DHCP is hard.

Felix Wilhelm | HITB Dubai 2018

A story of 5 bugs

Intro

- **D**ynamic **H**ost **C**onfiguration **P**rotocol (DHCP) is a 25 year old protocol designed to dynamically assign an IP address and other related config parameters to devices on a local network.
- DHCP**v6** is it’s 15 year old “successor” that almost nobody uses...
  - .. but everyone supports just in case

Intro

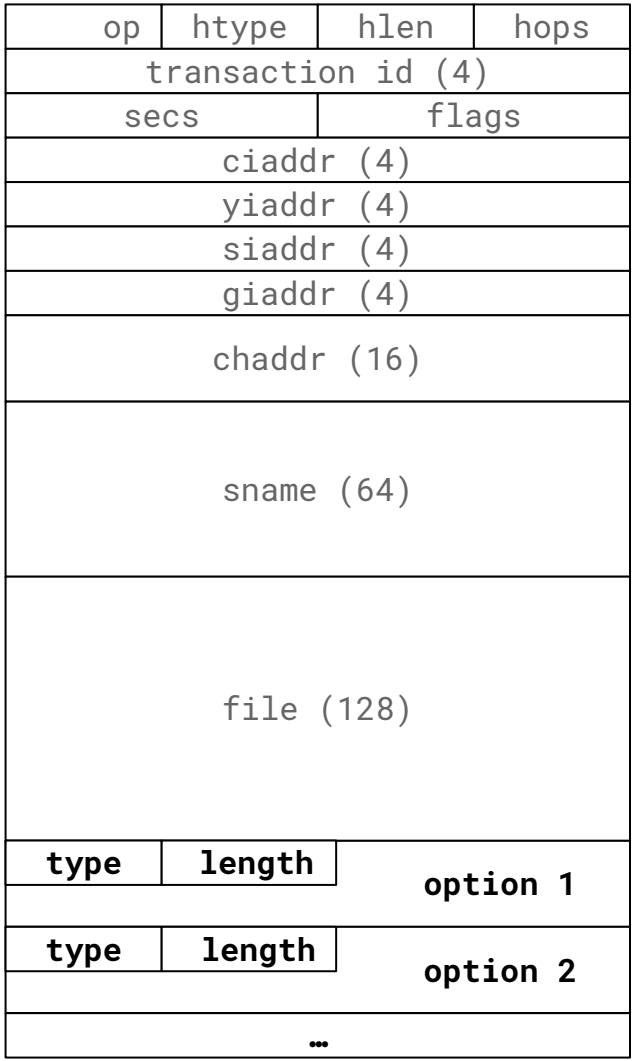
- Both protocols look pretty harmless on first glance
  - Small feature set
  - Easy to parse packet format
  - Reasonably small attack surface
  - .. turns out reality is a bit different

An interesting target

- Using DHCP for MitM is trivial and not part of this talk
- Focus is on **implementation** bugs
  - Memory safety, code injection, file handling bugs..
  - Goal is code execution
  - Against DHCP servers and more importantly **clients**
- Client bugs are an interesting attack surface
  - Lateral movement in local networks
  - Malicious access points
  - Physical access and connecting to a malicious network

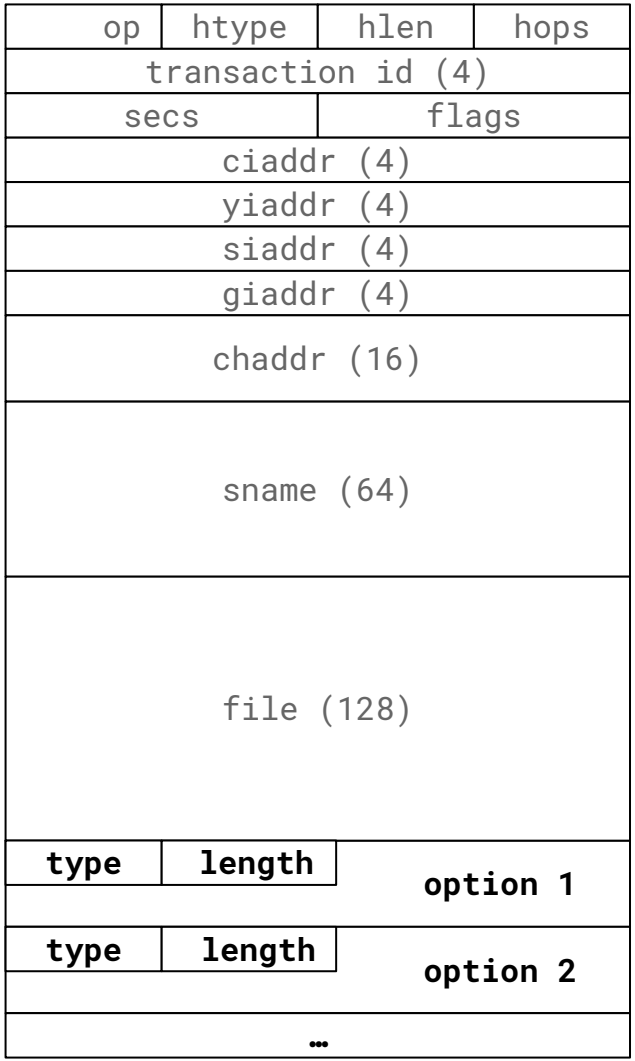
It’s all about options

- Most fields in a DHCP packets are pretty uninteresting for us
  - Fixed sizes
  - Often deprecated
  - BOOTP legacy
- Options are different
  - **T**ype - **L**ength - **V**alue encoding
  - Some options are required (e.g DHCP message type) others optional
- DHCPv6 removed almost everything besides options from the packet format.



