Can we exploit buggy P4 programs?

Mihai-Valentin Dumitru, Dragoș Dumitrescu, Costin Raiciu

University Politehnica of Bucharest

Funding from
Cornet H2020

Bugs and undefined behaviors

Undefined behaviors have unpredictable, unreliable effects

What *really happens* depends on the platform

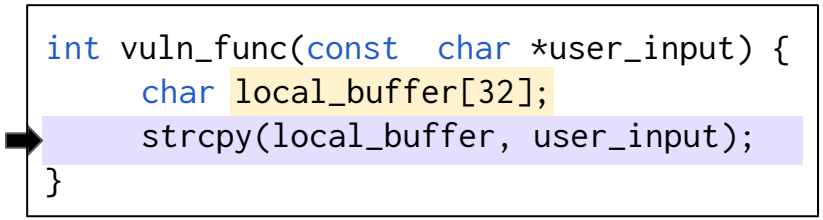
Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

Bugs and exploits in the world of C



Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

What if `strlen(user_input) >= 32` ?

Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

What if `strlen(user_input) >= 32` ?

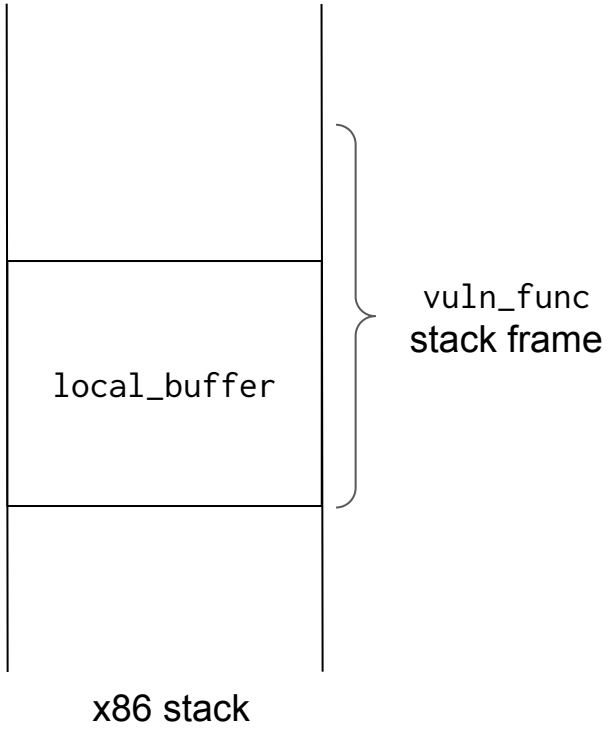
Undefined behavior

Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

What if `strlen(user_input) >= 32` ?

Undefined behavior

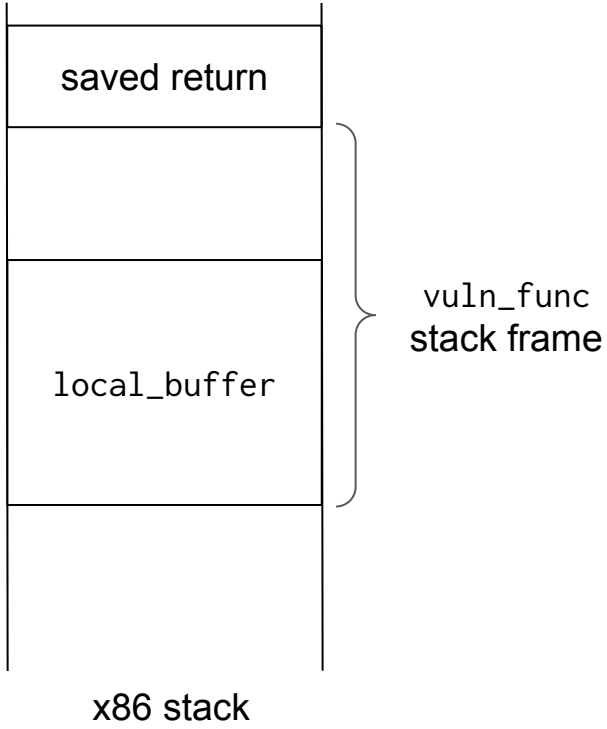


Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

What if `strlen(user_input) >= 32` ?

Undefined behavior

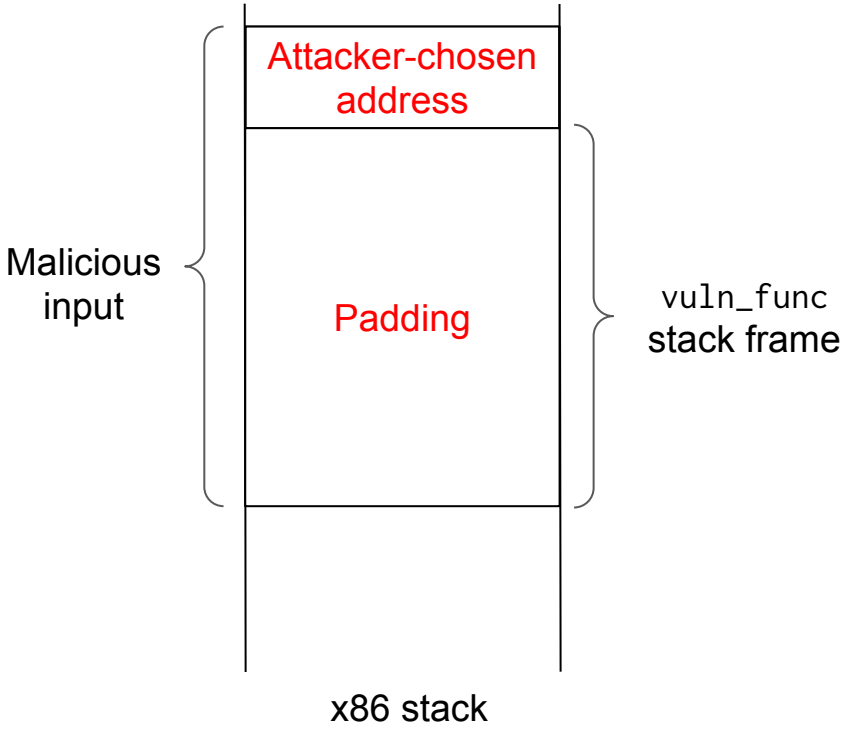


Bugs and exploits in the world of C

```
int vuln_func(const  char *user_input) {
     char local_buffer[32];
     strcpy(local_buffer, user_input);
}
```

What if `strlen(user_input) >= 32` ?

Undefined behavior



What about the P4 world?

Our Work

Document actual manifestations of undefined behaviors on:

- BMv2 simple_switch
- P4-NetFPGA
- Barefoot Tofino

Examples of exploits on P4 programs

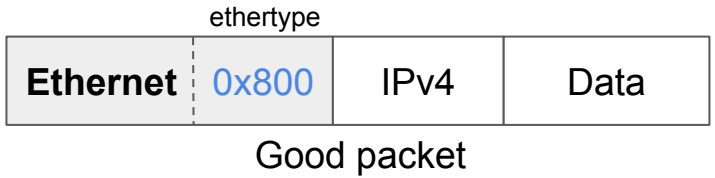
P4 headers

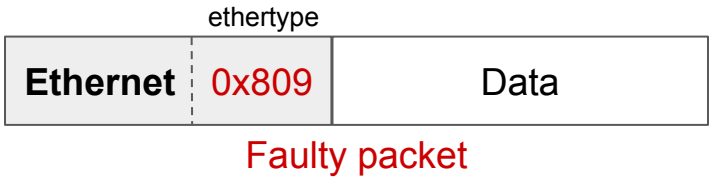
```
header ethernet_t {
     bit<48> dst;
     bit<48> src;
     bit<16> type;
}
```

```
struct headers {
     ethernet_t ethernet;
     ipv4_t ipv4;
     ipv6_t ipv6;
}
```

```
state parse_ethernet {
     packet.extract(hdr.ethernet);
     transition select(hdr.ethernet.type) {
          0x0800: parse_ipv4;
          0x86DD: parse_ipv6;
     }
}
```

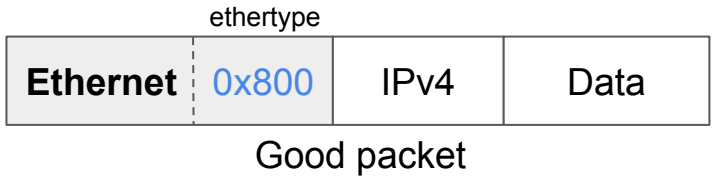
Reads from invalid headers

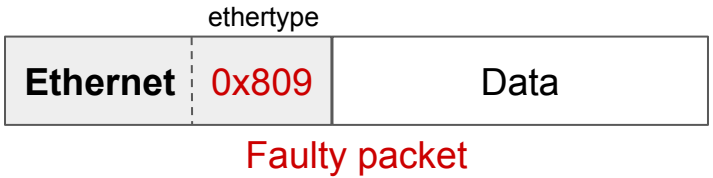





```
apply {
    if (!hdr.ipv4.isValid())
        hdr.ethernet.ether_type = hdr.ipv4.checksum;
        send_back();
}
```

