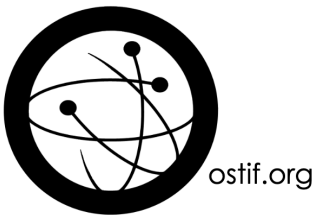
ADALOGICS

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

ADALOGICS

Table of Contents

ADALOGICS

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

ADA LOGICS

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
26 security issues found
9 CVEs • 1 critical severity • 4 high severity • 3 moderate severity • 1 low severity
7 fuzzers added
Review of fixes of previous security audit

ADALOGICS

Project Information

Document History

Version	Date	Details
1.0	03/05/2022	First version shared with the Argo team
1.1	13/07/2022	Final version of report delivered.

Contacts

Ada Logics

Name	**Title**	**Role**	**Email address**
Adam Korczynski	Security Engineer	Lead	Adam@adalogics.com
David Korczynski	Security Researcher	Support	David@adalogics.com

Open Source Technology Improvement Fund

Name	**Email**
Amir Montazery	Amir@ostif.org
Derek Zimmer	Derek@ostif.org

Argo Team

Name	**Email address**
Alex Collins	Alex_Collins@intuit.com
Derek Wang	Derek_Wang@intuit.com
Hari Rongali	Hari_Rongali@intuit.com
Henrik Blixt	Henrik_Blixt@intuit.com
Jann Fischer	Jann@mistrust.net
Jesse Suen	Jesse@akuity.io

ADALOGICS

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

Fuzzer Name	**Link**
`FuzzStripeEventsource`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/events_eventsource_stripe_fuzzer.go
`FuzzConstructPayload`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/events_triggers_fuzzer.go
`FuzzgetDependencyExpression`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/sensors_fuzzer.go
`FuzzExpr`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/events_expr_fuzzer.go
`FuzzValidateSensor`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/events_controllers_sensor_fuzzer.go
`FuzzArgoWorkflowTriggerExecute`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/events_argo_workflow_fuzzer.go
`FuzzGetExpression`	https://github.com/cncf/cncf-fuzzing/blob/main/projects/argo/common_fuzzer.go

[1] https://github.com/argoproj/argoproj/blob/dd7cae43d81c5a11f21ff4ea0a4afadcae4799c7/docs/audit_fuzzer_adalogics_2022.pdf

ADALOGICS

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

Issue #	CVE	Github Advisory	Severity (CVSS)
ADA-ARGO-SA-02	CVE-2022-31035	GHSA-h4w9-6x78-8vrj	Critical (9.0)
ADA-ARGO-SA-09	CVE-2022-31102	GHSA-pmjg-52h9-72qv	Low (2.6)
ADA-ARGO-SA-12	CVE-2022-31054	GHSA-5q86-62xr-3r57	High (7.5)
ADA-ARGO-SA-13	CVE-2022-25856	GHSA-qpgx-64h2-gc3c	High (7.5)
ADA-ARGO-SA-15	CVE-2022-31036	GHSA-q4w5-4gq2-98vm	Moderate (4.3)
ADA-ARGO-SA-16	CVE-2022-24904	GHSA-6gcg-hp2x-q54h	Moderate (4.3)
ADA-ARGO-SA-17	CVE-2022-31016	GHSA-jhqp-vf4w-rpwq	Moderate (6.5)
ADA-ARGO-SA-24	CVE-2022-31034	GHSA-2m7h-86qq-fp4v	High (8.3)
ADA-ARGO-SA-25	CVE-2022-31105	GHSA-7943-82jg-wmw5	High (8.3)

All issues

Issue #	CVE	Fixed
ADA-ARGO-SA-01	ADA-ARGO-SA-01: XSS in Argo CD Chat URL	No
ADA-ARGO-SA-02	ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript	Yes
ADA-ARGO-SA-03	XSS in Argo Workflows chat url	No
ADA-ARGO-SA-04	XSS in Workflows workflow link	No
ADA-ARGO-SA-05	XSS in Pod Logs link	No
ADA-ARGO-SA-06	XSS in sensor logs link	No
ADA-ARGO-SA-07	XSS in Event Source logs url	No
ADA-ARGO-SA-08	XSS in Pod link url	No

ADA-ARGO-SA-09	XSS through server side rendering	Yes
ADA-ARGO-SA-10	Denial of service from malicious trigger parameters	No
ADA-ARGO-SA-11	Insufficient logging when executing a trigger	No
ADA-ARGO-SA-12	8 Uses of deprecated API can be used to cause DoS in user-facing endpoints	Yes
ADA-ARGO-SA-13	Insecure path traversal in Git Trigger Source can lead to arbitrary file read	Yes
ADA-ARGO-SA-14	Insufficient logging if OIDC Provider is created with nil TLS config	No
ADA-ARGO-SA-15	Path traversal leading to arbitrary YAML file read with malicious Helm chart	Yes
ADA-ARGO-SA-16	Path traversal leading to reading of out of bounds manifest files	Yes
ADA-ARGO-SA-17	DoS through large manifest files	Yes
ADA-ARGO-SA-18	nil-pointer in 3rd-party library 2	No
ADA-ARGO-SA-19	DoS through well-crafted Boolean Expression	Yes
ADA-ARGO-SA-20	Unmaintained 3rd-party dependencies	No
ADA-ARGO-SA-21	Unused function parameters	Yes
ADA-ARGO-SA-22	Functions always returning nil	Yes
ADA-ARGO-SA-23	Nil-pointer dereferences in sensor controller	Yes
ADA-ARGO-SA-24	Use of math/rand in production code	Yes
ADA-ARGO-SA-25	Improper use of InsecureSkipVerify	Partially
ADA-ARGO-SA-26	Insecure production use of crypto/ssh.InsecureIgnoreHostKey	Yes

ADALOGICS

ADA-ARGO-SA-01: XSS in Argo CD Chat URL

Severity	Medium
Difficulty	High
Target	Argo CD
Fix	Not fixed. Tracked in https://github.com/argoproj/argo-cd/issues/9956

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

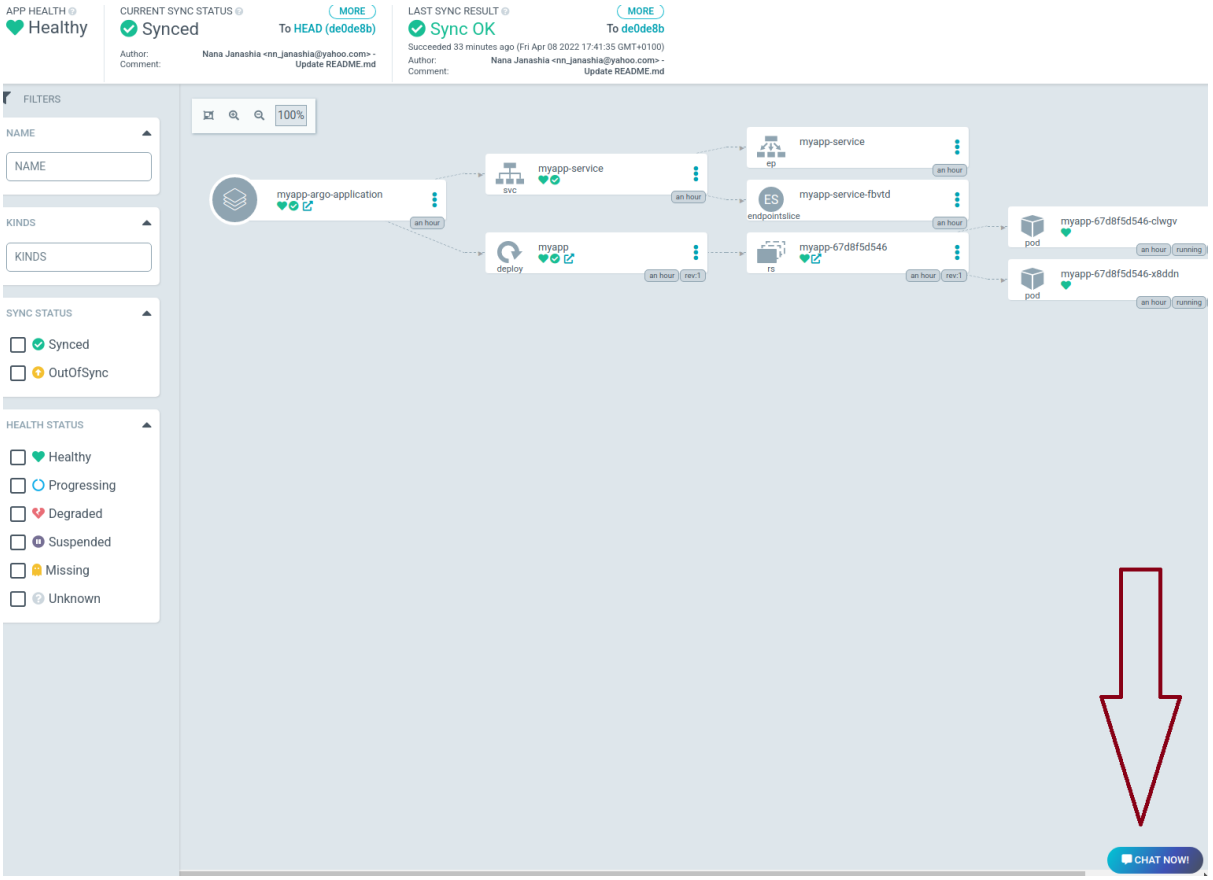
1: An attacker controlling the `ConfigMap` places the payload in a Config Map in `data` -> `help.chatUrl`. The payload will run whenever the victim clicks on the chat button in the bottom right corner of the page. For the sake of demonstration in this scenario, the attacker sets the URL value to "`javascript: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

```
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"

  statusbadge.url: "https://cd-status.apps.argoproj.io/"
  users.anonymous.enabled: "true"
  users.session.duration: "24h"

  passwordPattern: "^.{8,32}$"


  ga.trackingid: "UA-12345-1"
```

```
  help.chatUrl: "javascript: alert(1337)"
  help.chatText: "Chat now!"


  help.download.linux-arm64: "path-or-url-to-download"
  help.download.darwin-amd64: "path-or-url-to-download"
  help.download.darwin-arm64: "path-or-url-to-download"
  help.download.windows-amd64: "path-or-url-to-download"
```

