CVE-2022-24112_Apache_APISIX_远程代码执行漏洞

漏洞描述

攻击者可以向 `batch-requests` 插件发送请求来绕过管理API的IP限制。Apache APISIX 的默认配置(带有默认的 API 密钥)容易受到远程代码执行的攻击。当管理密钥更改或管理 API 端口更改为与数据面板不同的端口时，影响较小。但是，绕过 Apache APISIX 数据面板的 IP 限制仍然存在风险。在 `batch-requests` 插件中有一个检查，它用真实的远程 IP 覆盖客户端 IP。但是由于代码中的一个 bug，这个检查可以被绕过。

batch-requests 插件介绍

`batch-requests` 插件可以一次接受多个请求并以 http pipeline 的方式在网关发起多个 http 请求，合并结果后再返回客户端，这在客户端需要访问多个接口时可以显著地提升请求性能。
在漏洞版本中、默认是启用状态。
https://apisix.apache.org/zh/docs/apisix/2.12/plugins/batch-requests/

复现思路：

1、搭建漏洞环境。(注意需要修改:conf.yaml/allow_admin)
2、查看 diff 记录、进行简单调试，漏洞分析。
3、漏洞测试。

漏洞版本

Apache APISIX 1.3 ~ 2.12.1 之间的所有版本（不包含 2.12.1 ）
Apache APISIX 2.10.0 ~ 2.10.4 LTS 之间的所有版本（不包含 2.10.4）

环境搭建

搭建方式	Docker 搭建		
	版本号	默认端口	默认 admin_key/用户
Apache APISIX	apisix:2.12.0-alpine	9080	edd1c9f034335f136f87ad84b625c8f1

Apache APISIX Dashboard	apisix-dashboard:2.10.1-alpine	9000	admin/admin

环境搭建可以使用下面这个 Github 地址

Github 地址 https://github.com/twseptian/cve-2022-24112/tree/main/apisix-docker

```
cd CVE-2022-24112-main/apisix-docker/example/
```

```
vi apisix_conf/config.yaml
```

将 allow_admin 修改为 127.0.0.0/24，此处修改的实际为 nginx.conf。

使用 docker-compose 创建 docker 容器。

```
docker-compose -p apisixCveTest up -d
```

在服务启动阶段、会读取 config.yaml、生成 nginx.conf 。

1、会基于 Lua 模板 apisix/cli/ngx_tpl.lua 文件生成 nginx.conf。(APISIX 架构介绍：https://blog.csdn.net/alex_yangchuansheng/article/details/122053371)

2、调用 `ngx_http_access_module` 模块。该模块限制客户端对某些地址的访问。(nginx 模块介绍：http://nginx.org/en/docs/http/ngx_http_access_module.html#allow)

```
apisix:
  node_listen: 9080              # APISIX listening port
  enable_ipv6: false

  allow_admin:                  # http://nginx.org/en/docs/http/ngx_http_access_module.h
    - 127.0.0.0/24              # We need to restrict ip access rules for security. 0.0.

  admin_key:
    - name: "admin"
      key: edd1c9f034335f136f87ad84b625c8f1
      role: admin                # admin: manage all configuration data
                                 # viewer: only can view configuration data
    - name: "viewer"
      key: 4054f7cf07e344346cd3f287985e76a2
      role: viewer

  enable_control: true
  control:
    ip: "0.0.0.0"
    port: 9092
```

进入到容器中查看，conf/nginx.conf。allow/deny 是设置允许与拒绝访问的地址。只允许 127.0.0.1/24 访问 /apisix/admin 接口。