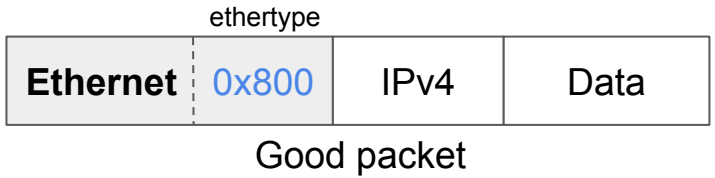
Reads from invalid headers

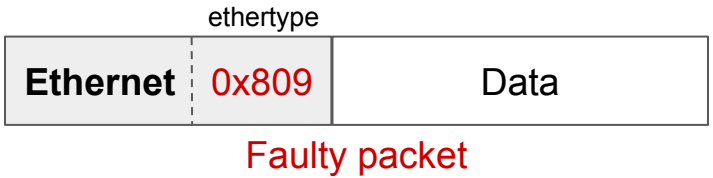





```
apply {
    if (!hdr.ipv4.isValid())
        hdr.ethernet.ether_type = hdr.ipv4.checksum;
        send_back();
}
```

Reads from invalid headers



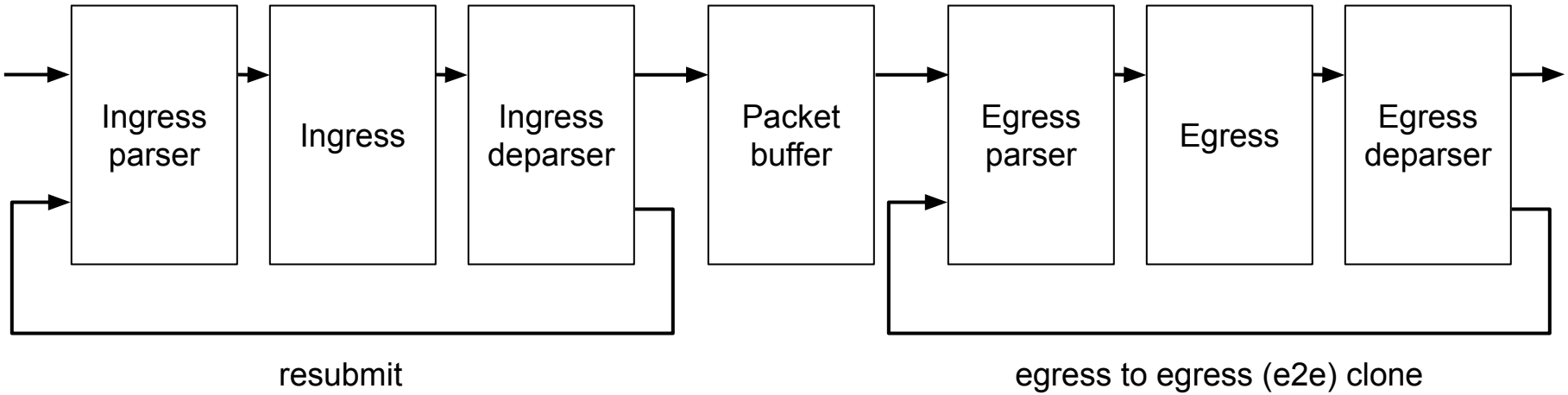



```
apply {
    if (!hdr.ipv4.isValid())
        hdr.ethernet.ether_type = hdr.ipv4.checksum;
        send_back();
}
```

Reads from invalid headers

	Value of ethertype for output packet
simple_switch	*IPv4 checksum of previous good packet*
P4-NetFPGA	Zero
Tofino	Zero

Infinite loops



Infinite loops

	resubmit	e2e clone
simple_switch	Incoming packets dropped	Flood egress port with clones
Tofino	Can only resubmit once	Flood egress port with clones
P4NetFPGA	N/A	N/A

Resurrecting dropped packets

Dropping means changing metadata:

```
meta.drop = 1;
```

OR

```
meta.output_port = DROP_PORT;
```

Resurrecting dropped packets

Dropping means changing metadata:

```
meta.drop = 1;
```

OR

```
meta.output_port = DROP_PORT;
```

```
apply(acl);    // mark as dropped => egress_spec = 511
apply(lpm);    // hit               => egress_spec = 2
```

Resurrecting dropped packets

Dropping means changing metadata:

```
meta.drop = 1;
```

OR

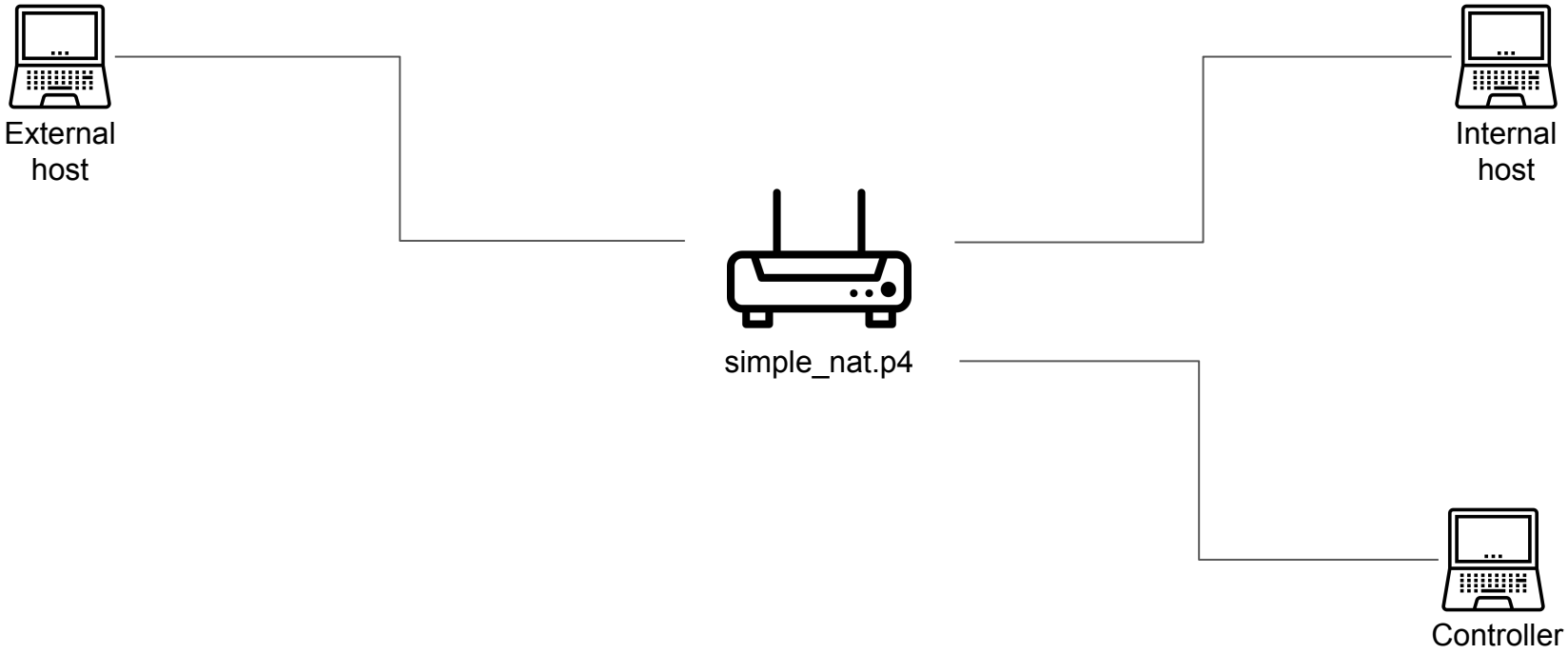
```
meta.output_port = DROP_PORT;
```

```
apply(acl);   // mark as dropped => egress_spec = 511
apply(lpm);   // hit                => egress_spec = 2
```