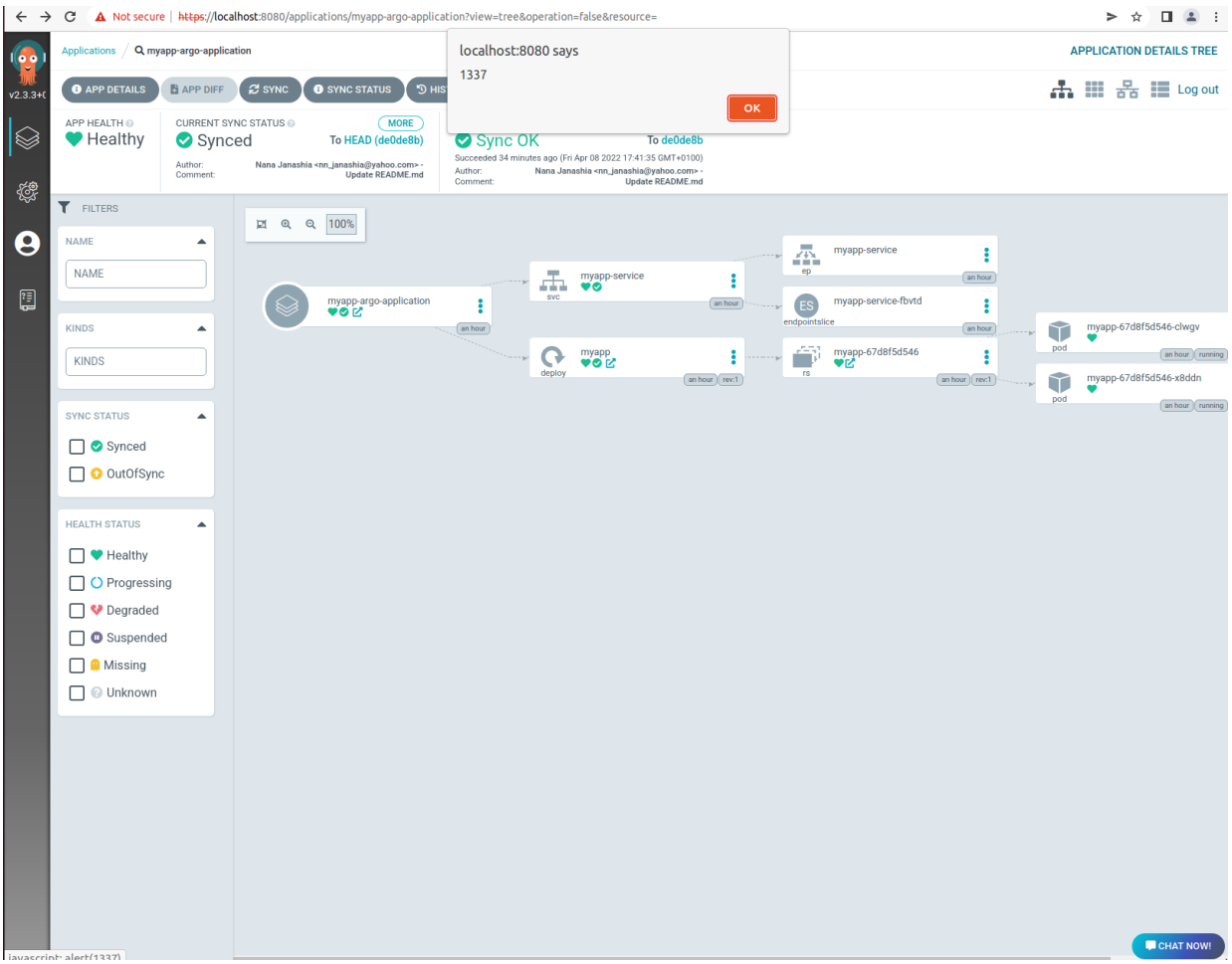
2: The `ConfigMap` is applied to the cluster with `kubectl apply -n argocd -f argocd-cm.yaml`.

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



4: XSS has been achieved:

ADALOGICS


Not secure | https://localhost:8080/applications/myapp-argo-application?view=tree&operation=false&resource=
Applications / myapp-argo-application
APPLICATION DETAILS TREE
APP DETAILS
APP DIFF
SYNC
SYNC STATUS
Log out
localhost:8080 says
1337
OK
APP HEALTH
Healthy
CURRENT SYNC STATUS
Synced
To HEAD (de0de8b)
Sync OK
To de0de8b
Succeeded 34 minutes ago (Fri Apr 08 2022 17:41:35 GMT+0100)
Author: Nana Janashia <nn_janashia@yahoo.com> -
Comment: Update README.md
FILTERS
NAME
KINDS
SYNC STATUS
Synced
OutOfSync
HEALTH STATUS
Healthy
Progressing
Degraded
Suspended
Missing
Unknown
100%
myapp-argo-application
myapp-service
myapp
myapp-service
myapp-service-fbvtd
myapp-67d8f5d546
myapp-67d8f5d546-clwgv
myapp-67d8f5d546-x8ddn
CHAT NOW!

ADALOGICS

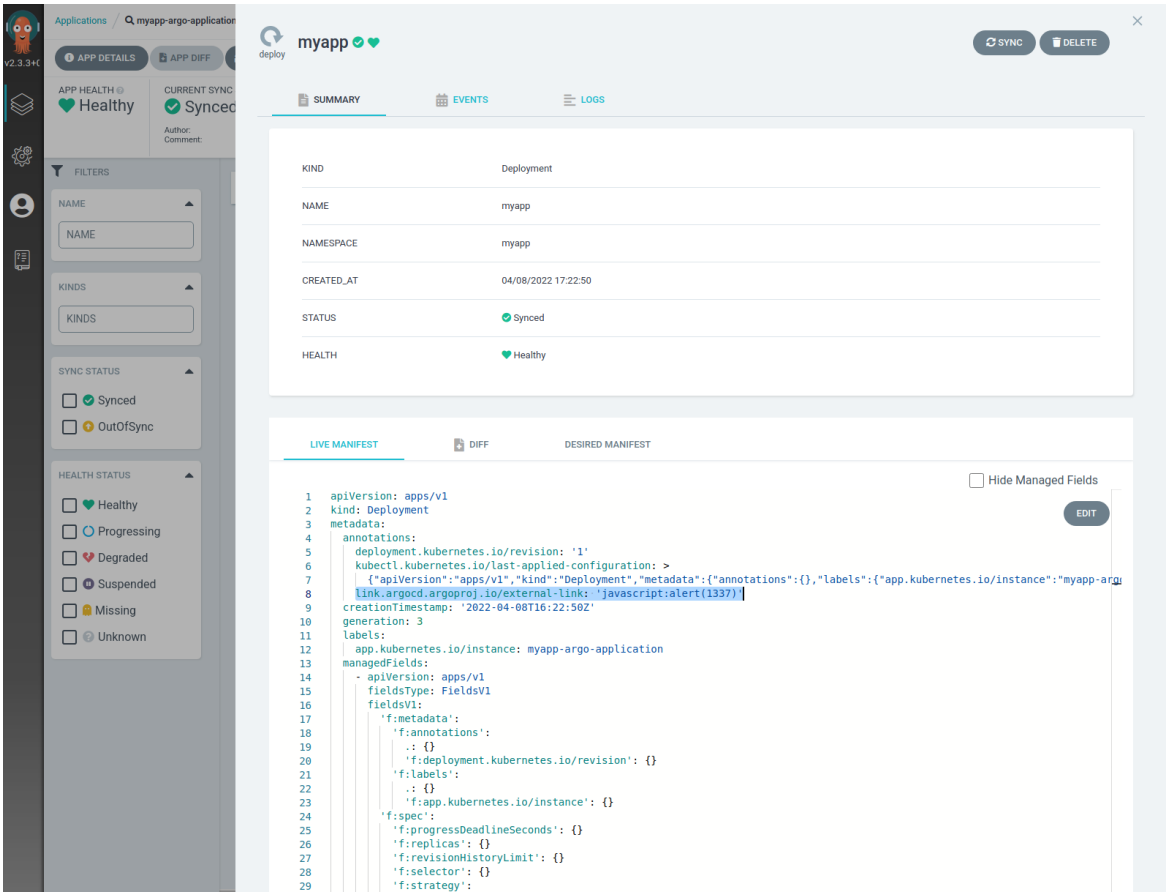
ADA-ARGO-SA-02: XSS: External URLs for Deployments can include javascript

