IT'S NOT SAFE ON THE STREETS... ESPECIALLY FOR YOUR 3DS!

EXPLORING A NEW ATTACK SURFACE ON THE 3DS

@MrNbaYoh

STATE OF 3DS HACKING

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)
- bootroms dumped (33c3)

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)
- bootroms dumped (33c3)
- derive secret AES keys as a result

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)
- bootroms dumped (33c3)
- derive secret AES keys as a result
- permanent unpatchable bootROM exploit (33c3 & 33.5c3)

STATE OF 3DS HACKING

- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)
- bootroms dumped (33c3)
- derive secret AES keys as a result
- permanent unpatchable bootROM exploit (33c3 & 33.5c3)

What is left unexplored?

STATE OF 3DS HACKING

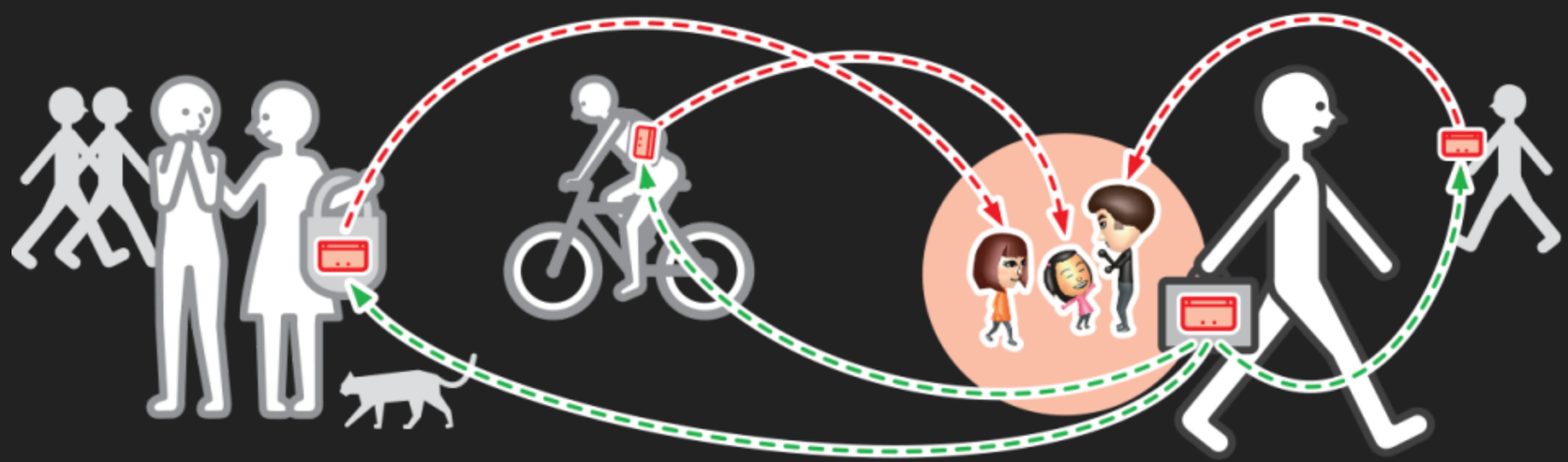
- many userland exploits, patched kernel flaws and lots of documentation
- hardware keyscrambler broken (32c3)
- bootroms dumped (33c3)
- derive secret AES keys as a result
- permanent unpatchable bootROM exploit (33c3 & 33.5c3)

What is left unexplored?

Could we use these keys to attack features that were protected until then?

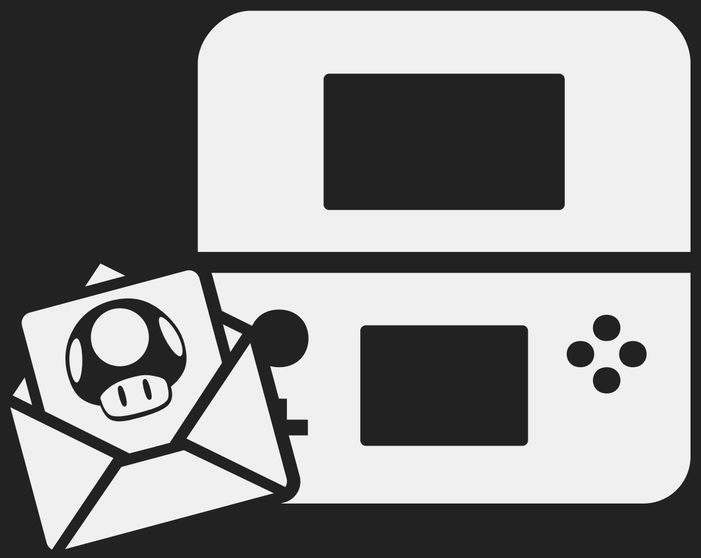
StreetPass

WHAT IS *StreetPass*?



- local wireless communication feature
- automatically communicates with nearby 3DS systems
- allows applications to exchange data (custom levels, messages, Miis, ...)

StreetPass FOR PLAYERS



StreetPass FOR PLAYERS

StreetPass FOR PLAYERS

StreetPass FOR PLAYERS

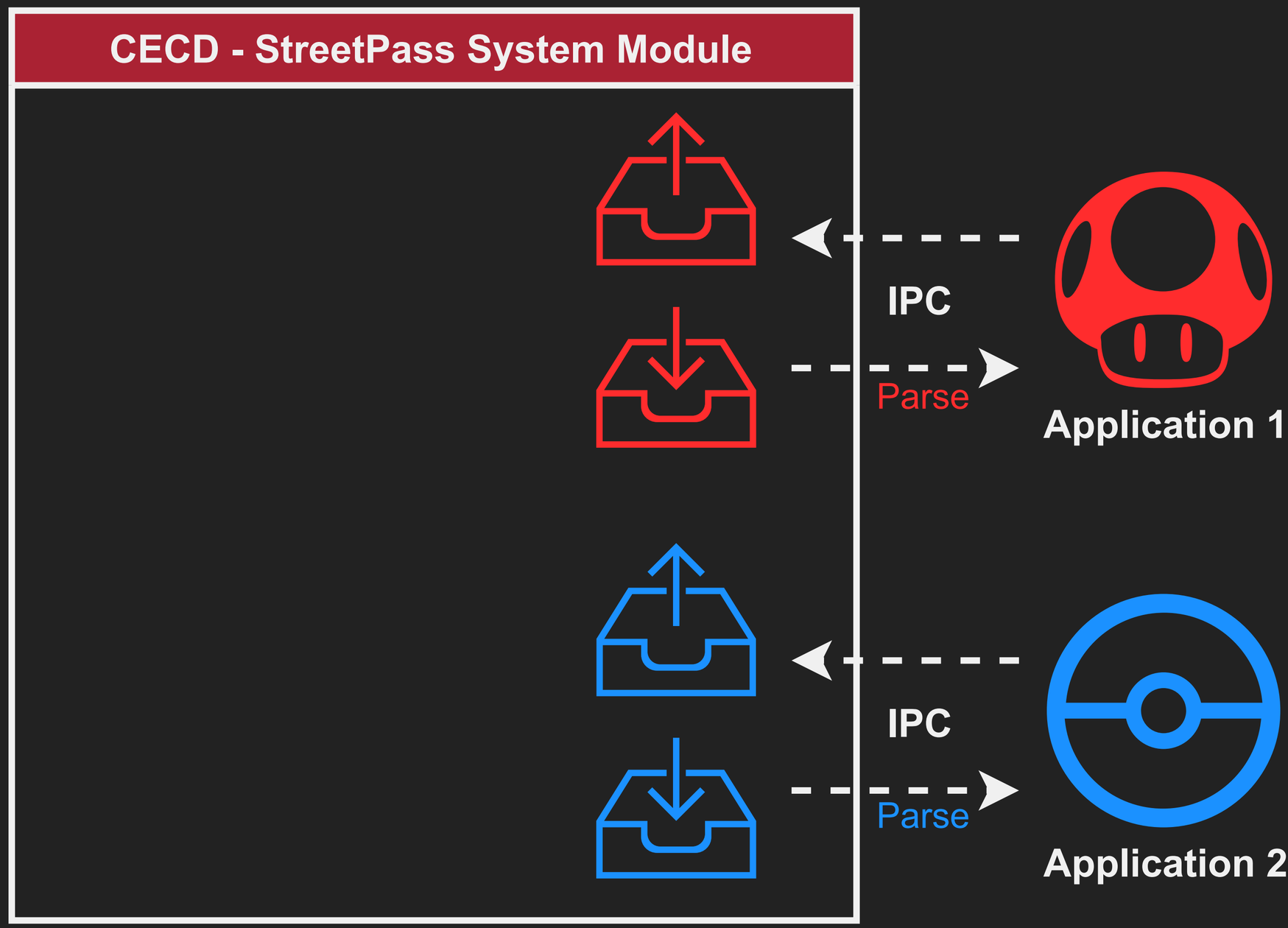
HACKER POINT OF VIEW...

HACKER POINT OF VIEW...

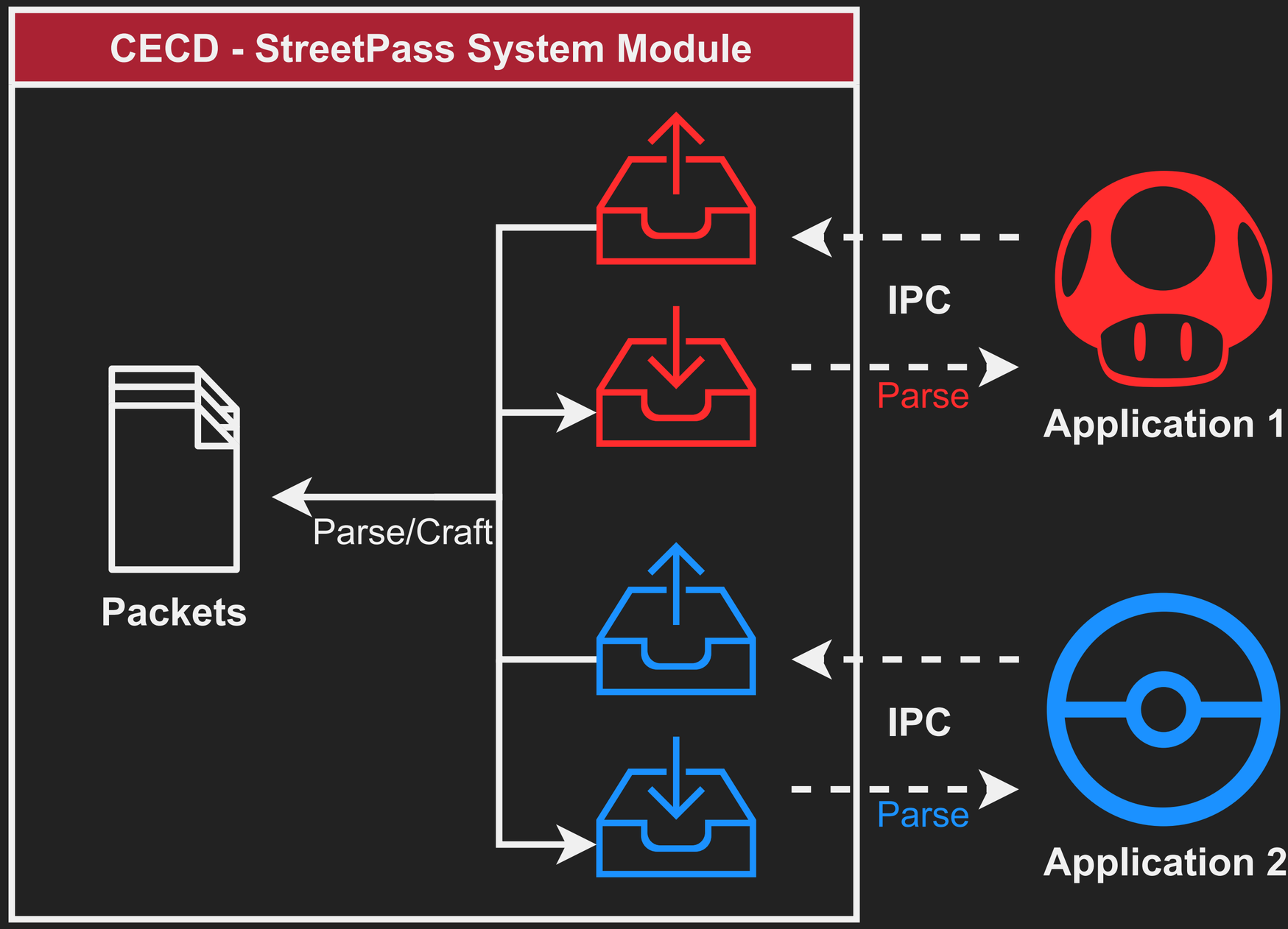
HACKER POINT OF VIEW...

HACKER POINT OF VIEW...

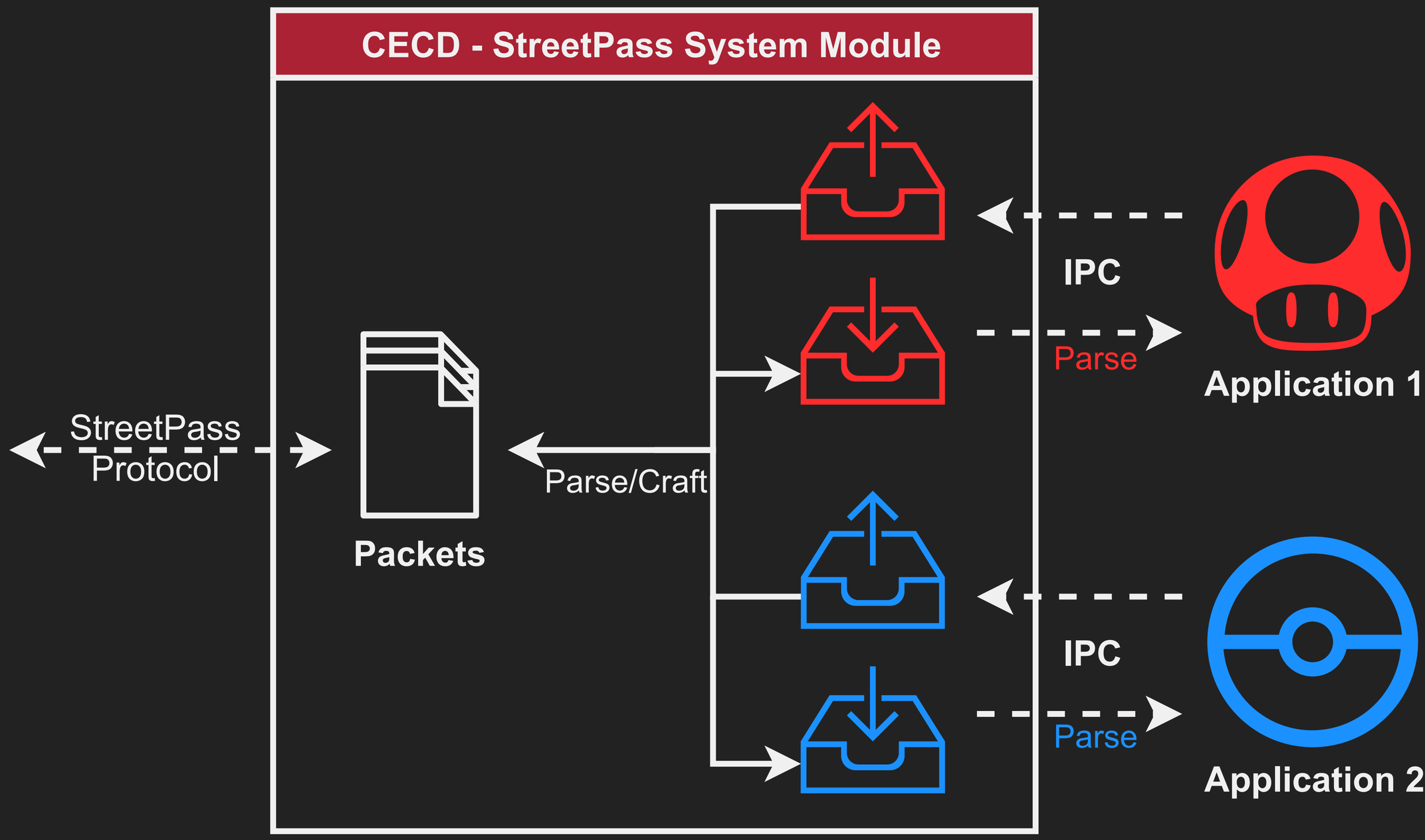
HOW DOES IT WORK?



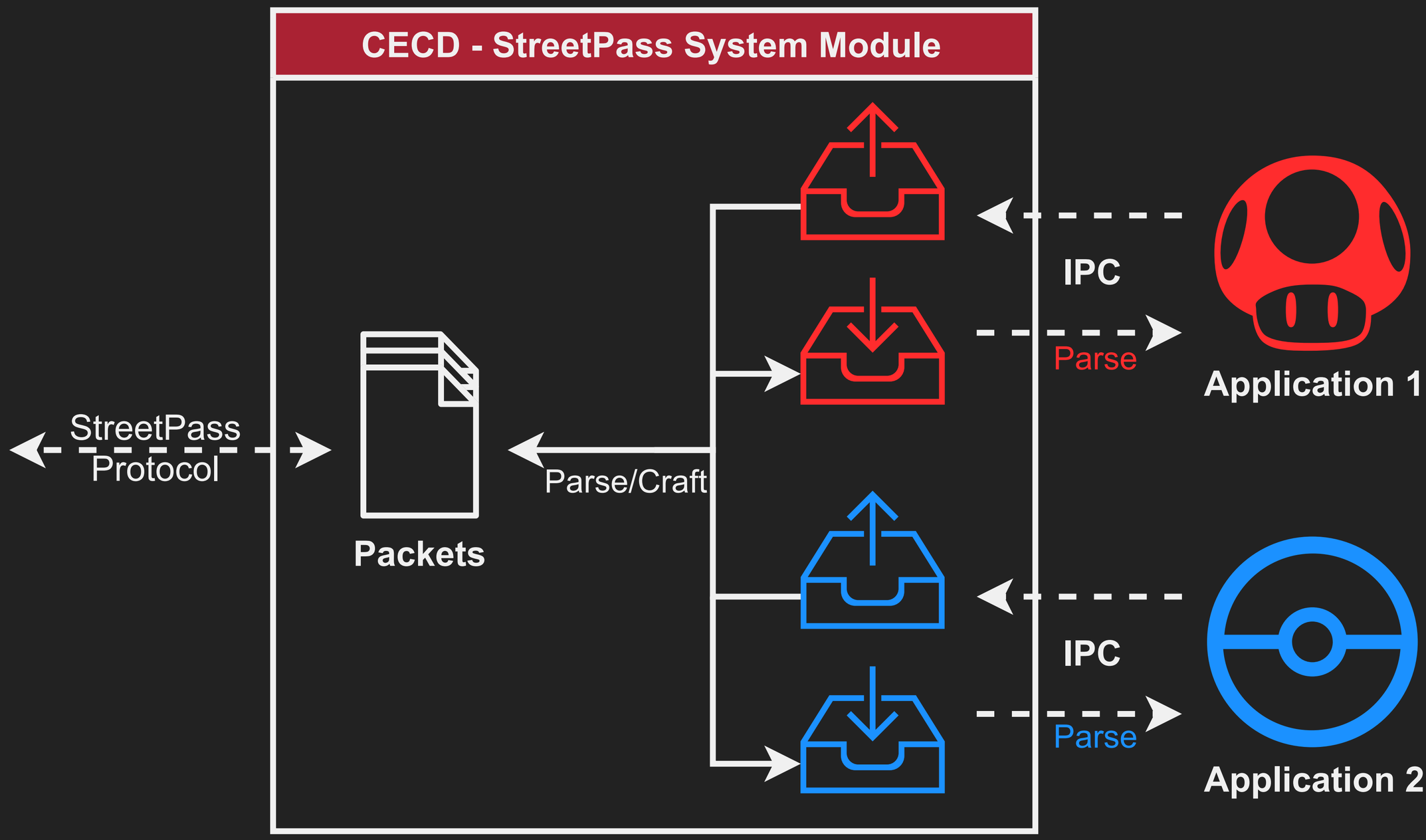
HOW DOES IT WORK?