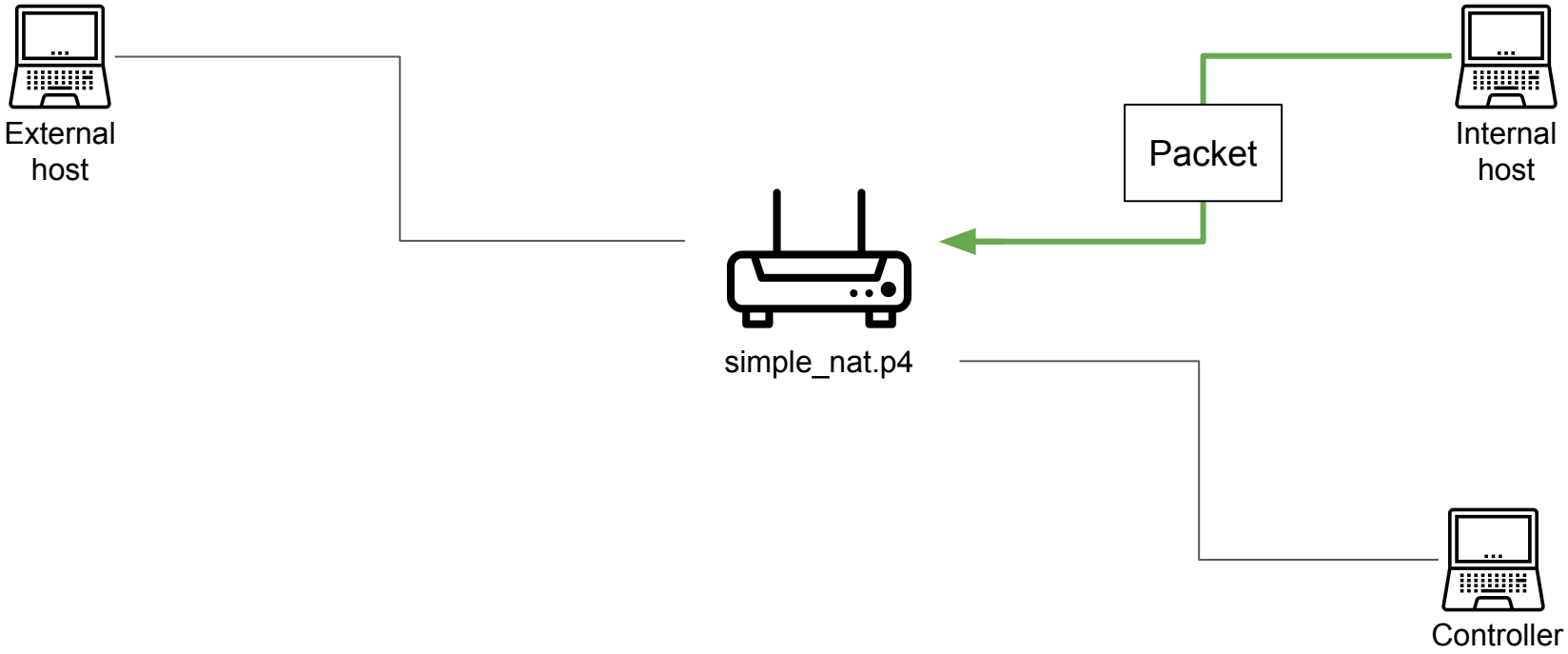
Resurrecting dropped packets

	Drop method
v1model	Set output port to a special value
SimpleSumeSwitch	Set output port to a special value
PSA	Set metadata boolean