Admin api 接口如下：

/apisix/admin/routes/{id}?ttl=0	Route 字面意思就是路由，通过定义一些规则来匹配客户端的请求，然后根据匹配结果加载并执行相应的插件，并把请求转发给到指定 Upstream。
/apisix/admin/services/{id}	`Service` 是某类 API 的抽象（也可以理解为一组 Route 的抽象）。它通常与上游服务抽象是一一对应的，`Route` 与 `Service` 之间，通常是 N:1 的关系
/apisix/admin/consumers/{username}	Consumer 是某类服务的消费者，需与用户认证体系配合才能使用。Consumer 使用 `username` 作为唯一标识，只支持使用 HTTP `PUT` 方法创建 Consumer。
/apisix/admin/upstreams/{id}	Upstream 是虚拟主机抽象，对给定的多个服务节点按照配置规则进行负载均衡。Upstream 的地址信息可以直接配置到 `Route`（或 `Service`）上，当 Upstream 有重复时，就需要用“引用”方式避免重复了。
/apisix/admin/ssl/{id}	SSL
/apisix/admin/global_rules/{id}	设置全局运行的插件。这一类插件在所有路

	由级别的插件之前优先运行。
/apisix/admin/plugin_configs/{id}	配置一组可以在路由间复用的插件。
/apisix/admin/plugin_metadata/{plugin_name}	插件元数据
/apisix/admin/plugins/{plugin_name}	插件
/apisix/admin/stream_routes/{id}	Stream Route 是用于 TCP/UDP 动态代理的路由。

漏洞分析：

diff ：

1. https://github.com/apache/apisix/pull/6254/commits/f5d44d1f58e0160132f009465e807f4093601a11
2. https://github.com/apache/apisix/pull/6251/commits/8f28aadc35362f417f89db505c18cd7f1d548c86
3. https://github.com/apache/apisix/pull/6204/commits/3d4b35b8e07f2c7cfcce7d7f56f24fad4e6e39de

```
165       local local_conf = core.config.local_conf()
166       local real_ip_hdr = core.table.try_read_attr(local_conf, "nginx_config", "http",
167                                                    "real_ip_header")
168   +   -- we don't need to handle '_' to '-' as Nginx won't treat 'X_REAL_IP' as 'X-Real-IP'
169   +   real_ip_hdr = str_lower(real_ip_hdr)
170
171       local outer_headers = core.request.headers(nil)
172       for i,req in ipairs(data.pipeline) do
```

查看 diff 记录后、修复方式是将 real_ip_hdr 转化为小写。
1、查看插件 batch-requests 代码，问题是出现在 set_common_header() 函数中

```
local function set_common_header(data)
    local local_conf = core.config.local_conf()
    local real_ip_hdr = core.table.try_read_attr(local_conf, "nginx_config", "http",
                                                 "real_ip_header")

    local outer_headers = core.request.headers( ctx: nil)
    for i,req in ipairs(data.pipeline) do
        for k, v in pairs(data.headers) do
            if not req.headers[k] then
                req.headers[k] = v
            end
        end

        if outer_headers then
            for k, v in pairs(outer_headers) do
                local is_content_header = str_find(k, needle: "content-") == 1
                -- skip header start with "content-"
                if not req.headers[k] and not is_content_header then
                    req.headers[k] = v
                end
            end
        end

        req.headers[real_ip_hdr] = core.request.get_remote_client_ip()
    end
end
```

函数参数为 data 是我们传入的请求体经过 json.decode()后的数据、类型为"table"。

```
        }
    end

    local data, err = core.json.decode(req_body)   data: table
    if not data then
        return 400, {
            error_msg = "invalid request body: " .. req_body .. ", err: " .. err
        }
    end
```

查看代码,**real_ip_hdr** 为调用 **try_read_attr**()函数获取到的返回值。通过遍历表结构、获取到 **real_ip_header** 的值然后再赋值给 **real_ip_hdr** 。

select('#', ...) 获取输入参数的数量,

select(i, ...) 获取第 n 个参数,

```
function _M.try_read_attr(tab, ...)
    local count = select( index : '#', ...)

    for i = 1, count do
        local attr = select(i, ...)
        if type(tab) ~= "table" then
            return nil
        end

        tab = tab[attr]
    end

    return tab
end
```

注意：此时的 real_ip_hdr 为 X-Real-IP 为大写。该值为在系统启动后、给定的默认值。

```
proxy_http_version 1.1;
proxy_set_header   Host              $upstream_host;
proxy_set_header   Upgrade           $upstream_upgrade;
proxy_set_header   Connection        $upstream_connection;
proxy_set_header   X-Real-IP         $remote_addr;
proxy_pass_header  Date;

### the following x-forwarded-* headers is to send to upstream server

set $var_x_forwarded_for        $remote_addr;
```