It’s all about options

- Value encoding is type specific
  - IP address, DNS encoded domain name
  - ASCII strings, Lists of X
- Options with the same type can be repeated
- Option overloading
  - `sname` and `file` can be used for storing more options
- Vendor-specific options
  - Popular way to extend the protocol with your own features
  - E.g WPAD == DHCP option 252



Let’s look at some code

dnsmasq - CVE-2017-14493

```
if ((opt = opt6_find(opts, end, OPTION6_CLIENT_MAC, 3)))
   {
     state->mac_type = opt6_uint(opt, 0, 2);
     state->mac_len = opt6_len(opt) - 2;
     memcpy(&state->mac[0], opt6_ptr(opt, 2), state->mac_len);
   }
```

dnsmasq - CVE-2017-14493

```
if ((opt = opt6_find(opts, end, OPTION6_CLIENT_MAC, 3)))
    {
      state->mac_type = opt6_uint(opt, 0, 2);
      state->mac_len = opt6_len(opt) - 2;
      memcpy(&state->mac[0], opt6_ptr(opt, 2), state->mac_len);
    }
```

discovered by Kevin Hamacher from Google Security

What about more widespread DHCP software?

ISC DHCP

- DHCP implementation by the Internet Systems Consortium
  - Also known for the BIND DNS server
- 1.0 Release in 1997
- One of the most widely used DHCP server implementation
- DHCP client: dhclient
  - Default DHCP client on most mainstream Linux distributions

ISC DHCP - CVE-2018-5733

```
struct option {
    const char *name;
    const char *format;
    struct universe *universe;
    unsigned code;
    int refcnt;
};
```

ISC DHCP - CVE-2018-5733

```
int parse_option_buffer (options, buffer, length, universe)
    struct option *option = NULL;
    ...
        // for every option in the buffer
        option_code_hash_lookup(&option, universe->code_hash, &code,
                    0, MDL);  // increase option->refcnt

        … // process option
        option_dereference(&option, MDL); //decrease option->refcnt
```

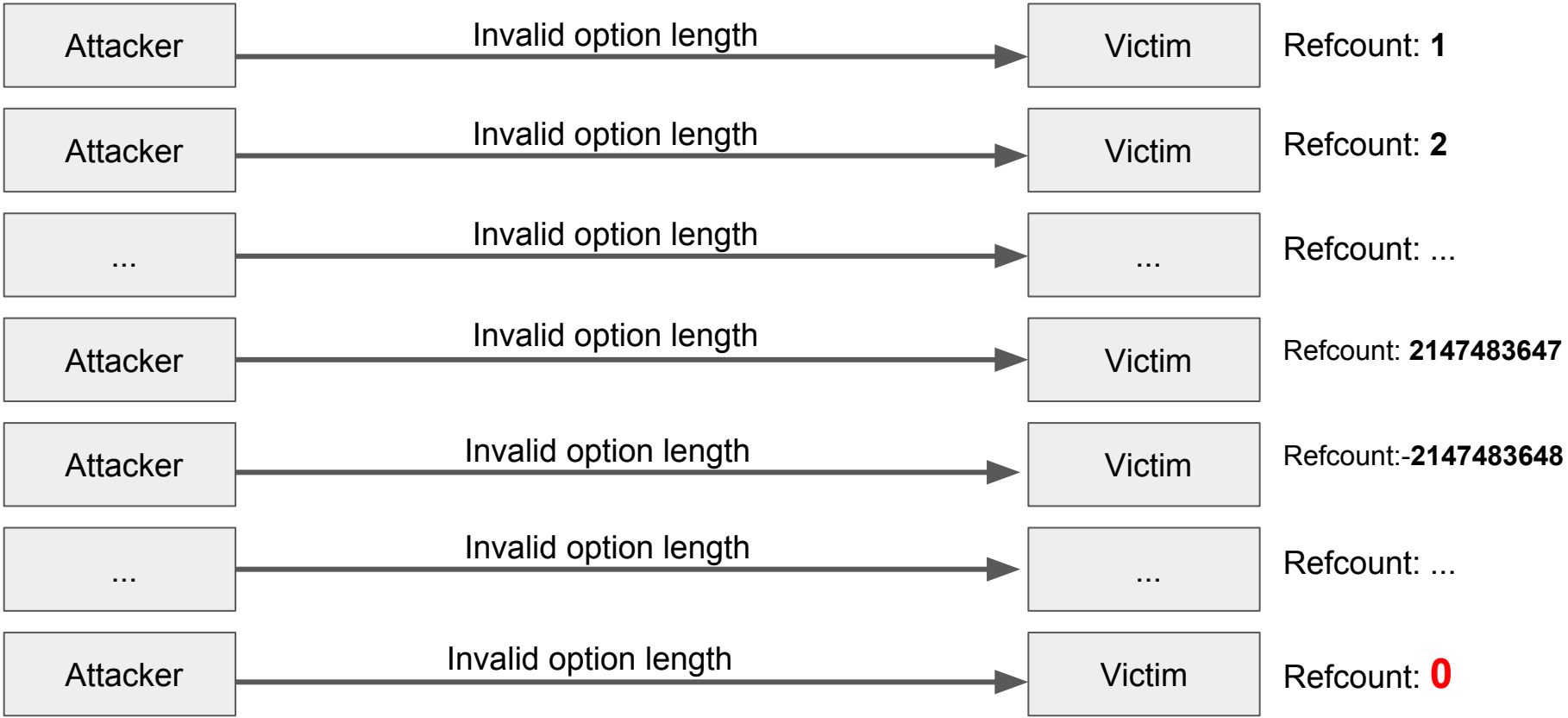
ISC DHCP - CVE-2018-5733

```
/* If the length is outrageous, the options are bad. */
if (offset + len > length) {
        reason = "option length exceeds option buffer length";
        [...]
        buffer_dereference (&bp, MDL);
        return 0;
}
```