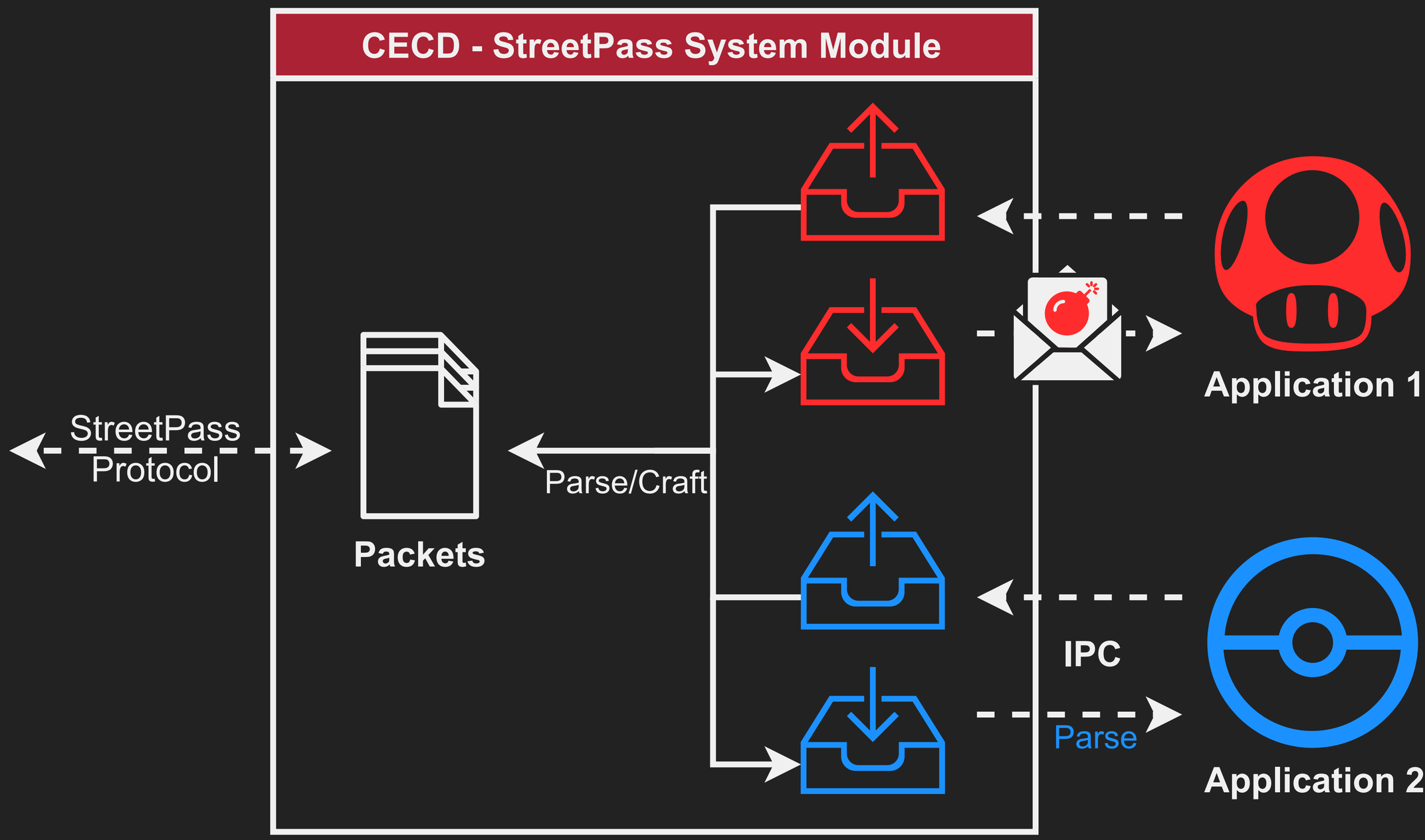
HOW DOES IT WORK?



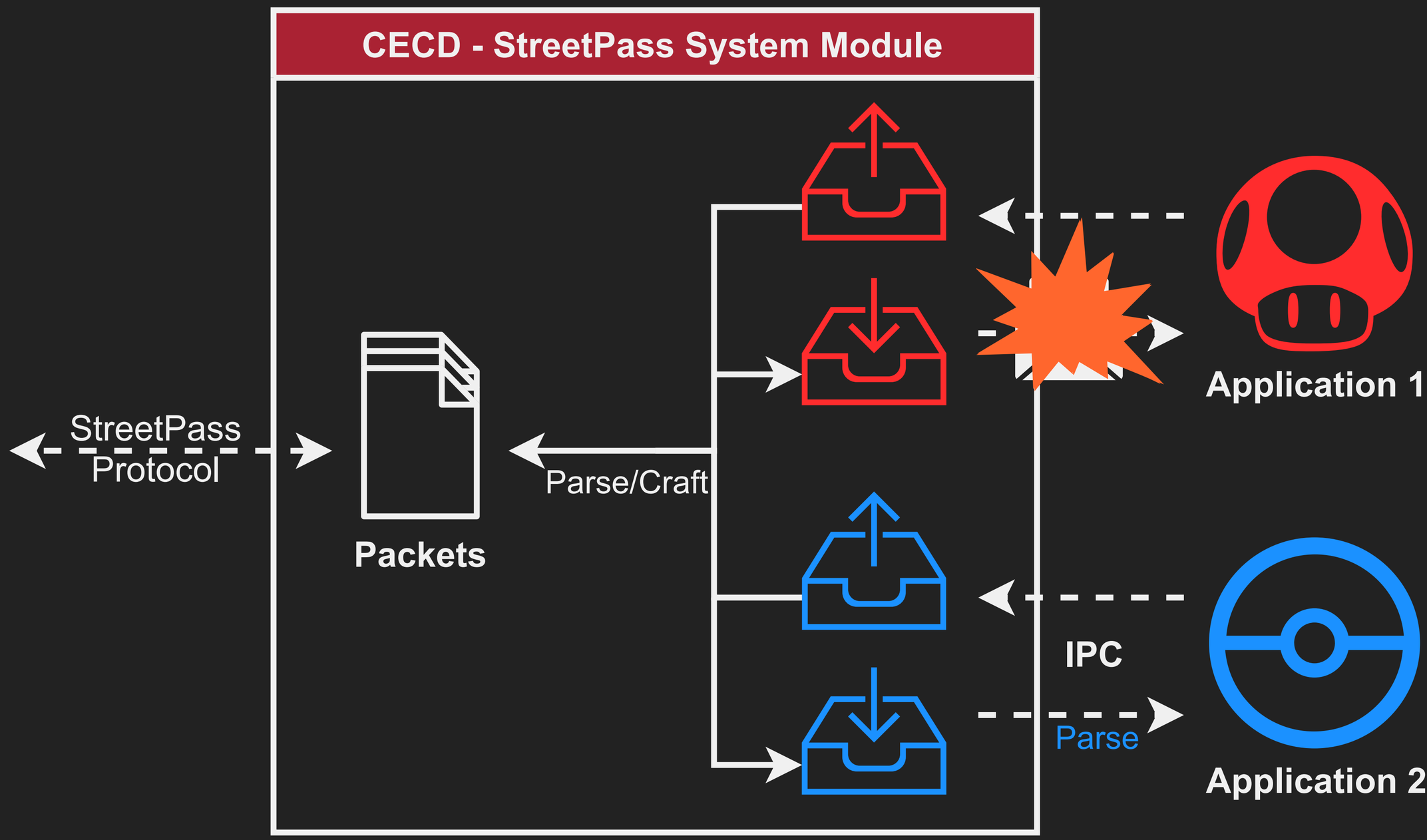
WHAT TO ATTACK?



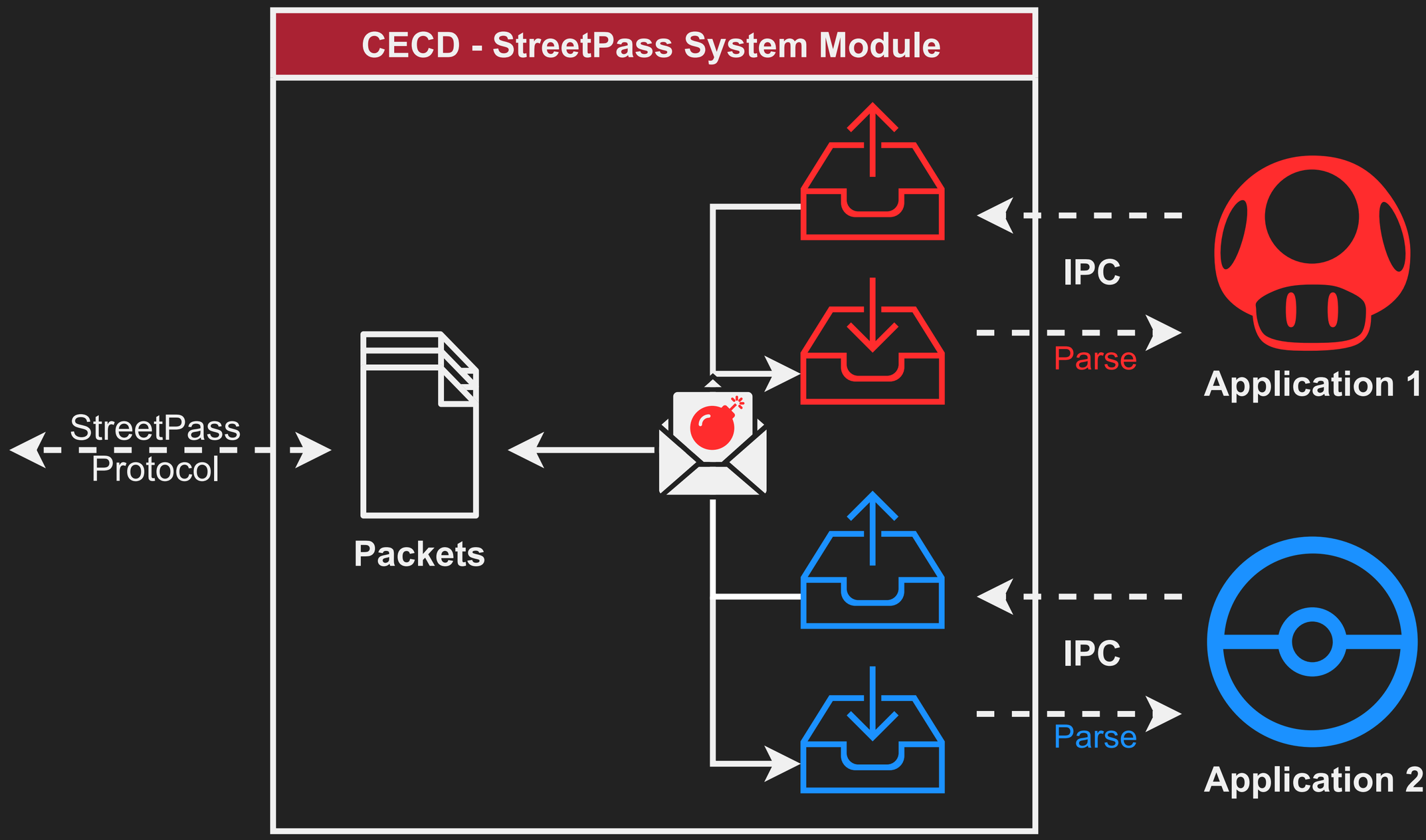
WHAT TO ATTACK?



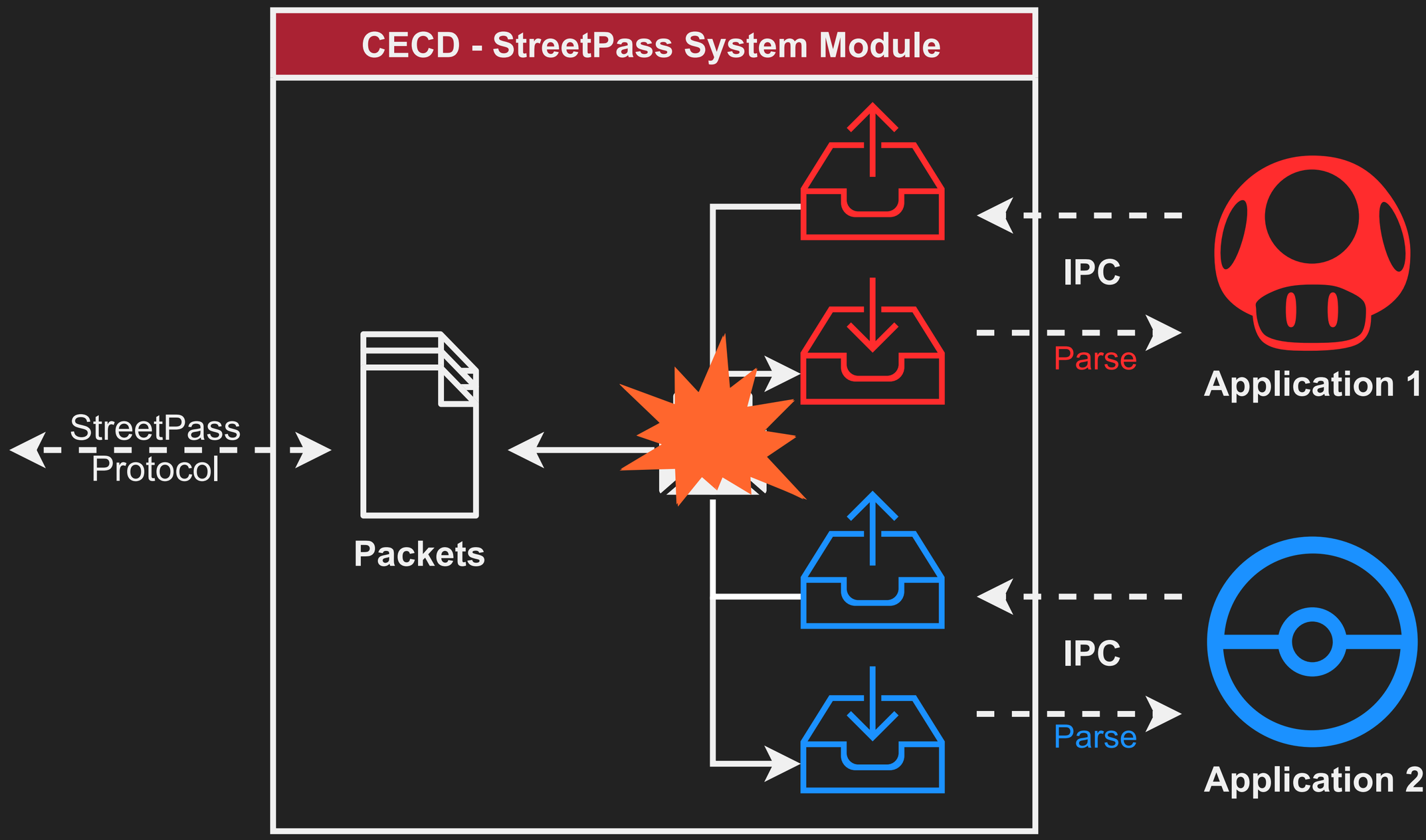
WHAT TO ATTACK?



WHAT TO ATTACK?



WHAT TO ATTACK?



STREETPASS PROTOCOL?

Nobody really knows how it works...

STREETPASS PROTOCOL?

Nobody really knows how it works...

- a bit of documentation on the pairing sequence
  - never been successfully reproduced...

STREETPASS PROTOCOL?

Nobody really knows how it works...

- a bit of documentation on the pairing sequence
  - never been successfully reproduced...
- unknown encrypted protocol operating according to IEEE 802.11 standards (**use AES keyslot 0x2E**)

STREETPASS PROTOCOL?

Nobody really knows how it works...

- a bit of documentation on the pairing sequence
  - never been successfully reproduced...
- unknown encrypted protocol operating according to IEEE 802.11 standards (**use AES keyslot 0x2E**)

We can get this key nowadays... let's reverse the protocol!

LET'S DIVE IN!

EQUIPMENT & TOOLS

Two hacked 3DS
running Luma3DS ❤
- debugger :)
- bootrom dump

Alfa Network
AWUS036NHA
- wifi adapter
- monitor mode
- packet injection

WIRESHARK

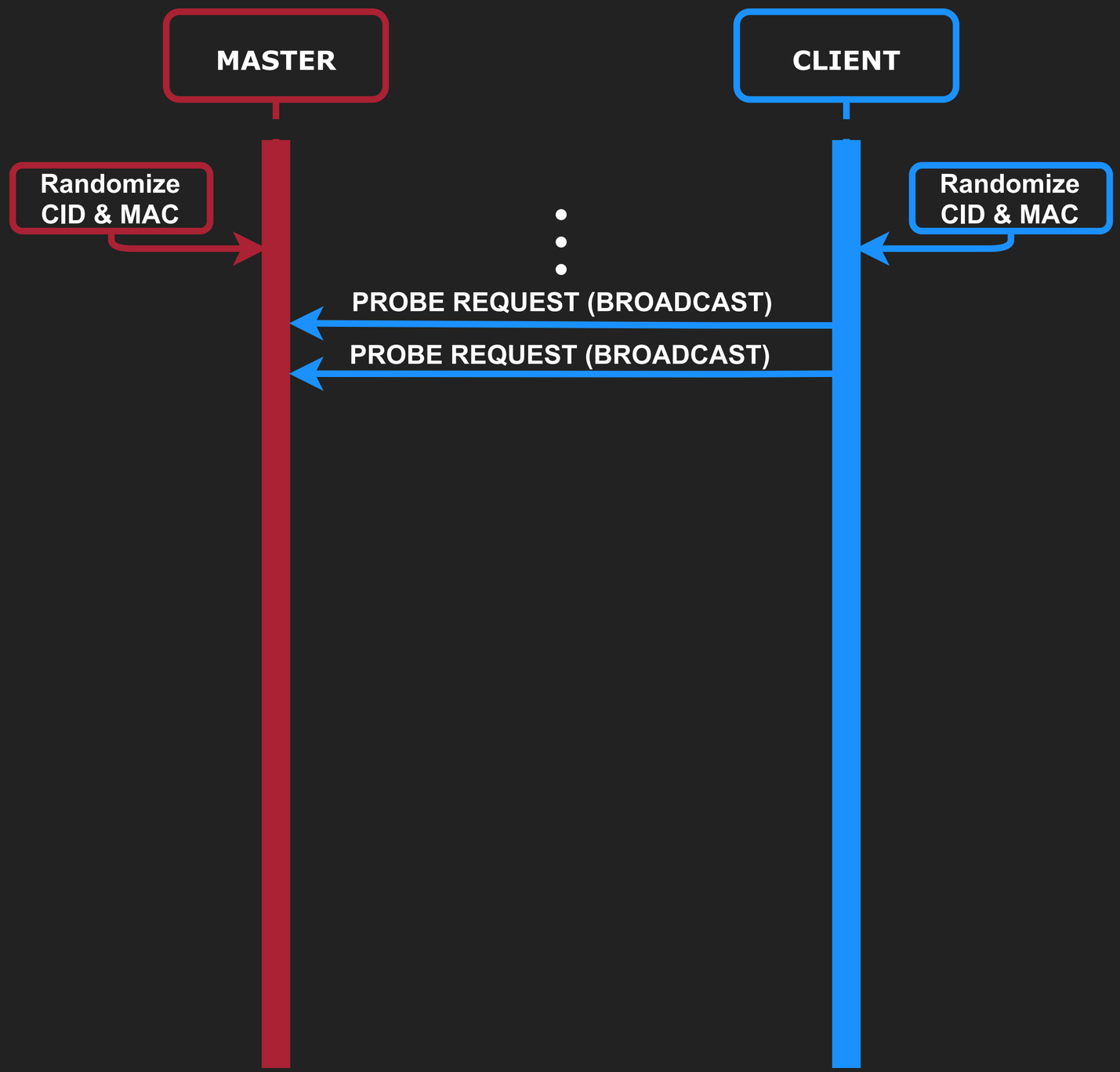
To sniff and analyze
packets...❤

IDA/Ghidra
Whichever you like
to reverse
engineer CECD...