Severity	Critical
Difficulty	Medium
Target	Argo CD
Github advisory	GHSA-h4w9-6x78-8vrj
CVE	CVE-2022-31035

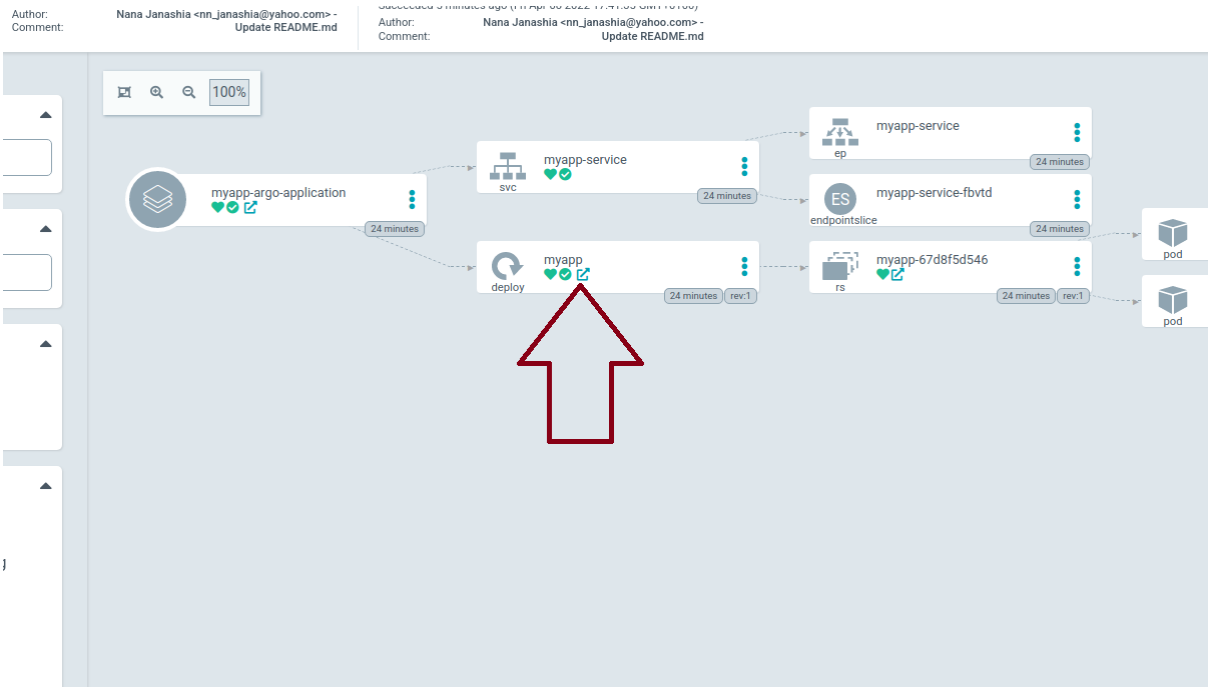
Argo CD allows users to add external links for Deployments: https://argo-cd.readthedocs.io/en/stable/user-guide/external-url/#add-external-url. This could be a link to monitoring pages or documentation for the related Deployment. The URL values are not sanitized. Therefore, an attacker who can specify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

Attack scenario

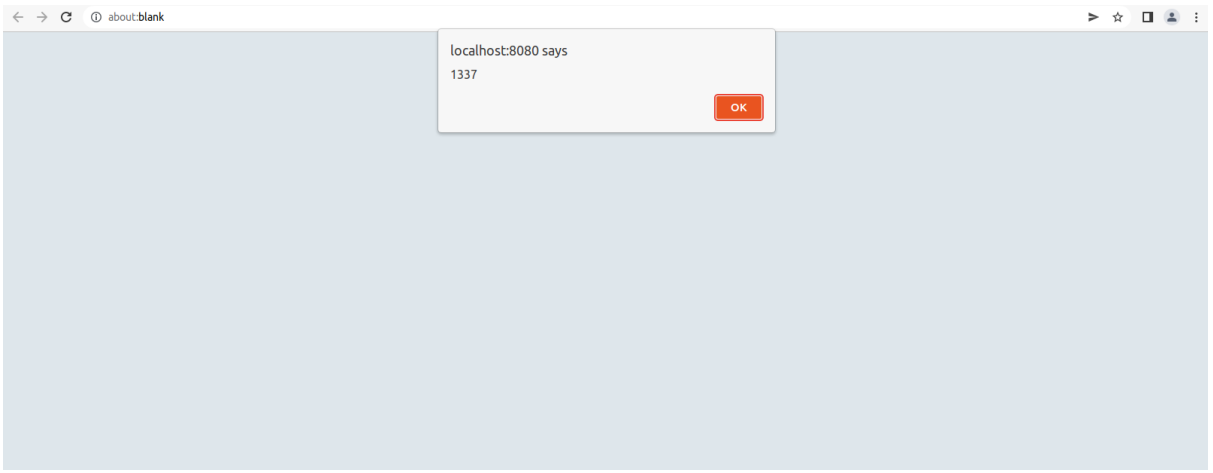
1: An attacker adds a payload as a URL for a Deployment. This is done through the `link.argocd.argoproj.io/external-link value` through the dashboard. For the sake of this demonstration, the attackers sets the URL value to `'javascript: alert(1337)'`:



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:



ADALOGICS

ADA-ARGO-SA-03: XSS in Argo Workflows chat url

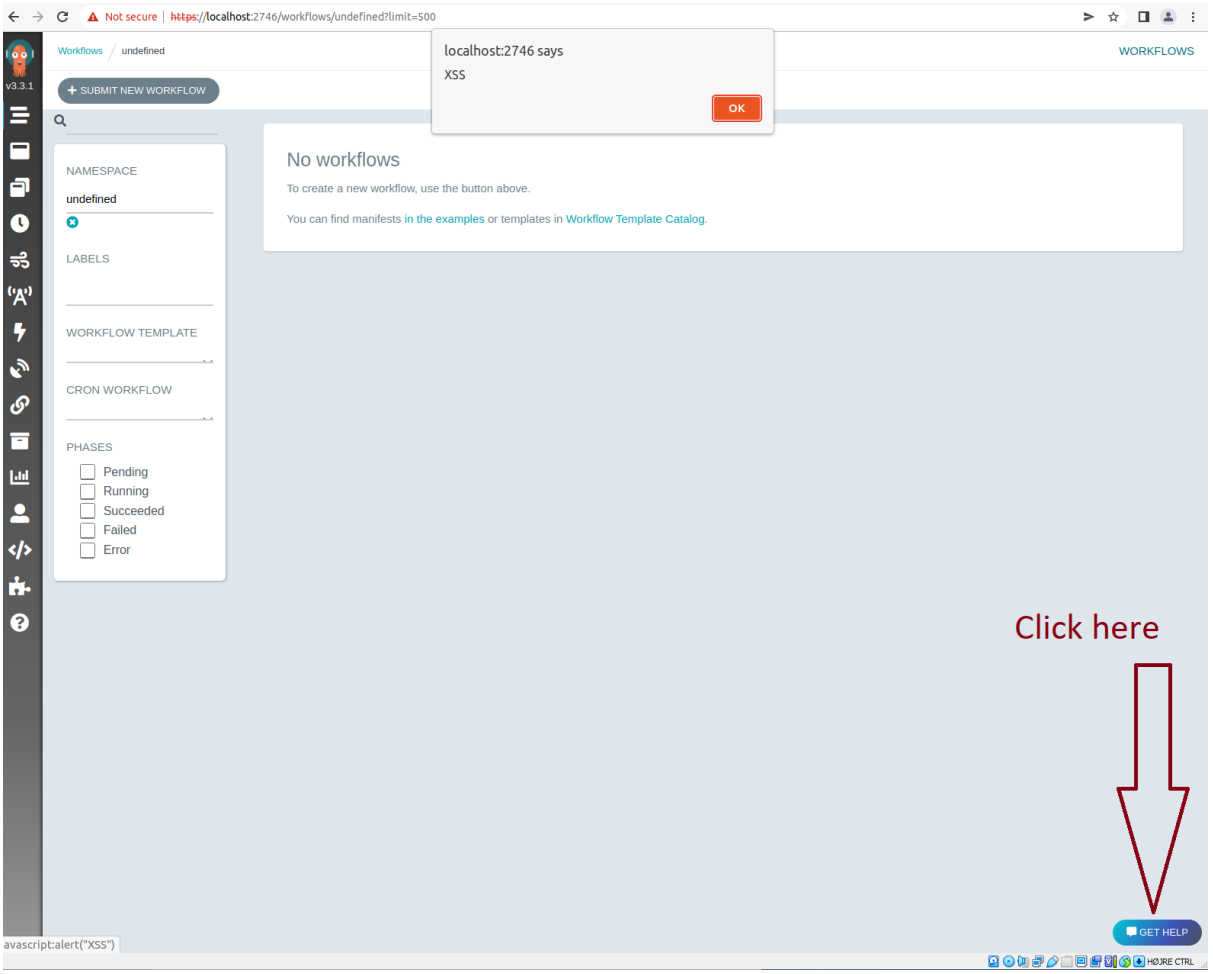
Severity	Medium
Difficulty	High
Target	Argo Workflows
Fix	Not fixed

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: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L73.
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.

ADALOGICS


Not secure | https://localhost:2746/workflows/undefined?limit=500
Workflows / undefined
WORKFLOWS
v3.3.1
+ SUBMIT NEW WORKFLOW
localhost:2746 says
XSS
OK
NAMESPACE
undefined
LABELS
WORKFLOW TEMPLATE
CRON WORKFLOW
PHASES
Pending
Running
Succeeded
Failed
Error
No workflows
To create a new workflow, use the button above.
You can find manifests in the examples or templates in Workflow Template Catalog.
Click here
GET HELP
avascript:alert("XSS")

ADALOGICS

ADA-ARGO-SA-04: XSS in Workflows workflow link

Severity	Medium
Difficulty	High
Target	Argo Workflows
Fix	Not fixed

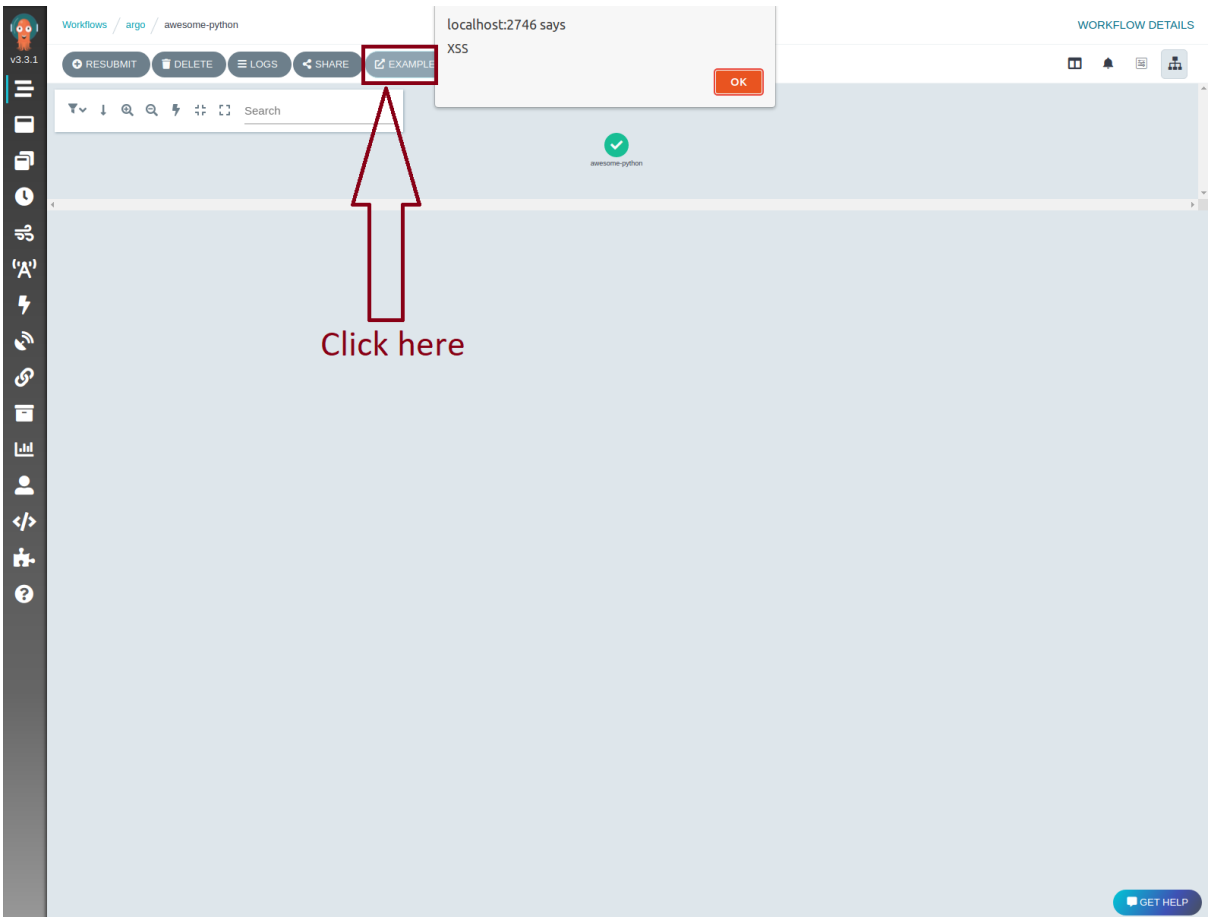
Argo Workflows allows users to add a button linking to any URL of their choosing on the Workflow page. This could be a link to an organization's logging facilities. The link is specified via a Config Map like is demonstrated here: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L56. 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.

ADALOGICS


v3.3.1
Workflows / argo / awesome-python
RESUBMIT
DELETE
LOGS
SHARE
EXAMPLE
Search
localhost:2746 says
XSS
OK
WORKFLOW DETAILS
awesome-python
Click here
GET HELP