ISC DHCP - CVE-2018-5733

```
if (offset + len > length) {
        reason = "option length exceeds option buffer length";
        [...]
        option_dereference(&option, MDL);  // decrease
option->refcnt
        buffer_dereference (&bp, MDL);
        return 0;
        }
```

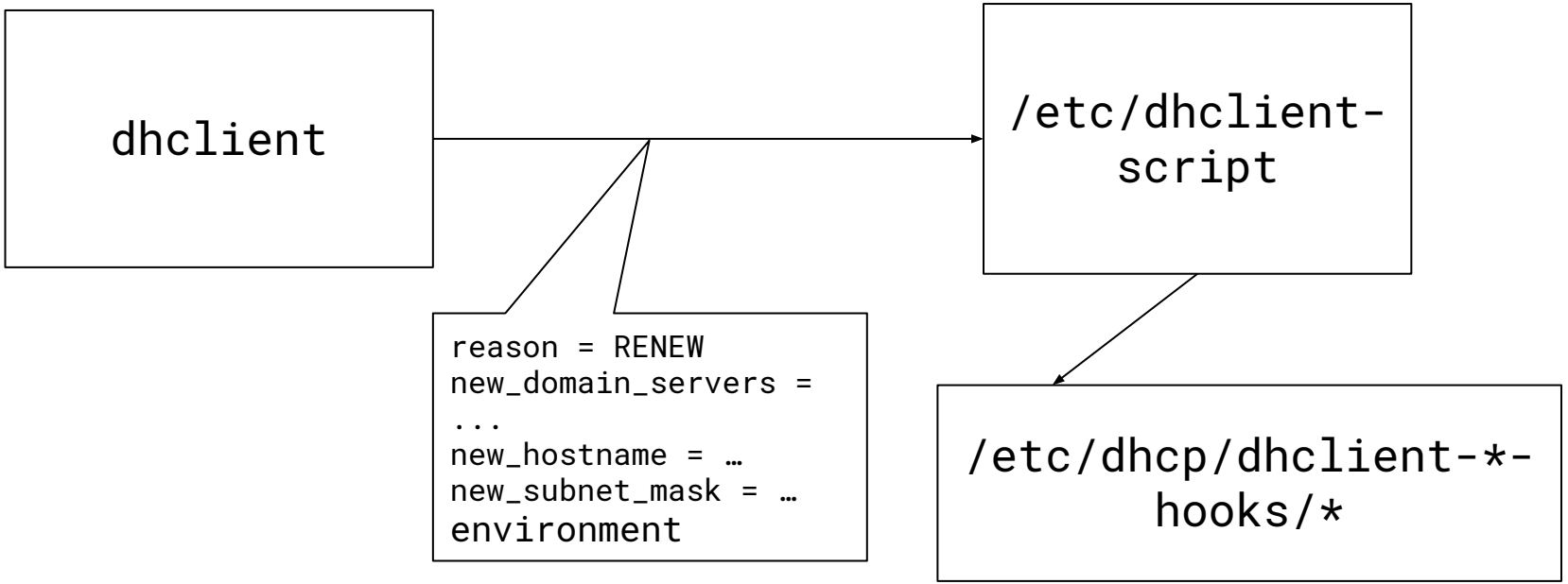
Refcount Overflow



ISC DHCP - CVE-2018-5733

- Can be turned into use-after-free by overflowing option refcount
  - .. but requires ~200GB of traffic
- DHCP option parsing gives perfect heap primitives
  - Exactly sized allocations and frees
  - Order of heap operations influenceable by option types
- Vulnerable code is part of Server and Client
  - Function happens before interface or transaction id is checked
  - Can be turned into RCE against Server
  - Client-side RCE is .. non-trivial