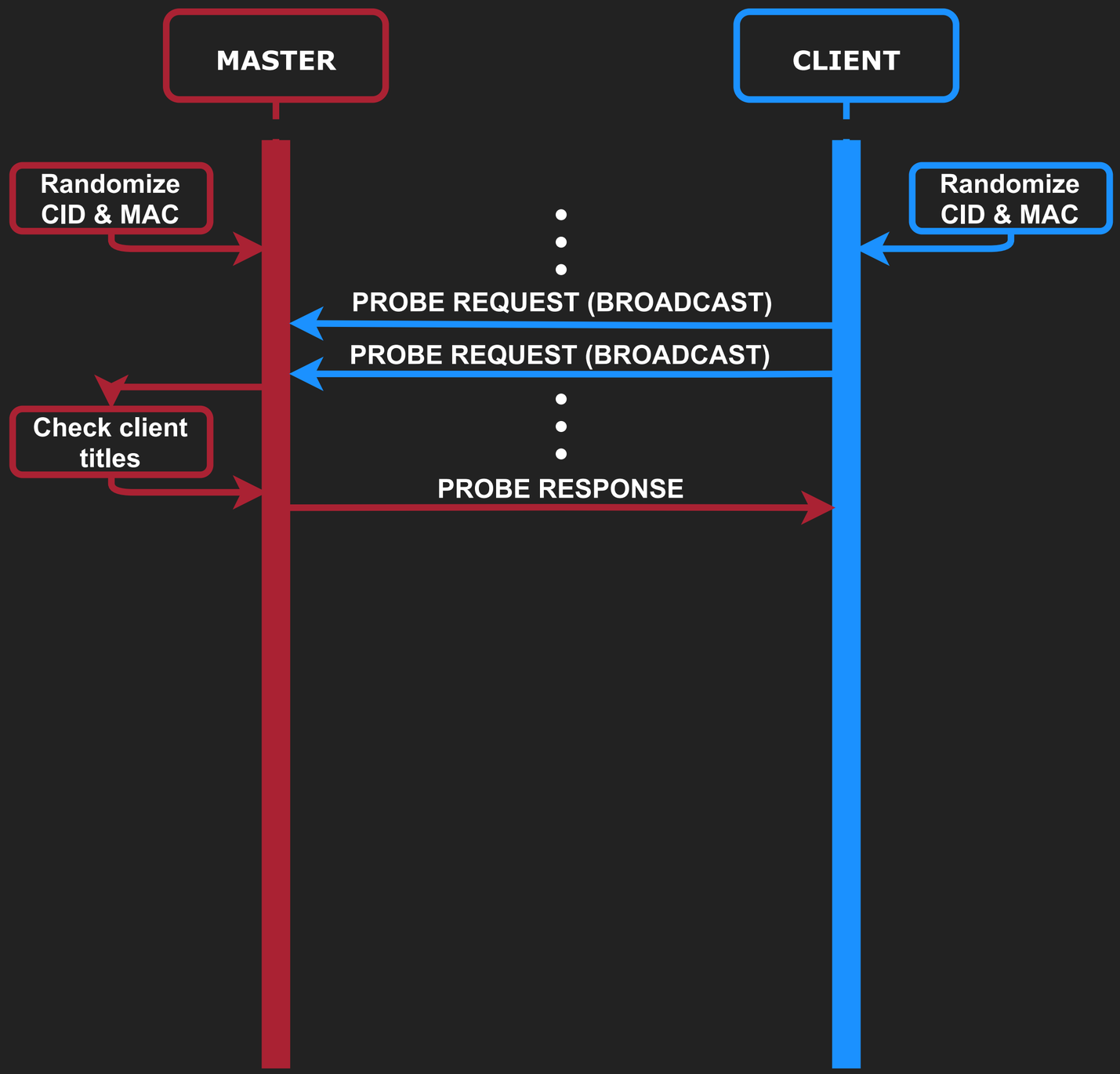
PAIRING

PAIRING SEQUENCE

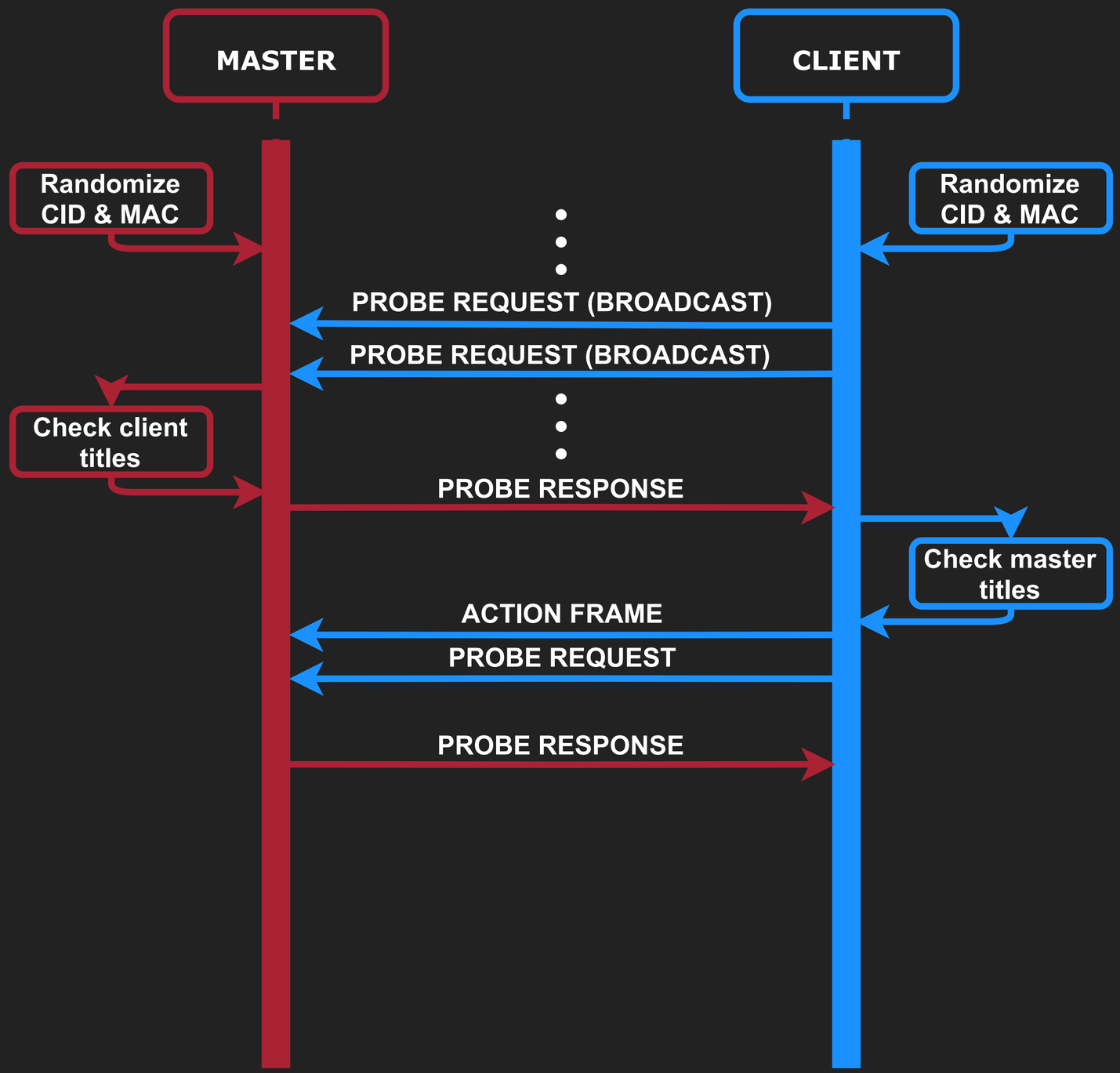
PAIRING SEQUENCE



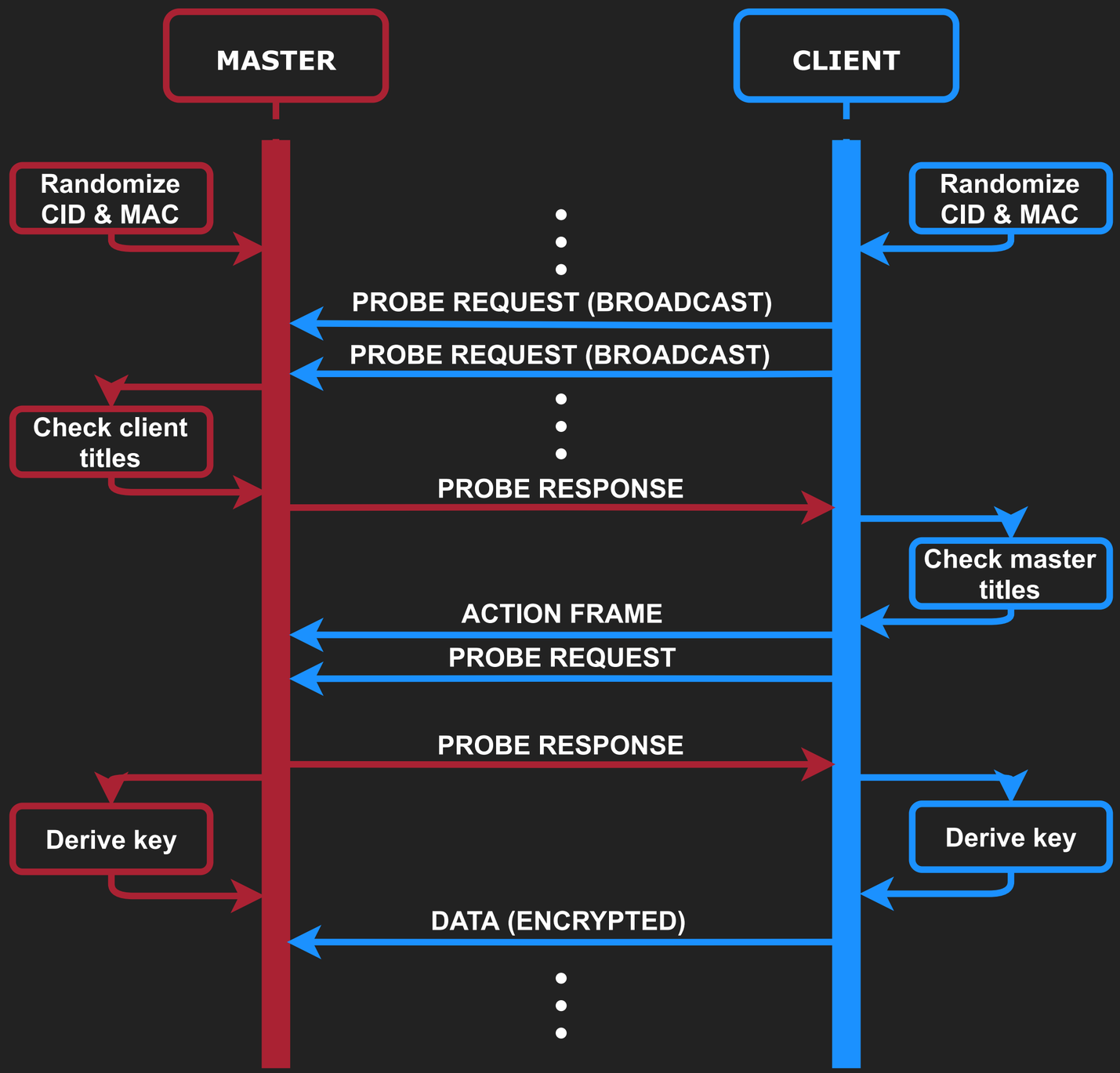
PAIRING SEQUENCE



PAIRING SEQUENCE



PAIRING SEQUENCE



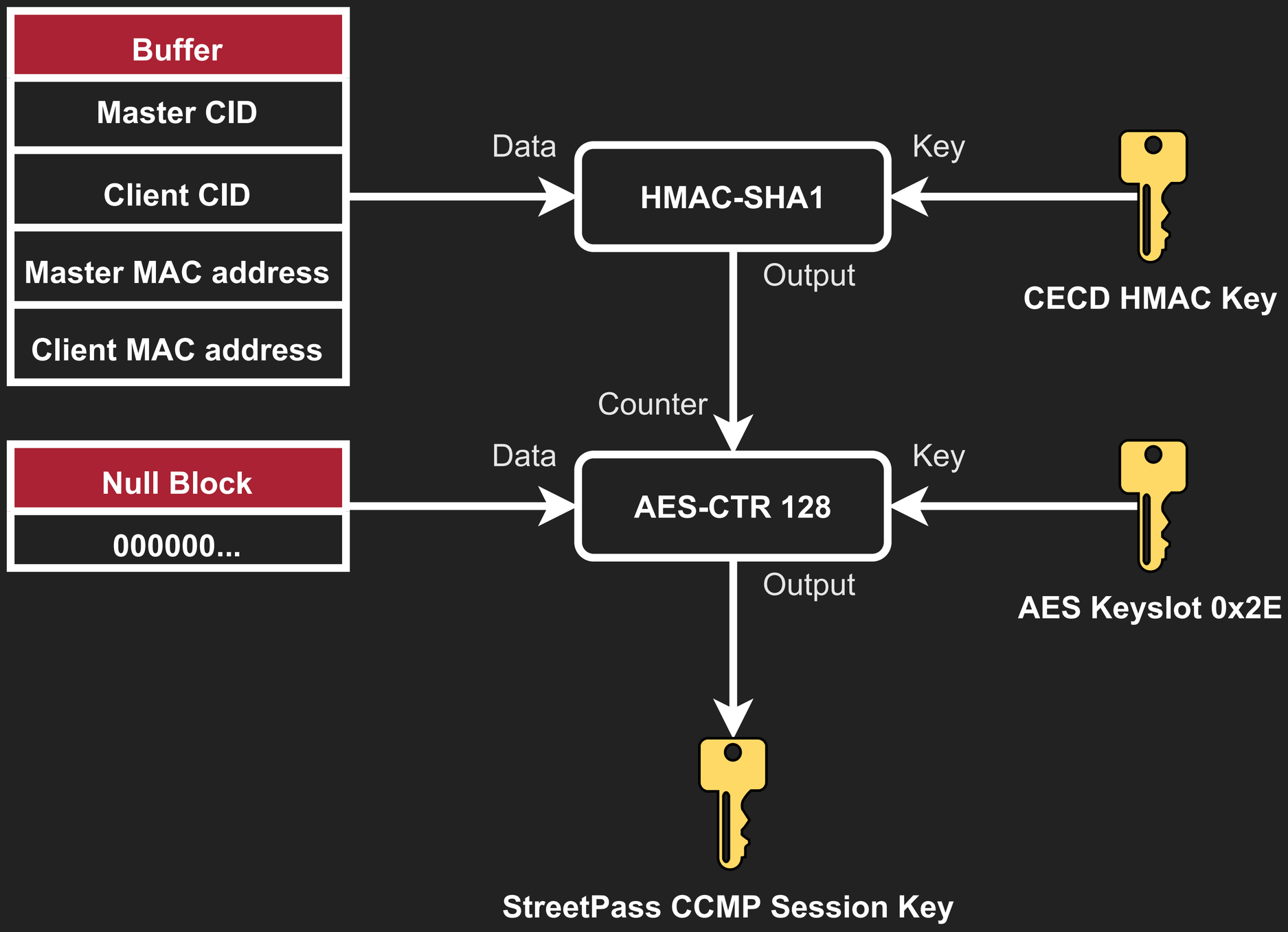
PAIRING ✓

- netlink protocol to communicate with drivers (libnl)
- use nl80211 to send/recv probe requests/responses
- everything else is handled by the driver :)

Yay! The 3DS starts sending encrypted data!

ENCRYPTION

SESSION KEY DERIVATION



DECRYPTION ✓

- uses AES-CCMP
- nl80211 lets you register CCMP keys
  - receive and send encrypted packets using raw sockets and send/recv syscalls :)

Source	Destination	Protocol	Length	Info
9e:e6:35:c9:fa:d4	7e:bb:8a:ac:b6:c5	IPv4	94	Bogus IPv4 version (0, must be 4)
9e:e6:35:c9:fa:d4	7e:bb:8a:ac:b6:c5	IPv4	94	Bogus IPv4 version (0, must be 4)
7e:bb:8a:ac:b6:c5	9e:e6:35:c9:fa:d4	IPv4	94	Bogus IPv4 version (0, must be 4)
9e:e6:35:c9:fa:d4	7e:bb:8a:ac:b6:c5	IPv4	94	Bogus IPv4 version (0, must be 4)
9e:e6:35:c9:fa:d4	7e:bb:8a:ac:b6:c5	IPv4	150	Bogus IPv4 version (0, must be 4)
7e:bb:8a:ac:b6:c5	9e:e6:35:c9:fa:d4	IPv4	94	Bogus IPv4 version (0, must be 4)
7e:bb:8a:ac:b6:c5	9e:e6:35:c9:fa:d4	IPv4	94	Bogus IPv4 version (0, must be 4)

REVERSING THE PROTOCOL

EARLY ANALYSIS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

```
59 59 68 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 00 00 43 00 00 00
00 00 00 00 10 00 00 00 1B 00 10 00 1A 80 0F 00
FF FF FF FF 74 54 12 00
```

```
59 59 56 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 FF FF 12 00 00 00
00 00 00 00 16 00 00 00 11 0A 00 05 16 00 30 00
02 08 00 00 F0 08 A3 E1 84 BD C0 05 BF 49
```

EARLY ANALYSIS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

```
59 59 68 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 00 00 43 00 00 00
00 00 00 00 10 00 00 00 1B 00 10 00 1A 80 0F 00
FF FF FF FF 74 54 12 00
```

```
59 59 56 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 FF FF 12 00 00 00
00 00 00 00 16 00 00 00 11 0A 00 05 16 00 30 00
02 08 00 00 F0 08 A3 E1 84 BD C0 05 BF 49
```

header

EARLY ANALYSIS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

```
59 59 68 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 00 00 43 00 00 00
00 00 00 00 10 00 00 00 1B 00 10 00 1A 80 0F 00
FF FF FF FF 74 54 12 00
```

```
59 59 56 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 FF FF 12 00 00 00
00 00 00 00 16 00 00 00 11 0A 00 05 16 00 30 00
02 08 00 00 F0 08 A3 E1 84 BD C0 05 BF 49
```

header

data

EARLY ANALYSIS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

```
59 59 68 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 00 00 43 00 00 00
00 00 00 00 10 00 00 00 1B 00 10 00 1A 80 0F 00
FF FF FF FF 74 54 12 00
```

```
59 59 56 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 38 B8 38 00 00 00 00 63 63 FF FF 12 00 00 00
00 00 00 00 16 00 00 00 11 0A 00 05 16 00 30 00
02 08 00 00 F0 08 A3 E1 84 BD C0 05 BF 49
```

header

data

header magic value

data magic value

TWO PROTOCOLS!

