ROMHACK 21

Fuzzing Apache HTTP Server for fun (and CVEs)

Antonio Morales

// WHO I AM

#define speaker Antonio Morales

#define job Security Researcher at 

#define twitter @nosoynadiemas 

using namespace RomHack;

int main(int argc, char argv[]){*

Security Lab

https://securitylab.github.com/

Chrome, Security, Exploit

Exploiting a textbook use-after-free in Chrome

In this post I'll give details about how to exploit CVE-2020-6449, a use-after-free (UAF)
WebAudio module of Chrome that I discovered in March 2020. I'll give an outline of the g
strategy to exploit this type of UAF to achieve a sandboxed RCE in Chrome by a single c
(and perhaps a 2 minute wait) on a malicious website.

Man Yue Mo

Android, Security, Fuzzing

Structured fuzzing Android's NFC

In this post I'll give some details of how to use libprotobuf-mutator on Android to fuzz th
component.

Man Yue Mo

Security, CVE, C/C++, CodeQL

Bug Hunting with CodeQL, an Rsyslog Case Study

Follow GitHub security researcher Agustin Gianni in his bug hunting process, from
modeling to variant analysis.

Agustin Gianni

CVE, Security

CVE-2020-5398 Reflected File Download in Spring MVC/WebFlux

Learn about Reflected File Downloads by reviewing how Spring MVC and WebFlux were affected.

Alvaro Muñoz

Java, Bean Validation, Expression Language, Security

Bean Stalking: Growing Java beans into RCE

In this post I'll show how input validation which should be used to prevent malformed inputs to enter our applications, open up the doors to Remote Code Execution (RCE).

Alvaro Muñoz

Announcement, CVE, C/C++, Security

VLC Vulnerabilities Discovered by the GitHub Security Research Team

GitHub Security Lab's research team discovers 11 bugs in VLC, the popular media player. The VLC vulnerability CVE-2019-14438 could potentially allow an attacker to take control of the user's computer.

Antonio Morales

https://securitylab.github.com

What I do

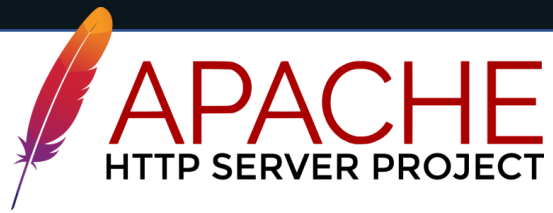
- Mainly focused on C/C++ projects
- Fuzzing enthusiast (AFL/AFL++ fan)
- Some of my work in the last year:

Talk outline

1. Apache architecture overview
2. Fuzzing workflow
3. Custom Mutators
4. Custom Interceptors
5. Vulnerabilities & Bugs

Apache architecture overview

Apache HTTP Server

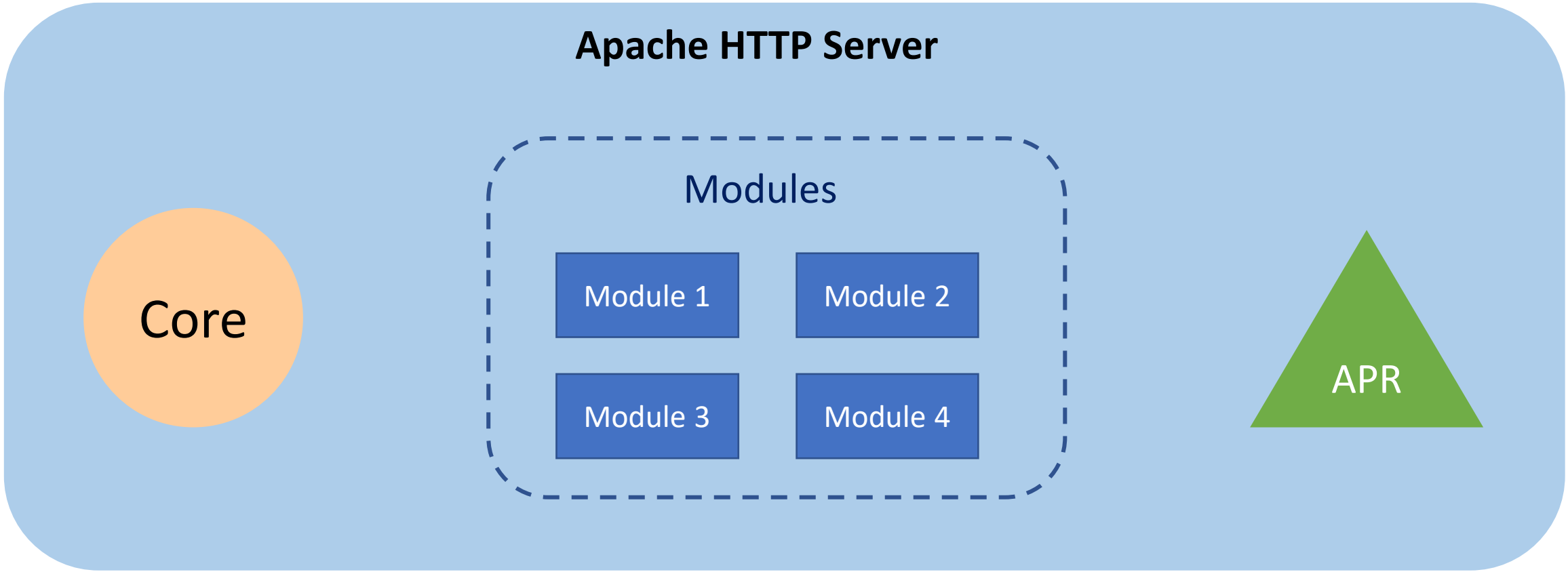


- As one of the most popular web servers out there, the Apache HTTP server doesn't need any introduction.
- Apache HTTP was one of the first HTTP servers, with development dating back to **1995**.
- Apache is free and Open Source and it's written in C
- With a market share of 26%, it is the second most used web server on the internet only behind Nginx (31%)