ADALOGICS

ADA-ARGO-SA-05: XSS in Pod Logs link

Severity	Medium
Difficulty	High
Target	Argo Workflows
Fix	Not fixed

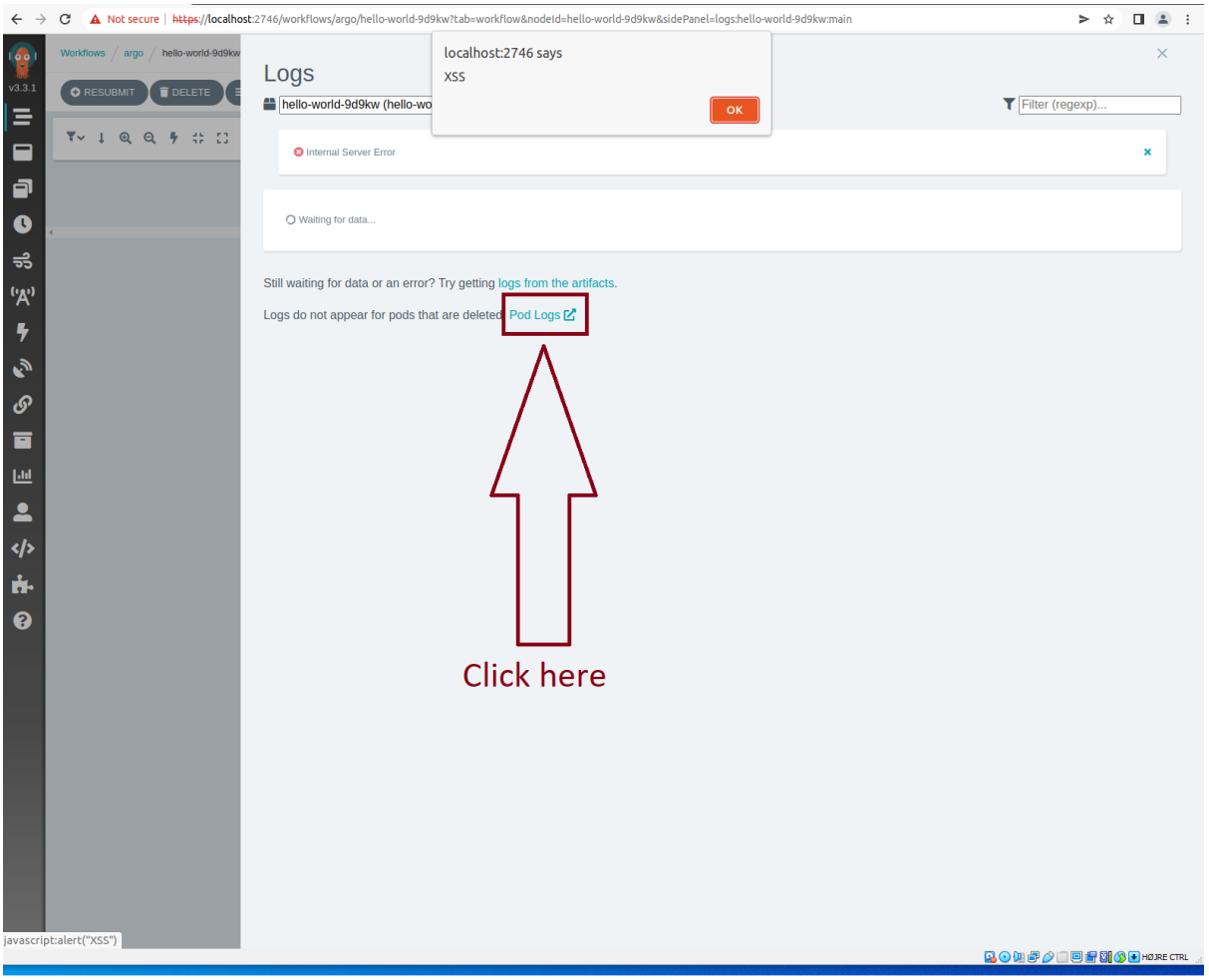
Argo Workflows allows users to add a link to any URL of their choosing on the pod logs page. This could be a link to the logging facilities of the given pod. Both the text of the link as well as the URL can be specified through a Config Map: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L60.
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:

ADALOGICS


Not secure | https://localhost:2746/workflows/argo/hello-world-9d9kw?tab=workflow&nodeId=hello-world-9d9kw&sidePanel=logs:hello-world-9d9kw:main
Workflows / argo / hello-world-9d9kw
v3.3.1
RESUBMIT
DELETE
Logs
hello-world-9d9kw (hello-wo
Filter (regexp)...
localhost:2746 says
xss
OK
Internal Server Error
Waiting for data...
Still waiting for data or an error? Try getting logs from the artifacts.
Logs do not appear for pods that are deleted Pod Logs
Click here
javascript:alert("XSS")
HØJRE CTRL

ADALOGICS

ADA-ARGO-SA-06: XSS in sensor logs link

Severity	Medium
Difficulty	High
Target	Argo Workflows
Fix	Not fixed

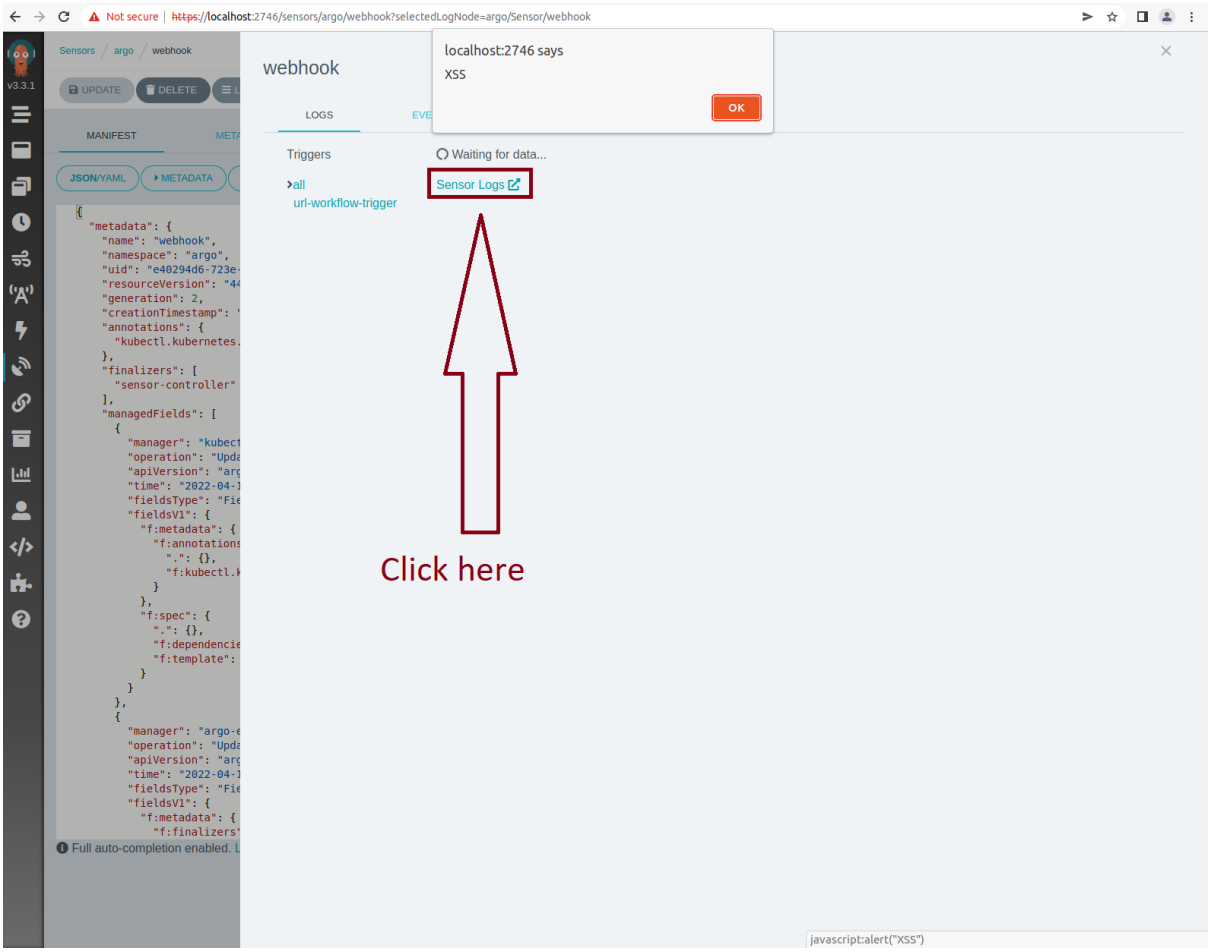
Argo Workflows allows users to add a link to the logs view of a sensor. This can be a link to the logs of a given sensor. The text and URL of the link can be specified via a Config Map as is demonstrated here: https://github.com/argoproj/argo-workflows/blob/master/docs/workflow-controller-configmap.yaml#L69.
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.

ADALOGICS


Not secure
https://localhost:2746/sensors/argo/webhook?selectedLogNode=argo/Sensor/webhook
Sensors / argo / webhook
v3.3.1
UPDATE
DELETE
MANIFEST
JSON/YAML
METADATA
webhook
LOGS
localhost:2746 says
xss
OK
Triggers
Waiting for data...
all
url-workflow-trigger
Sensor Logs
Click here
Full auto-completion enabled.
javascript:alert("XSS")