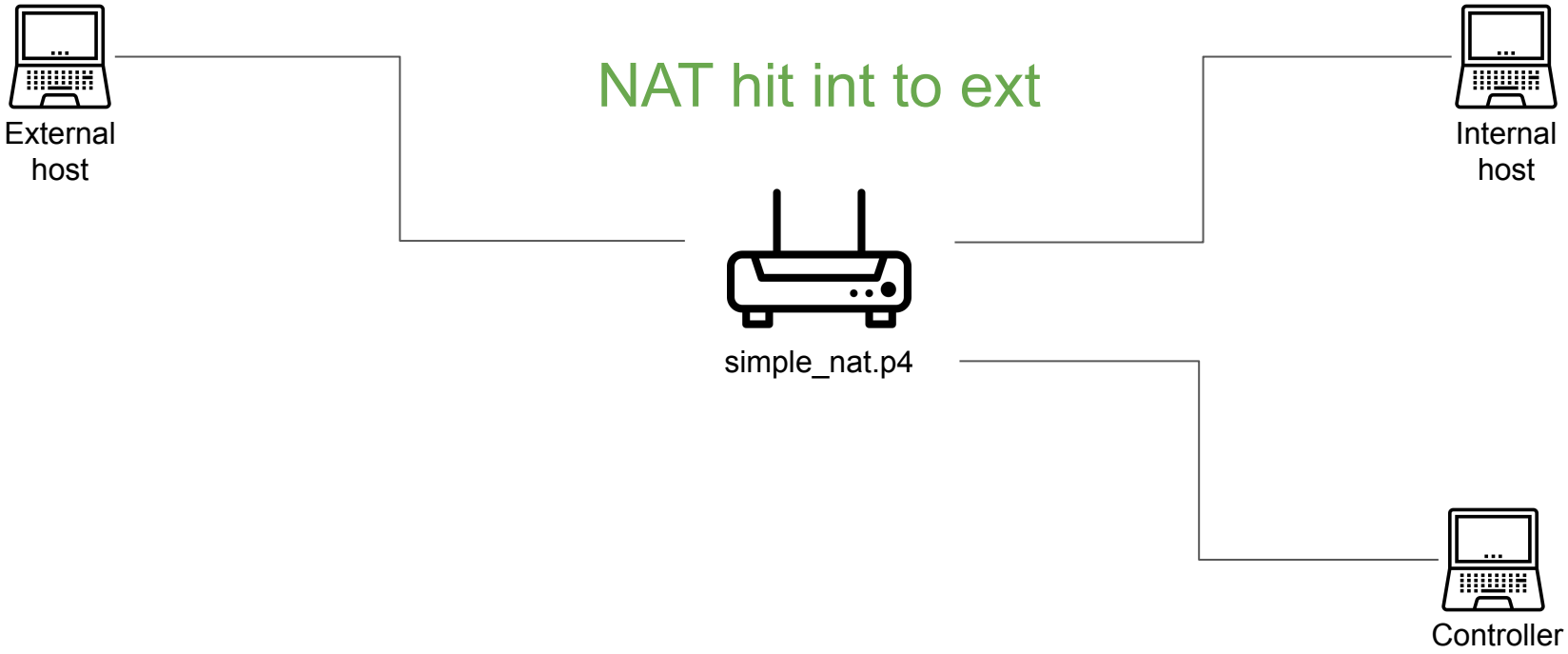
Simple NAT setup



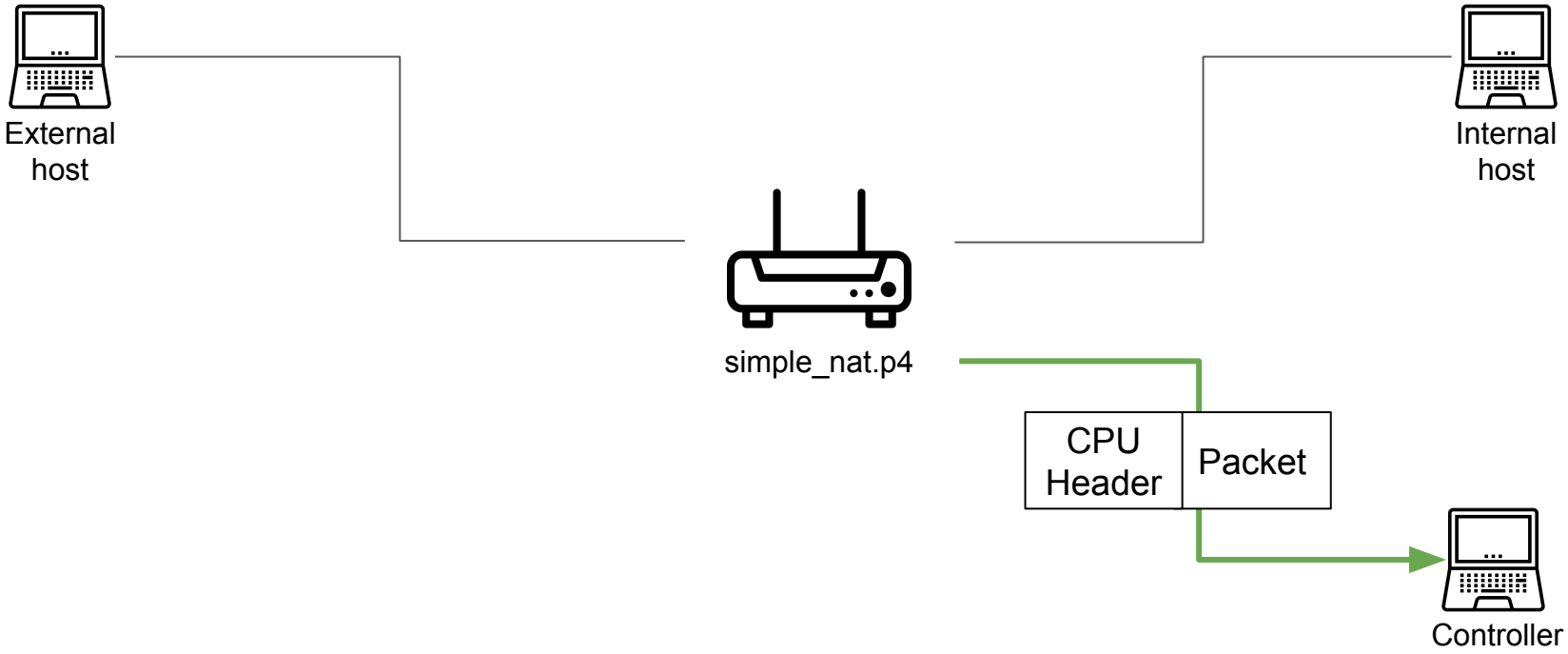
Simple NAT setup



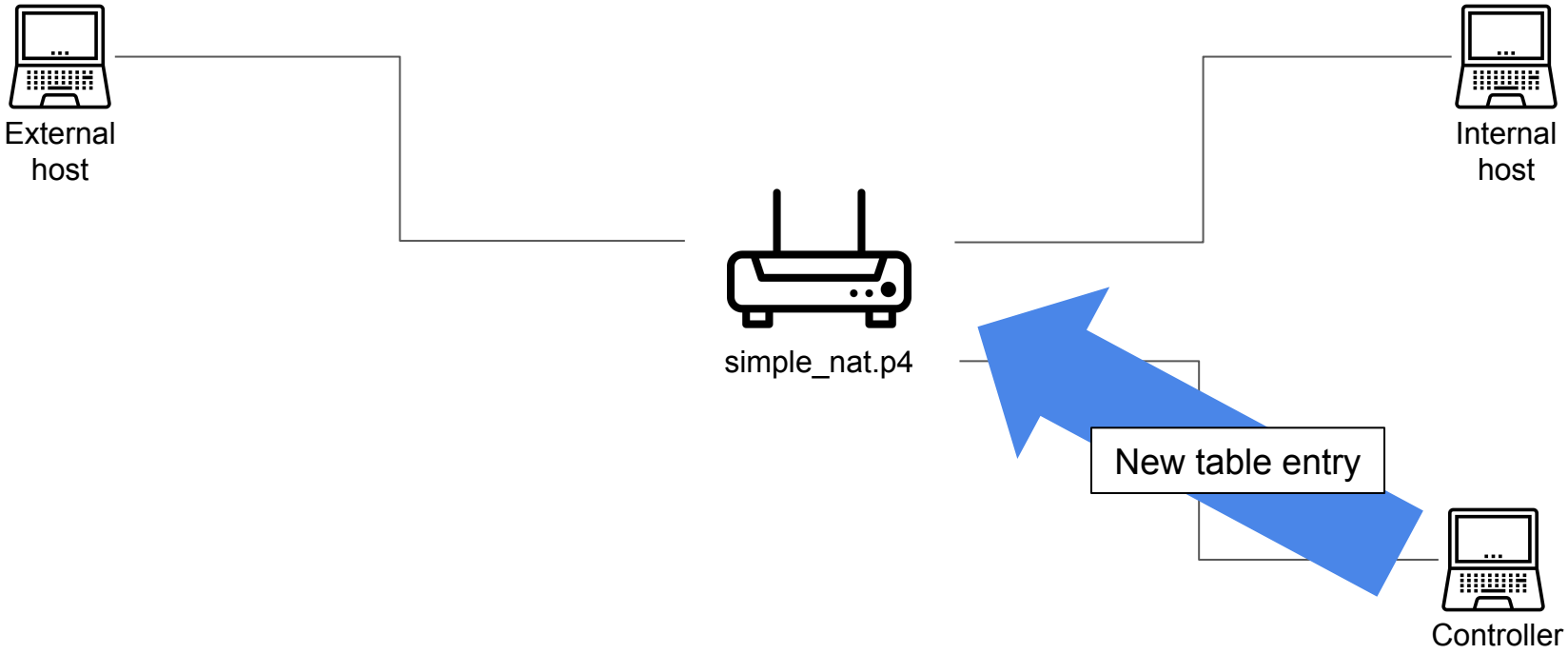
Simple NAT setup



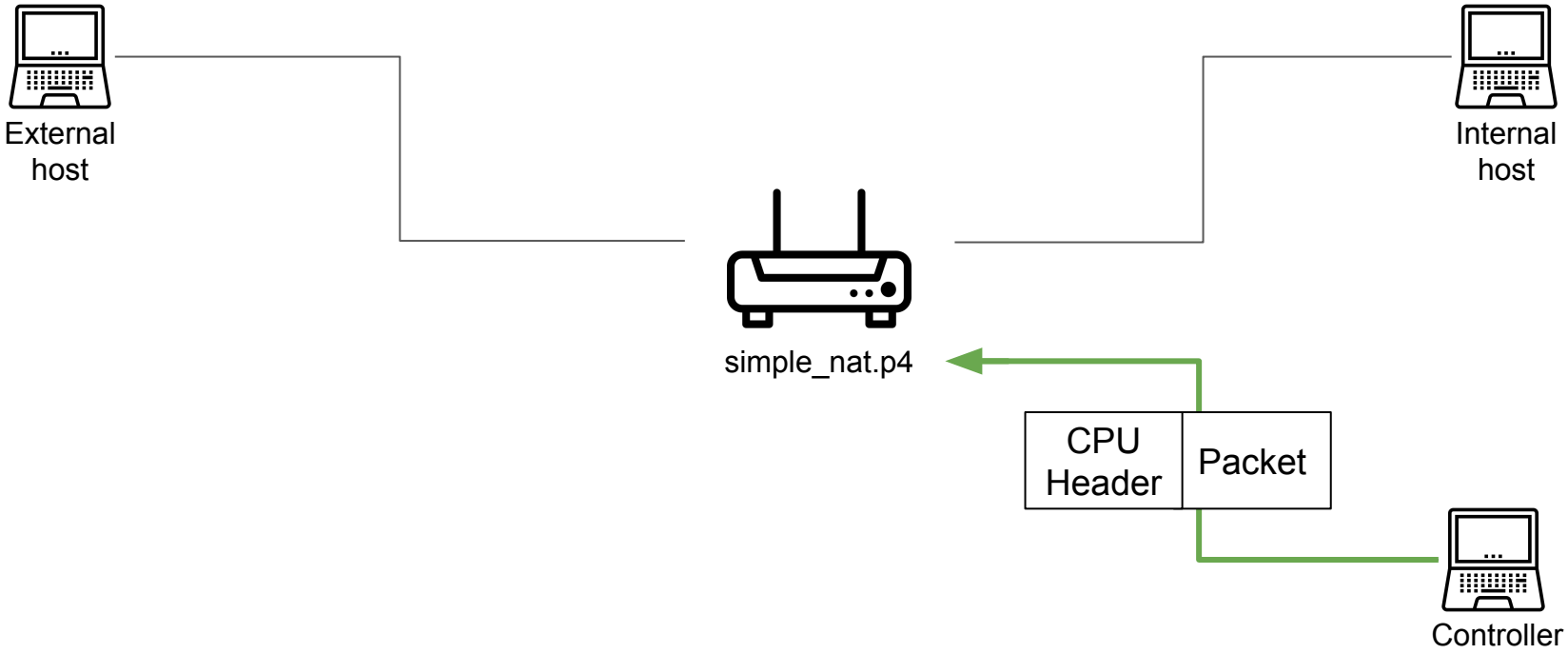
Simple NAT setup



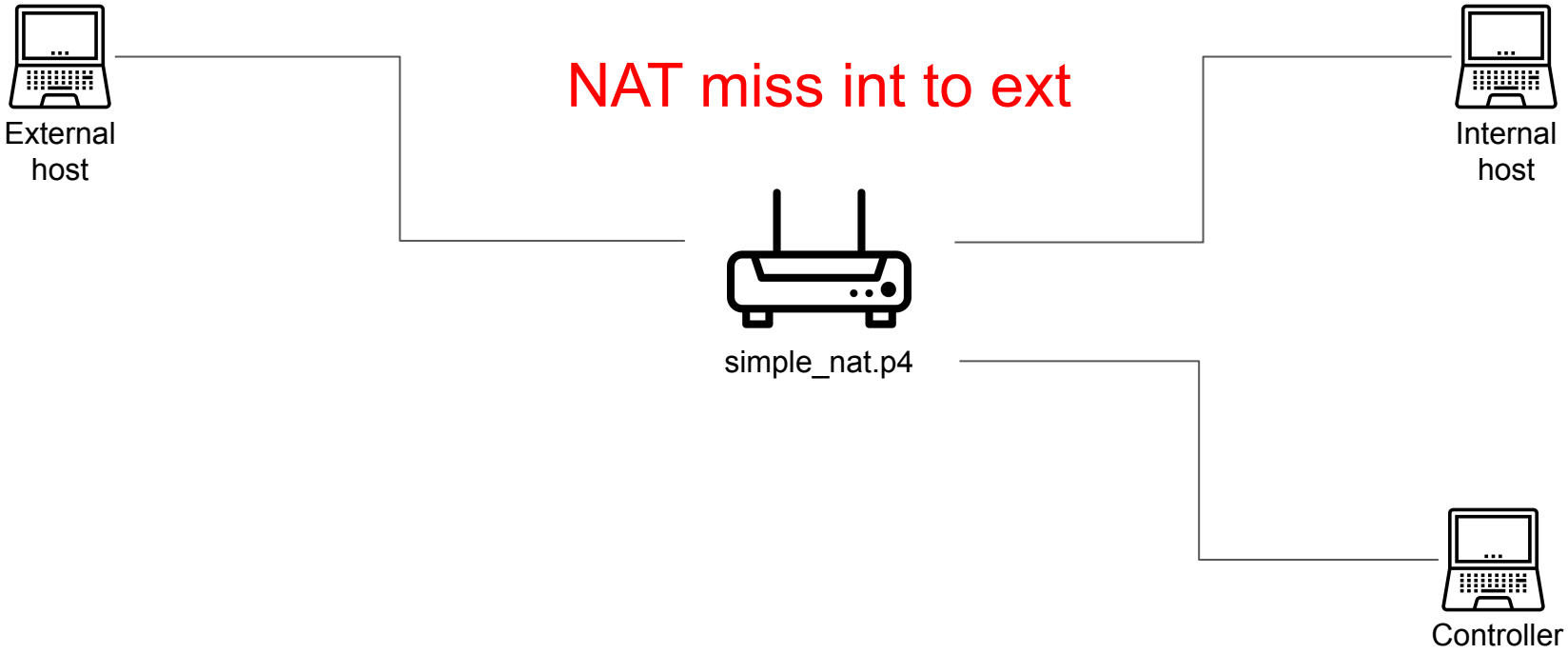
Simple NAT setup



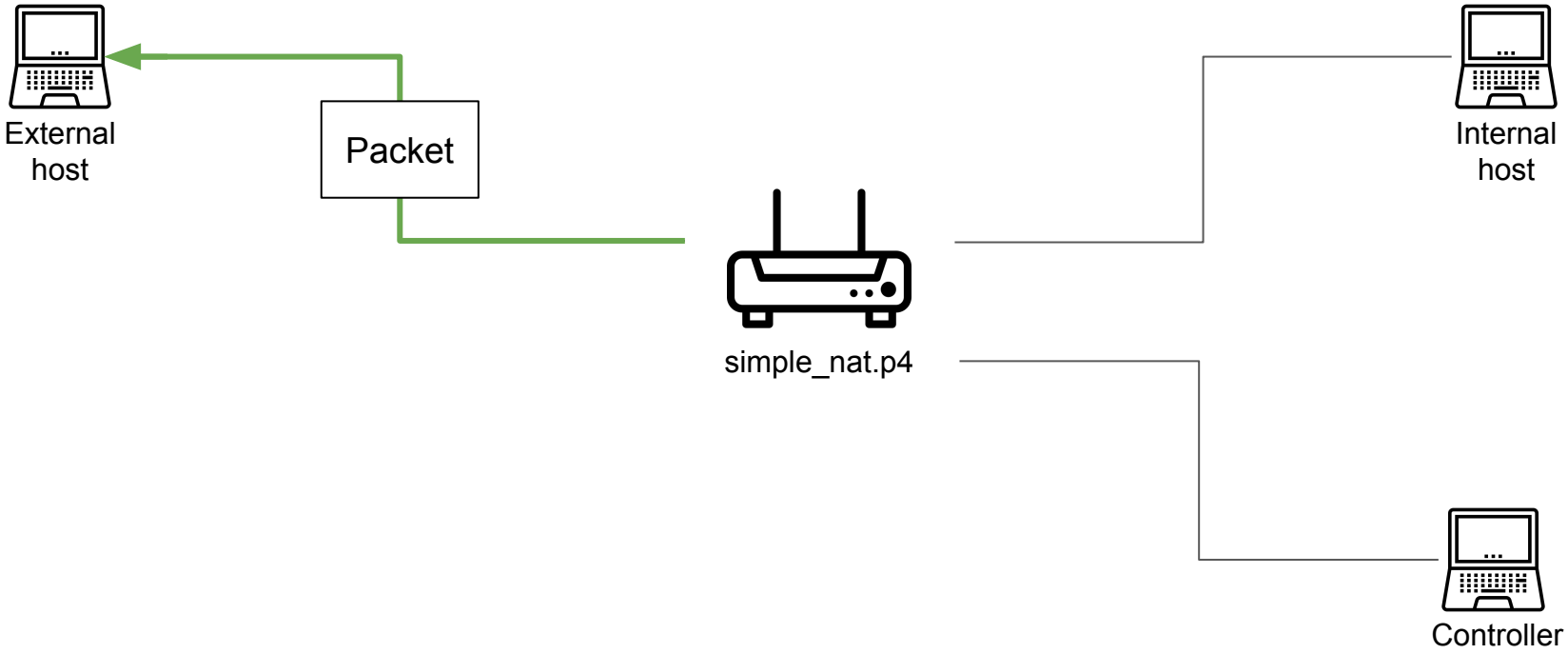
Simple NAT setup



Simple NAT setup

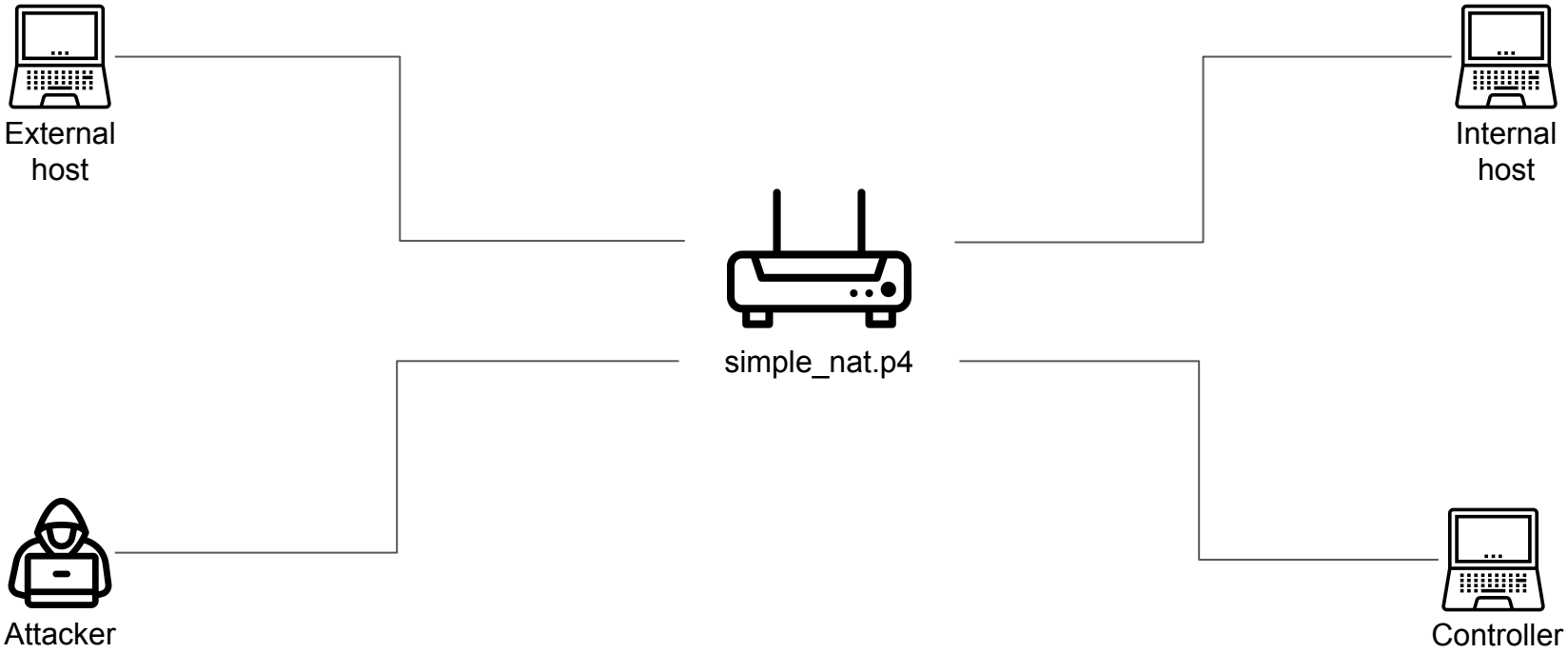


Simple NAT setup

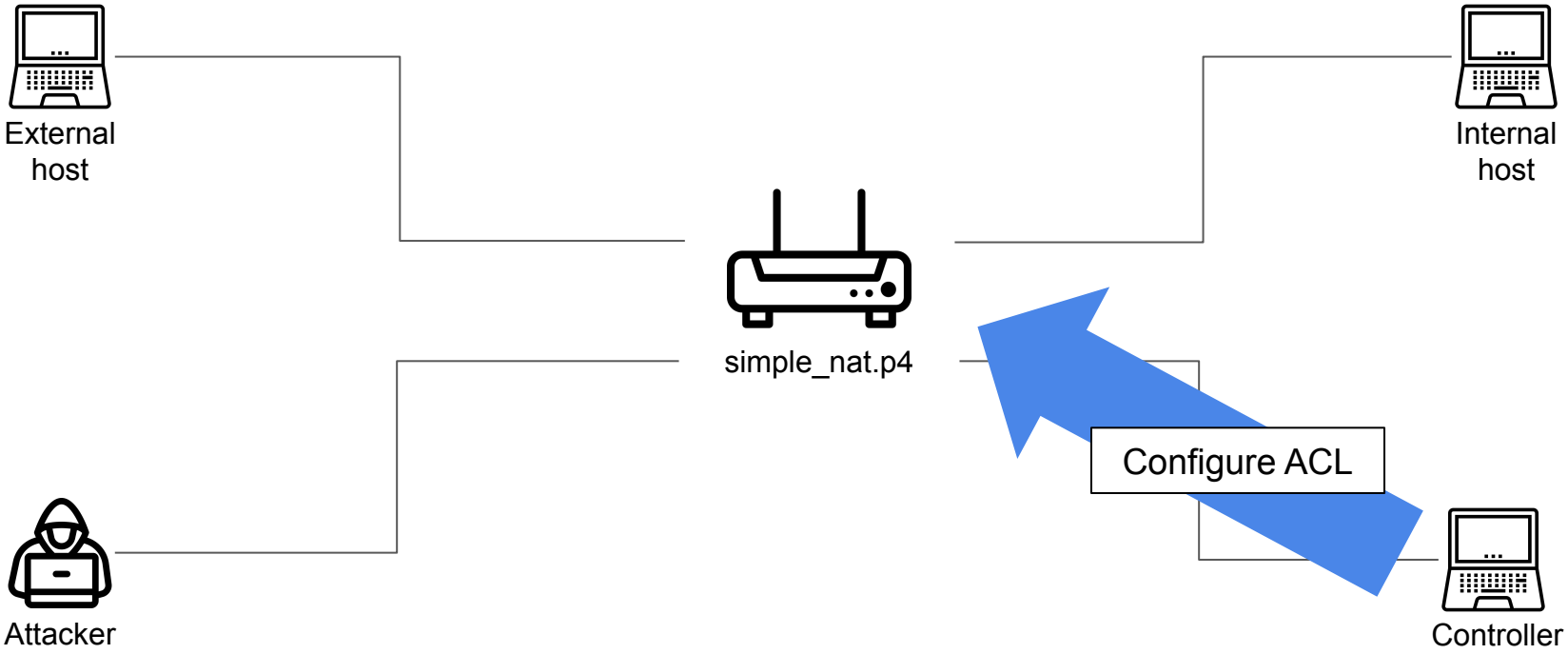


Simple NAT setup

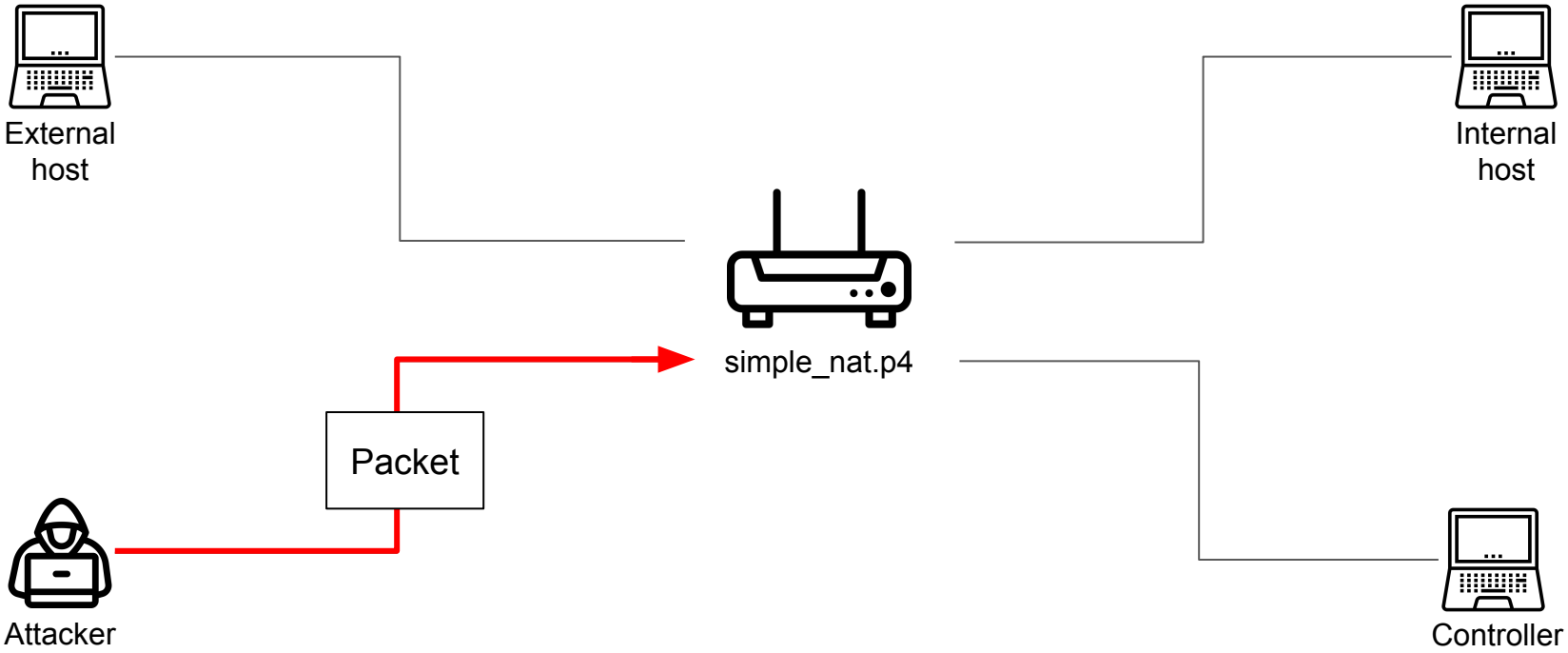
Attacker model



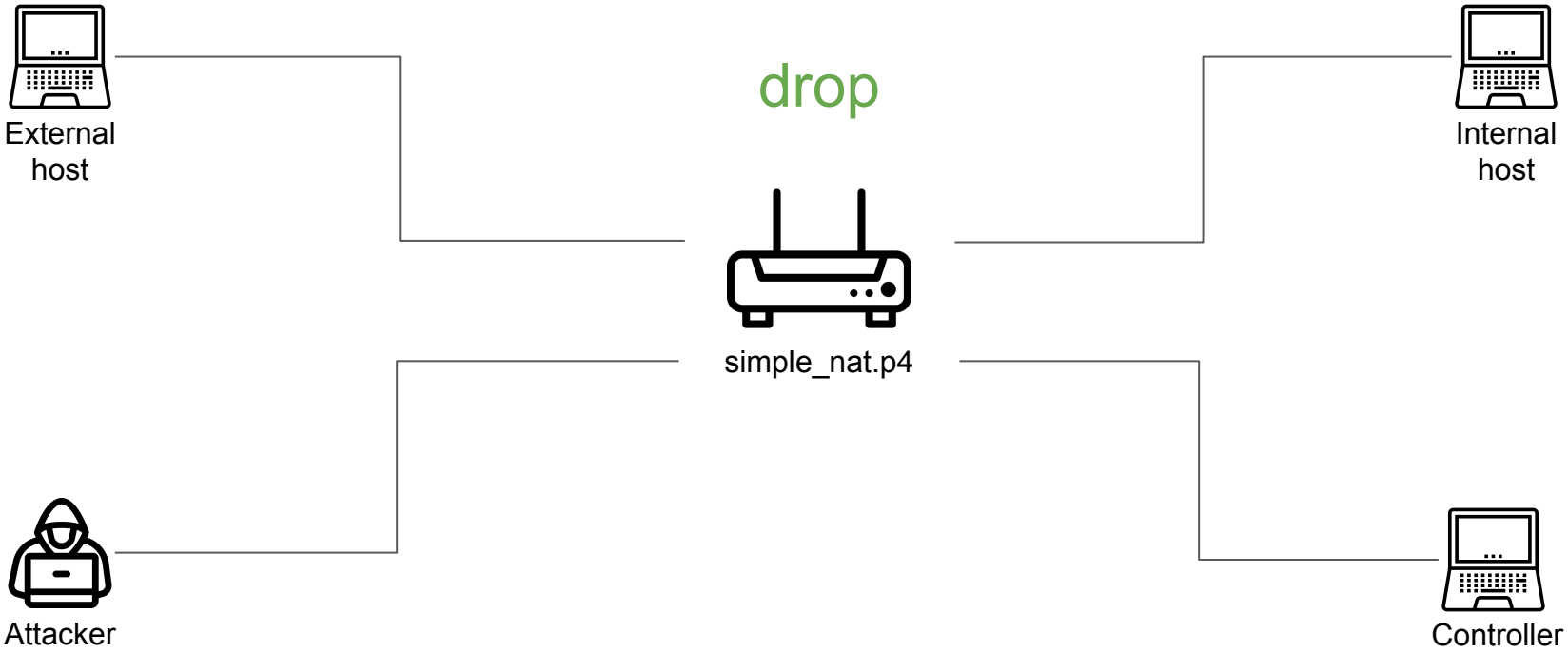
Attacker model



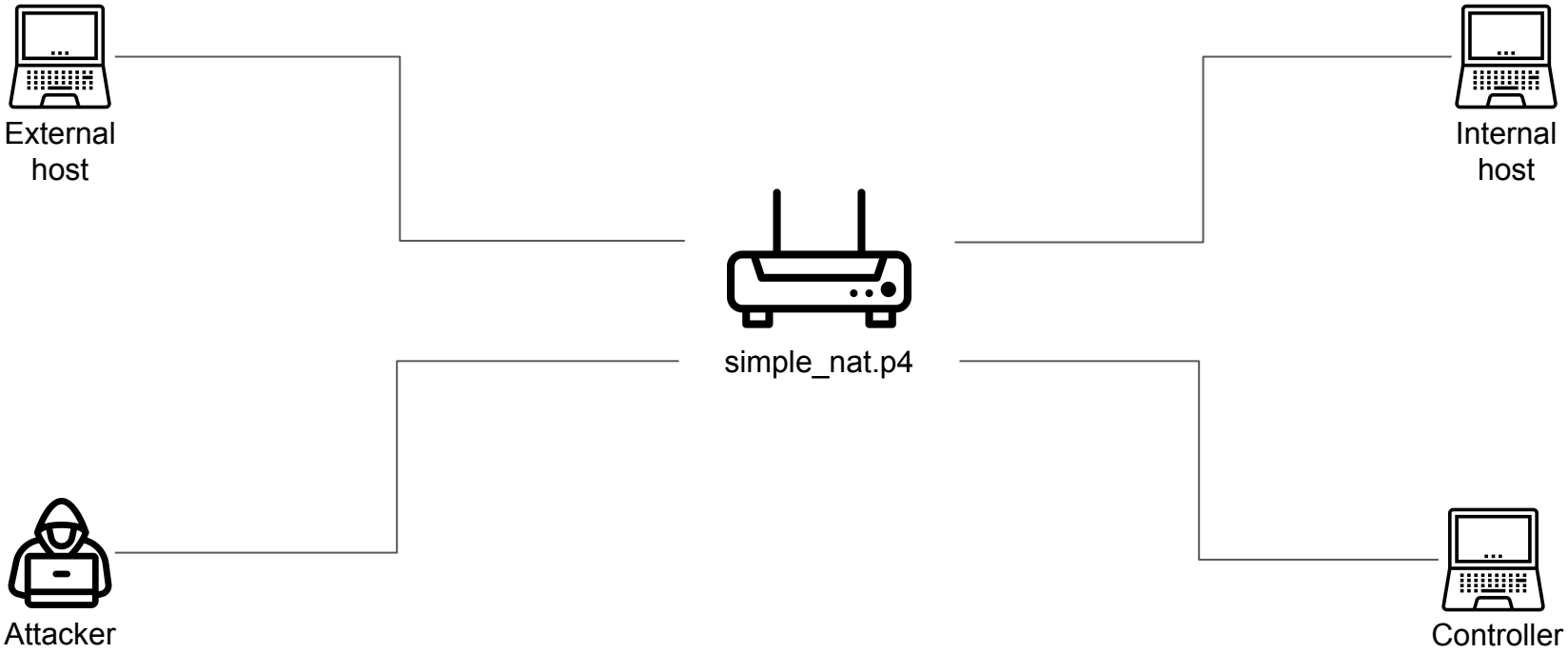
Attacker model



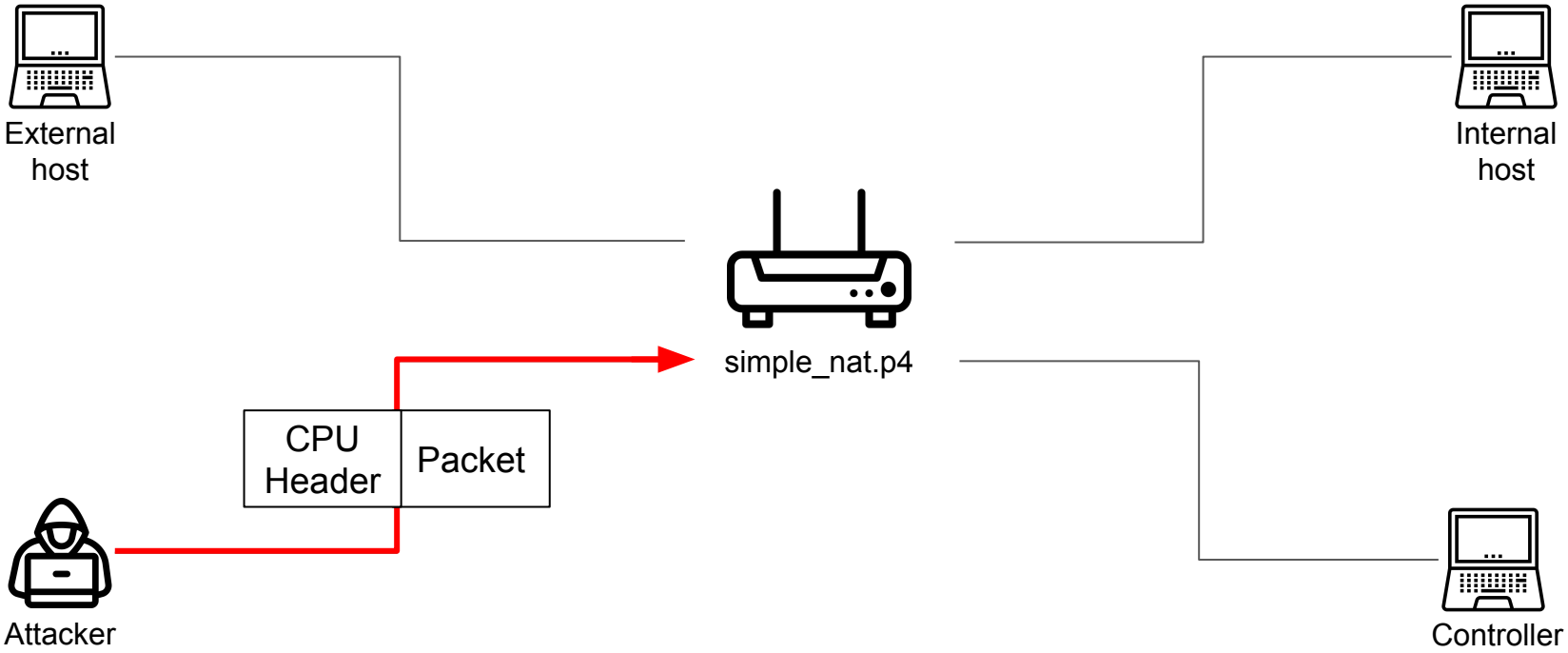
Attacker model



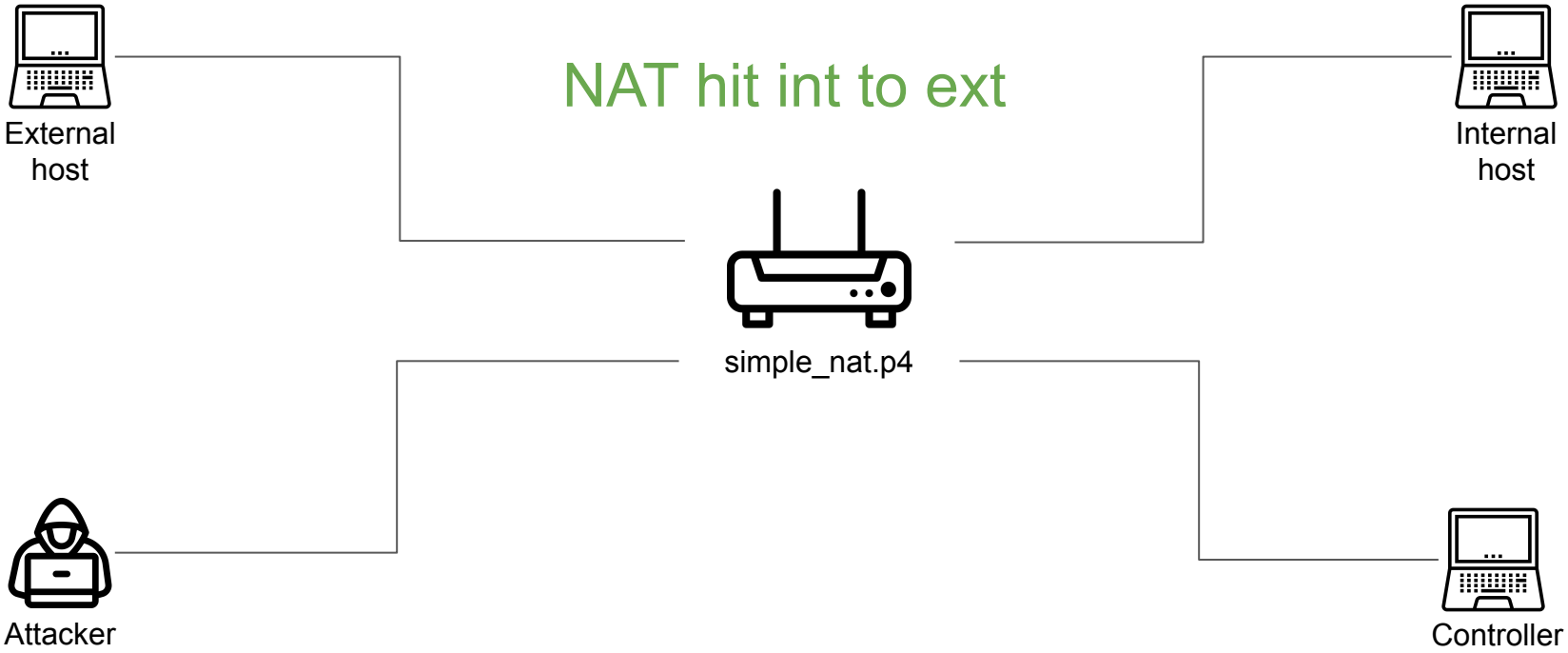
CPU header spoofing



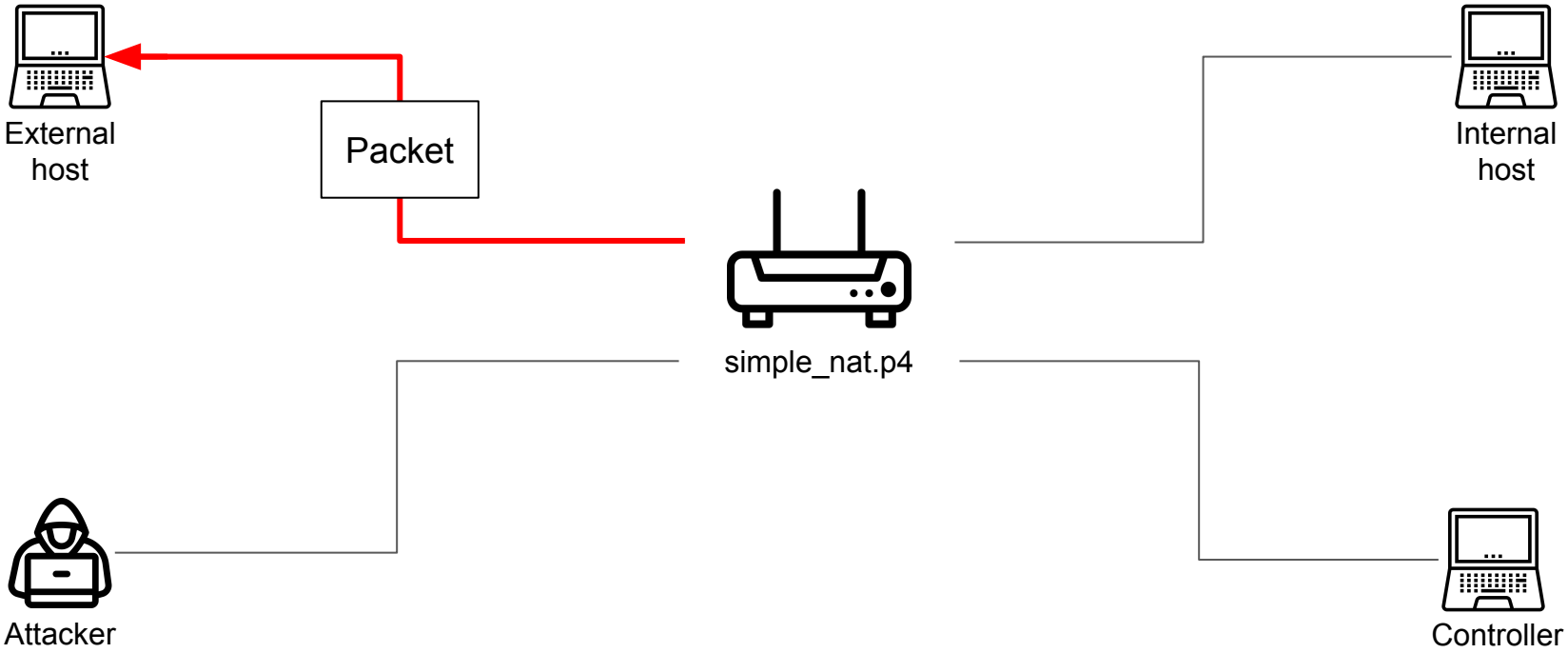
CPU header spoofing



CPU header spoofing



CPU header spoofing



CPU header spoofing