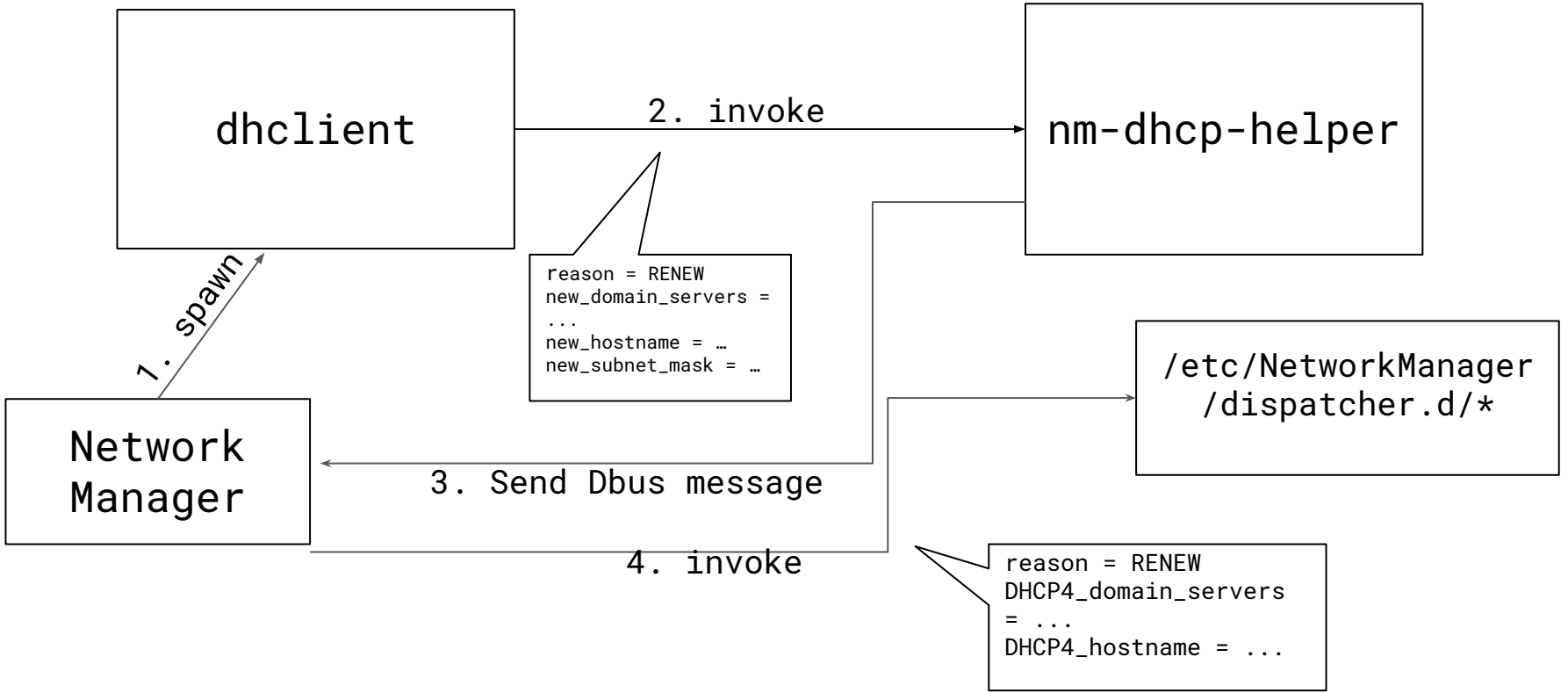
ISC DHCP - Architecture



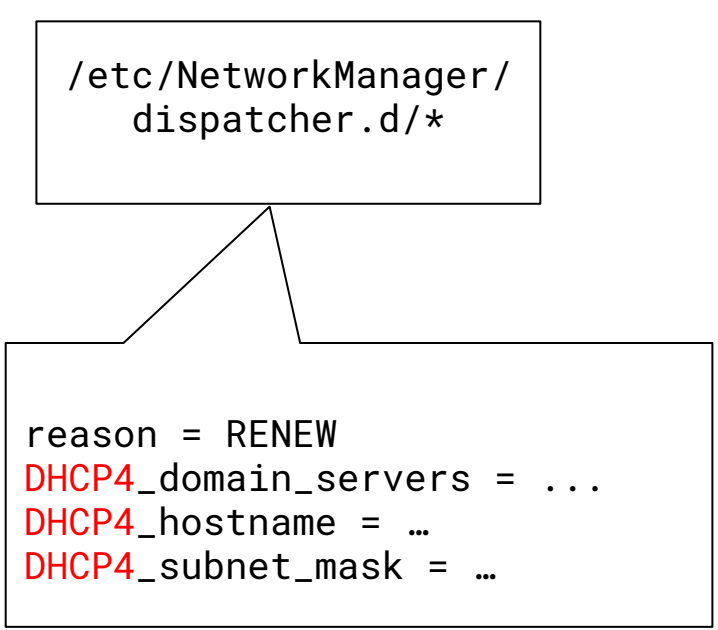
ISC DHCP - CVE-2018-5732

```
/* Format the specified option so that a human can easily read it.*/
const char *pretty_print_option (...)
{
    static char optbuf [32768]; /* XXX */
    char *op = optbuf;
    … // for every byte of the option
        case 'X':
        case 'x':
                sprintf (op, "%x", *dp++);
                break;
        op += strlen (op);
```

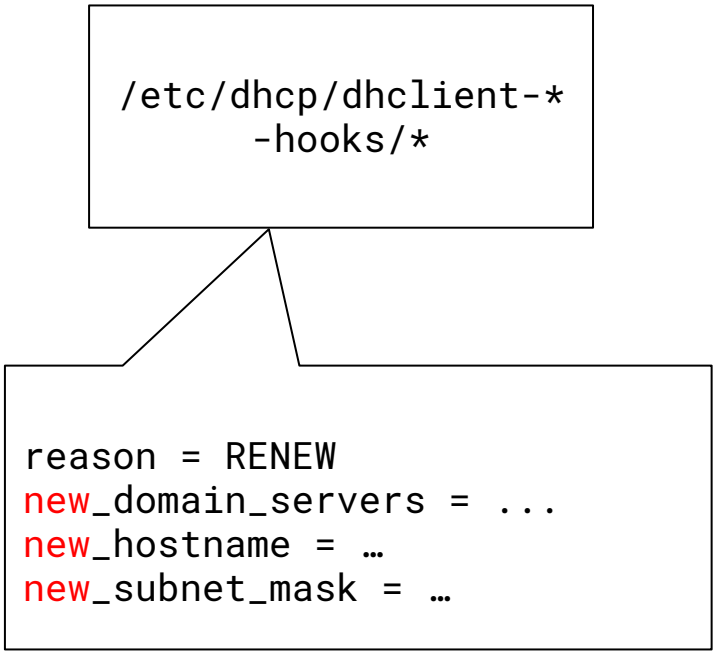
ISC DHCP - Real Architecture



ISC DHCP - Real Architecture



VS.



11dhclient

- `11dhclient`
  - Script in /etc/NetworkManager/dispatcher.d/
  - Responsible for running /etc/dhcp/dhclient-*-hooks/*
- Used by default on RHEL, Fedora, CentOS
  - Even if you don’t have any dhclient hooks
- Needs to translate environment variables
  - DHCP4_OPTION to new_option

CVE-2018-1111 (also known as dynoroot )

```
#!/bin/bash
# run dhclient.d scripts in an emulated environment
…
eval "$(
declare | LC_ALL=C grep '^DHCP4_[A-Z_]*=' | while read opt;
do
    optname=${opt%%=*}
    optname=${optname,,}
    optname=new_${optname#dhcp4_}
    optvalue=${opt#*=}
    echo "export $optname=$optvalue"
done)"
```

CVE-2018-1111 (also known as dynoroot )

```
#!/bin/bash
# run dhclient.d scripts in an emulated environment
…
eval "$(
declare | LC_ALL=C grep '^DHCP4_[A-Z_]*=' | while read -r
opt; do
    optname=${opt%%=*}
    optname=${optname,,}
    optname=new_${optname#dhcp4_}
    optvalue=${opt#*=}
    echo "export $optname=$optvalue"
done)"
```