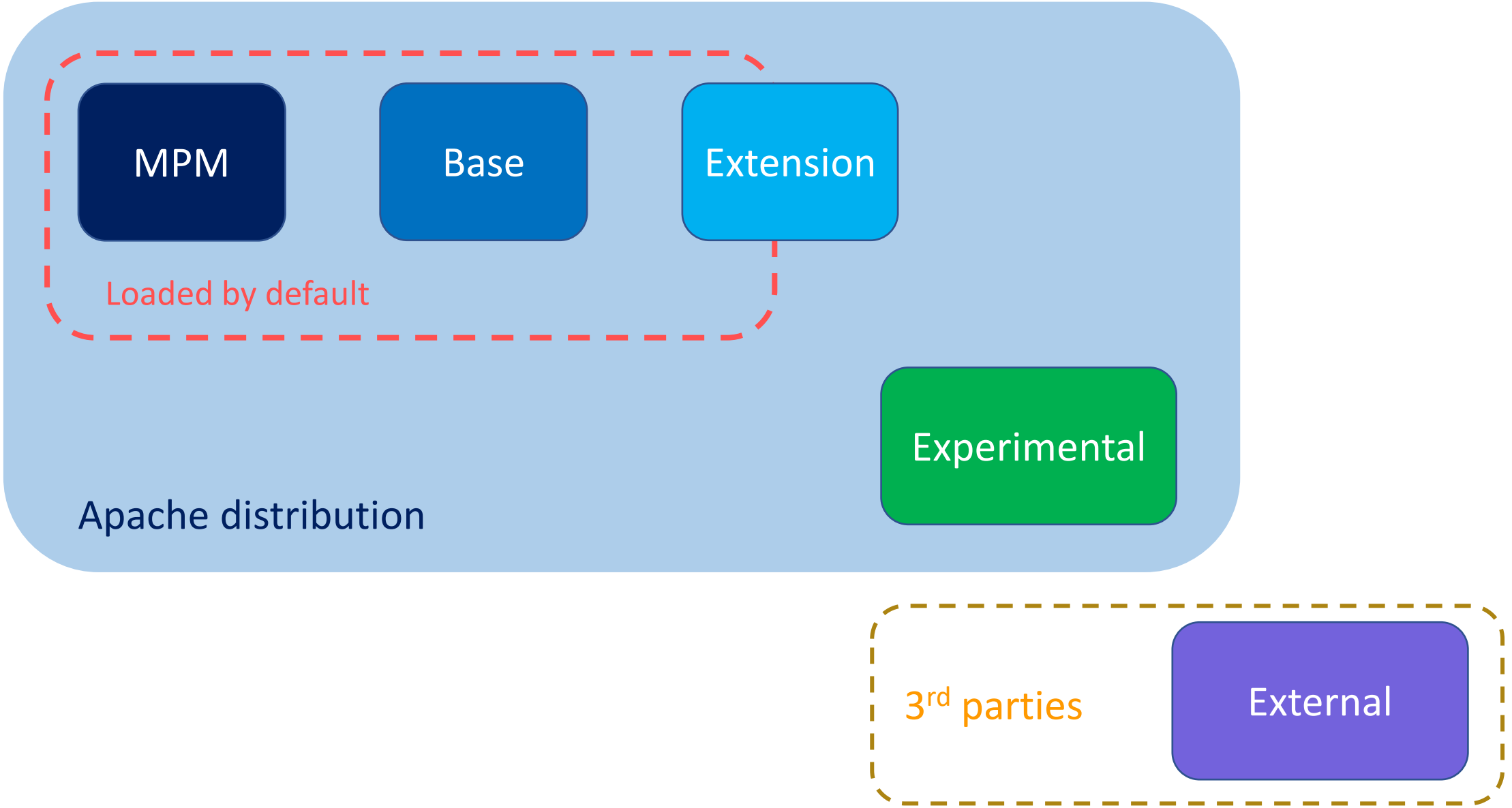
Architecture

The following information is related to the latest released version of Apache HTTP (2.4.X)

- Apache's architecture is composed of 3 main parts:



Modules


MPM
Base
Extension
Loaded by default
Experimental
Apache distribution
3rd parties
External

Multi-processing modules (MPM)

- The MPM provides the interface between the running Apache server and the underlying operating system.

- MPM handle important core tasks such as:
  - Creating network connections (binding to ports)
  - Listening for / accepting requests from clients (browsers)
  - Creating child processes or threads to handle the requests

- It also allows different Apache models to coexist even within a single operating system, thus providing users with options for different usages.

- You can only have one MPM loaded at the time

Multi-processing modules (MPM)

3 main MPM modules on Unix systems:

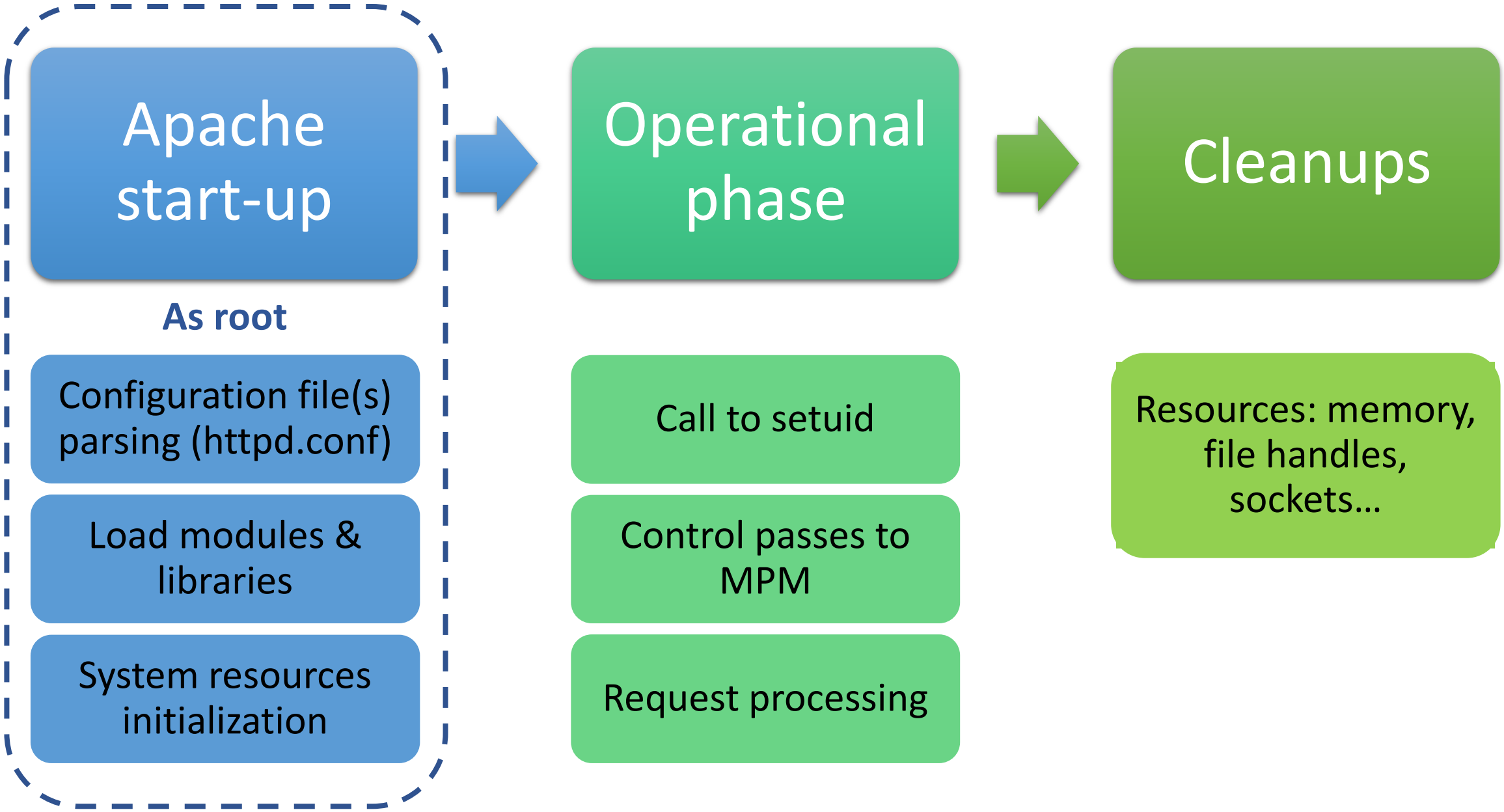
- **Prefork**:
  - Implements a non-threaded, pre-forking web server (similar to Apache 1.X).
  - Each server process in the server pool may answer incoming requests.
  - The best choice for isolating each request, so that a problem with a single request will not affect any other.
- **Worker**:
  - Implements a hybrid multi-process multi-threaded server.
  - Each child process creates a fixed number of server threads.
  - By using threads, it can serve more requests with fewer system resources than Prefork
- **Event**:
  - An asynchronous, event-based implementation.
  - It can decouple the server thread from the connection
  - Can handle more requests simultaneously. Ideal for busy servers (>1000 req/s)