Not official names obviously...

TWO PROTOCOLS!

Not official names obviously...

- **StreetPass Transmission Control Protocol (SPTCP)**
  - quite similar to TCP but for local communication
  - ensures reliability
  - handles data segmentation

TWO PROTOCOLS!

Not official names obviously...

- **StreetPass Transmission Control Protocol (SPTCP)**
  - quite similar to TCP but for local communication
  - ensures reliability
  - handles data segmentation

- **StreetPass Message Transfer Protocol (SPMTP)**
  - sends packets over SPTCP
  - handles exchanging streetpass messages

SPTCP

SPTCP HEADER & FLAGS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

SPTCP HEADER & FLAGS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

magic

0x5959

SPTCP HEADER & FLAGS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

magic
0x5959

constants
0xdead 0xbeaf

SPTCP HEADER & FLAGS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

magic
0x5959

constants
0xdead 0xbeaf

flags
frame types

SPTCP HEADER & FLAGS

```
59 59 30 00 AD DE AF BE 00 00 00 00 00 00 00 00
0C 10 B8 38 00 00 00 00
```

magic	constants	flags
0x5959	0xdead 0xbeaf	frame types

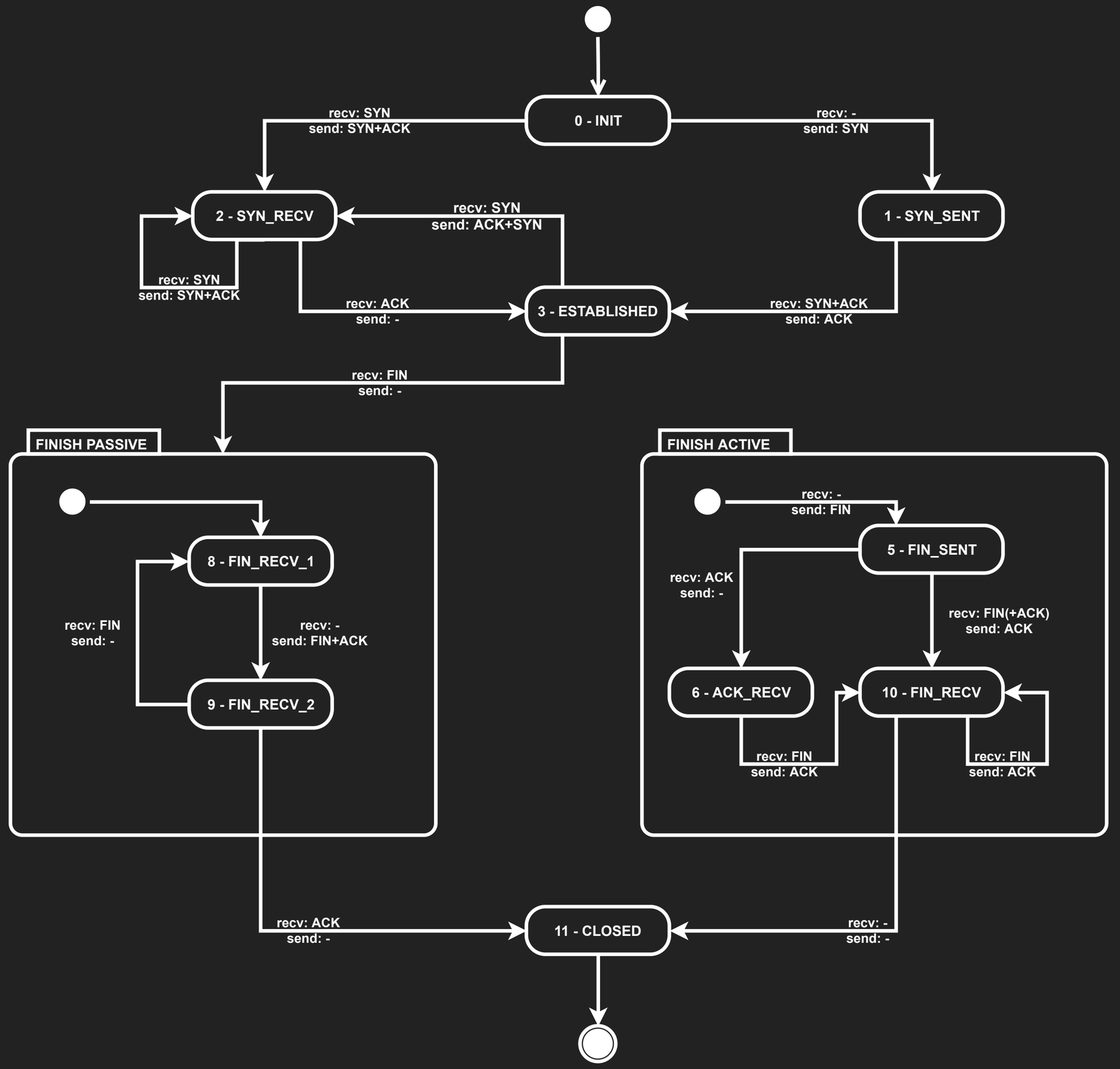
Almost the same as TCP flags...

```
ACK|PSH: 0 1 1 0 0 0 = 0x18
SYN|ACK: 0 1 0 0 1 0 = 0x12
```

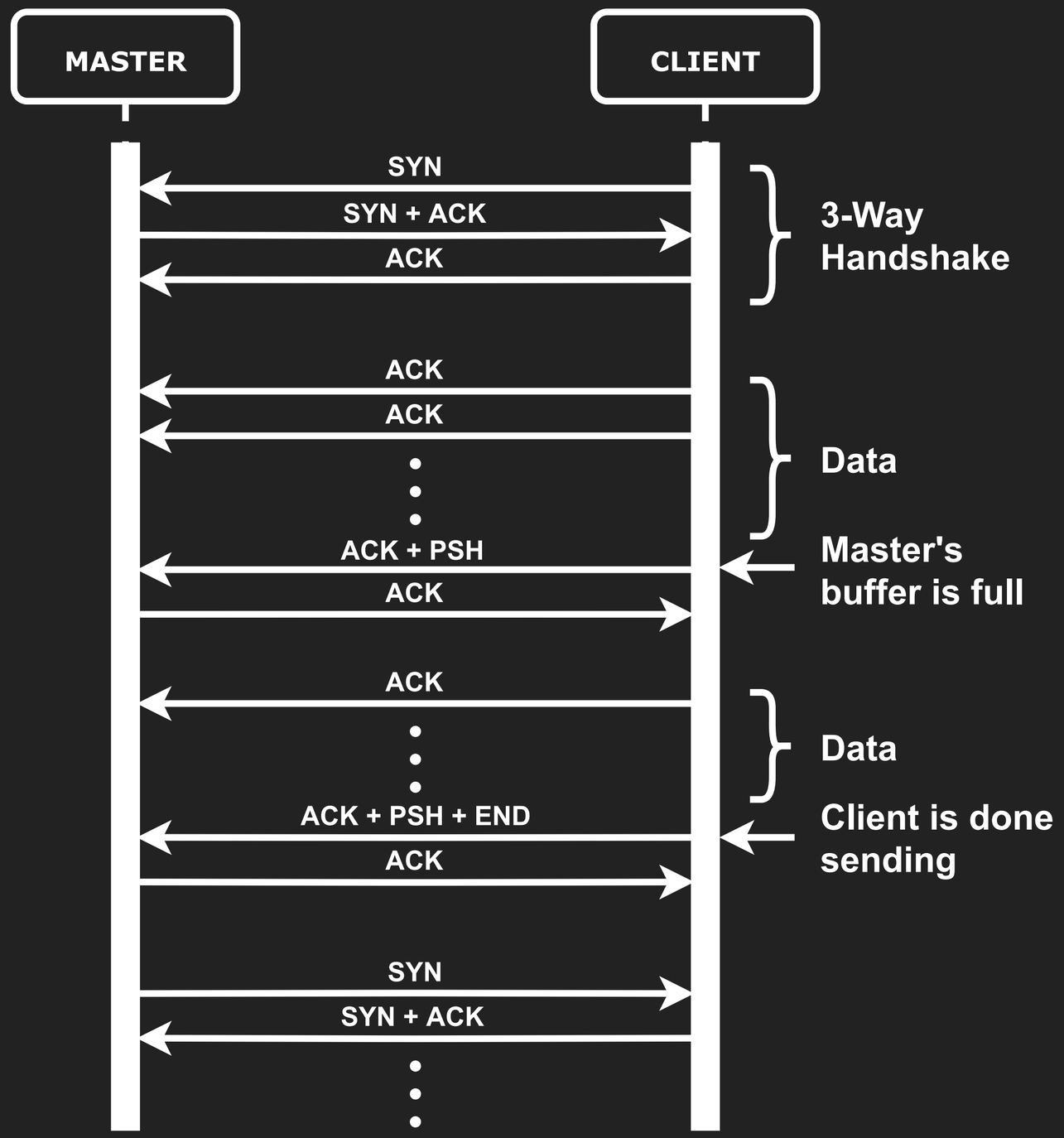
~~URG~~ END ACK PSH RST SYN FIN

Once you know that, it's fairly easy to build the state diagram and understand the protocol...

SPTCP STATE DIAGRAM



TYPICAL SPTCP EXCHANGE



SPTCP SECURITY

Hey it seems to be okay.

- found no deadly bug
  - maybe some minor unexploitable ones
- attack surface is not that large
  - SPMTP is much more interesting!

SPMTP

TWO TYPES OF PACKETS

There are two different magic values!

```
63 63 00 00 43 00 00 00 00 00 00 00 10 00 00 00
1B 00 10 00 1A 80 0F 00 FF FF FF FF 74 54 12 00
```

```
61 61 00 00 80 30 00 00 00 16 05 00 00 00 00 00
01 00 00 00 60 60 00 00 6C 30 00 00 48 0D 00 00
04 23 00 00 00 16 05 00 00 00 00 00 00 00 00 00
00 00 00 00 12 1D AA 64 90 76 5A CB 00 00 ...
```

TWO TYPES OF PACKETS

There are two different magic values!

```
63 63 00 00 43 00 00 00 00 00 00 00 10 00 00 00
1B 00 10 00 1A 80 0F 00 FF FF FF FF 74 54 12 00
```

```
61 61 00 00 80 30 00 00 00 16 05 00 00 00 00 00
01 00 00 00 60 60 00 00 6C 30 00 00 48 0D 00 00
04 23 00 00 00 16 05 00 00 00 00 00 00 00 00 00
00 00 00 00 12 1D AA 64 90 76 5A CB 00 00 ...
```

info packet

share information
like part of a handshake

message box packet

contains messages for
a specific application!

TWO TYPES OF PACKETS

There are two different magic values!

```
63 63 00 00 43 00 00 00 00 00 00 00 10 00 00 00
1B 00 10 00 1A 80 0F 00 FF FF FF FF 74 54 12 00
```

```
61 61 00 00 80 30 00 00 00 16 05 00 00 00 00 00
01 00 00 00 60 60 00 00 6C 30 00 00 48 0D 00 00
04 23 00 00 00 16 05 00 00 00 00 00 00 00 00 00
00 00 00 00 12 1D AA 64 90 76 5A CB 00 00 ...
```

info packet

share information
like part of a handshake

message box packet

contains messages for
a specific application!

CECD message file magic value!

we reached application data!

INFO PACKETS

A bunch of data sent here...

INFO PACKETS

A bunch of data sent here...

- fixed size data
  - nothing fancy...
  - Friend Code, MAC address, date & time, etc.

INFO PACKETS

A bunch of data sent here...

- fixed size data
  - nothing fancy...
  - Friend Code, MAC address, date & time, etc.
- variable size data
  - application list, message box metadata list, etc.
  - much more interesting! any buffer overflow in sight?

BOX METADATA LIST PARSING

Where is the deadly bug here?

```
void copy_box_info_list(box_info_list* dst, box_info_list* src)
{
  memcpy(dst, src, sizeof(box_info_list_header));
  if(dst->header.magic == 0x6565)
  {
    dst->box_count = dst->header.count;

    for(int i = 0; i < dst->box_count; i++)
      memcpy(&dst->box_info[i], &src->box_info[i],
                          sizeof(box_info_entry));

  }
}
```

BOX METADATA LIST PARSING

Where is the deadly bug here?

```
void copy_box_info_list(box_info_list* dst, box_info_list* src)
{
  memcpy(dst, src, sizeof(box_info_list_header));
  if(dst->header.magic == 0x6565)
  {
    dst->box_count = dst->header.count;

    for(int i = 0; i < dst->box_count; i++)
      memcpy(&dst->box_info[i], &src->box_info[i],
                          sizeof(box_info_entry));

  }
}
```

BOX METADATA LIST PARSING

Where is the deadly bug here?

```
void copy_box_info_list(box_info_list* dst, box_info_list* src)
{
  memcpy(dst, src, sizeof(box_info_list_header));
  if(dst->header.magic == 0x6565)
  {
    dst->box_count = dst->header.count;

    for(int i = 0; i < dst->box_count; i++)
      memcpy(&dst->box_info[i], &src->box_info[i],
                          sizeof(box_info_entry));

  }
}
```

They do not check the number of entries in the list!

BOX METADATA LIST PARSING

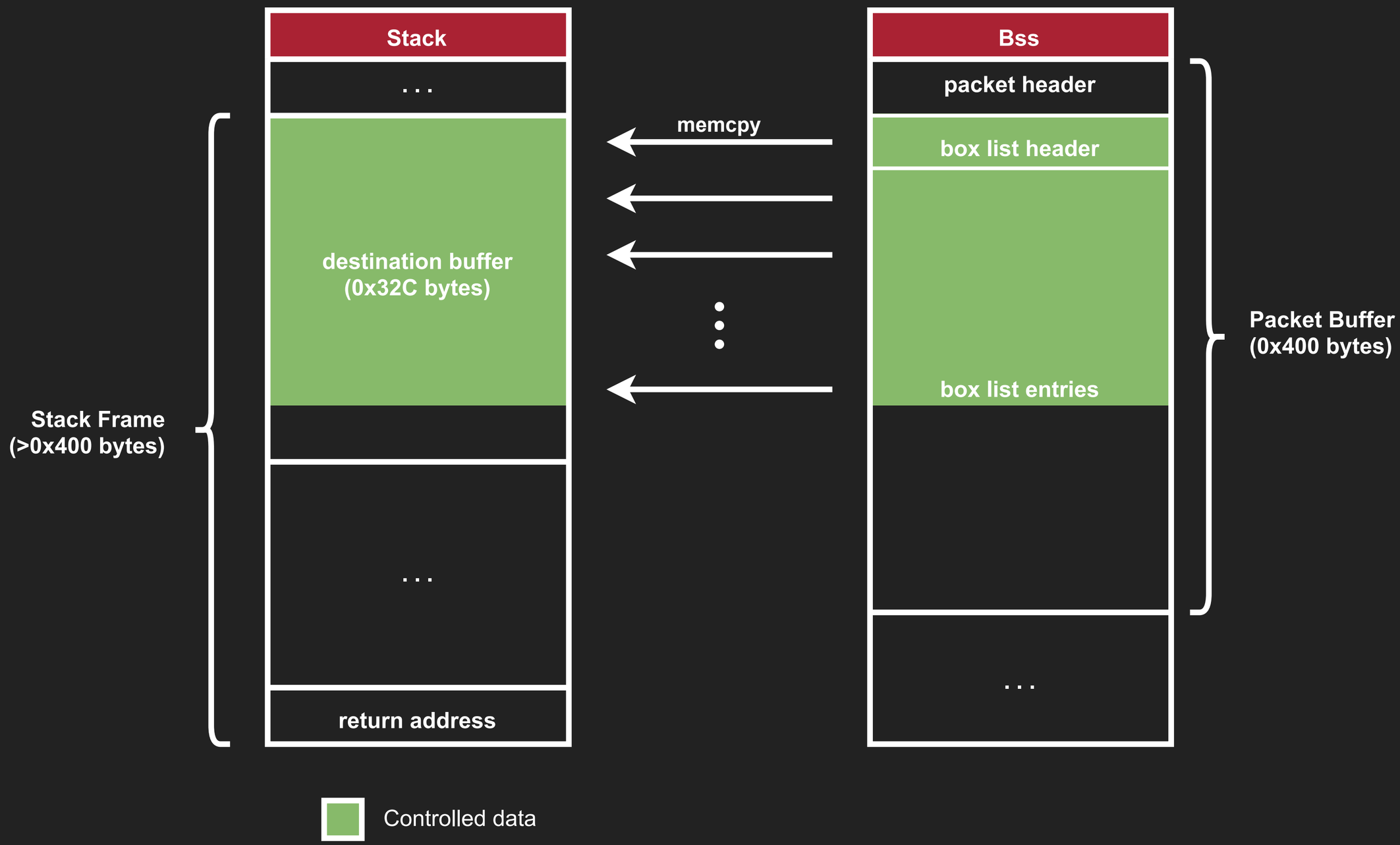
Where is the deadly bug here?

```
void copy_box_info_list(box_info_list* dst, box_info_list* src)
{
  memcpy(dst, src, sizeof(box_info_list_header));
  if(dst->header.magic == 0x6565)
  {
    dst->box_count = dst->header.count;

    for(int i = 0; i < dst->box_count; i++)
      memcpy(&dst->box_info[i], &src->box_info[i],
                          sizeof(box_info_entry));

  }
}
```

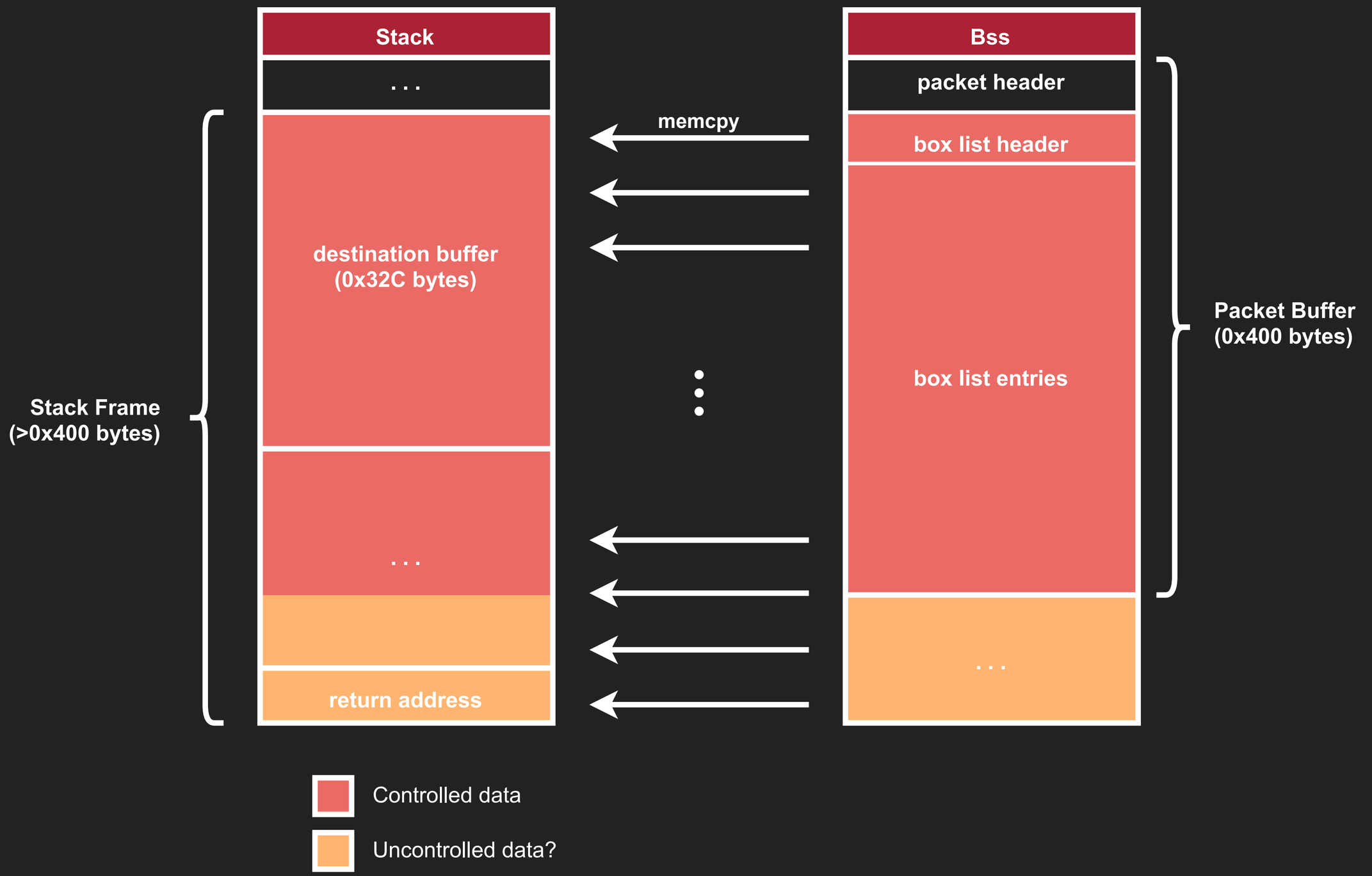
They do not check the number of entries in the list!