2、通过遍历 **data.pipeline** 和 **data.headers** 、将 data.headers 出现的头信息赋值给 data.pipline。发现 **data.headers** 中出现了 **x-real-ip:127.0.0.1** ，这是我们调用 batch-requests 插件 传递的头信息，而此时系统默认的 **X-Real-IP 为大写。**

在请求头覆盖中、因为 real_ip_hdr 为 **X-Real-IP** ，函数 **core.request.get_remote_clinte_ip()获取远程客户端 ip、不能将 x-real-ip 給覆盖。**

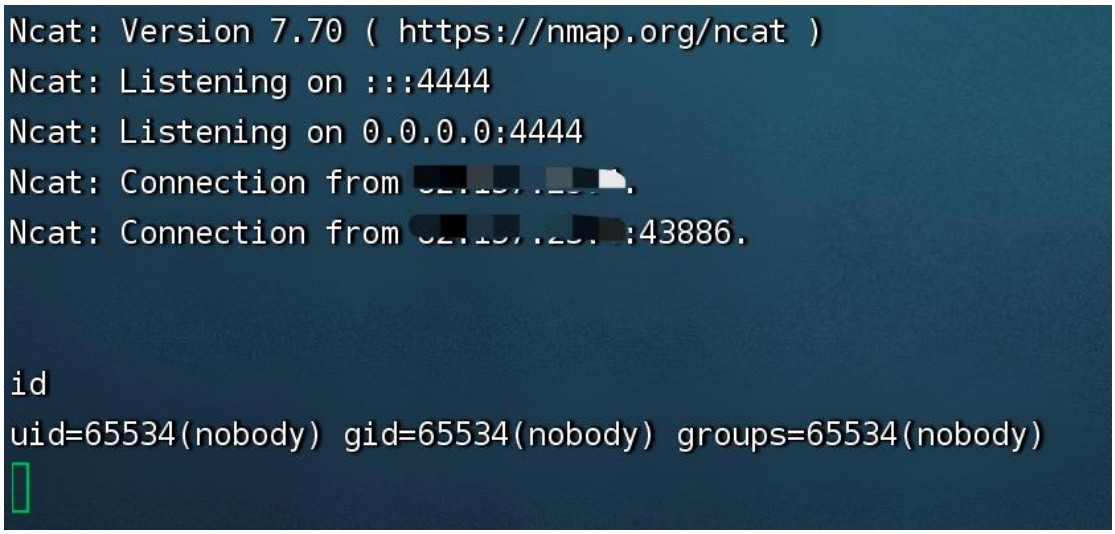
```
    if outer_headers then
        for k, v in pairs(outer_headers) do
            local is_content_header = str_find(k, "content-") == 1
            -- skip header start with "content-"
            if not req.headers[k] and not is_content_header then
                req.headers[k] = v
            end
        end
    end

    req.headers[real_ip_hdr] = core.request.get_remote_client_ip()
end
end
```

查看此时的栈数据。在 data.headers 中出现了 x-real-ip 与 X-Real-IP。

最后经由 batch-requests 插件、使用 PUT 方法将请求发送到 /apisix/admin/xxxx 注册新的路由。通过 **filter_func** 参数可写入一段 lua 代码、造成远程代码执行。

filter_func	可选	匹配规则	用户自定义的过滤函数。可以使用它来实现特殊场景的匹配要求实现。该函数默认接受一个名为 vars 的输入参数，可以用它来获取 Nginx 变量。	function(vars) return vars["arg_name"] == "json" end

漏洞测试：



利用条件

- `batch-requests` 插件默认开启状态。

- 用户使用了 Apache APISIX 默认配置（启用 Admin API ，使用默认 Admin Key 且没有额外分配管理端口），攻击者可以通过 `batch-requests` 插件调用 Admin API 。

修复意见

- 更新至最新版本
- 禁用 `batch-requests` 插件