Apache Portable Runtime (APR)

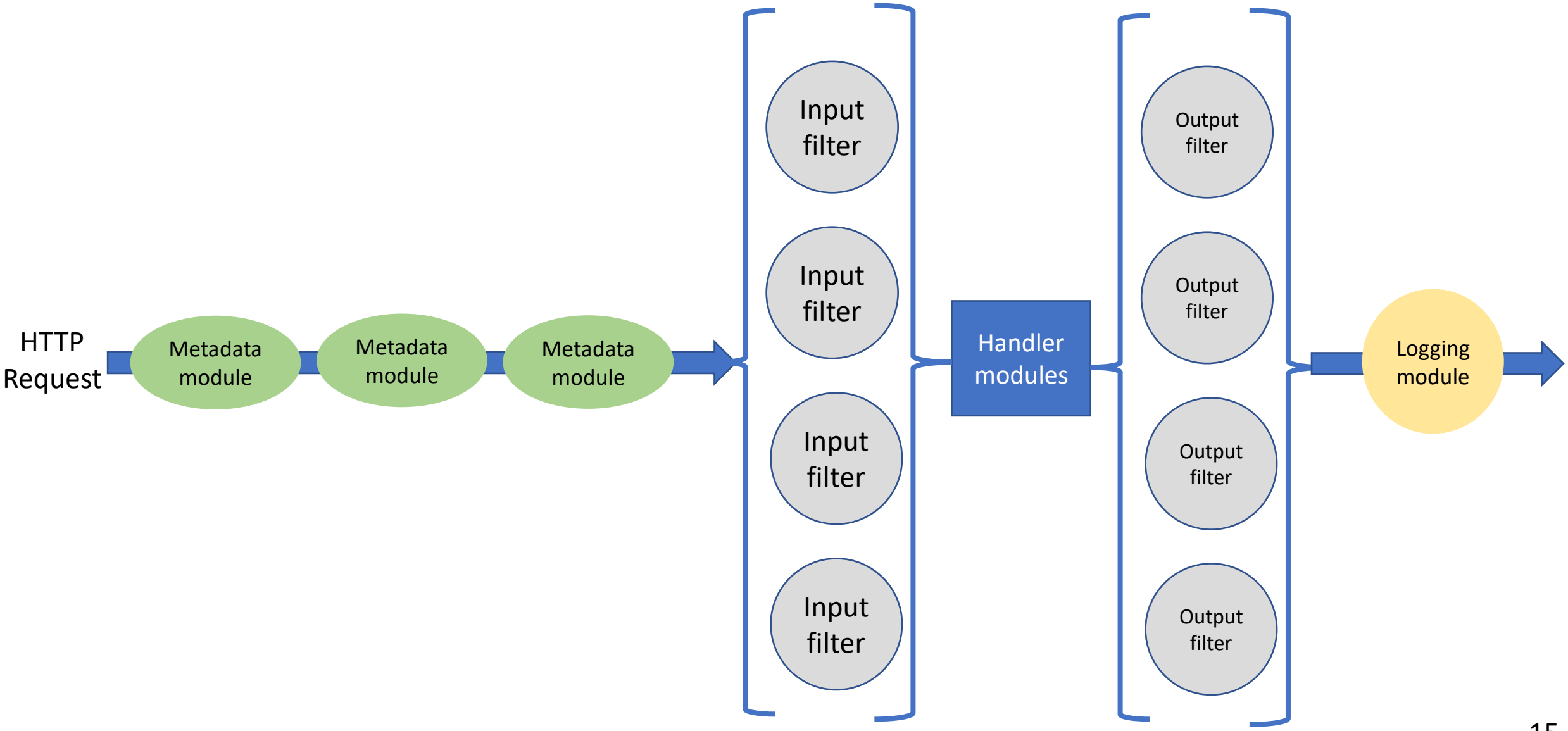
Apache Portable Runtime

- APR is a supporting library for Apache HTTP
- It provides a set of independent-platform APIs that map to the underlying operating system so, in that way programmers can use the APR to make a program portable across platforms
- APR includes the following capabilities:
  - Memory pools & memory allocation
  - File I/O
  - Shared memory
  - Multi-threading & mutex
  - Time routines