Is it exploitable?

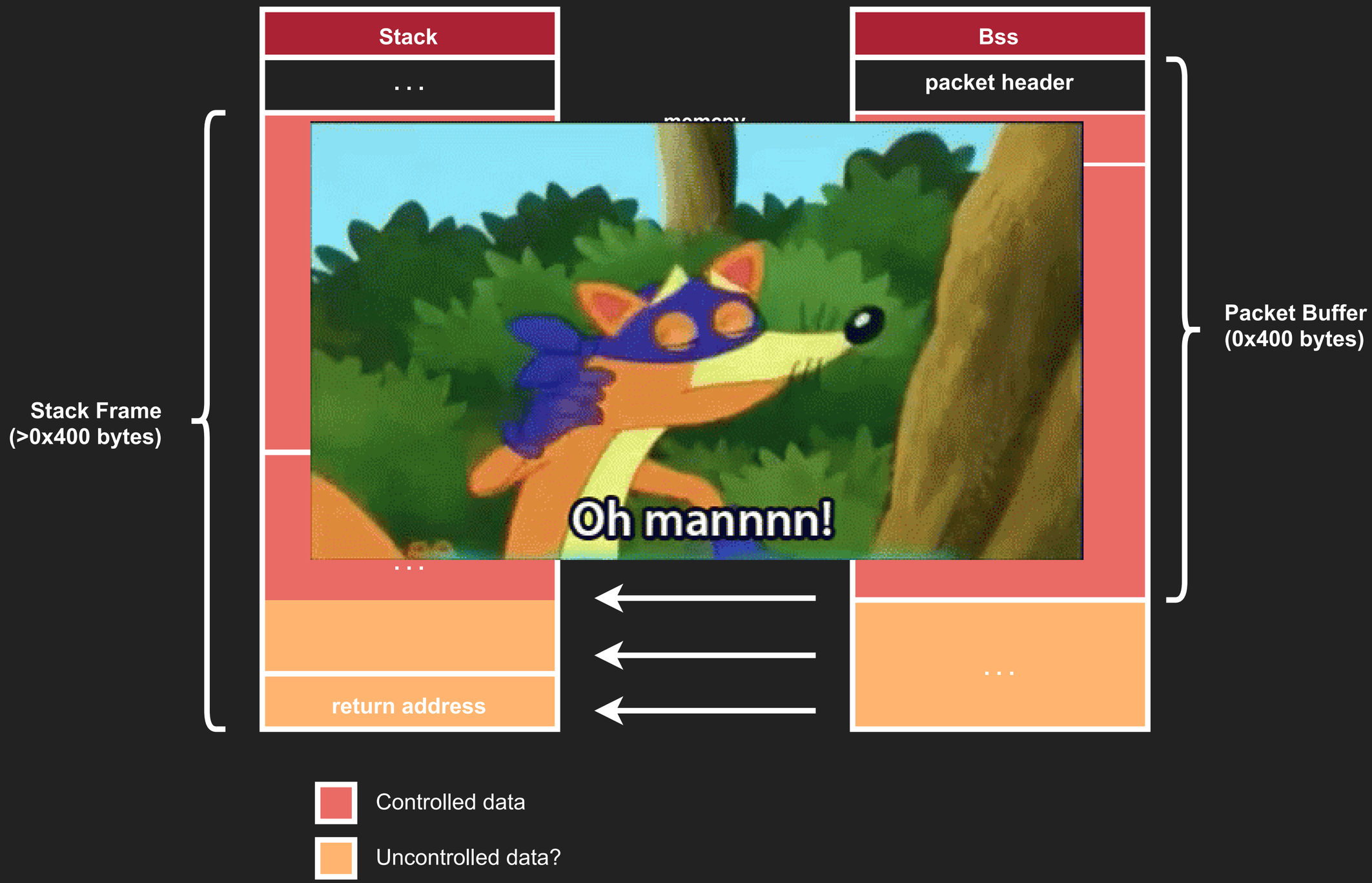
BOX METADATA LIST PARSING



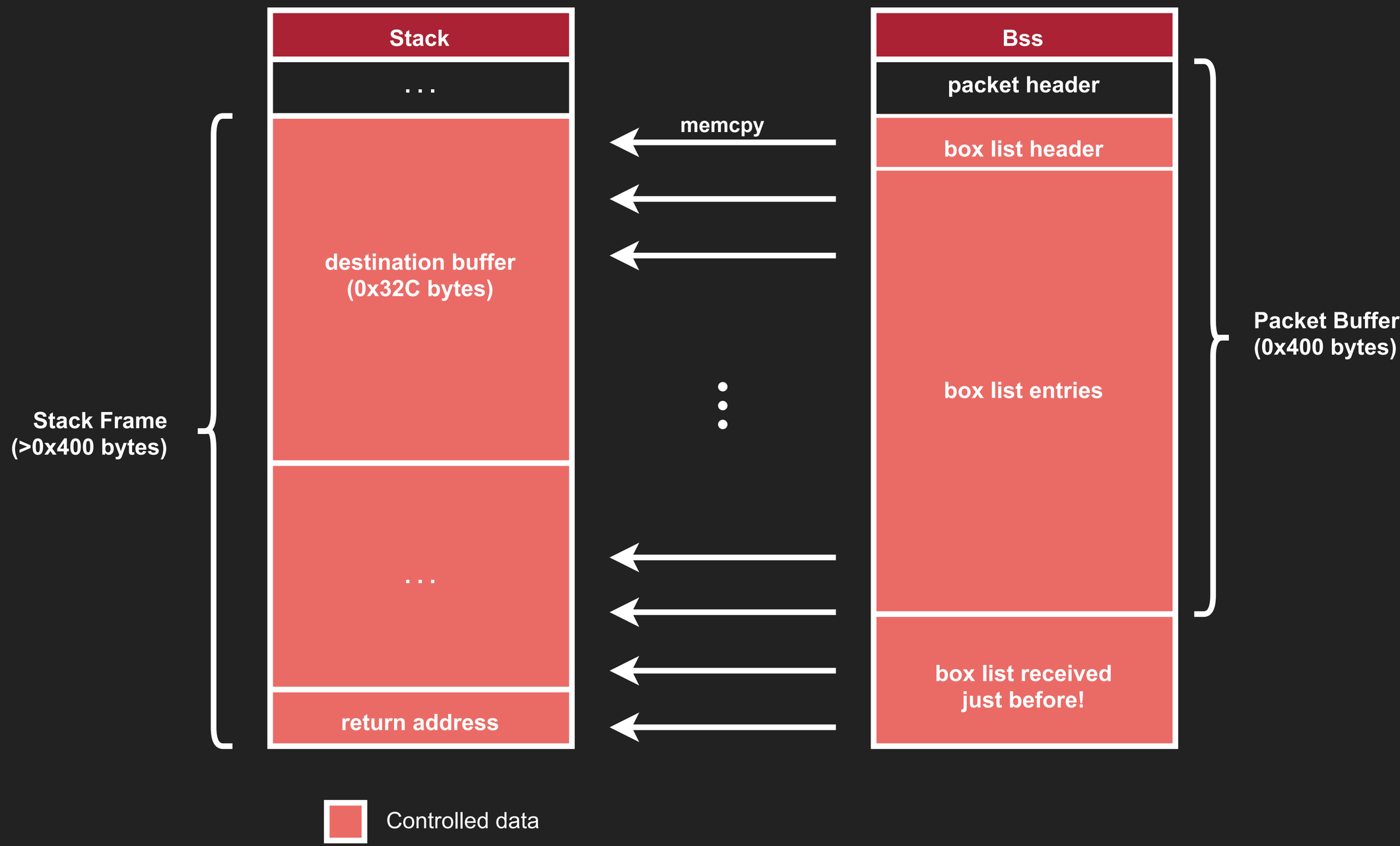
BOX METADATA LIST PARSING



BOX METADATA LIST PARSING



BOX METADATA LIST PARSING



HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

1. embed a small ROP-chain in the box list

HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

1. embed a small ROP-chain in the box list
2. send another one in a packet

HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

1. embed a small ROP-chain in the box list
2. send another one in a packet
3. stack-pivot to the second chain

HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

1. embed a small ROP-chain in the box list
2. send another one in a packet
3. stack-pivot to the second chain

RCE in cecd ✓

HOW DO WE EXPLOIT IT?

NX Bit	Stack Cookie	ASLR
✓	✗	✗

1. embed a small ROP-chain in the box list
2. send another one in a packet
3. stack-pivot to the second chain

RCE in cecd ✓

This one was easy... let's move on!

MESSAGE BOX PACKETS

- list of StreetPass messages (max 64)
- stored in temporary files ("TMP_XX")
  - let's call them "TMP Box"
- parsed once the communication is over

MESSAGE BOX PACKETS

- list of StreetPass messages (max 64)
- stored in temporary files ("TMP_XX")
  - let's call them "TMP Box"
- parsed once the communication is over

Let's take a look at that parser!

TMP BOX FILE LOADING

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  ...

  message* current_msg = file_buffer->messages;
  while(dst->header.msg_count > i && end_of_file > current_msg) {
    uint32_t msg_size = message_get_size(current_msg);
    dst->msg_pointers[i] = current_msg;
    dst->msg_sizes[i] = msg_size;
    current_msg += msg_size;
    i++;
    glob_tmp_box_alloc_mode  = POINTER_MODE; // POINTER_MODE  = 0
  }

  dst->header.msg_count = i;
  return 0;
}
```

TMP BOX FILE LOADING

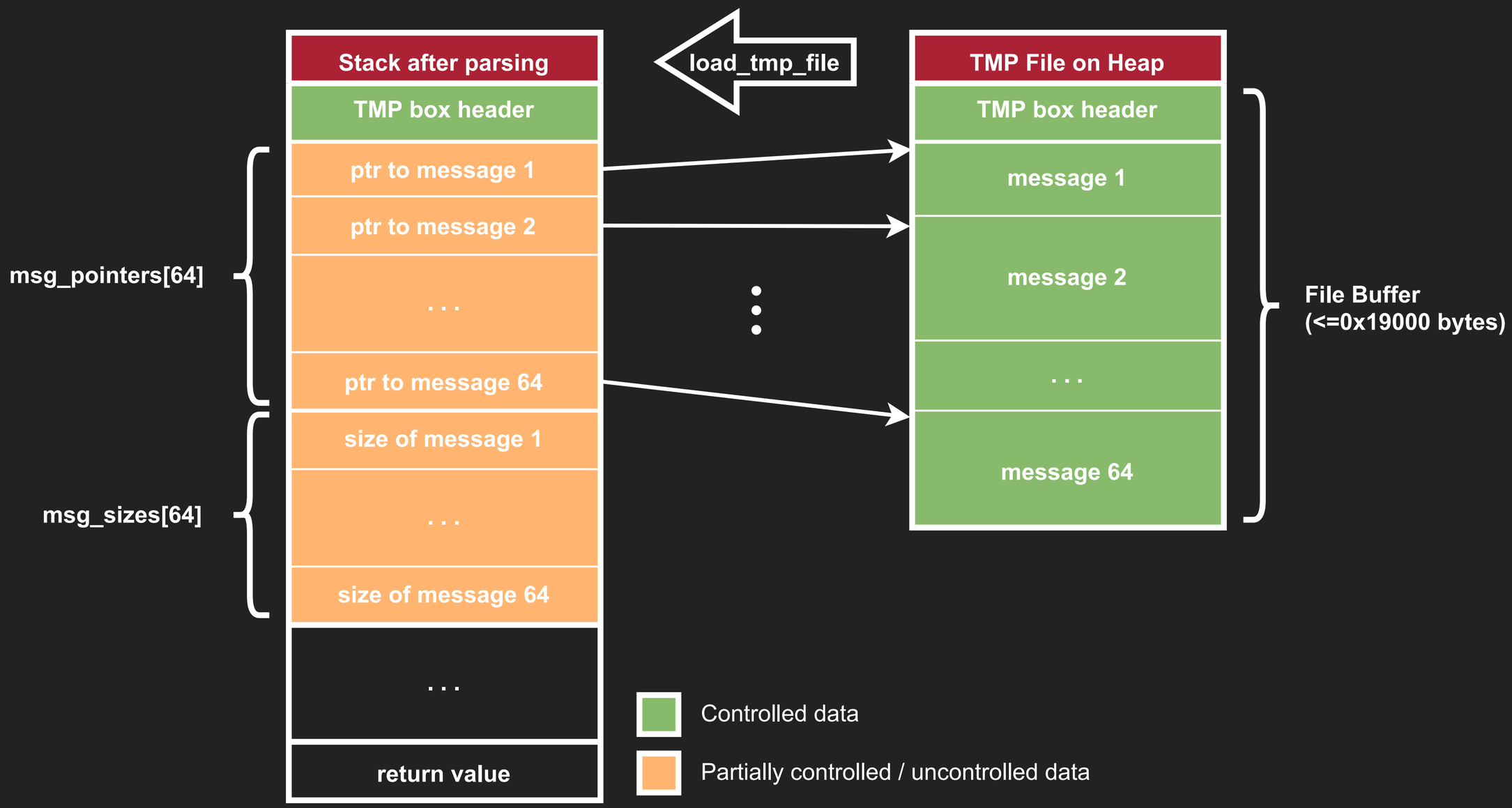
```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  ...

  message* current_msg = file_buffer->message
  while(dst->header.msg_count > i && end_of_f
    uint32_t msg_size = message_get_size(curr
    dst->msg_pointers[i] = current_msg;
    dst->msg_sizes[i] = msg_size;
    current_msg += msg_size;
    i++;
    glob_tmp_box_alloc_mode  = POINTER_MODE;
  }

  dst->header.msg_count = i;
  return 0;
}
```

TMP BOX FILE LOADING

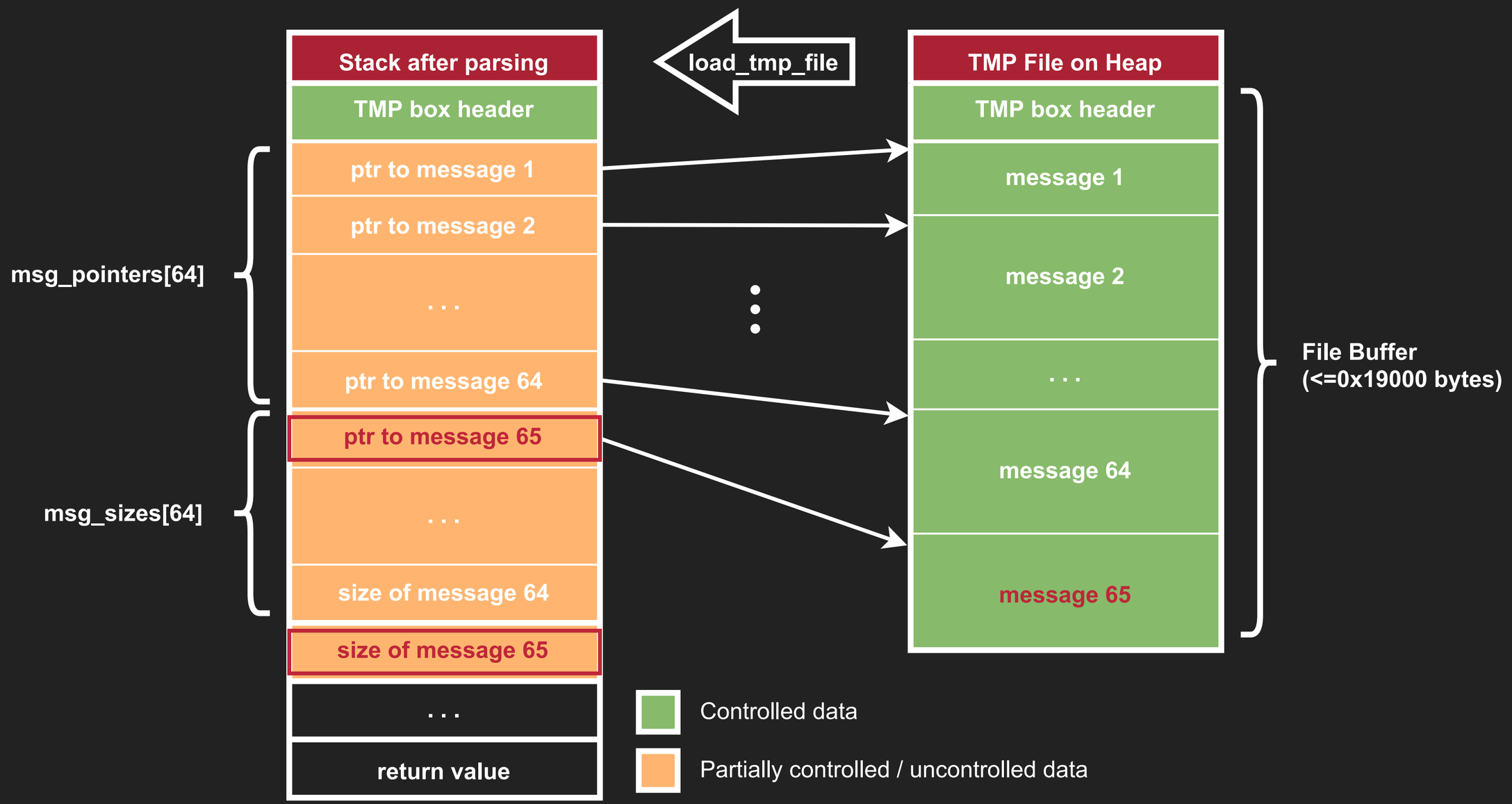
```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  ...

  message* current_msg = file_buffer->message
  while(dst->header.msg_count > i && end_of_f
    uint32_t msg_size = message_get_size(curr
    dst->msg_pointers[i] = current_msg;
    dst->msg_sizes[i] = msg_size;
    current_msg += msg_size;
    i++;
    glob_tmp_box_alloc_mode  = POINTER_MODE;
  }

  dst->header.msg_count = i;
  return 0;
}
```

They do not check the number of messages in the box!

TMP BOX OVERFLOW



TMP BOX OVERFLOW



WHAT CAN WE DO?

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  ...

  message* current_msg = file_buffer->messages;
  while(dst->header.msg_count > i && end_of_file > current_msg) {
    uint32_t msg_size = message_get_size(current_msg);
    dst->msg_pointers[i] = current_msg;
    dst->msg_sizes[i] = msg_size;
    current_msg += msg_size;
    i++;
    glob_tmp_box_alloc_mode  = POINTER_MODE; // POINTER_MODE  = 0
  }

  dst->header.msg_count = i;
  return 0;
}
```

WHAT CAN WE DO?