CVE-2018-1111 (also known as dynoroot )

- `read` without flags eats backslashes.
  - Foo\’ ⇒ Foo’
  - “read -r” is the correct one
- This removes required escaping for option values
  - eval block becomes vulnerable
- Remote root code execution by setting
  - WPAD (option 252) to something like `foo’ & touch /tmp/test123 #`

Systemd networkd

- NetworkManager replacement under the systemd umbrella
- Included in the default installation of Ubuntu 18.X
  - also on some RHEL images and container distributions
- No dependencies on external DHCP clients
  - Contains a self-written implementation
  - Modern C
  - Clean rewrite
  - No historical baggage

Systemd networkd (CVE-2018-15688)

```
int dhcp6_option_append_ia(uint8_t **buf, size_t *buflen, const
DHCP6IA *ia) { [...]
        len = DHCP6_OPTION_IA_NA_LEN;
        if (*buflen < len)
                return -ENOBUFS;
        ia_hdr = *buf;
        ia_buflen = *buflen;

        *buf += sizeof(DHCP6Option);
        *buflen -= sizeof(DHCP6Option);
        memcpy(*buf, (char*) ia + iaid_offset, len);

        *buf += len;
        *buflen -= len;
```

Systemd networkd (CVE-2018-15688)

```
int dhcp6_option_append_ia(uint8_t **buf, size_t *buflen, const
DHCP6IA *ia) { [...]
        len = DHCP6_OPTION_IA_NA_LEN;
        if (*buflen < len + sizeof(DHCP6Option))
                return -ENOBUFS;
        ia_hdr = *buf;
        ia_buflen = *buflen;

        *buf += sizeof(DHCP6Option);
        *buflen -= sizeof(DHCP6Option);
        memcpy(*buf, (char*) ia + iaid_offset, len);

        *buf += len;
        *buflen -= len;
```

Systemd networkd (CVE-2018-15688)

- Vulnerable Code is part of the DHCPv6 functionality… which is enabled by default

*Note that **DHCPv6 will by default be triggered by Router Advertisement**, if that is enabled, regardless of this parameter. By enabling DHCPv6 support explicitly, the DHCPv6 client will be started regardless of the presence of routers on the link, or what flags the routers pass.*
*(**man systemd.network**)*

Exploiting systemd

Exploiting systemd

- Largely controlled OOB write in a fixed size heap buffer
- Target system: Ubuntu 18.04 LTS
  - `systemd 237-3ubuntu10`
  - `glibc 2.27-3ubuntu1`
  - Full ASLR and PIE
- Crash triggers immediate restart

Infoleak - client_parse_message

```
int client_parse_message( …, DHCP6Message *message,size_t len,...)
 {
      len -= sizeof(DHCP6Message);
       while (pos < len) {
            DHCP6Option *option = (DHCP6Option *)&message->options[pos];

              if (len < offsetof(DHCP6Option, data) ||
                  len < offsetof(DHCP6Option, data) + be16toh(option->len))
                                    return -ENOBUFS;
            // process option
            ///
            pos += offsetof(DHCP6Option, data) + optlen;
```

Infoleak - client_parse_message

```
int client_parse_message( …, DHCP6Message *message,size_t len,...)
 {
      len -= sizeof(DHCP6Message);
       while (pos < len) {
            DHCP6Option *option = (DHCP6Option *)&message->options[pos];

          if (len < pos + offsetof(DHCP6Option, data) ||
              len < pos + offsetof(DHCP6Option, data) + be16toh(option->len))
                                        return -ENOBUFS;
            // process option
            ///
            pos += offsetof(DHCP6Option, data) + optlen;
```

Triggering the Infoleak: server -> client

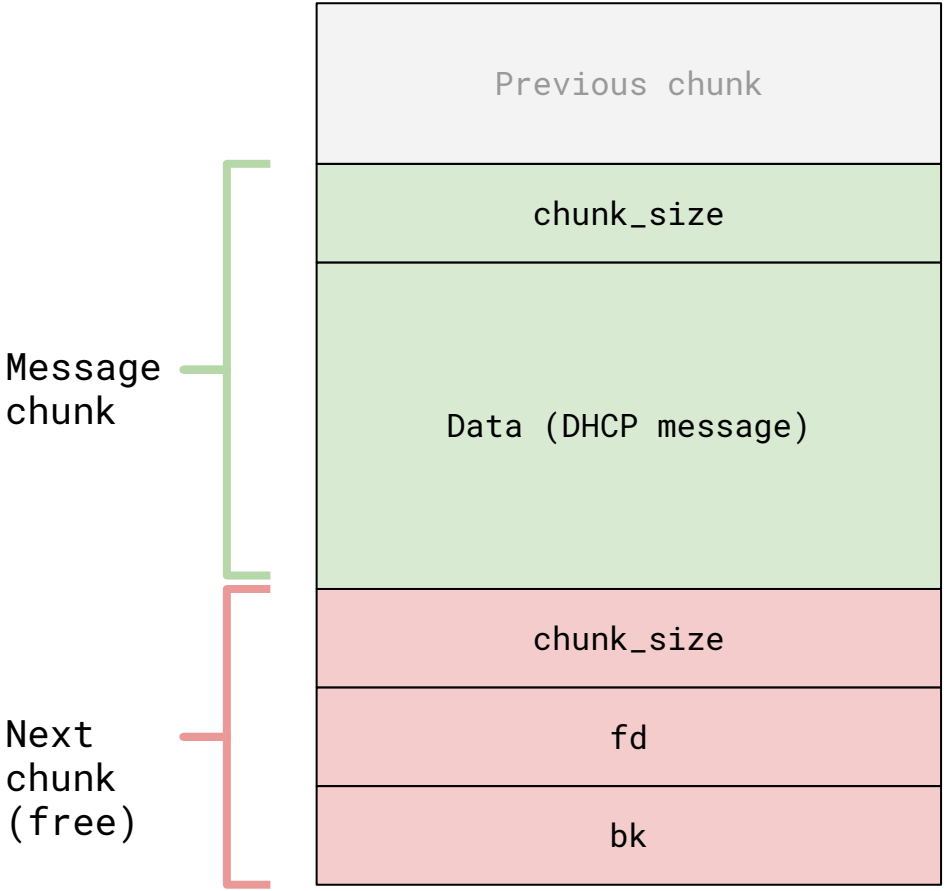
DHCP6 Header		
Option Type 1337	Option Length 50	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Option Type 1337	Option Length 50	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Option Type 1337	Option Length 50	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Option Type ServerID	Option Length **150**	B