Apache Operation



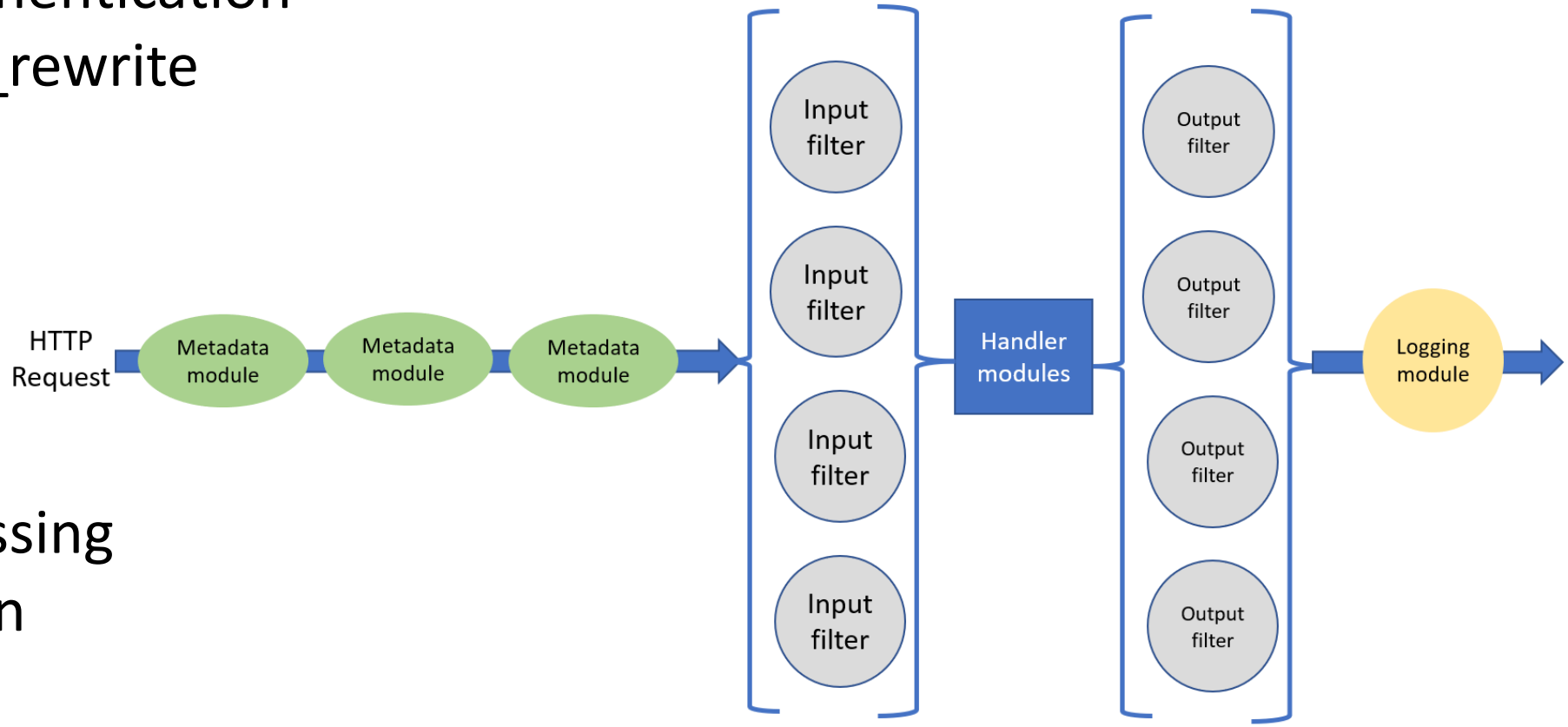
Request processing



Request processing

Metadata modules examples:

- URL pre-parsing for choosing a content generator
- HTTP access/authentication
- mod_alias/mod_rewrite

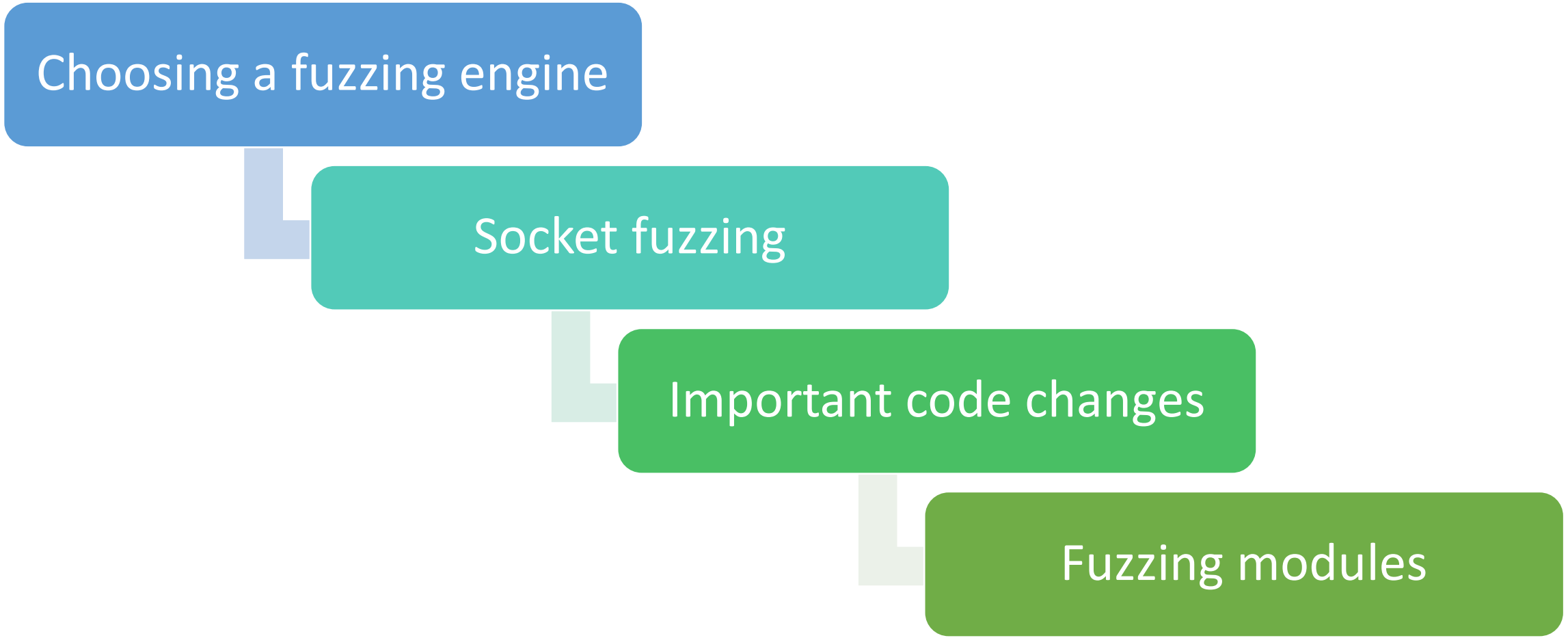


Filters examples:

- XML/XSLT processing
- Gzip compression
- Encryption (SSL)
- Server-side includes (SSI)

Fuzzing workflow

Fuzzing workflow



Choosing a fuzzing engine

AFL++

- AFL++ is an AFL fork, that includes a lot of changes, optimizations and new features in comparison to the original AFL

- Today, AFL++ may be considered as a **fuzzing framework** rather than a fuzzing tool:
  - Includes some external custom-mutators (ex. MOPt and Radamsa) and supports C/C++ and Python custom mutators
  - Multiple instrumentation modes
  - Integrates AFLfast's power schedules

Fuzzing sockets

- The AFL fuzzing process is usually straightforward when the input is file based (image libraries such as libpng, libjpg).
- In these cases, few or no changes to the targeted source code are required.
- However, when dealing with networked, interactive servers (such as HTTP servers), the process is not that simple.
- In that case, my approach was to add a code snippet to the MPM function that create a connection to the web server itself and sent the fuzzing input through that.