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
table if_info {
     reads {
          meta.if_index : exact;
     } actions {
          drop;
          set_if_info;
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

Controller

```
if_info: meta.if_index == ATTACKER_PORT
      => drop()
```

```
table if_info {
     reads {
          meta.if_index : exact;
     } actions {
          drop;
          set_if_info;
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

Controller

```
if_info: meta.if_index == ATTACKER_PORT
      => drop()
```

```
table if_info {
     reads {
          meta.if_index : exact;
     } actions {
          drop;
          set_if_info;
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
table nat {
    reads {
        meta.is_ext_if : exact;
        ipv4.srcAddr: ternary;
        ipv4.dstAddr: ternary;
        tcp.srcPort : ternary;
        tcp.dstPort : ternary;
    } actions {
        nat_miss_int_to_ext;
        nat_hit_int_to_ext;
        ...
    }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
table nat {
     reads {
          meta.is_ext_if : exact;
          ipv4.srcAddr: ternary;
          ipv4.dstAddr: ternary;
          tcp.srcPort : ternary;
          tcp.dstPort : ternary;
     } actions {
         nat_miss_int_to_ext;
         nat_hit_int_to_ext;
         ...
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
table nat {
     reads {
          meta.is_ext_if : exact;
          ipv4.srcAddr: ternary;
          ipv4.dstAddr: ternary;
          tcp.srcPort : ternary;
          tcp.dstPort : ternary;
     } actions {
         nat_miss_int_to_ext;
         nat_hit_int_to_ext;
         ...
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
action nat_hit_int_to_ext(dstAddr, dstPort)
{
    meta.do_forward = 1;
    ...
}
```

```
table nat {
    reads {
        meta.is_ext_if : exact;
        ipv4.srcAddr: ternary;
        ipv4.dstAddr: ternary;