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  ...

  message* current_msg = file_buffer->messages;
  while(dst->header.msg_count > i && end_of_file > current_msg) {
    uint32_t msg_size = message_get_size(current_msg);
    dst->msg_pointers[i] = current_msg;
    dst->msg_sizes[i] = msg_size;
    current_msg += msg_size;
    i++;
    glob_tmp_box_alloc_mode  = POINTER_MODE; // POINTER_MODE  = 0
  }

  dst->header.msg_count = i;
  return 0;
}
```

The "size" of the last message can be an arbitrary value!

We can totally control only one 32-bit value on the stack...

WHAT TO OVERWRITE?

The only interesting one you can overwrite without crashing...

```
void parse_all_TMP() {
  [...]
  tmp_box tmp_box;
  critical_section* lock; // overwritten by overflow in parse_TMP_file!
  [...]
  for(int i = 0; i < TMP_file_count; i++) {
    enter_critical_section(&lock, &global_lock); // restore lock value!
    file_buffer = malloc(TMP_file_size[i]);
    [...] // file reading, etc.
    parse_TMP_file(&tmp_box, file_buffer, TMP_file_size[i]);
    write_messages_from_tmp_box(...);
    [...] // deleting file, etc.
    free_tmp_box(tmp_box);
    leave_critical_section(&lock); // lock = arbitrary value!
  }
  [...]
}
```

WHAT TO OVERWRITE?

The only interesting one you can overwrite without crashing...

```
void parse_all_TMP() {
  [...]
  tmp_box tmp_box;
  critical_section* lock; // overwritten by overflow in parse_TMP_file!
  [...]
  for(int i = 0; i < TMP_file_count; i++) {
    enter_critical_section(&lock, &global_lock); // restore lock value!
    file_buffer = malloc(TMP_file_size[i]);
    [...] // file reading, etc.
    parse_TMP_file(&tmp_box, file_buffer, TMP_file_size[i]);
    write_messages_from_tmp_box(...);
    [...] // deleting file, etc.
    free_tmp_box(tmp_box);
    leave_critical_section(&lock); // lock = arbitrary value!
  }
  [...]
}
```

CRITICAL SECTION?

```
void leave_critical_section(critical_section** lock_ptr) {
        *lock_ptr->count--;
        [...] // actual unlocking code...
}
```

CRITICAL SECTION?

```
void leave_critical_section(critical_section** lock_ptr) {
        *lock_ptr->count--;
        [...] // actual unlocking code...
}
```

By overwriting lock_ptr we can decrement a value at an arbitrary address!

ALLOCATION MODES?

```
void free_tmp_box(tmp_box* box) {
  if(box->header.msg_count && glob_tmp_box_alloc_mode != POINTER_MODE){
    for(int i = 0; i < box->header.msg_count; i++) {
      if(box->msg_pointers[i]) {
        free(box->msg_pointers[i]);
        box->msg_pointers[i] = NULL;
      }
      [...]
  }
}
```

ALLOCATION MODES?

```
void free_tmp_box(tmp_box* box) {
  if(box->header.msg_count && glob_tmp_box_alloc_mode != POINTER_MODE){
    for(int i = 0; i < box->header.msg_count; i++) {
      if(box->msg_pointers[i]) {
        free(box->msg_pointers[i]);
        box->msg_pointers[i] = NULL;
      }
      [...]
  }
}
```

We can decrement **glob_tmp_box_alloc_mode**...
We fully control data pointed by **msg_pointers**...

We can make it free some crafted fake chunks!

WE HAVE A PROBLEM...

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  [...]

  memcpy(dst, file_buffer, sizeof(tmp_box_header)); // copy header
  if(dst->header.size != file_size)                 // if invalid size
    return 0xC8E1086A;                              // return error

  while(dst->header.msg_count > i && end_of_file > current_msg) {
    [...]
    glob_tmp_box_alloc_mode = POINTER_MODE;
  }
  [...]
}
```

WE HAVE A PROBLEM...

...the allocation mode is restored when loading a TMP file!

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  [...]

  memcpy(dst, file_buffer, sizeof(tmp_box_header)); // copy header
  if(dst->header.size != file_size)                 // if invalid size
    return 0xC8E1086A;                              // return error

  while(dst->header.msg_count > i && end_of_file > current_msg) {
    [...]
    glob_tmp_box_alloc_mode = POINTER_MODE;
  }
  [...]
}
```

WE HAVE A PROBLEM...

...the allocation mode is restored when loading a TMP file!

```
int load_TMP_file(tmp_box* dst, tmp_file* file_buffer,
        size_t file_size) {
  [...]

  memcpy(dst, file_buffer, sizeof(tmp_box_header)); // copy header
  if(dst->header.size != file_size)                 // if invalid size
    return 0xC8E1086A;                              // return error

  while(dst->header.msg_count > i && end_of_file > current_msg) {
    [...]
    glob_tmp_box_alloc_mode = POINTER_MODE;
  }
  [...]
}
```

We could bypass this by crafting an invalid header, but an error is returned...

WHATEVER...

```
void parse_all_TMP() {
  [...]
  tmp_box tmp_box;
  critical_section* lock;
  [...]
  for(int i = 0; i < TMP_file_count; i++) {
    enter_critical_section(&lock, &global_lock);
    file_buffer = malloc(TMP_file_size[i]);
    [...] // file reading, etc.
    parse_TMP_file(&tmp_box, file_buffer, TMP_file_size[i]);
    write_messages_from_tmp_box(...);
    [...] // deleting file, etc.
    free_tmp_box(tmp_box);
    leave_critical_section(&lock);
  }
  [...]
}
```

WHATEVER...