Triggering the Infoleak: client -> server

DHCP6 Header		
Option Type ServerID	Option Length 150	BE806B55BBB9EE512BF954A1458F9310715812588A59452E9C F538E1D8360BCEFC5432ACC7D26AF76F871EE57B81F085F….
Option Type ...	Option Length ...	...
Option Type ...	Option Length ...	...

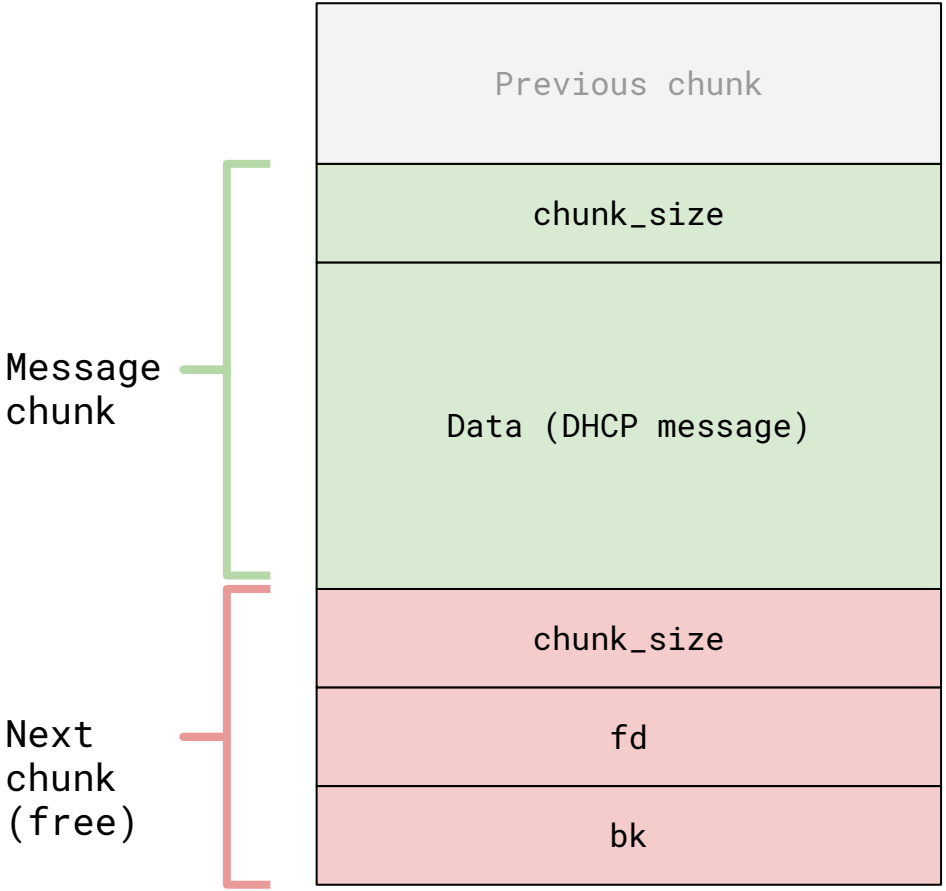
Leaking a glibc pointer

- If chunk following our message buffer is free we can leak heap metadata
- If the free chunk is the only entry in an empty glibc bin (freelist) `fd` and `bk` point to an address inside `glibc`
  - `main_arena + xyz`



Leaking a glibc pointer

- Leaked pointer can be used to calculate `glibc` base address ⇒ Break ASLR
- Changes for the desired heap state are pretty high for a fresh heap
  - We can just crash systemd and try again