ADALOGICS

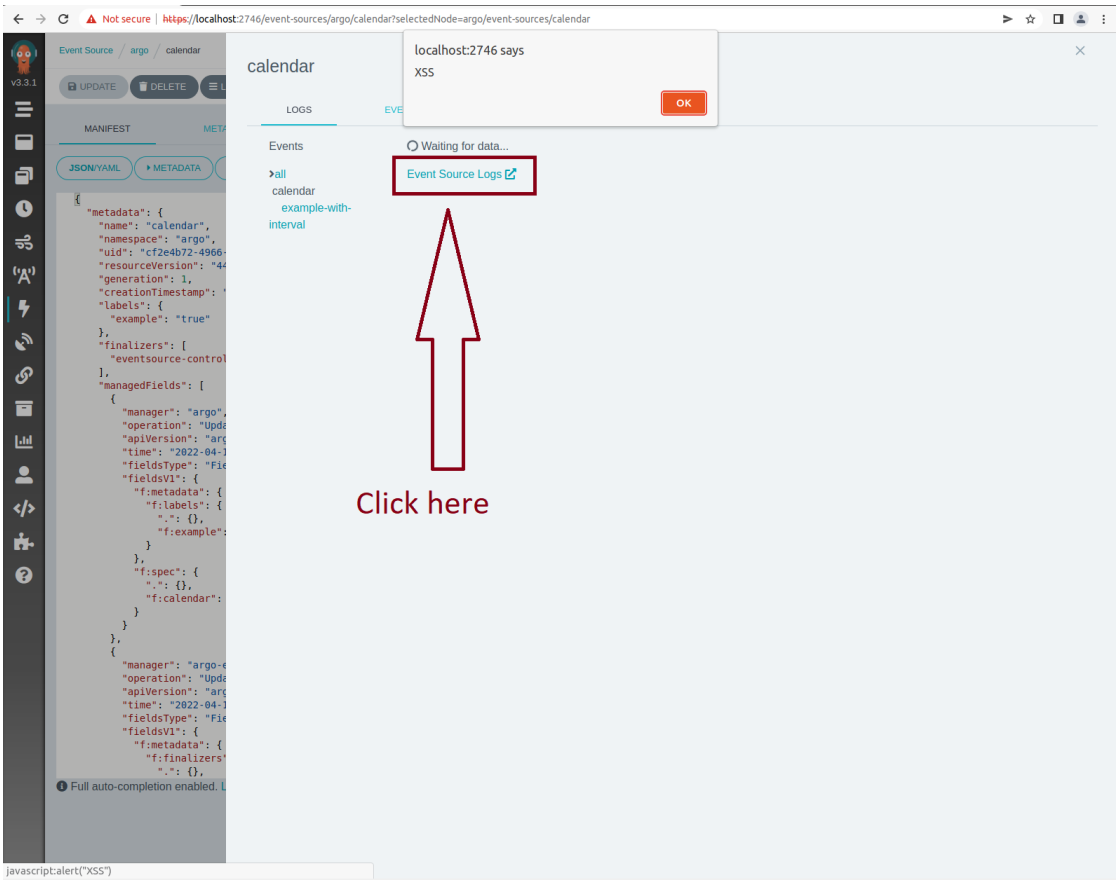
ADA-ARGO-SA-07: XSS in Event Source logs url

Severity	Medium
Difficulty	High
Target	Argo Workflows

Argo 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:



ADALOGICS

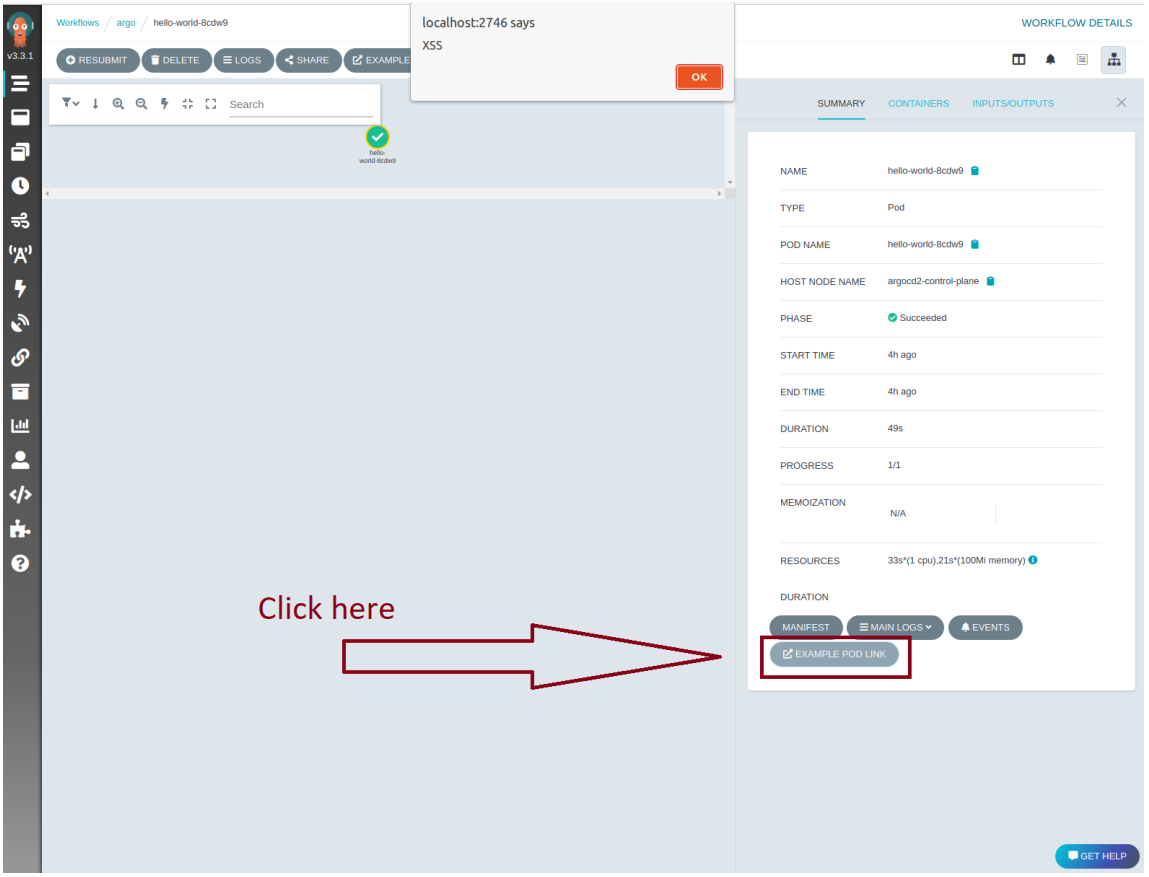
ADA-ARGO-SA-08: XSS in Pod link url

Severity	Medium
Difficulty	High
Target	Argo Workflows

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:



ADALOGICS

ADA-ARGO-SA-09: XSS through server side rendering

Severity	Low
Difficulty	High
Target	Argo CD
Github advisory	GHSA-pmjg-52h9-72qv
CVE	CVE-2022-31102

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:

- https://github.com/argoproj/argo-cd/blob/master/util/oidc/oidc.go#L374
- https://github.com/argoproj/argo-cd/blob/master/util/oidc/templates.go#L16

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:

```
<script>
var hash = window.location.hash.substr(1);
var result = hash.split('&').reduce(function (result, item) {
      var parts = item.split('=');
```

```
        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:

```
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
}
```

ADALOGICS

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

Severity	Medium
Difficulty	High
Target	Argo Events
Fix	Not fixed

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
Reproducing OSS-fuzz crashes:
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

```
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: \n%+v\n", events)
        fmt.Printf("parameters: \n%+v\n", parameters)
```

```
        _, _ = ConstructPayload(events, parameters)
        return 1
}
```

Parameters

```
events:
map[:{"context:" "{"id:" "", "source": "", "specversion": "", "type": "",
"datacontenttype": "", "subject": "", "time": "0001-01-01 00:00:00 +0000 UTC"}",
"data": ""}]
parameters:
[{Src:&TriggerParameterSource{DependencyName:,ContextKey:,ContextTemplate:-',Dat
aKey:,DataTemplate:,Value:*,}
Dest:444444444444444444444444444444444444444444444444
Operation:4444444444444444444444444444444444444444444444}]
```

ADA LOGICS

ADA-ARGO-SA-11: Insufficient logging when executing a trigger

Severity	Informational
Difficulty	High
Target	Argo Events
Fix	Not fixed

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:

```
// 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

4. HTTP
5. Kafka
6. NATS
7. Pulsar
8. Custom Trigger

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 `ContructPayload()` including the payload.

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

```
      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.

ADALOGICS

ADA-ARGO-SA-12: 8 Uses of deprecated API can be used to cause DoS in user-facing endpoints

Severity	High
Difficulty	Medium
Target	Argo Events
Github advisory	https://github.com/argoproj/argo-events/security/advisories/GHSA-5q86-62xr-3r57
Fix	https://github.com/argoproj/argo-events/pull/1966

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

Since `io/ioutil` has ceased maintenance we recommend discontinuing all use of this package.

ADA LOGICS

ADA-ARGO-SA-13: Insecure path traversal in Git Trigger Source can lead to arbitrary file read

Severity	High
Difficulty	High
Target	Argo Events
Github advisory	GHSA-qpgx-64h2-gc3c
Fix	https://github.com/argoproj/argo-events/pull/1965

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

```
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:

```
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

ADALOGICS

ADA-ARGO-SA-14: Insufficient logging if OIDC Provider is created with `nil` TLS config

Severity	Informational
Difficulty	High
Target	Argo CD
Fix	Not fixed. Tracked in https://github.com/argoproj/argo-cd/issues/9957

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), where the `ClientApp.client.Transport.TLSClientConfig` can be `nil`:

```
...
	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.