Fuzzing sockets

```
//MYCHANGE
if(selfconnect("127.0.0.1", currentPort, 'p') == -1)
    continue;
//-----------------------------------
```

```
tmp = connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
if (tmp < 0){
    printf("\nConnection Failed \n");
    perror("Error: ");
    return -1;
}

if(send(sock, filedata, filesize, 0) < 0){
    printf("\nSend Failed \n");
    perror("Error: ");
    return -1;
}

if(shutdown(sock, SHUT_WR) < 0){
    printf("\nSend Failed \n");
    perror("Error: ");
    return -1;
}
```

Fuzzing sockets

- In these cases, few or no changes to the targeted source code are required.
- However, when dealing with networked, interactive servers (such as HTTP servers), the process is not that simple.
- In that case, my approach was to add a code snippet to the MPM function that create a connection to the web server itself and sent the fuzzing input through that.
- **2 fuzzing configurations**:
  - Prefork MPM: 80% of the time of fuzzing; 1 process at a time
  - *Event MPM*: For detecting concurrency issues

Important code changes

To address some limitations, I have made some changes to the Apache code. Let's see some of these changes:

- Changes aimed at **reducing entropy**: Ex. Replacing "random" and "rand" by constant seeds:

```
5652  5655        rng = apr_random_standard_new(p);
5653  5656        do {
5654        -         rv = apr_generate_random_bytes(seed, sizeof(seed));
      5657  +       //MYCHANGE
      5658  +         //rv = apr_generate_random_bytes(seed, sizeof(seed));
      5659  +       char constant_seed[] = {0x78,0xAB,0xF5,0xDB,0xE2,0x7F,0xD2,0x8A};
      5660  +         memcpy(seed, constant_seed, sizeof(seed));
      5661  +         rv = APR_SUCCESS;
      5662  +         //---------------------------------------------
```

Important code changes

- Changes aimed at **reducing delays**: Ex. Removing some of the “sleep()” and “select()” calls:

```
APR_DECLARE(void) apr_sleep(apr_interval_time_t t)
{
#ifdef OS2
    DosSleep(t/1000);
#elif defined(BEOS)
    snooze(t);
#elif defined(NETWARE)
    delay(t/1000);
#else
    struct timeval tv;
    tv.tv_usec = t % APR_USEC_PER_SEC;
    tv.tv_sec = t / APR_USEC_PER_SEC;
    //MYCHANGE
    //select(0, NULL, NULL, NULL, &tv);
    //-----------------------------------
```

Important code changes

- Changes in **crypto routines**: Ex. disabling checksums

```
-                   if (ctx->crc != compCRC) {
+                   //MYCHANGE
+                   if(1==0){
+                   //if (ctx->crc != compCRC) {
                        inflateEnd(&ctx->stream);
                        ap_log_rerror(APLOG_MARK, APLOG_WARNING, 0, r, APLOGNO(01394)
                                      "Zlib: CRC error inflating data");
                        return APR_EGENERAL;
                    }
                    compLen = getLong(buf + VALIDATION_SIZE / 2);
                    /* gzip stores original size only as 4 byte value */
-                   if ((ctx->stream.total_out & 0xFFFFFFFF) != compLen) {
+                   //MYCHANGE
+                   if(1==0){
+                   //if ((ctx->stream.total_out & 0xFFFFFFFF) != compLen) {
                        inflateEnd(&ctx->stream);
                        ap_log_rerror(APLOG_MARK, APLOG_WARNING, 0, r, APLOGNO(01395)
                                      "Zlib: Length %" APR_UINT64_T_FMT
```

Fuzzing modules

- The main configuration file is usually called **httpd.conf** and it contains one directive per line

- I followed an **incremental approach**:
  1. I started with a small set of modules enabled **(--enable-mods-static=few**)
  2. After reaching a stable fuzzing workflow, I enabled a new module and tested the fuzzing stability again.
  3. Repeat 2

```
<Location "/p">
    SetInputFilter DEFLATE
</Location>

<Location "/pp">
    SetOutputFilter INFLATE
</Location>

<Location "/ppp">
    SetOutputFilter BROTLI_COMPRESS
</Location>

CryptoDriver openssl

<Location "/j">
    CryptoKey file:./conf/file.key
    SetHandler crypto-key
    SetOutputFilter ENCRYPT
</Location>
```

Custom Mutators

Choosing the right mutator

What's the problem?

- By default AFL/Libfuzzer implements basic mutators.

- **AFL/Libfuzzer generic mutators**:
  - Bit flipping
  - Byte increments/decrements
  - Simple Arithmetics
  - Known integers
  - Block deletion/merge/duplication/memset
  - Dictionaries

Choosing the right mutator

What's the problem?

- **Pros**: These mutators provide overall good results, especially in binary formats

- **Cons**: They have limited success when applied to text-based and/or complex formats, such as HTTP protocol

Solution: Create some additional **custom mutators** specifically for the task of fuzzing the HTTP protocol

You can find the code of all the following custom mutators on https://github.com/antonio-morales/Apache-HTTP-Fuzzing inside the "Custom Mutators" folder

Custom mutators

- **Piece swapping** (between all input corpus files):
  - **Line swapping**: Swap lines of two different HTTP requests

Before

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: 127.0.0.1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
1 POST /cgi-bin/process.cgi HTTP/1.1
2 User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
3 Host: 127.0.0.1
4 Content-Type: application/x-www-form-urlencoded
5 Content-Length: length
6 Accept-Language: en-us
7 Accept-Encoding: gzip, deflate
8 Connection: Keep-Alive
```

After

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
1 POST /cgi-bin/process.cgi HTTP/1.1
2 User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
3 Host: 127.0.0.1
4 Accept-Language: en-us
5 Content-Length: length
6 Accept-Language: en-us
7 Accept-Encoding: gzip, deflate
8 Connection: Keep-Alive
```

Custom mutators

- **Piece swapping** (between all input corpus files):
  - **Word swapping**: Swap words of two different HTTP requests

Before

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
1  POST /cgi-bin/process.cgi HTTP/1.1
2  User-Agent: Mozilla/4.0 (compatible
```

After