        tcp.srcPort : ternary;
        tcp.dstPort : ternary;
    } actions {
        nat_miss_int_to_ext;
        nat_hit_int_to_ext;
        ...
    }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
action nat_hit_int_to_ext(dstAddr, dstPort)
{
     meta.do_forward = 1;
     ...
}
```

```
table nat {
     reads {
          meta.is_ext_if : exact;
          ipv4.srcAddr: ternary;
          ipv4.dstAddr: ternary;
          tcp.srcPort : ternary;
          tcp.dstPort : ternary;
     } actions {
         nat_miss_int_to_ext;
         nat_hit_int_to_ext;
         ...
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
if (meta.do_forward == 1 && ipv4.ttl > 0) {
    apply(ipv4_lpm);
    apply(forward);
}
```

```
action nat_hit_int_to_ext(dstAddr, dstPort)
{
     meta.do_forward = 1;
     ...
}
```

```
table nat {
     reads {
          meta.is_ext_if : exact;
          ipv4.srcAddr: ternary;
          ipv4.dstAddr: ternary;
          tcp.srcPort : ternary;
          tcp.dstPort : ternary;
     } actions {
         nat_miss_int_to_ext;
         nat_hit_int_to_ext;
         ...
     }
}
```

Bypassing ACLs with revived packets

```
apply(if_info);
apply(nat);
```

```
{
    meta.do_forward = 1;
    ...
}
```

```
table nat {
    reads {
```

```
        nat_miss_int_to_ext;
        nat_hit_int_to_ext;
        ...
    }
}
```

Packet from attacker is forwarded

What can and cannot be done exploit-wise?

Impossible attacks

- immutable code => no code injection
- no instruction pointer => no code reuse
- no dynamic memory > no use-after-free

Possible attacks

- data leakage
- Denial-of-Service
- spoofing

Summary

Assess exploitability of buggy P4 programs

Document undefined behaviors on real targets

Provide examples of exploits

Ongoing work:

- Explore weaker attackers
- Provide more exploits on real P4 programs
- Automate bug discovery and exploit generation