```
void parse_all_TMP() {
  [...]
  tmp_box tmp_box;
  critical_section* lock;
  [...]
  for(int i = 0; i < TMP_file_count; i++) {
    enter_critical_section(&lock, &global_lock);
    file_buffer = malloc(TMP_file_size[i]);
    [...] // file reading, etc.
    parse_TMP_file(&tmp_box, file_buffer, TMP_file_size[i]);
    write_messages_from_tmp_box(...);
    [...] // deleting file, etc.
    free_tmp_box(tmp_box);
    leave_critical_section(&lock);
  }
  [...]
}
```

...they do not check the return value anyway!

WHAT CAN WE DO SO FAR?

WHAT CAN WE DO SO FAR?

- send a first TMP box
  - overwrite the lock & decrement the alloc mode

WHAT CAN WE DO SO FAR?

- send a first TMP box
  - overwrite the lock & decrement the alloc mode
- send a second TMP box (invalid header)
  - parser returns early so **msg_pointers** is not updated
  - all pointers in **msg_pointers** are freed

WHAT CAN WE DO SO FAR?

- send a first TMP box
  - overwrite the lock & decrement the alloc mode
- send a second TMP box (invalid header)
  - parser returns early so **msg_pointers** is not updated
  - all pointers in **msg_pointers** are freed

Pointers in **msg_pointers** still point to the freed chunk of the first TMP file...

WHAT CAN WE DO SO FAR?

- send a first TMP box
  - overwrite the lock & decrement the alloc mode
- send a second TMP box (invalid header)
  - parser returns early so **msg_pointers** is not updated
  - all pointers in **msg_pointers** are freed

Pointers in **msg_pointers** still point to the freed chunk of the first TMP file...

...which gets reallocated for the second TMP file!

WHAT CAN WE DO SO FAR?

- send a first TMP box
  - overwrite the lock & decrement the alloc mode
- send a second TMP box (invalid header)
  - parser returns early so **msg_pointers** is not updated
  - all pointers in **msg_pointers** are freed

Pointers in **msg_pointers** still point to the freed chunk of the first TMP file...

...which gets reallocated for the second TMP file!

So it frees pointers to our controlled buffer!

WHAT DO WE DO NEXT?

We can craft fake heap chunks which will get freed...

WHAT DO WE DO NEXT?

We can craft fake heap chunks which will get freed...

- the 3DS heap is insecure
  - classic unsafe-unlink
  - one arbitrary write for each chunk you free

WHAT DO WE DO NEXT?

We can craft fake heap chunks which will get freed...

- the 3DS heap is insecure
  - classic unsafe-unlink
    - one arbitrary write for each chunk you free
- rewrite the heap free-list head pointer
  - make it point to the stack
  - next malloc call will return a pointer to the stack!

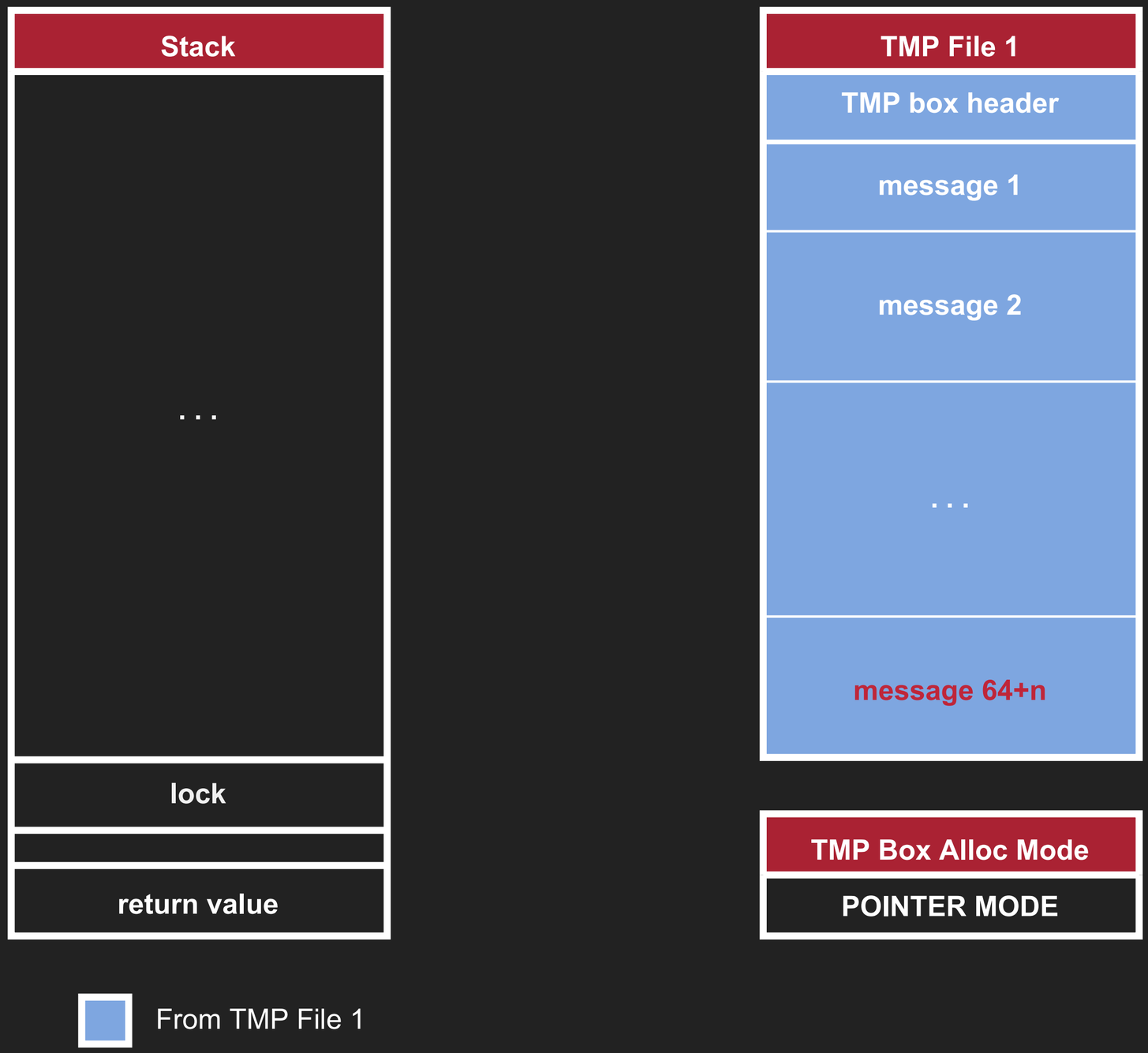
WHAT DO WE DO NEXT?

We can craft fake heap chunks which will get freed...

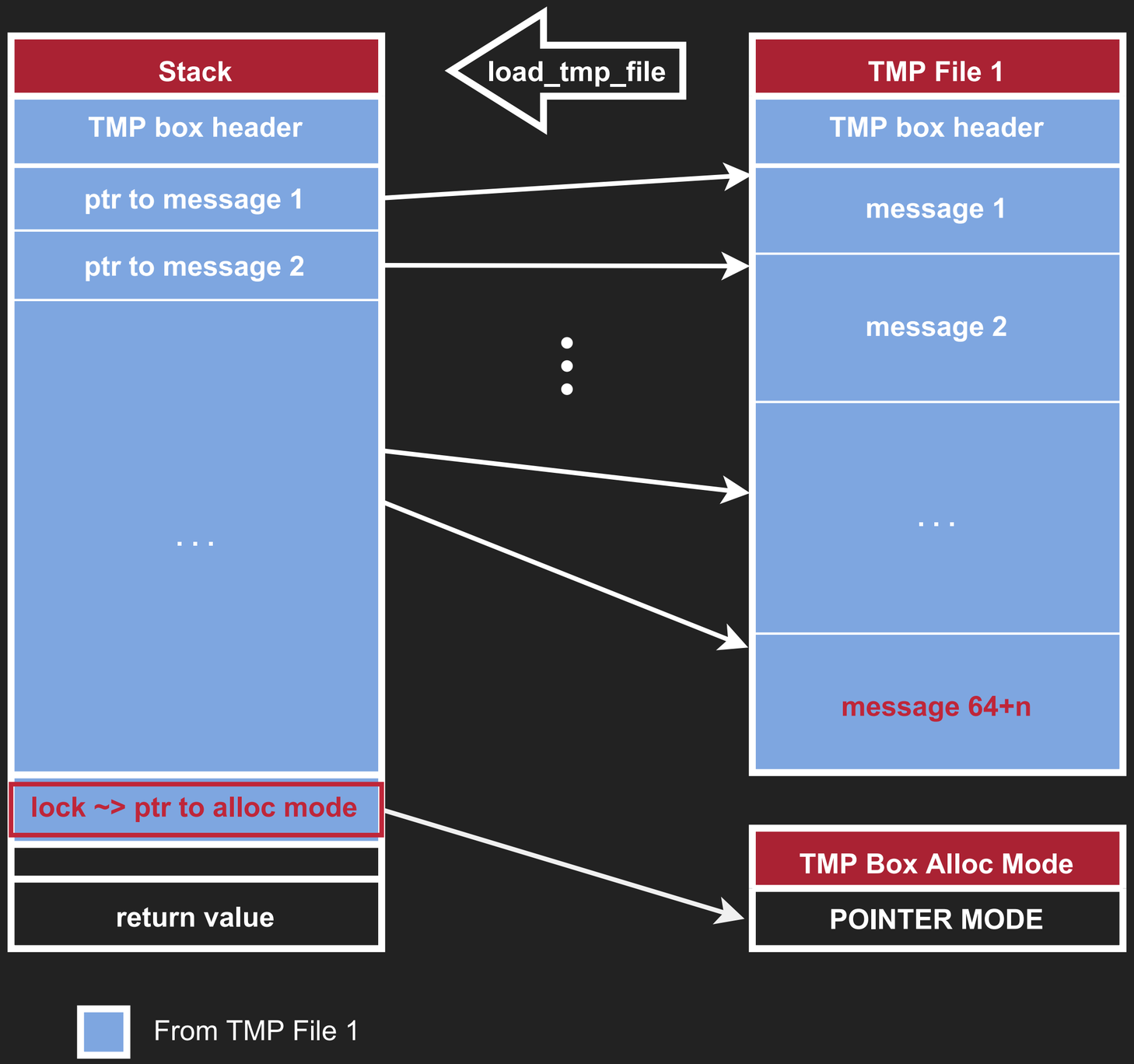
- the 3DS heap is insecure
  - classic unsafe-unlink
    - one arbitrary write for each chunk you free
- rewrite the heap free-list head pointer
  - make it point to the stack
  - next malloc call will return a pointer to the stack!

The third TMP file buffer will be allocated on the stack!

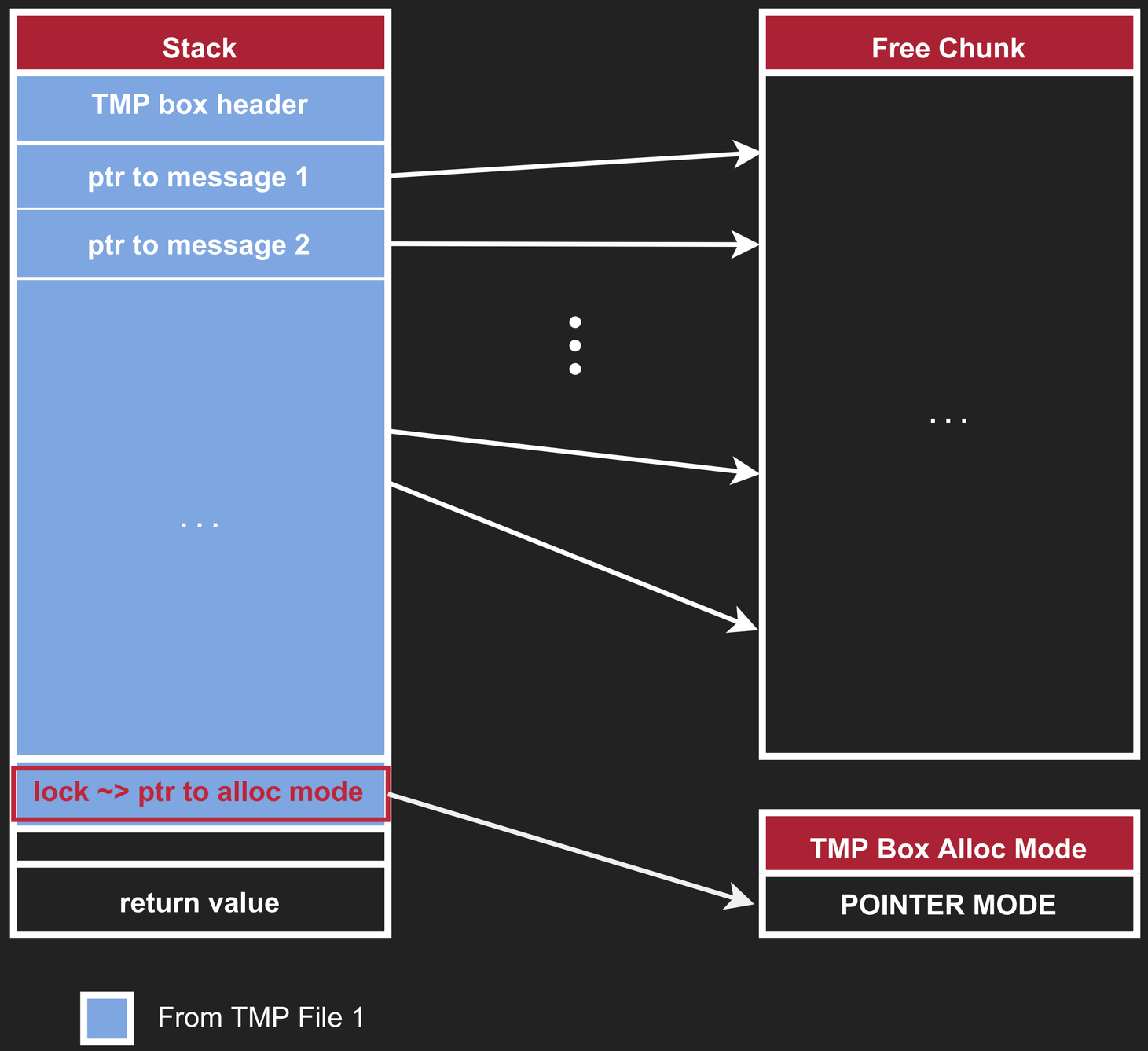
EXPLOIT SCENARIO



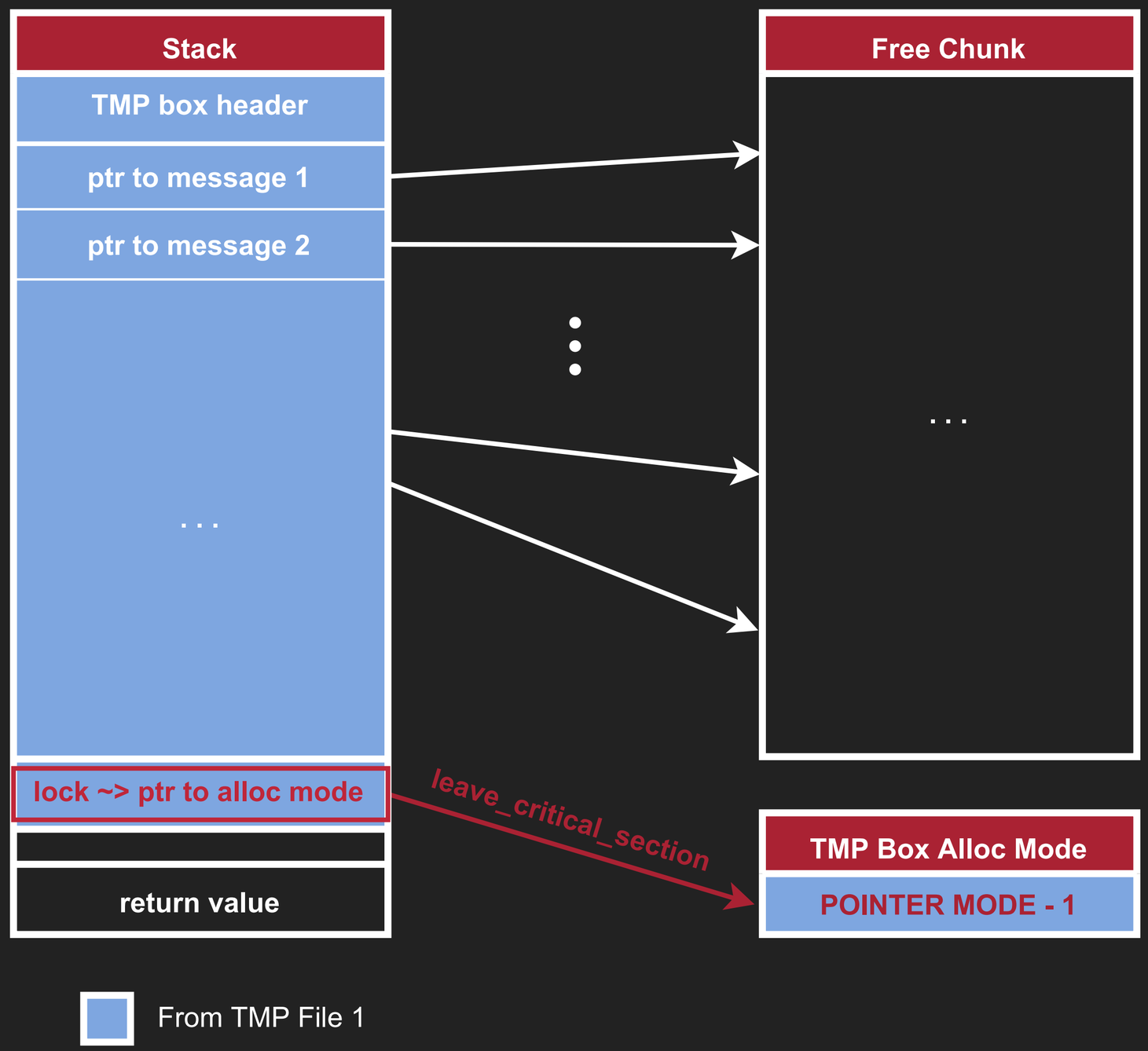
EXPLOIT SCENARIO



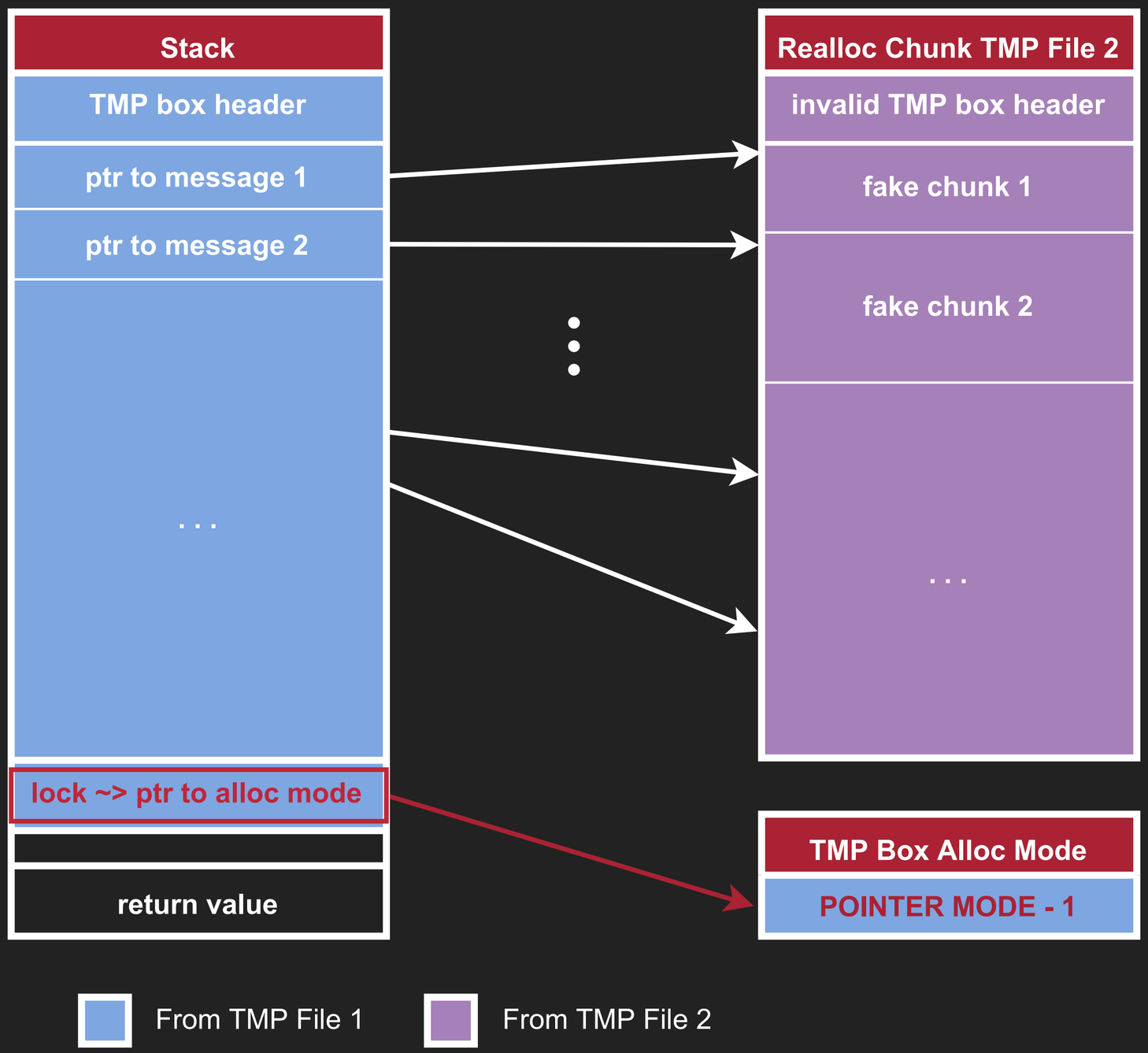
EXPLOIT SCENARIO



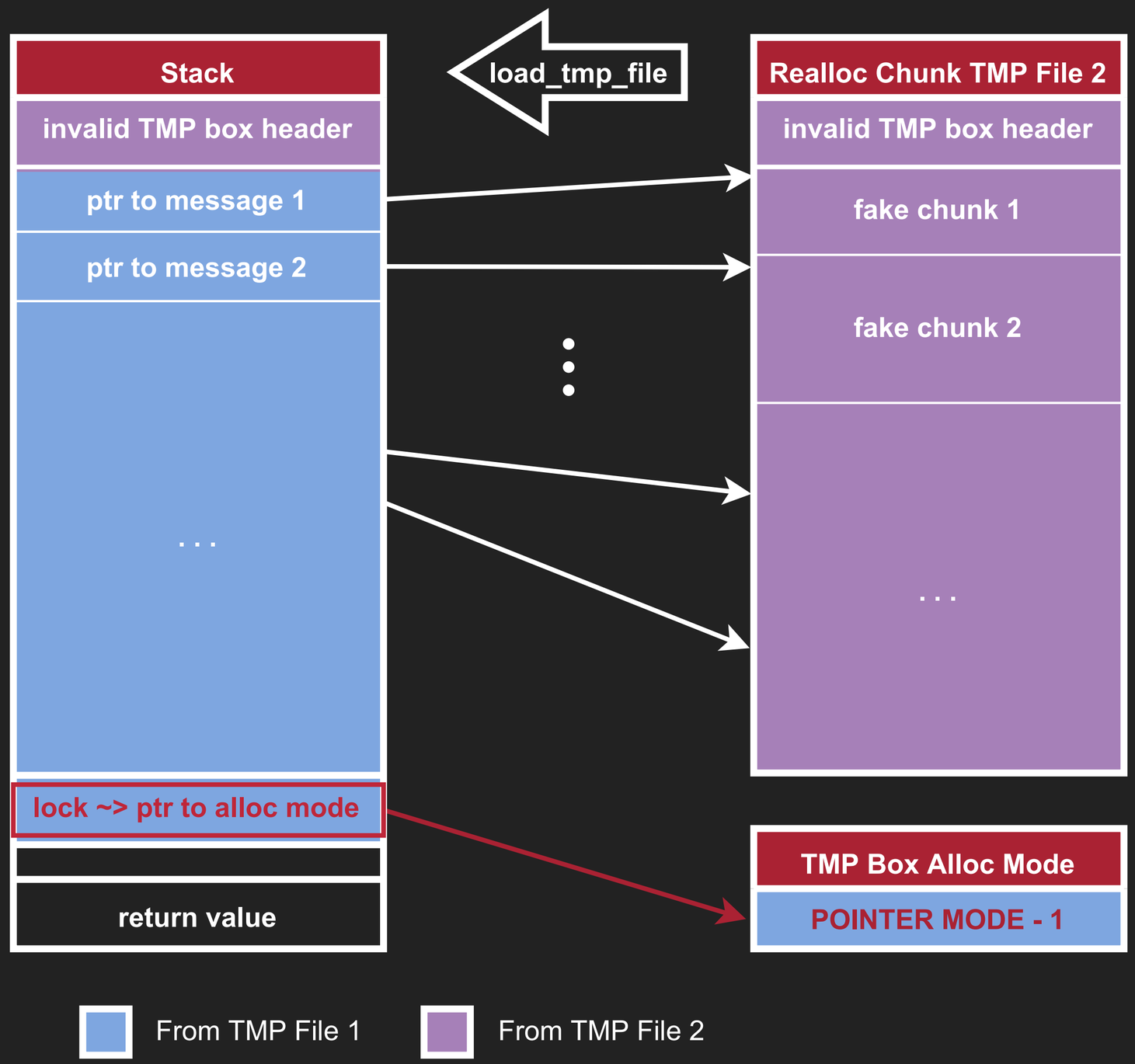
EXPLOIT SCENARIO



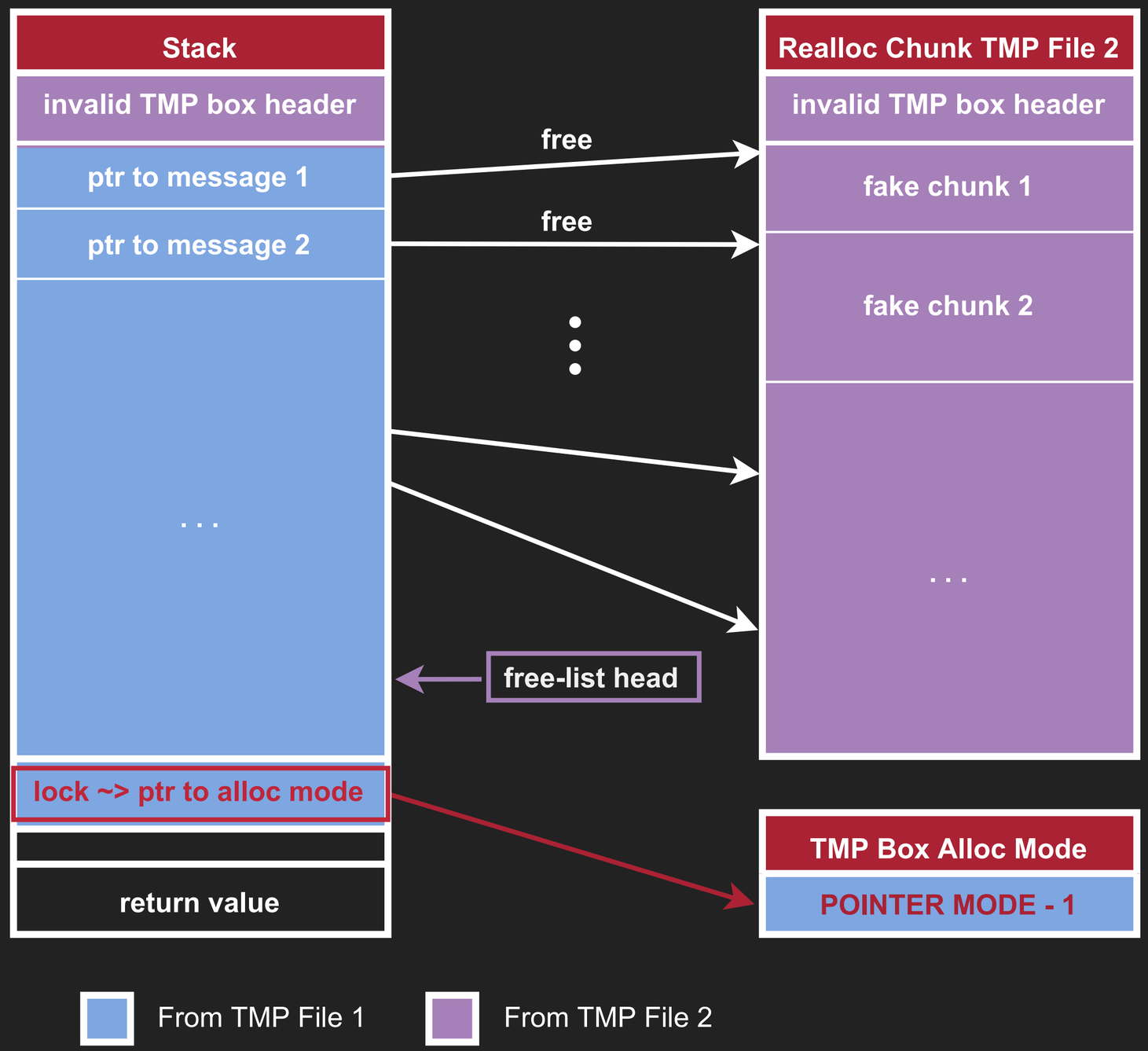
EXPLOIT SCENARIO



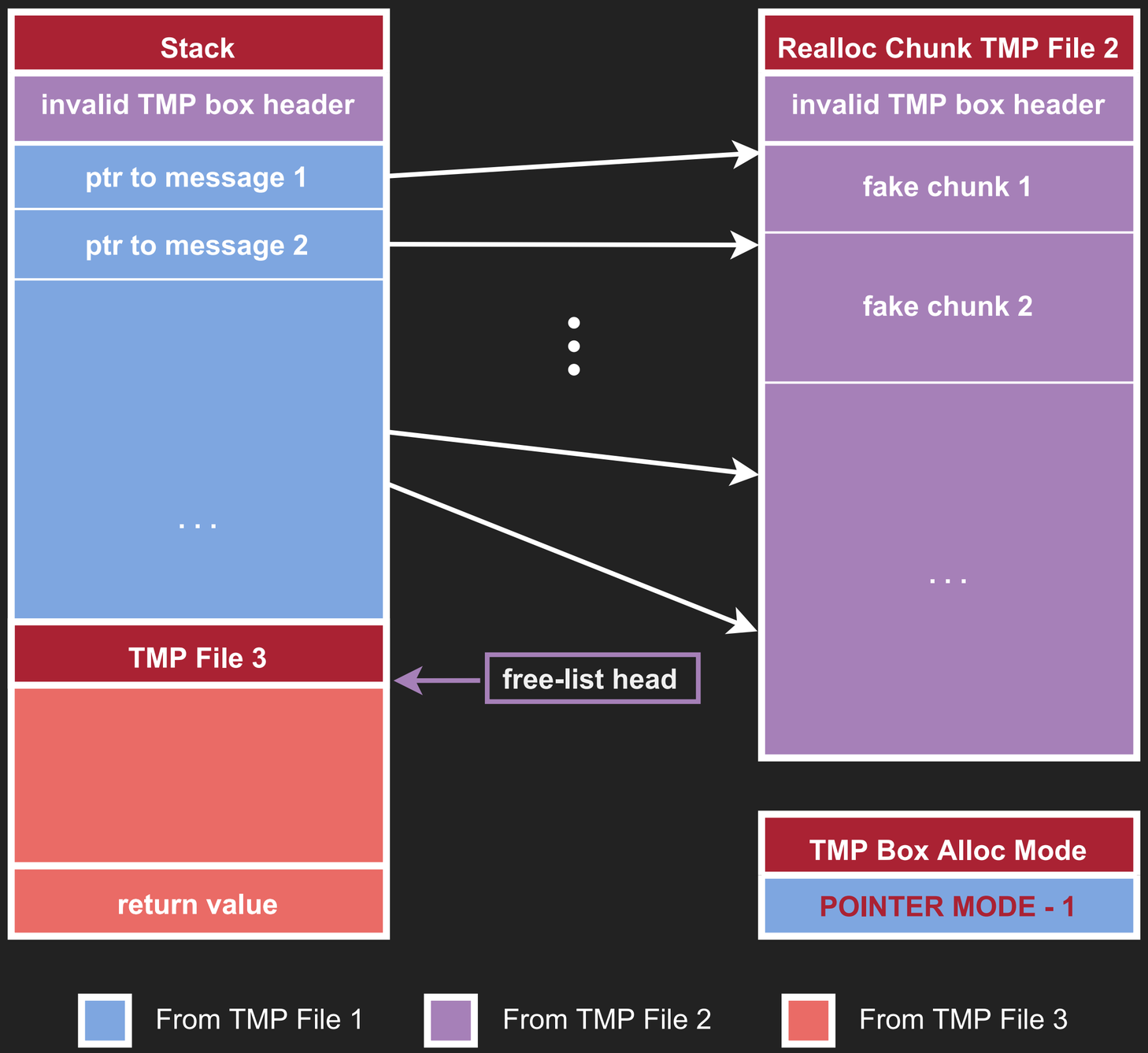
EXPLOIT SCENARIO



EXPLOIT SCENARIO



EXPLOIT SCENARIO



SECOND RCE IN CECD ✓

This one was trickier!


AND
ANOTHER ONE

AGAIN?

AGAIN?

There is another one in the message parser...

AGAIN?

There is another one in the message parser...

which is a SDK function...

AGAIN?

There is another one in the message parser...
which is a SDK function...
so any application using StreetPass is vulnerable!

AGAIN?

There is another one in the message parser...
which is a SDK function...
so any application using StreetPass is vulnerable!

This problem is left as an exercise for the reader...

THIRD RCE IN CECD ✓

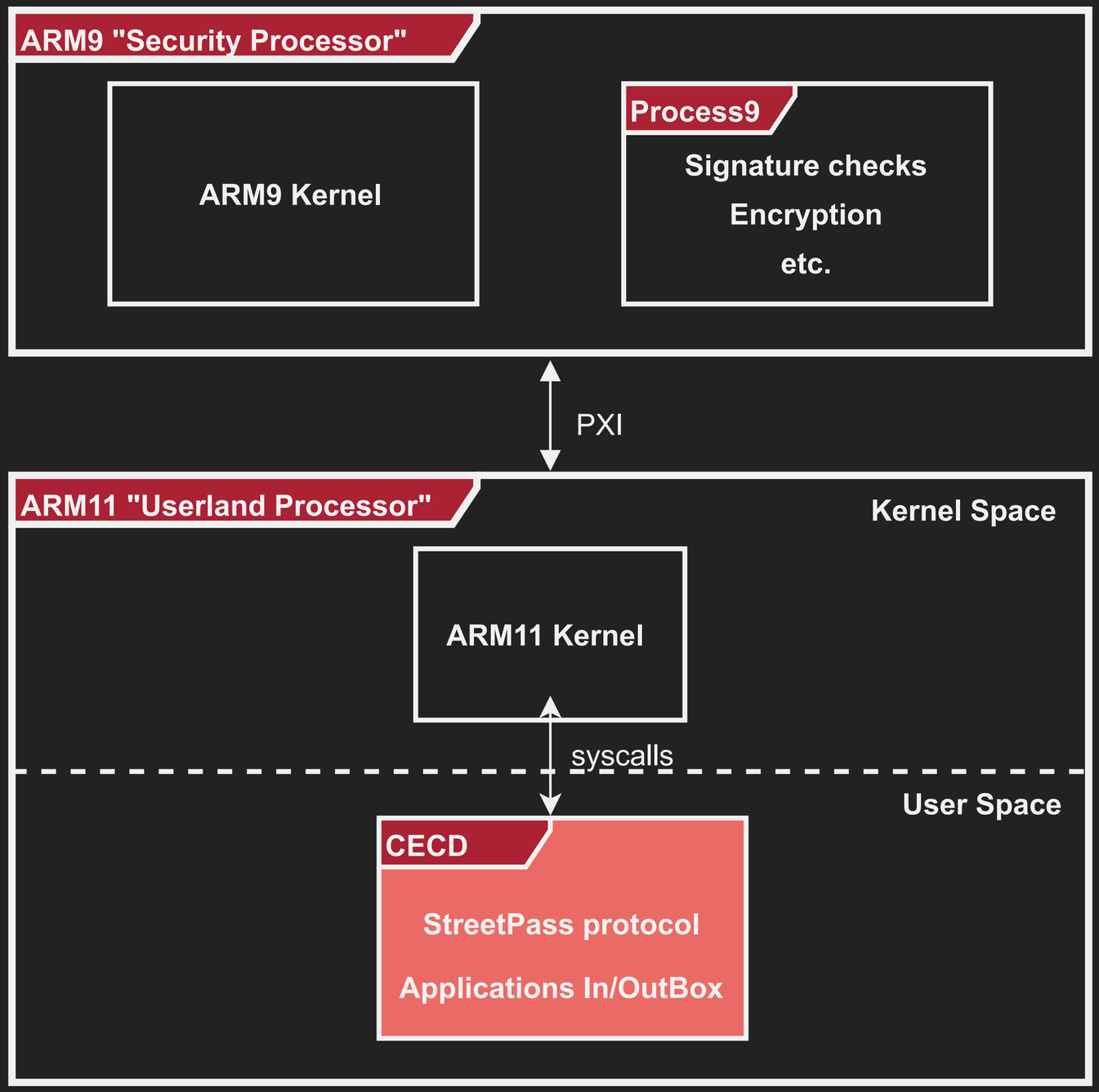
Code execution in any application using StreetPass ✓

Persistent backdoor in CECD ✓

POST-EXPLOITATION

SANDBOX ESCAPE

CECD does not have much privileges... we want more!



TAKING OVER THE HOME MENU

OUTBOX INDEX READER

Another SDK vulnerability!