```
GET /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
1  POST /hello.htm HTTP/1.1
2  User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
3  Host: 127.0.0.1
4  Accept-Language: en-us
5  Content-Length: length
6  Accept-Language: en-us
7  Accept-Encoding: gzip, deflate
8  Connection: Keep-Alive
```

Custom mutators

- **Charsets bruteforce** [0-9][a-z][A-Z][:;,./<>=@%]
- Based on **hashcat** rules approach
- Some examples:
  - 1-byte bruteforce: 0x00 – 0xFF
  - 2 bytes bruteforce: 0x0000 – 0xFFFF
  - 3 letters bruteforce: [a-z]{3}
  - 4 digits bruteforce: [0-9]{4}
  - 3 letters & numbers bruteforce: ([a-z][0-9){3}
  - 3bytes / 4 bytes strings bruteforce: it does bruteforce using all 3/4 bytes strings in the input file.

Custom grammar

- In addition to using custom mutators I used a **custom grammar** for fuzzing HTTP.

- For this, I used a tool that was recently added to AFL++: **Grammar-Mutator**.

- The grammar file is specified in JSON and is shown as a collection of key/value pairs.

Custom grammar

- In my case, I created a simplified HTTP grammar specification:

```
"<A>": [["<START_LINE>", "\r\n", "<HEADERS>", "<BODY>", "\r\n\r\n"]],

"<START_LINE>": [["<METHOD>", " ", "<URI>", " ", "<VERSION>"]],

"<METHOD>": [["GET"], ["HEAD"], ["POST"], ["PUT"], ["DELETE"], ["CONNECT"],
], ["COPY"], ["LABEL"], ["LINK"], ["LOCK"], ["MERGE"], ["MKACTIVITY"], ["MKC
"PROPFIND"], ["PROPPATCH"], ["REBIND"], ["REPORT"], ["SEARCH"], ["UNBIND"],

"<URI>": [["<SCHEME>" , ":", "<HIER>", "<QUERY>", "<FRAGMENT>"]],

"<SCHEME>": [["http"], ["https"], ["shttp"], ["dav"], ["about"], ["attachmen

"<HIER>": [["//", "<AUTHORITY>", "<PATH>"]],

"<AUTHORITY>": [["<USERINFO>", "<HOST>"]],

"<PATH>": [["/", "<DIR>"]],

"<DIR>": [[], ["<CHAR>", "/", "<DIR>"]],
```

Custom Interceptors

Motivation

But why do we need to implement custom interceptors?

Motivation

- As previously noted, Apache uses custom memory pools to improve management of program-dynamic memory.

- That is why, if we want to allocate memory inside a memory pool, we should invoke ***apr_palloc*** instead of ***malloc***.

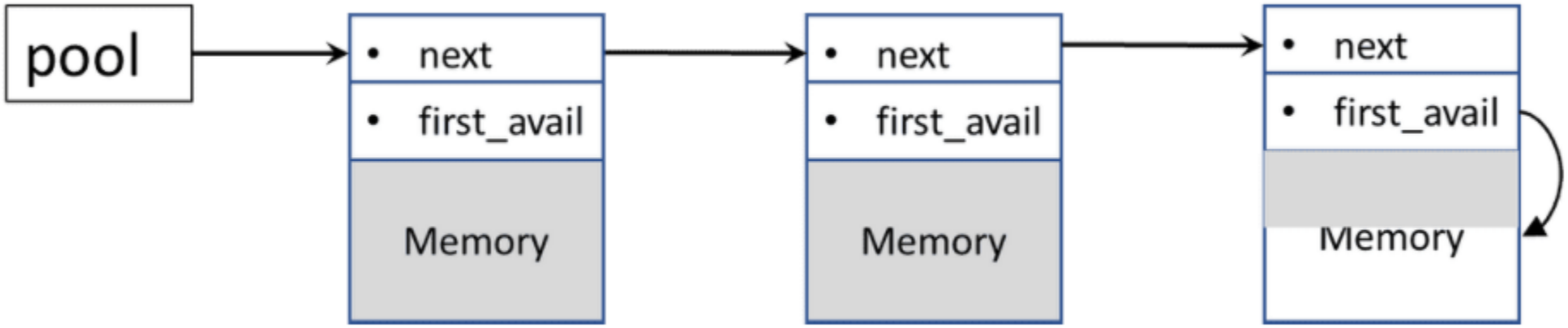
- Consider the following code snippet, where a call to *apr_palloc* was made in order to allocate memory:

```
if (p->pool_size < p->bytes+1) {
    unsigned char *np = apr_palloc(g->apr_pool,(p->bytes+1)*2);
```

- In this case, the value of the second argument is 126 (in_size = 126) or, in other words, **our aim is to allocate 126 bytes inside the g->apr_pool memory pool**.

Motivation

- Apache HTTP memory pools consist of a linked-list of memory nodes as follows:



- The program will add new nodes to this linked-list as additional space is needed.

- When the free space of a node is not enough to meet *apr_palloc* demand, then ***allocator_alloc*** function is called: it will create a new node and add it to the linked list.

Motivation

```
apr_memnode_t *allocator_alloc(apr_allocator_t *allocator, apr_size_t in_size)
{
    apr_memnode_t *node, **ref;
    apr_size_t max_index, upper_index;
    apr_size_t size, i, index;

    /* Round up the block size to the next boundary, but always
     * allocate at least a certain size (MIN_ALLOC).
     */
    size = allocator_align(in_size);
    if (!size) {
        return NULL;
    }
```

- But, as you can see here, this allocation size is always rounded up to ***MIN_ALLOC = 8192 bytes***, and that will be a problem for **ASan**.

Motivation

- So, we find ourselves facing a scenario in which we made a call to ***apr_palloc*** with size = 126…

```
APR_DECLARE(void *) apr_palloc(apr_pool_t *pool, apr_size_t in_size)
{
    apr_memnode_t *active
    void *mem;
```

Expression	Type	Value
in_size	apr_size_t	126

- …but ASAN has poisoned a memory area of size **8192**:

```
140 INTERCEPTOR(void*, malloc, uptr size) {
141   if (UNLIKELY(UseLocalPoo
142     // Hack: dlsym calls m
143     return AllocateFromLoc
144   ENSURE_ASAN_INITED();
```

Expression	Type	Value
size	__sanitizer::uptr	8192

Motivation

- The end result is a total of **8192-126 = 8066 bytes** marked as writable by ASAN when it is not real allocated memory but rather free space in the node.
- So, a memcpy(np, source, 5000) call would lead to an out-of-bound write, overwriting the rest of the node's memory.
- However, we wouldn't see any ASAN alert message, which would cause us to miss memory corruption bugs even with ASAN enabled.
- Errors like these can result in vulnerabilities such as the one I published a year ago in ProFTPD: CVE-2020-9273.

Custom ASan interceptors

- To solve this, we can define our own **custom ASan interceptors**.
- In this way, we'll be able to catch these type of bugs in the Apache memory pools.
- To define our own ASAN Interceptors, we need to perform the following steps:
  - Define *INTERCEPTOR(int, foo, const char *bar, double baz) { ... }*, where ***foo*** is the name of the function we want to intercept
  - Call *ASAN_INTERCEPT_FUNC (foo)* prior to the first call of function *foo*

apr_palloc example

```
INTERCEPTOR(void*, apr_palloc, void *pool, SIZE_T in_size){

    ENSURE_ASAN_INITED();
    GET_STACK_TRACE_MALLOC;

    void* libc_addr = REAL(apr_palloc)(pool, in_size);

    /** APR alignment */
    #define APR_ALIGN(size, boundary) \
        (((size) + ((boundary) - 1)) & ~((boundary) - 1))

    #define APR_ALIGN_DEFAULT(size) APR_ALIGN(size, 8)

    SIZE_T size = APR_ALIGN_DEFAULT(in_size);

    void* asan_addr = asan_malloc(in_size, &stack);

    //We need to link libc addresses to asan addresses to free-ing them later
    addr[size][0] = libc_addr;
    addr[size][1] = asan_addr;
    size++;

    return asan_addr;
}
```

apr_palloc example

```
static APR_INLINE
void allocator_free(apr_allocator_t *allocator, apr_memnode_t *node
{
        freelist = node->next;
#if APR_ALLOCATOR_USES_MMAP
        munmap((char *)node - GUARDPAGE_SIZE,
               2 * GUARDPAGE_SIZE + ((node->index+1) << BOUNDARY_I
#else
        //free(node);

        //Free asan-malloced addresses
        for(int i=0; i<addr_size; i++){

            if(addr[i][0] >= node &&  addr[i][0] <= node->endp){

                free(addr[i][1]);
                addr[i][0] = NULL;
                addr[i][1] = NULL;
            }
        }

        __libc_free(node);
```

Vulnerabilities & Bugs

NULL dereference in session_identity_decode

- The bug is triggered when you set a cookie with a null key and value.

```
GET /1 HTTP/1.1
Host: foo.example
Accept: */*
Cookie: session=choco&admin-user=2&=; min-pw=3;
```

- The *session_identity_decode* function performs a call to *apr_strtok*(split a string into separate tokens) with *pair* as its 1st argument.

```
414         const char *psep = "=";
415         char *key = apr_strtok(pair, psep, &plast);
416         char *val = apr_strtok(NULL, psep, &plast);
417         if (key && *key) {
```

- This argument represents the value of **cookie key-value pair**

NULL dereference in session_identity_decode

- So, if this cookie pair is NULL, then str = 0 and a null dereference will be happen

```
27 APR_DECLARE(char *) apr_strtok(char *str, const char *sep, char **last)
28 {
29     char *token;
30
31     if (!str)           /* subsequent call */
32         str = *last;    /* start where we left off */
33
34     /* skip characters in sep (will terminate at '\0') */
35     while (*str && strchr(sep, *str))
36         ++str;
37
38     if (!*str)          /* no more tokens */
39         return NULL;
```

- Mod_session enabled
- May lead to DoS (child level)

Off-by-one (stack-based) in check_nonce

- Mod_auth_digest enabled and digest authentication
- The bug is triggered when you assign a specific set of values to the nonce field:

```
GET http://127.0.0.1/i?proxy=yes HTTP/1.1
Host: foo.example
Accept: */*
Authorization: Digest username="2",
                      realm="private area",

                      nonce="d2hhdGFzdXJwcmlzZXhkeGR4ZHhkeGR4ZHhkeGR4ZHhkeGR4ZA==",
                      uri="http://127.0.0.1:80/i?proxy=yes",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="53849ce65ba787cd0a07a272ece3bba6",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Off-by-one (stack-based) in check_nonce

- The *check_nonce* function performs a call to

  apr_base64_decode_binary(nonce_time.arr, resp->nonce)

  where *nonce_time.arr* is a local array of size **8**.

```
APR_DECLARE(int) apr_base64_decode_binary(unsigned char *bufplain,
                                          const char *bufcoded)
{
    int nbytesdecoded;
    register const unsigned char *bufin;
    register unsigned char *bufout;
    register apr_size_t nprbytes;

    bufin = (const unsigned char *) bufcoded;
    while (pr2six[*(bufin++)] <= 63);
    nprbytes = (bufin - (const unsigned char *) bufcoded) - 1;
    nbytesdecoded = (((int)nprbytes + 3) / 4) * 3;

    bufout = (unsigned char *) bufplain;
```

- But *(((int)nprbytes + 3) / 4) * 3* calculation is wrong, giving **9** as a result

Off-by-one (stack-based) in check_nonce

```
APR_DECLARE(int) apr_base64_decode_binary(unsigned char *bufplain,
                                          const char *bufcoded)
{
    int nbytesdecoded;
    register const unsigned char *bufin;
    register unsigned char *bufout;
    register apr_size_t nprbytes;

    bufin = (const unsigned char *) bufcoded;
    while (pr2six[*(bufin++)] <= 63);
    nprbytes = (bufin - (const unsigned char *) bufcoded) - 1;
    nbytesdecoded = (((int)nprbytes + 3) / 4) * 3;

    bufout = (unsigned char *) bufplain;
```

- So, 9 bytes will be written into bufpoint
- As a result, the program is writing 1 byte outside of the boundaries of the local array nonce_time.arr, overwriting 1 byte in the program stack (aka Off-by-one).

