://github.com/argoproj/argo-cd/pull/5786</a></td>
</tr>
<tr>
<td><strong>TOB-ARGO-006</strong></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-rollouts/pull/1168">https://github.com/argoproj/argo-rollouts/pull/1168</a></td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td></td>
</tr>
<tr>
<td>Examples of unfixed cases:</td>
<td></td>
</tr>
<tr>
<td>This is being tracked in <a href="https://github.com/argoproj/argo-cd/issues/9963">https://github.com/argoproj/argo-cd/issues/9963</a></td>
<td></td>
</tr>
<tr>
<td><strong>TOB-ARGO-007</strong></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>Wontfix</td>
</tr>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-workflows/issues/5860">https://github.com/argoproj/argo-workflows/issues/5860</a></td>
</tr>
<tr>
<td><strong>TOB-ARGO-008</strong></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
 [https://github.com/argoproj/argo-cd/pull/5789](https://github.com/argoproj/argo-cd/pull/5789) |
| **Comments** | |
| The following areas were marked as issues in the previous report and have not been fixed. We don’t consider these to be necessary to fix. | |
 [https://github.com/argoproj/argo-workflows/blob/master/examples/validator.go#L44](https://github.com/argoproj/argo-workflows/blob/master/examples/validator.go#L44) |
<p>| <strong>TOB-ARGO-009</strong> | Fixed |
|  | Yes |
| Relevant links | <a href="https://github.com/argoproj/argo-workflows/pull/5211">https://github.com/argoproj/argo-workflows/pull/5211</a> |
| <strong>TOB-ARGO-010</strong> | Fixed |
|  | Check pending |
| Relevant links |  |
| <strong>TOB-ARGO-011</strong> | Fixed |
|  | No |</p>
<table>
<thead>
<tr>
<th>Relevant links</th>
<th><a href="https://github.com/argoproj/argo-workflows/pull/6985">https://github.com/argoproj/argo-workflows/pull/6985</a></th>
</tr>
</thead>
</table>

**Comments**
The initial fix in [https://github.com/argoproj/argo-workflows/pull/6985](https://github.com/argoproj/argo-workflows/pull/6985) has been replaced in [https://github.com/argoproj/argo-workflows/pull/8292/files](https://github.com/argoproj/argo-workflows/pull/8292/files) which does not prove that TOB-ARGO-011 remains fixed. This should be done with a regression test, and because no regression test exists to prove that the HTTP artifact fetcher will not fail on self-signed certificates, we mark this as not fixed.

**TOB-ARGO-012**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-workflows/pull/6985">https://github.com/argoproj/argo-workflows/pull/6985</a></td>
</tr>
</tbody>
</table>

**Comments**
The initial fix in [https://github.com/argoproj/argo-workflows/pull/6985](https://github.com/argoproj/argo-workflows/pull/6985) has been replaced in [https://github.com/argoproj/argo-workflows/pull/8292/files](https://github.com/argoproj/argo-workflows/pull/8292/files) which does not prove that TOB-ARGO-012 remains fixed. This should be done with a regression test, and because no regression test exists to prove that the HTTP artifact fetcher will not use TLS by default, we mark this as not fixed.

**TOB-ARGO-013**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
</table>

**Comments**
The issue was tracked via the link above which has not been closed. Therefore this issue is considered to not be fixed.

**TOB-ARGO-014**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-workflows/issues/6520">https://github.com/argoproj/argo-workflows/issues/6520</a></td>
</tr>
</tbody>
</table>

**Comments**
According to the Relevant Link, no tracker has been set up for TOB-ARGO-014, and it has not been marked “Fixed”. Therefore this issue is considered to not be fixed.

**TOB-ARGO-015**

<table>
<thead>
<tr>
<th>Fixed</th>
<th>Not by the end of this Audit. Argo may have fixed it later here: [<a href="https://github.com/ar">https://github.com/ar</a> gobroj/argo-rollouts/pull/1155](<a href="https://github.com/ar">https://github.com/ar</a> gobroj/argo-rollouts/pull/1155) but due to timing this fix was not confirmed by Ada Logics.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td>n/a</td>
</tr>
</tbody>
</table>

**TOB-ARGO-016**
<table>
<thead>
<tr>
<th>TOB-ARGO-017</th>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/security/advisories/GHSA-2m7h-86qq-fp4v">https://github.com/argoproj/argo-cd/security/advisories/GHSA-2m7h-86qq-fp4v</a></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Argo CD still used the insecure <code>math/rand</code> here: <a href="https://github.com/argoproj/argo-cd/blob/master/util/rand/rand.go">https://github.com/argoproj/argo-cd/blob/master/util/rand/rand.go</a>. This was found to not be fixed by Ada Logics and was fixed during this audit.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOB-ARGO-018</th>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/security/advisories/GHSA-6gcg-hp2x-q54h">https://github.com/argoproj/argo-cd/security/advisories/GHSA-6gcg-hp2x-q54h</a></td>
<td></td>
</tr>
</tbody>
</table>
| Comments | • No Github issue had been created to track this finding.  
• No tests had been added.  
• No check had been added for a symlink for `path` in `filepath.Walk()` in `findManifests` ([https://github.com/argoproj/argo-cd/blob/master/reposerver/repository/repository.go#L1078](https://github.com/argoproj/argo-cd/blob/master/reposerver/repository/repository.go#L1078)). This was fixed during this audit. |

<table>
<thead>
<tr>
<th>TOB-ARGO-019</th>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOB-ARGO-020</th>
<th>Fixed</th>
<th>Will not fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant links</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOB-ARGO-021</th>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TOB-ARGO-022</th>
<th>Fixed</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Status</td>
<td>Relevant links</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>TOB-ARGO-023</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-024</td>
<td>Fixed</td>
<td>No</td>
</tr>
<tr>
<td>TOB-ARGO-025</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-026</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-027</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-028</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-029</td>
<td>Fixed</td>
<td>No</td>
</tr>
<tr>
<td>TOB-ARGO-030</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>TOB-ARGO-031</td>
<td>Fixed</td>
<td>No</td>
</tr>
<tr>
<td>TOB-ARGO-033</td>
<td>Fixed</td>
<td>Yes</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>Relevant links</td>
<td><a href="https://github.com/argoproj/argo-cd/pull/5477">https://github.com/argoproj/argo-cd/pull/5477</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOB-ARGO-034</th>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
</table>
| Relevant links | - [https://github.com/alexec/argo-workflows/commit/4b5adc7f8f2f1e10ff0618b223ab9eb126846031](https://github.com/alexec/argo-workflows/commit/4b5adc7f8f2f1e10ff0618b223ab9eb126846031)
- [https://github.com/argoproj/argo-events/pull/1153](https://github.com/argoproj/argo-events/pull/1153) |

**Comments**

“Releases” links in documentation for Argo CD and Argo Rollouts are links to releases and not documentation:

- [https://argoprod.github.io/argo-rollouts/](https://argoprod.github.io/argo-rollouts/)

<table>
<thead>
<tr>
<th>TOB-ARGO-035</th>
<th>Fixed</th>
<th>Yes</th>
</tr>
</thead>
</table>

**Comments**

Fixed by adding the requirement to resolve all critical- and high-severity vulnerabilities (as reported by `snyk test`) before each release. Golang and NPM dependencies are tested when each PR is merged, and maintainers monitor the build for failures.