atxsinn3r NONE
Posted 7 months ago

atxsinn3r

Multiple Stack Buffer Overflows Spotted in create_http_request_header() #51

Greetings,

There are multiple stack buffer overflows in the code that can be triggered with local user-supplied inputs. Since the inputs are local, at first glance the risk would probably be not high. However a concern is that httping could be used in consumer products, hence the local-only inputs would likely become available remotely (for example: a web wrapper that allows a remote user to provide some arguments for httping), and these overflow problems would turn remote.

The exact issues I noticed are in the create_http_request_header function, specifically:

```
char auth_string[256] = { 0 };
char b64_auth_string[512] = { 0 };

sprintf(auth_string, "%s:%s", auth_usr, auth_password);
```

And:

```
char ppa_string[256] = { 0 };
char b64_ppa_string[512] = { 0 };

sprintf(ppa_string, "%s:%s", proxy_user, proxy_password);
```

The user can supply something that is longer than what the stack buffer can handle, and overflows. For example:

```
$ ./httping -A -U user -P
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA -g http://127.0.0.1:8181

 *** -A is no longer required ***

PING 127.0.0.1:8181 (/):
=================================================================
==17975==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffc4b94aca0 at pc 0x7fc73b44c715 bp 0x7ffc4b94a920
sp 0x7ffc4b94a0b0
WRITE of size 1030 at 0x7ffc4b94aca0 thread T0
    #0 0x7fc73b44c714 in vsprintf (/lib/x86_64-linux-gnu/libasan.so.5+0x9e714)
    #1 0x7fc73b44cbce in sprintf (/lib/x86_64-linux-gnu/libasan.so.5+0x9ebce)
    #2 0x563fac3e7828 in create_http_request_header /home/tester/CLionProjects/httping/main.c:350
    #3 0x563fac3eed15 in main /home/tester/CLionProjects/httping/main.c:1689
    #4 0x7fc73b02a0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #5 0x563fac3e214d in _start (/home/tester/CLionProjects/httping/httping+0x1114d)

Address 0x7ffc4b94aca0 is located in stack of thread T0 at offset 480 in frame
    #0 0x563fac3e6f49 in create_http_request_header /home/tester/CLionProjects/httping/main.c:292

  This frame has 6 object(s):
    [32, 40) 'request' (line 294)
    [64, 192) 'pb' (line 295)
    [224, 480) 'auth_string' (line 347)
    [544, 800) 'ppa_string' (line 359) <== Memory access at offset 480 partially underflows this variable
    [864, 1376) 'b64_auth_string' (line 348) <== Memory access at offset 480 partially underflows this variable
    [1440, 1952) 'b64_ppa_string' (line 360) <== Memory access at offset 480 partially underflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/lib/x86_64-linux-gnu/libasan.so.5+0x9e714) in vsprintf
Shadow bytes around the buggy address:
  0x100009721540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100009721550: 00 00 00 00 00 00 00 00 f1 f1 f1 f1 00 f2 f2 f2
  0x100009721560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100009721570: f2 f2 f2 f2 00 00 00 00 00 00 00 00 00 00 00 00
  0x100009721580: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x100009721590: 00 00 00 00[f2]f2 f2 f2 f2 f2 f2 f2 00 00 00 00
  0x1000097215a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000097215b0: 00 00 00 00 00 00 00 00 00 00 00 00 f2 f2 f2 f2
  0x1000097215c0: f2 f2 f2 f2 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000097215d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000097215e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
```

There could be more issues but I did not look further. I'm sure there are a bunch of ways to fix this so I will just let the open source community have fun with it.