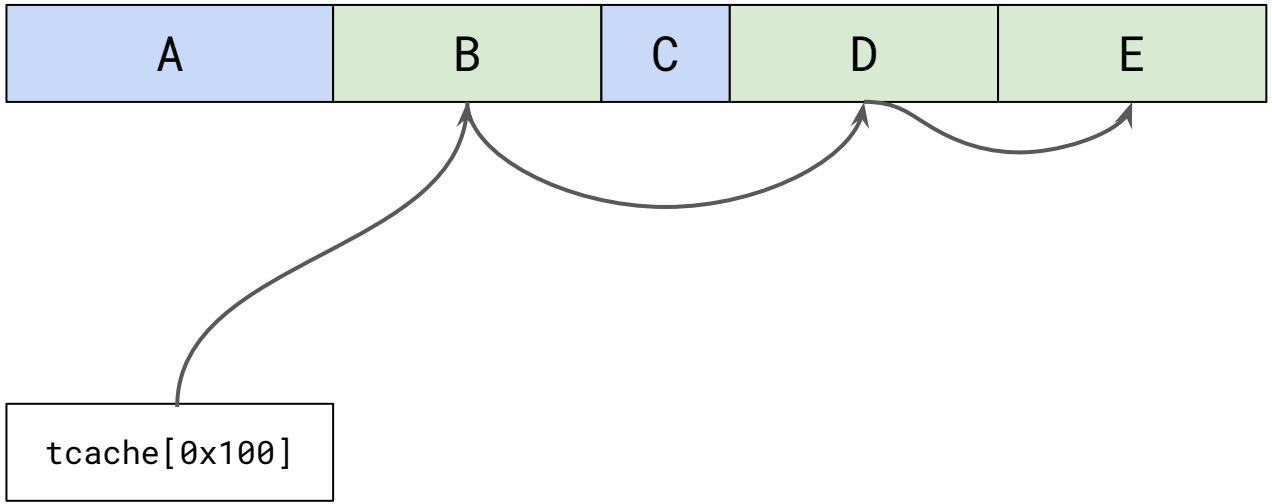
Heap Overflow to Arbitrary Write

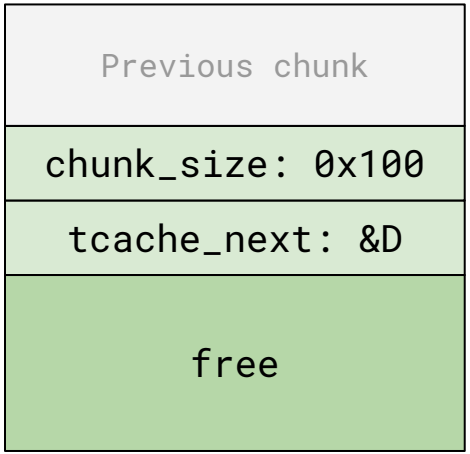
- To get closer to code execution we want to turn our Heap Overflow primitive into a controlled Arbitrary Write
  - Write of data behind our buffer ⇒ Write of data at an arbitrary address
- One way: Trick glibc's `malloc` into returning an attacker controlled address.
  - House of {Spirit, Lore, Force, ..}
- Got much easier on recent versions of glibc: **tcache poisoning**

tcache

- Thread local cache for recently used memory chunks
  - Introduced in glibc 2.26
  - See http://tukan.farm/2017/07/08/tcache/ for a detailed write up
- Cached chunks are stored in a singly linked list
  - One list for every chunk size between 24 - 1032 byte
  - Stores up to last 7 chunks per size
  - Last In First Out
- No integrity checks!
  - Basically negates any attempt at glibc malloc hardening over the last years.

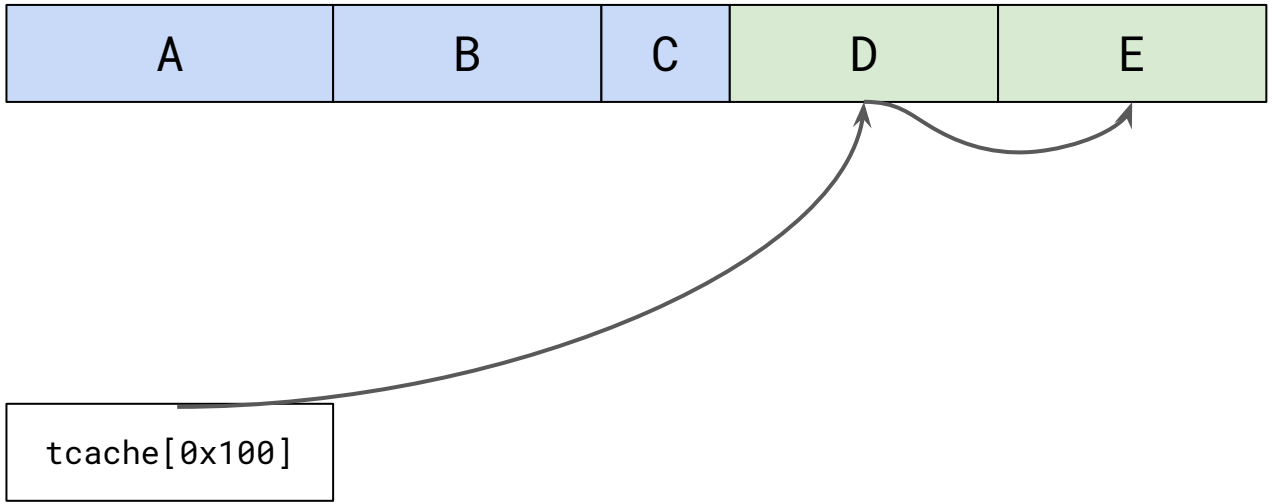
tcache


A
B
C
D
E
tcache[0x100]