ADALOGICS

ADA-ARGO-SA-15: Path traversal leading to arbitrary YAML file read with malicious Helm chart

Severity	Moderate
Difficulty	High
Target	Argo CD
Github advisory	GHSA-q4w5-4gq2-98vm
CVE	CVE-2022-31036

A path traversal attack was found in func `(s *Service).GetAppDetails()`[2] 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():

```
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
```

[2] 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()`:

```
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
	}
	h, err := helm.NewHelmApp(appPath, getHelmRepos(q.Repos), false, version,
q.Repo.Proxy, 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/passwd2`. The contents of that file was:

```
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/passwd2` 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:

```
{
    "type": "Helm",
    "helm": {
        "valueFiles": [
            "values.yaml"
        ],
        "parameters": [
            {
                "name": "name",
                "value": "Some Name"
            },
            {
                "name": "section.password",
                "value": "1234"
            }
        ]
    }
}
```

ADALOGICS

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

```
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 {
		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`:

```
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 {
```

```
                        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 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`:

```
        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/argo-cd/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.

ADALOGICS

ADA-ARGO-SA-17: DoS through large manifest files

Severity	Moderate
Difficulty	High
Target	Argo CD
Github advisory	GHSA-jhqp-vf4w-rpwq
CVE	CVE-2022-31016

In `github.com/argoproj/argo-cd/v2/reposerver/repository.findManifests()`, files are read via the 3rd-party API `github.com/TomOnTime/utfutil.ReadFile()`. This API is a thin wrapper around `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()` 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()` 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()`:

```
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 {
		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)
      }
```

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.

ADALOGICS

ADA-ARGO-SA-18: nil-pointer in 3rd-party library 2

Severity	Low
Difficulty	High
Target	Argo Events
Fix	Not fixed

`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

```
connection.evaluableExpression, err =
govaluate.NewEvaluableExpression(strings.ReplaceAll(dependencyExpression
, "-", "\\-"))
```

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

To reproduce:

```
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/lexerStrea
m.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)
```

```
github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/parsing.go:
296 +0x39
github.com/Knetic/govaluate.readToken(0x10c0002c6f30, {0x0, 0x1, 0x0, {0x2b132c0, 0xa, 0xa}},
0x3)

github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/parsing.go:
240 +0xeaf
github.com/Knetic/govaluate.parseTokens({0x10c0002ccd68, 0x3}, 0x0)

github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/parsing.go:
28 +0x189
github.com/Knetic/govaluate.NewEvaluableExpressionWithFunctions({0x10c0002ccd68, 0x3},
0x1b83b65)

github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/EvaluableE
xpression.go:99 +0x94
github.com/Knetic/govaluate.NewEvaluableExpression(...)

github.com/Knetic/govaluate@v3.0.1-0.20171022003610-9aa49832a739+incompatible/EvaluableE
xpression.go:47
```

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/sensors/listener.go#L131
- https://github.com/argoproj/argo-events/blob/master/eventbus/stan/sensor/trigger_conn.go#L324
- https://github.com/argoproj/argo-events/blob/master/sensors/dependencies/filter.go#L149

ADALOGICS

ADA-ARGO-SA-19: DoS through well-crafted Boolean Expression

Severity	Low
Difficulty	High
Target	Argo Events
Fix	https://github.com/argoproj/argo-events/pull/1958

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.

```
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,0x58,0x58,0x35,0x26,0
x26,0x5a,0x58,0x58,0x26,0x26,0x78,0x73,0x26,0x26,0x5a,0x58,0x58,0x73,0x26,0x26,0x58,0x73
,0x26,0x26,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x5a,0x58,0x58,0x26,0x
26,0x78,0x73,0x26,0x26,0x5a,0x58,0x58,0x73,0x26,0x26,0x58,0x73,0x26,0x26,0x54,0x54,0x54,
0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x54,0x5
4,0x5a,0x58,0x58,0x26,0x26,0x5a,0x29,0x28,0x58,0x58,0x26,0x26,0x29,0x58,0x73,0x26,0x26,0
x5a,0x58,0x58,0x73,0x26,0x26,0x5a,0x26,0x26,0x29,0x5a,0x58,0xa}
        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.

ADALOGICS

ADA-ARGO-SA-20: Unmaintained 3rd-party dependencies

Severity	Low
Difficulty	Low
Target	Argo Events
Fix	Not fixed

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
2. github.com/joncalhoun/qson: No longer maintained. Is used in the Storage Grid Event Source: https://github.com/argoproj/argo-events/blob/master/eventsources/sources/storagegrid/start.go

ADALOGICS

ADA-ARGO-SA-21: Unused function parameters

Severity	Informational
Difficulty	High
Target	Argo Events
Fix	https://github.com/argoproj/argo-events/pull/1963

The following functions accept parameters that are not used in the functions body:

Function signature	File	Unused parameter
`func (e *expr) evaluatePostfix(vars []string, set term, postfix []string) bool`	https://github.com/argoproj/argo-events/blob/master/common/boolminifier.go	`vars`
`func (r *reconciler) reconcile(ctx context.Context, eventSource *v1alpha1.EventSource) error`	https://github.com/argoproj/argo-events/blob/master/controllers/eventsource/controller.go	`ctx`
`func (el *EventListener) processMessage(ctx context.Context, message *sqslib.Message, dispatch func([]byte, ...eventsourcecommon.Option) error, ack func(), log *zap.SugaredLogger)`	https://github.com/argoproj/argo-events/blob/master/eventsources/sources/awssqs/start.go	`ctx`
`getHook := func(hooks []*gitlab.ProjectHook, url string, events []string) *gitlab.ProjectHook`	https://github.com/argoproj/argo-events/blob/master/eventsources/sources/gitlab/start.go#L218	`events`
`func schema_argo_events_pkg_apis_common_Amount(ref common.ReferenceCallback) common.OpenAPIDefinition`	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	`ref`
`func schema_argo_events_pkg_apis_common_Int64OrString(ref common.ReferenceCallback) common.OpenAPIDefinition`	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	`ref`
`func schema_argo_events_pkg_apis_common_Metadata(ref common.ReferenceCallback) common.OpenAPIDefinition`	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	`ref`
`func schema_argo_events_pkg_apis_common_Resource(ref common.ReferenceCallback) common.OpenAPIDefinition`	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	`ref`

func schema_argo_events_pkg_apis_common_S3Bucket(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	ref
func schema_argo_events_pkg_apis_common_S3Filter(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/common/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_AMQPConsumeConfig(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_AMQPExchangeDeclareConfig(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_AMQPQueueBindConfig(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_BitbucketServerRepository(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_CatchupConfiguration(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_ConfigMapPersistence(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_EventSourceFilter(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_KafkaConsumerGroup(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref

func schema_pkg_apis_eventsource_v1alpha1_OwnedRepositories(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_Selector(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_StorageGridFilter(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_eventsource_v1alpha1_WatchPathConfig(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/eventsource/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_ConditionsResetByTime(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_DataFilter(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_EventDependencyTransformer(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_FileArtifact(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_GitRemoteConfig(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func schema_pkg_apis_sensor_v1alpha1_LogTrigger(ref common.ReferenceCallback) common.OpenAPIDefinition	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func	https://github.com/argopro	ref

schema_pkg_apis_sensor_v1alpha1_PayloadField(ref common.ReferenceCallback) **common.OpenAPIDefinition**	j/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	
func **schema_pkg_apis_sensor_v1alpha1_RateLimit**(ref common.ReferenceCallback) **common.OpenAPIDefinition**	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func **schema_pkg_apis_sensor_v1alpha1_StatusPolicy**(ref common.ReferenceCallback) **common.OpenAPIDefinition**	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func **schema_pkg_apis_sensor_v1alpha1_TimeFilter**(ref common.ReferenceCallback) **common.OpenAPIDefinition**	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func **schema_pkg_apis_sensor_v1alpha1_TriggerParameterSource**(ref common.ReferenceCallback) **common.OpenAPIDefinition**	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref
func **schema_pkg_apis_sensor_v1alpha1_URLArtifact**(ref common.ReferenceCallback) **common.OpenAPIDefinition**	https://github.com/argoproj/argo-events/blob/master/pkg/apis/sensor/v1alpha1/openapi_generated.go	ref

ADALOGICS

ADA-ARGO-SA-22: Functions always returning nil

Severity	Informational
Difficulty	High
Target	Argo Events
Fix	https://github.com/argoproj/argo-events/pull/1960

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

Function signature	File
`func setConfigDefaults(config *rest.Config) error`	https://github.com/argoproj/argo-events/blob/master/pkg/client/eventbus/clientset/versioned/typed/eventbus/v1alpha1/eventbus_client.go
`func (sensorCtx *SensorContext) triggerActions(ctx context.Context, sensor *v1alpha1.Sensor, events map[string]cloudevents.Event, trigger v1alpha1.Trigger) error`	https://github.com/argoproj/argo-events/blob/master/sensors/listener.go

ADALOGICS

ADA-ARGO-SA-23: Nil-pointer dereferences in sensor controller

Severity	Low
Difficulty	High
Target	Argo Events
Fix	https://github.com/argoproj/argo-events/pull/1961
CVE	CVE-2022-31034

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

`github.com/argoproj/argo-events/controllers/sensor.Validate()`

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

Parameter 1

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

If this parameter is `nil`, a nil-pointer dereference will induce a panic on the marked line below:

```
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

```
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")
		}
		if b.Spec.NATS != nil {
			// For STAN, EventSourceName + EventName can not be
referenced more than once in one Sensor object.
			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.