Use-After-Free in cleanup_tables

- The *cleanup_tables* function performs a call to *apr_rmm_destroy(client_rmm)*, but *client_rmm* address has been previously freed (in *apr_allocator_destroy)*.

```
static apr_status_t cleanup_tables(void *not_used)
{
    ap_log_error(APLOG_MARK, APLOG_INFO, 0, NULL, APLOGNO(01756)
                 "cleaning up shared memory");

    if (client_rmm) {
        apr_rmm_destroy(client_rmm);
        client_rmm = NULL;
    }

    if (client_shm) {
        apr_shm_destroy(client_shm);
        client_shm = NULL;
    }
```

- Only in ONE_PROCESS mode

OOB-write (heap-based) in ap_escape_quotes

- OOB write in *ap_escape_quotes* (given a string, replace any bare " with \“)
- Calculation mismatch between input string length and malloced *outstring* size

```
while (*inchr != '\0') {
    newlen++;
    if (*inchr == '"') {
        newlen++;
    }
    /*
     * If we find a slosh, and it's not the last byte in the string,
     * it's escaping something - advance past both bytes.
     */
    if ((*inchr == '\\') && (inchr[1] != '\0')) {
        inchr++;
        newlen++;
    }
    inchr++;
}
outstring = apr_palloc(p, newlen + 1);
```

OOB-write (heap-based) in ap_escape_quotes

```
/*
 * Now copy the input string to the output string, inserting a slosh
 * in front of every " that doesn't already have one.
 */
while (*inchr != '\0') {
    if ((*inchr == '\\') && (inchr[1] != '\0')) {
        *outchr++ = *inchr++;
        *outchr++ = *inchr++;
    }
    if (*inchr == '"') {
        *outchr++ = '\\';
    }
    if (*inchr != '\0') {
        *outchr++ = *inchr++;
    }
}
*outchr = '\0';
return outstring;
```

- As a result, it's possible to write out of the *outchr* array (1 byte PoC)
- **Bug collision** (previously reported by Google OSSFuzz) ☹

Race condition leading to UAF

- During my fuzzing work, I found multiple non-reproducible UAF crashes.
- This seems to be the result of a race condition between calls to a*pr_allocator_destroy* and *allocator_alloc .* These functions **might not be thread safe** in concurrent scenarios.
- Seems like in certain circumstances the APR memory pool can be corrupted. That's why the program try to release a block that's already present in the pool as a free block.
- This bug shares some similarities with the bug I reported in ProFTPD (CVE-2020-9273).
- Only happens when MPM = Event
- At this moment, I've not been able to reproduce the bug **in a deterministic way**.

Race condition leading to UAF

- I've a lot of different Asan reports:

```
=================================================================
==106820==ERROR: AddressSanitizer: heap-use-after-free on address 0x625000091100 at pc 0x7ffff7d2ff4d bp 0x7fffffffd800
sp 0x7fffffffd7f8
READ of size 8 at 0x625000091100 thread T0
    #0 0x7ffff7d2ff4c in apr_allocator_destroy /home/antonio/Downloads/httpd-
trunk/srclib/apr/memory/unix/apr_pools.c:197:26
    #1 0x7ffff7d3306c in apr_pool_terminate /home/antonio/Downloads/httpd-
trunk/srclib/apr/memory/unix/apr_pools.c:756:5
    #2 0x7ffff77aeba6 in __run_exit_handlers /build/glibc-5mDdLG/glibc-2.30/stdlib/exit.c:108:8
    #3 0x7ffff77aed5f in exit /build/glibc-5mDdLG/glibc-2.30/stdlib/exit.c:139:3
    #4 0x5b1ae8 in clean_child_exit /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:777:5
    #5 0x5b19a5 in child_main /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:2957:5
    #6 0x5afa7b in make_child /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:2981:9
    #7 0x5af005 in startup_children /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:3046:13
    #8 0x5a74c1 in event_run /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:3407:9
    #9 0x6212b1 in ap_run_mpm /home/antonio/Downloads/httpd-trunk/server/mpm_common.c:100:1
    #10 0x5e67e6 in main /home/antonio/Downloads/httpd-trunk/server/main.c:891:14
    #11 0x7ffff778c1e2 in __libc_start_main /build/glibc-5mDdLG/glibc-2.30/csu/../csu/libc-start.c:308:16
    #12 0x44da7d in _start ??:0:0
```

Race condition leading to UAF

- I've a lot of different Asan reports:

```
=================================================================
==42672==ERROR: AddressSanitizer: heap-use-after-free on address 0x6250006569c8 at pc 0x7ffff7d58e56 bp 0x7fffffffd7e0 sp 0x7fffffffd7d8
READ of size 8 at 0x6250006569c8 thread T0
    #0 0x7ffff7d58e55 in apr_socket_timeout_set /home/antonio/Downloads/httpd-trunk/srclib/apr/network_io/unix/sockopt.c:86:25
    #1 0x59c065 in abort_socket_nonblocking /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:558:5
    #2 0x591e16 in close_worker_sockets /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:589:13
    #3 0x591e16 in signal_threads /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:668:9
    #4 0x590321 in child_main /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:0:0
    #5 0x58edc9 in make_child /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:2988:9
    #6 0x58e355 in startup_children /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:3053:13
    #7 0x58743b in event_run /home/antonio/Downloads/httpd-trunk/server/mpm/event/event.c:3414:9
    #8 0x5fcd11 in ap_run_mpm /home/antonio/Downloads/httpd-trunk/server/mpm_common.c:100:1
    #9 0x5c23b6 in main /home/antonio/Downloads/httpd-trunk/server/main.c:891:14
    #10 0x7ffff778c1e2 in __libc_start_main /build/glibc-5mDdLG/glibc-2.30/csu/../csu/libc-start.c:308:16
    #11 0x44da7d in _start ??:0:0
```

Race condition leading to UAF

- This is not a new problem. It has been going on for many years.

- I found that **Hanno Böck (@hanno)** has been reporting similar type of issues since June 2018. You can check Hanno previous reports in https://github.com/hannob/apache-uaf

- For now, the position of Apache security team is to not consider these security issues unless we can show a practical exploit.

Minor bugs

- Other minor bugs have been found during the fuzzing process.
- As example we can mention this **integer overflow in session_identity_decode**.
- To reproduce it you only need to send a LOCK request with a long "Timeout" value:

```
LOCK /dav/c HTTP/1.1
Host: 127.0.0.1
Timeout: Second-410000000004100000000
Content-Type: text/xml; charset="utf-8"
Content-Length: XXX
Authorization: Basic Mjoz

<?xml version="1.0" encoding="utf-8" ?>
<d:lockinfo xmlns:d="DAV:">
```

Minor bugs

```
while ((val = ap_getword_white(r->pool, &timeout))
    if (!strncmp(val, "Infinite", 8)) {
        return DAV_TIMEOUT_INFINITE;
    }

    if (!strncmp(val, "Second-", 7)) {
        val += 7;
        /* ### We need to handle overflow better:
         * ### timeout will be <= 2^32 - 1
         */
        expires = atol(val);
        now     = time(NULL);
        return now + expires;
    }
}
```

- **Mod_dav enabled**

One last thought

One last thought

- Apache HTTP security has been extensively studied by researchers.
- But there is still room for discovering new vulnerabilities (ex. through fuzzing).
- In the coming weeks I'll release new further details regarding Apache HTTP security, including new bugs. So stay tuned!

Thanks!

Antonio Morales

Twitter: @nosoynadiemas
GitHub: @antonio-morales
Email: antonio-morales@github.com