Previous chunk
chunk_size: 0x100
tcache_next: &D
free

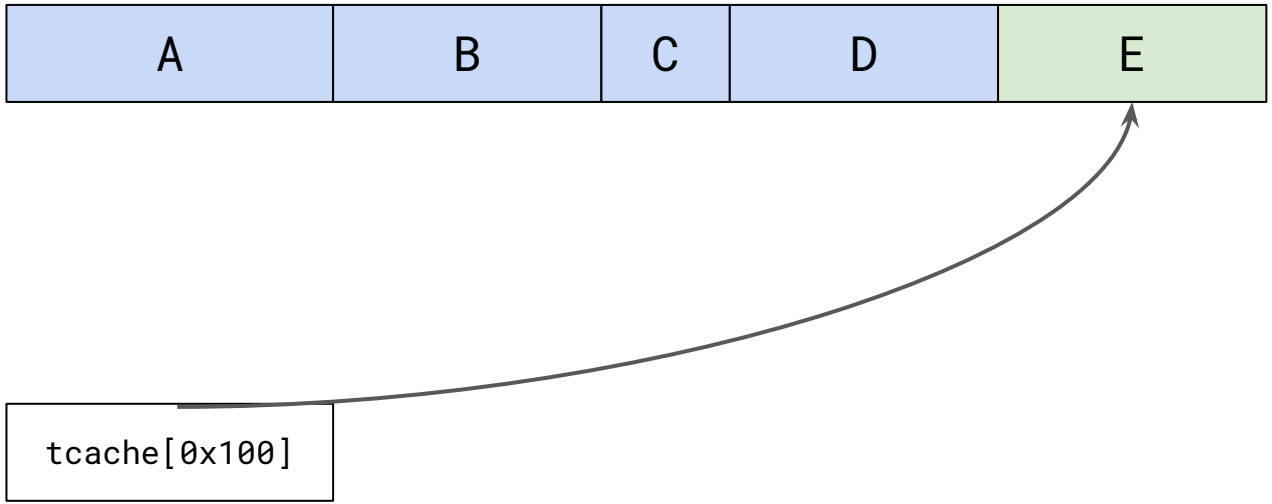
tcache

malloc(0x100)=&**B**

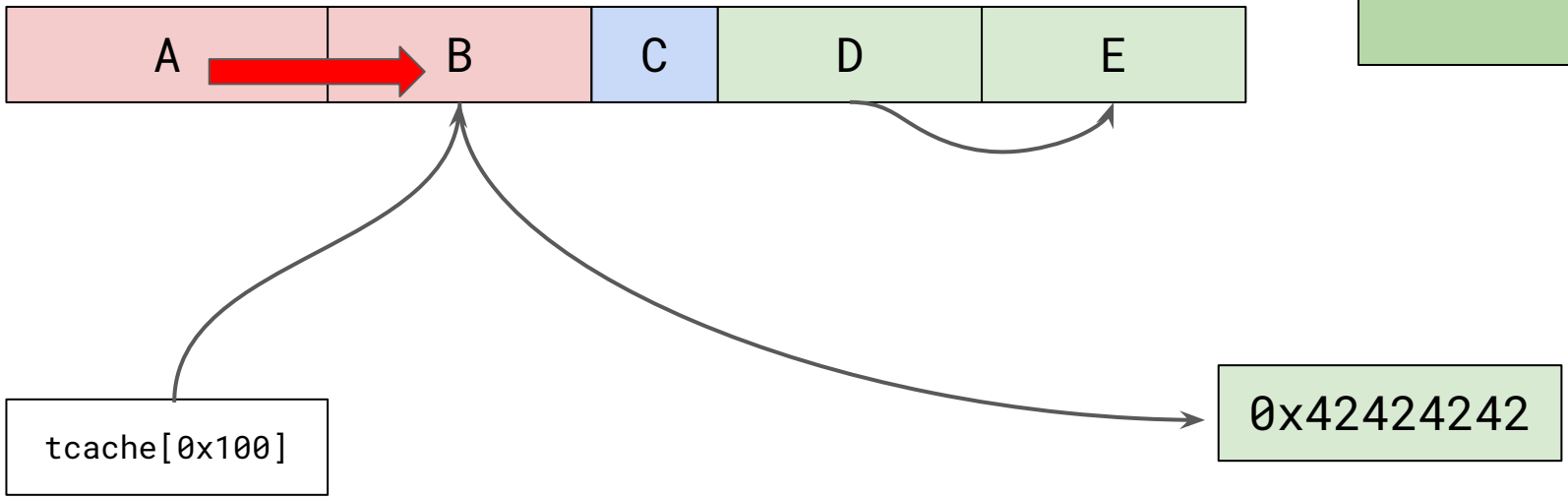


tcache

malloc(0x100)=&**D**



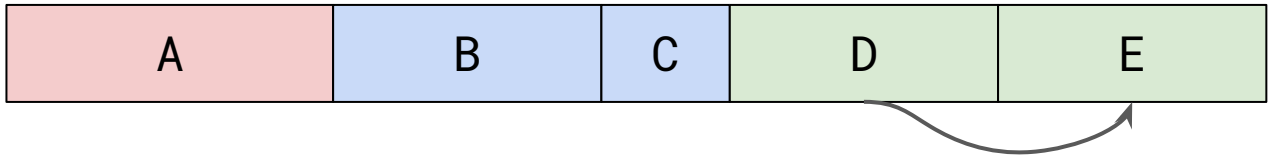
tcache Poisoning



Previous chunk
chunk_size: 0x414141
tcache_next: 0x42424242
free

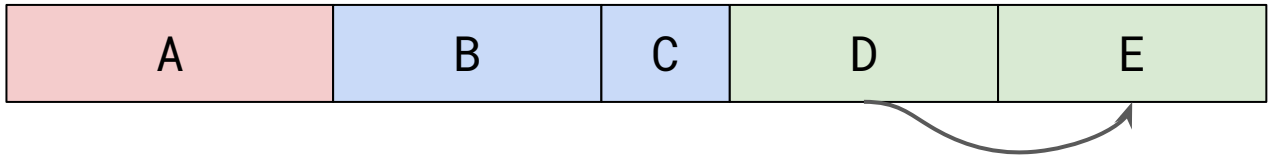
tcache Poisoning

`malloc(0x100)=&B`



tcache Poisoning

`malloc(0x100)=`**`0x42424242`**



`tcache[0x100]`

`0x42424242`

Arbitrary Write + Infoleak → RCE

- Quick and dirty approach: Overwrite `__free_hook` glibc with pointer to `system`
- Every time a memory address is freed, glibc will call system($address) instead.
- Command Execution once an attacker controlled chunk is freed.

Putting it all together

1. Trigger systemd networkd DHCPv6 client using router advertisement packets
2. Leak pointer to `glibc` using option size OOB read and calculate libc base address
3. Trigger heap overwrite to perform `tcache` poisoning
4. Allocate fake chunk at address of `__free_hook` and write address of `system` to it
5. Trigger `free()` of chunk with your payload in it.
6. $shell

Demo

Conclusion

- DHCP looks easy at first glance, but reality is different.
- No one uses DHCPv6 but everyone supports it
- Backwards compatibility increases attack surface

Conclusion

- Sufficient powerful memory corruptions still lead to RCE
- Please don’t write new network daemons in C