```
size_t index_file_size;
CECD_open_file(..., &size); //done through IPC
[...]
void* index_buffer = malloc(0x800); //what if size is > 0x800?
[...]
CECD_read_file(..., index_buffer, size, ...); //done through IPC
```

OUTBOX INDEX READER

Another SDK vulnerability!

```
size_t index_file_size;
CECD_open_file(..., &size); //done through IPC
[...]
void* index_buffer = malloc(0x800); //what if size is > 0x800?
[...]
CECD_read_file(..., index_buffer, size, ...); //done through IPC
```

We are CECD now, we can provide a file larger than 0x800 bytes... and trigger a heap overflow!

OUTBOX INDEX READER

Another SDK vulnerability!

```
size_t index_file_size;
CECD_open_file(..., &size); //done through IPC
[...]
void* index_buffer = malloc(0x800); //what if size is > 0x800?
[...]
CECD_read_file(..., index_buffer, size, ...); //done through IPC
```

We are CECD now, we can provide a file larger than 0x800 bytes... and trigger a heap overflow!

Enough to take over the home menu!

...and any application that uses this function...

ESCAPE TO THE HOME MENU ✓

Access to the internet ✓

Access to the SD card ✓

Drawing on screen ✓

...

TAKING OVER THE ARM11 KERNEL

IPC: STATIC BUFFERS

How does one send data from a sender process to a receiver process?

IPC: STATIC BUFFERS

How does one send data from a sender process to a receiver process?

- for regular large buffers
  - map parts of the sender's memory into the receiver's

IPC: STATIC BUFFERS

How does one send data from a sender process to a receiver process?

- for regular large buffers
  - map parts of the sender's memory into the receiver's
- for regular small buffers
  - receiver can register some static buffers
  - copy from the sender's buffer to the receiver's buffer done by the **ARM11 kernel**

IPC: STATIC BUFFERS

How does one send data from a sender process to a receiver process?

- for regular large buffers
  - map parts of the sender's memory into the receiver's
- for regular small buffers
  - receiver can register some static buffers
  - copy from the sender's buffer to the receiver's buffer done by the **ARM11 kernel**
- for buffers sent to the ARM9 (over PXI)
  - ARM11 kernel writes pairs of {physical address, size} to static buffers for the ARM9 side to understand
  - copy of data done by **Process9 (ARM9 side)** using the given **physical address**

LAZYPIXIE

Vulnerability found by **@TuxSH**!

How does the kernel handle the "PXI buffers" case?

1. check alignment of the destination static buffer
2. check size of the destination static buffer
3. check permissions for the source buffer
5. cache operations, etc.
6. copy metadata to the destination static buffer

LAZYPIXIE

Vulnerability found by **@TuxSH**!

How does the kernel handle the "PXI buffers" case?

1. check alignment of the destination static buffer
2. check size of the destination static buffer
3. check permissions for the source buffer
4. **check permissions for the destination buffer**
5. cache operations, etc.
6. copy metadata to the destination static buffer

The destination can be an arbitrary physical address!

LAZYPIXIE

Vulnerability found by **@TuxSH**!

How does the kernel handle the "PXI buffers" case?

1. check alignment of the destination static buffer
2. check size of the destination static buffer
3. check permissions for the source buffer

4. check permissions for the destination buffer

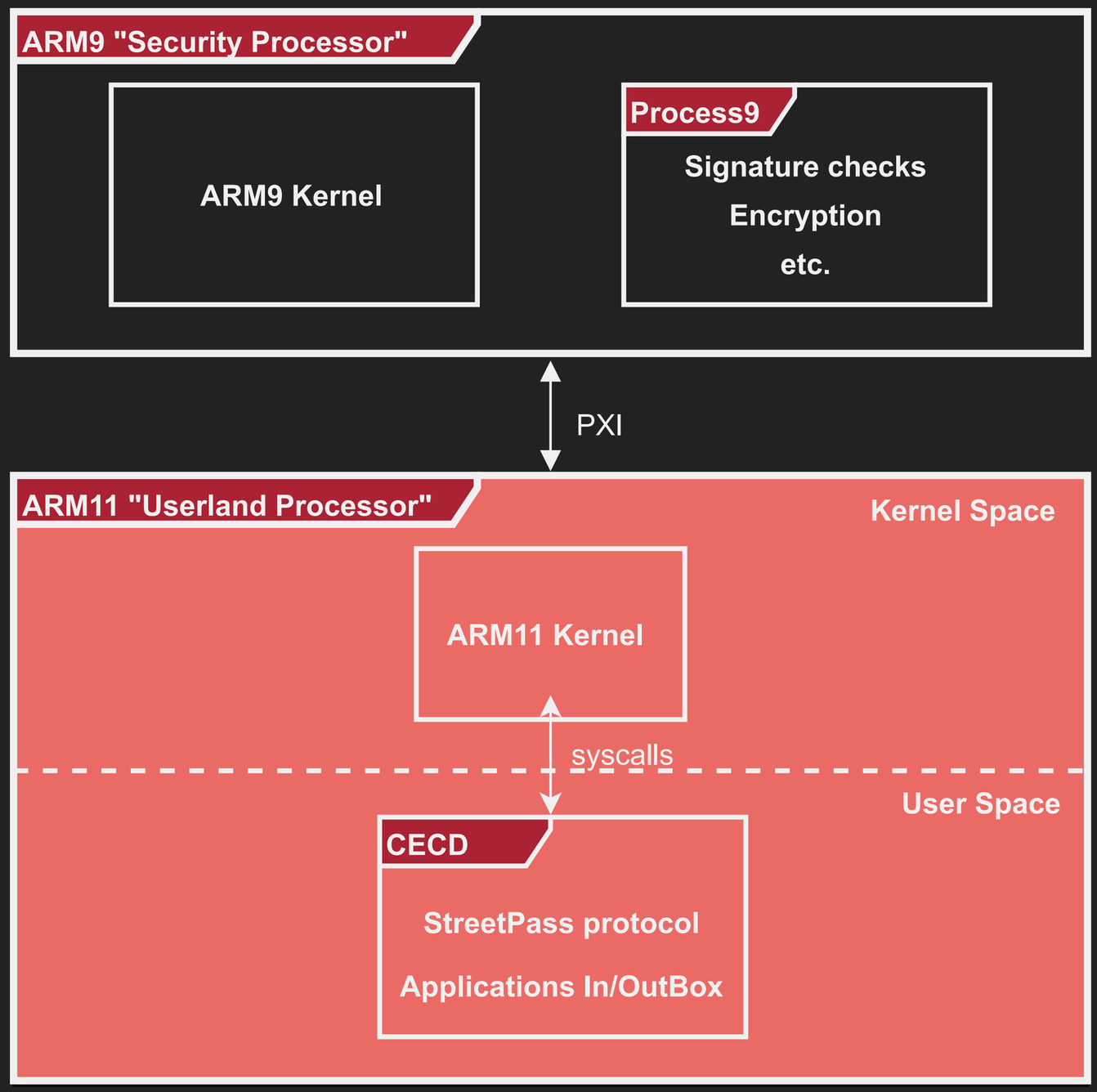
5. cache operations, etc.
6. copy metadata to the destination static buffer

The destination can be an arbitrary physical address!

Just overwrite the MMU table and make the kernel

ARM11 KERNEL HAS FALLEN!

... but we still want more!



ROAD TO FULL SYSTEM CONTROL

...or why pwning CECD was the best idea ever!

SAFEHAX

- race condition in firmware header parsing
  - take over ARM9 if you control ARM11 kernel
- fixed in version 9.5.0 for regular (native) firmware
  - remains unfixed in safe mode firmware
- mitigated in version 11.3.0 and 11.4.0

SAFEHAX

- race condition in firmware header parsing
  - take over ARM9 if you control ARM11 kernel
- fixed in version 9.5.0 for regular (native) firmware
  - remains unfixed in safe mode firmware
- mitigated in version 11.3.0 and 11.4.0

Mitigated? Not Patched?

SAFEHAX

- race condition in firmware header parsing
  - take over ARM9 if you control ARM11 kernel
- fixed in version 9.5.0 for regular (native) firmware
  - remains unfixed in safe mode firmware
- mitigated in version 11.3.0 and 11.4.0

Mitigated? Not Patched?

How do they prevent it?

SAFEHAX "MITIGATION"

SAFEHAX "MITIGATION"

- add a global boolean flag on ARM9 side
  - set to 1 ⇒ panics when trying to launch safe mode firmware

SAFEHAX "MITIGATION"

- add a global boolean flag on ARM9 side
  - set to 1 ⇒ panics when trying to launch safe mode firmware
- flag set to 1 when applications are launched
  - except for home menu and system modules

SAFEHAX "MITIGATION"

- add a global boolean flag on ARM9 side
  - set to 1 ⇒ panics when trying to launch safe mode firmware
- flag set to 1 when applications are launched
  - except for home menu and system modules

We can get RCE in cecd without launching any application...

SAFEHAX "MITIGATION"

- add a global boolean flag on ARM9 side
  - set to 1 ⇒ panics when trying to launch safe mode firmware
- flag set to 1 when applications are launched
  - except for home menu and system modules

We can get RCE in cecd without launching any application...
...with a ARM11 kernel exploit we can leverage safehax!

FULL CONTROL RCE ✓

FULL CONTROL RCE ✓

WITHOUT USER INTERACTION ✓

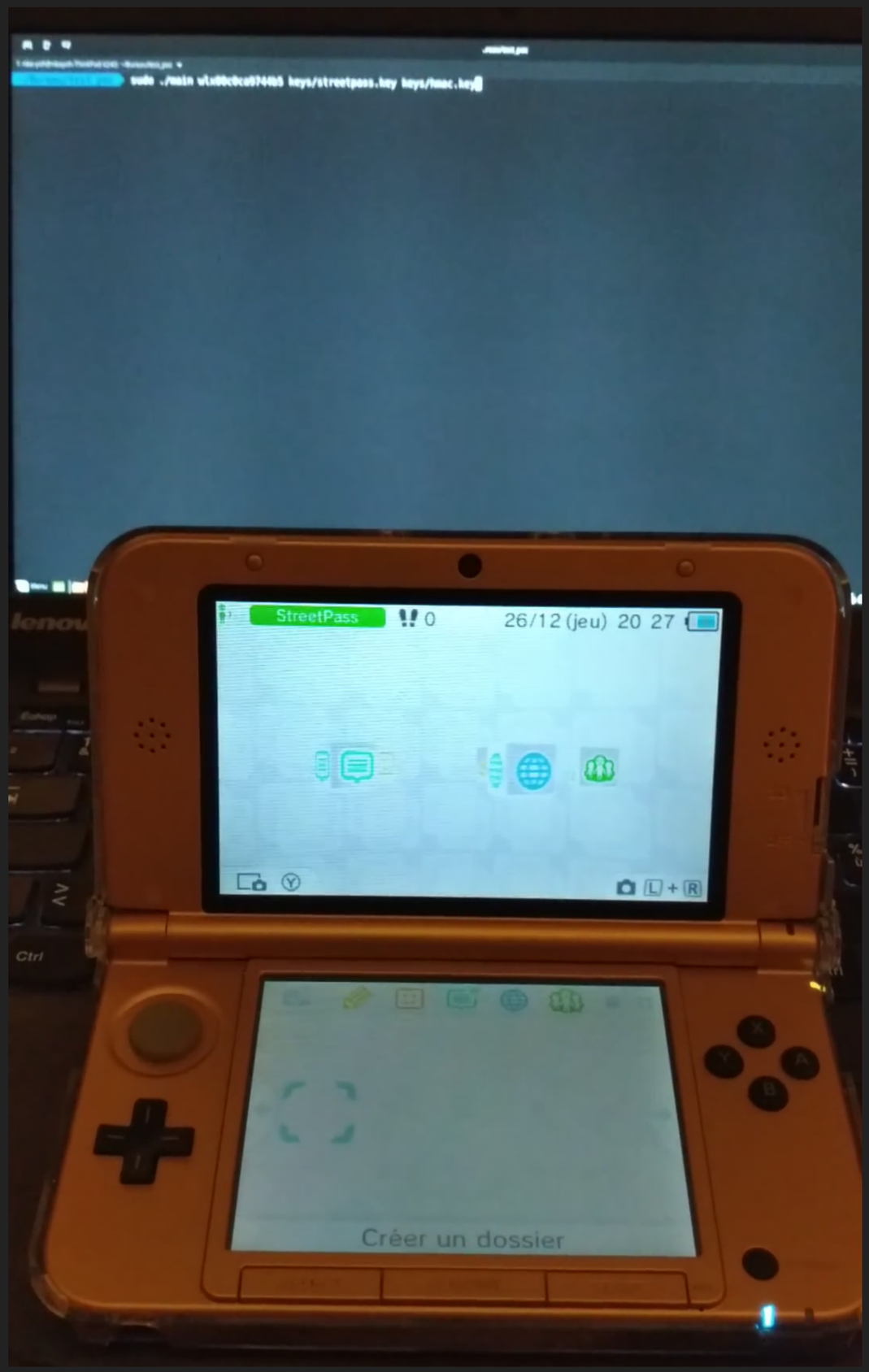
FULL CONTROL RCE ✓

WITHOUT USER INTERACTION ✓

ON ANY FIRMWARE VERSION ✓

at the time this was developed... fixed on version 11.12!

DEMO TIME!


sudo ./main
StreetPass
0
26/12 (jeu) 20 27
Créer un dossier

SOME TAKEAWAYS

- you'd better check your return values
- don't hide behind cryptography
  - your encryption might get broken faster than you think
- assessing hard-to-reach features is arduous but can lead to amazing (yet dangerous) results!
- fix you flaws
  - don't implement poor mitigations...
- there're still things to do on 3DS!
  - amazing system to work on
  - check out the documentation on **3DBrew**!

ACKNOWLEDGEMENTS

- **@TuxSH**: LazyPixie, joint effort on leveraging safehax
- **@hedgeberg**: recurrent support, help with so many things it would not fit in the slide...
- **3DBrew contributors**: amazing documentation
- **Nintendo**: allowing me to talk about these great things and patching the flaws

CONTACT

I'm a first year master's degree student looking for a great research internship!

Twitter: **@MrNbaYoh**

Email: **mrnbayoh(at)gmail(dot)com**