ADALOGICS

ADA-ARGO-SA-24: Use of `math/rand` in production code

Severity	High
Difficulty	Moderate
Target	Argo Events
Github advisory	GHSA-2m7h-86qq-fp4v
Fix	https://github.com/argoproj/argo-events/pull/1959

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/bitbucketserver/start.go
- 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
- https://github.com/argoproj/argo-events/blob/master/eventbus/jetstream/eventsource/source_conn.go

[3] https://pkg.go.dev/math/rand

ADALOGICS

ADA-ARGO-SA-25: Improper use of `InsecureSkipVerify`

Severity	High
Difficulty	High
Target	Argo Events & Argo CD
Fix	Partially (Fixed in ArgoCD but not Argo Events)
Github advisory	GHSA-7943-82jg-wmw5
CVE	CVE-2022-31105

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

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

API	File
`func (reader *URLReader) Read() ([]byte, error)`	https://github.com/argoproj/argo-events/blob/master/sensors/artifacts/url.go
`func (stream *Jetstream) MakeConnection() (*JetstreamConnection, error)`	https://github.com/argoproj/argo-events/blob/master/eventbus/jetstream/base/jetstream.go
`func GetTLSConfig(config *apicommon.TLSConfig) (*tls.Config, error)`	https://github.com/argoproj/argo-events/blob/master/common/util.go
`func (e *natsEventBusElector) RunOrDie(ctx context.Context, callbacks LeaderCallbacks)`	https://github.com/argoproj/argo-events/blob/master/common/leaderelection/leaderelection.go

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.

ADALOGICS

ADA-ARGO-SA-26: Insecure production use of `crypto/ssh.InsecureIgnoreHostKey`

Severity	Low
Difficulty	High
Target	Argo Events
Fix	https://github.com/argoproj/argo-events/pull/1982

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

ADALOGICS

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

TOB-ARGO-TM1

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/6742

TOB-ARGO-TM2

Fixed	Partially fixed
Relevant links	https://github.com/argoproj/argo-cd/issues/9968

TOB-ARGO-TM3

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/issues/1364

TOB-ARGO-TM4

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/5477

Comments
Documentation should be updated here:
https://github.com/argoproj/argo-cd/blob/a809469d9af10c626449bfcb8b9a09a9d2dc9065/util/session/sessionmanager.go#L153 - To include information about the admin user.

TOB-ARGO-TM5

Fixed	Will not fix
Relevant links	n/a

TOB-ARGO-TM6

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/7071

TOB-ARGO-TM7

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/7094

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

TOB-ARGO-TM8

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/5477 https://github.com/argoproj/argo-cd/issues/9962

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

```
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

TOB-ARGO-TM9

Fixed	Will not fix
Relevant links	n/a

TOB-ARGO-TM10

Fixed	Will not fix
Relevant links	n/a

TOB-ARGO-TM11

Fixed	Yes
Relevant links	https://argo-cd.readthedocs.io/en/stable/operator-manual/user-management/#failed-logins-rate-limiting

TOB-ARGO-TM12

Fixed	Yes
Relevant links	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

TOB-ARGO-TM16

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-workflows/pull/6515

TOB-ARGO-TM17

Fixed	Yes
Relevant links	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

TOB-ARGO-TM19

Fixed	Fixed
Relevant links	https://github.com/argoproj/argo-events/pull/1585

	https://github.com/argoproj/argo-events/pull/1090

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

TOB-ARGO-TM21

Fixed	Yes
Relevant links	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

TOB-ARGO-001

Fixed	Fixed
Relevant links	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
Relevant links	https://github.com/argoproj/argo-cd/blob/master/util/cache/cache.go

Comments
We recommend avoiding storing credentials in environment variables.

TOB-ARGO-003

Fixed	Fixed
Relevant links	https://github.com/argoproj/argo-cd/pull/6742

TOB-ARGO-004

Fixed	Fixed

Relevant links	https://github.com/argoproj/argo-cd/pull/5786

TOB-ARGO-005

Fixed	Fixed
Relevant links	https://github.com/argoproj/argo-rollouts/pull/1168

TOB-ARGO-006

Fixed	No
Relevant links	https://github.com/argoproj/argo-workflows/issues/5312 https://github.com/argoproj/argo-cd/issues/9963

Comments

Examples of unfixed cases:

1. https://github.com/argoproj/argo-cd/blob/master/util/gpg/gpg.go#L167
2. https://github.com/argoproj/argo-cd/blob/master/util/gpg/gpg.go#L187

This is being tracked in https://github.com/argoproj/argo-cd/issues/9963

TOB-ARGO-007

Fixed	Wontfix
Relevant links	https://github.com/argoproj/argo-workflows/issues/5860

TOB-ARGO-008

Fixed	Yes
Relevant links	https://github.com/argoproj/gitops-engine/pull/243 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/hack/docgen.go#L164
- https://github.com/argoproj/argo-workflows/blob/master/examples/validator.go#L44

TOB-ARGO-009

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-workflows/pull/5211

TOB-ARGO-010

Fixed	Check pending
Relevant links	

TOB-ARGO-011

Fixed	No

Relevant links	https://github.com/argoproj/argo-workflows/pull/6985

Comments
The initial fix in https://github.com/argoproj/argo-workflows/pull/6985 has been replaced in 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

Fixed	No
Relevant links	https://github.com/argoproj/argo-workflows/pull/6985

Comments
The initial fix in https://github.com/argoproj/argo-workflows/pull/6985 has been replaced in 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

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/issues/5778

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

Fixed	No
Relevant links	https://github.com/argoproj/argo-workflows/issues/6520

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

Fixed	Not by the end of this Audit. Argo may have fixed it later here: https://github.com/argoproj/argo-rollouts/pull/1155 but due to timing this fix was not confirmed by Ada Logics.
Relevant links	n/a

TOB-ARGO-016

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/6361

TOB-ARGO-017

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/security/advisories/GHSA-2m7h-86qq-fp4v

Comments
Argo CD still used the insecure `math/rand` here:
https://github.com/argoproj/argo-cd/blob/master/util/rand/rand.go.
This was found to not be fixed by Ada Logics and was fixed during this audit.

TOB-ARGO-018

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/security/advisories/GHSA-6gcg-hp2x-q54h

- 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)

This was fixed during this audit.

TOB-ARGO-019

Fixed	No
Relevant links	n/a

TOB-ARGO-020

Fixed	Will not fix
Relevant links	n/a

TOB-ARGO-021

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/issues/9964

TOB-ARGO-022

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/issues/9969

TOB-ARGO-023

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/6064

TOB-ARGO-024

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/issues/9966

TOB-ARGO-025

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/9057

TOB-ARGO-026

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/8943

TOB-ARGO-027

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/8943

TOB-ARGO-028

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-events/pull/1815

TOB-ARGO-029

Fixed	No
Relevant links	https://github.com/argoproj/argo-cd/pull/9967

TOB-ARGO-030

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/5774

TOB-ARGO-031

Fixed	No
Relevant links	n/a

TOB-ARGO-032

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/issues/5433

TOB-ARGO-033

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/pull/5477

TOB-ARGO-034

Fixed	Yes
Relevant links	• https://github.com/alexec/argo-workflows/commit/4b5adc7f8f2f1e10ff0618b223ab9eb126846031 • 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://argoproj.github.io/argo-rollouts/
- https://argo-cd.readthedocs.io/en/stable/

TOB-ARGO-035

Fixed	Yes
Relevant links	https://github.com/argoproj/argo-cd/issues/6917

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.