STUMPING THE MOBILE CHIPSET

New 0days from down under

Adam Donenfeld
AGENDA

- Android chipsets overview in ecosystem
- Qualcomm chipset subsystem’s overview
- New kernel vulnerabilities
- Exploitation of a new kernel vulnerability
- Conclusions
ADAM DONENFELD

- Years of experience in research (both PC and mobile).
- Vulnerability assessment
- Vulnerability exploitation
- Senior security researcher at Check Point
- In meiner Freizeit, lerne ich Deutsch gern 😊
How **Android** gets to your device

- Carrier
- OEM
- Chipset code
- Android Project
- Linux Kernel
Qualcomm’s chipset subsystems

- IPC Router
- GPU
- Audio
- Performance
- Thermal
- QSEECOM

[Protected] Non-confidential content
The Rooting Zoo
ASHmenian Devil (ashmem vulnerability)  
CVE-2016-5340

- Qualcomm ‘expands’ ashmem for the GPU
  - Map ashmem to GPU
- Passing ashmem fd to map
ASHmenian devil (ashmem vulnerability)

```c
int get_ashmem_file(int fd, 
   struct file **filp, 
   struct file **vm_file, 
   unsigned long *len)
{
    int ret = -1;
    struct ashmem_area *asma;
    struct file *file = fget(fd);
    if (is_ashmem_file(file)) {
        asma = file->private_data;
        *filp = file;
        *vm_file = asma->file;
        *len = asma->size;
        ret = 0;
    } else {
        fput(file);
    }
    return ret;
}
```
ASHmenian devil (ashmem vulnerability)

- Qualcomm ‘expands’ ashmem for the GPU
  - Map ashmem to GPU
- Passing ashmem fd to map

- Is our fd an ashmem file descriptor?
ASHmenian devil (ashmem vulnerability)
ASHmenian devil (ashmem vulnerability)

```c
static int is_ashmem_file(struct file *file)
{
    char fname[256], *name;
    name = dentry_path(file->f_dentry, fname, 256);
    return strcmp(name, "~/ashmem") == 0; /* Oh my god */
}
```
ASHmenian devil – PoC

- Filename on root path == “ashmem”
- / is read-only 😞
- /sdcard is a symlink
- Obb (Opaque Binary Blob)
ASHmenian devil – PoC

● Create an OBB
  ➢ With “ashmem” in it’s root directory

● Mount the OBB

● Map “ashmem” memory to the GPU
  ➢ Pass a fd to your fake ashmem file
Qualaroot (IPC Router vulnerability)  
CVE-2016-2059

- Qualcomm’s IPC router
- Special socket family
  - AF_MSM_IPC (27)
- Unique features
  - “Whitelist” for services that are permitted to communicate
  - Everyone gets an “address” for communication
  - Creation/destruction can be monitored by anyone
- Requires no permission 😊
Qualaroot

- AF_MSM_IPC socket types
  - CLIENT_PORT
  - SERVER_PORT
  - IRSC_PORT
  - CONTROL_PORT
    - Conversion via IPC_ROUTER_IOCTL_BIND_CONTROL_PORT

- Each new socket is a CLIENT_PORT socket
static int msm_ipc_router_ioctl(
    struct sock *sock,
    unsigned int cmd,
    unsigned long arg)
{
    struct sock *sk = sock->sk;
    struct msm_ipc_port *port_ptr;

    lock_sock(sk);
    port_ptr = msm_ipc_sk_port(sock->sk);
    switch (cmd) {
        ...
        case IPC_ROUTER_IOCTL_BIND_CONTROL_PORT:
            msm_ipc_router_bind_control_port(port_ptr);
            ...
        }
    release_sock(sk);
    ...
}
int msm_ipc_router_bind_control_port(struct msm_ipc_port *port_ptr)
{
    if (!port_ptr)
        return -EINVAL;

    /* Lock clients list */
    down_write(&local_ports_lock_lhc2);

    /* Remove our socket from its current list */
    list_del(&port_ptr->list);

    /* Unlock clients list */
    up_write(&local_ports_lock_lhc2);

    /* Lock control ports list */
    down_write(&control_ports_lock_lha5);

    /* Add our socket to the control ports list */
    list_add_tail(&port_ptr->list, &control_ports);

    /* Unlock control ports list */
    up_write(&control_ports_lock_lha5);

    return 0;
}
Client list

Client list

down_write(&local_ports_lock_lhc2);
list_del(&port_ptr->list);
up_write(&local_ports_lock_lhc2);
down_write(&control_ports_lock_lha5);
list_add_tail(&port_ptr->list, &control_ports);
up_write(&control_ports_lock_lha5);
Client list

Control list

```
down_write(&local_ports_lock_lhc2);
list_del(&port_ptr->list);
up_write(&local_ports_lock_lhc2);
down_write(&control_ports_lock_lha5);
list_add_tail(&port_ptr->list, &control_ports);
up_write(&control_ports_lock_lha5);
```
Qualaroot – the vulnerability

- `control_ports` list is modified without lock!
- Deleting 2 objects from `control_ports` simultaneously!

RACE CONDITION
Qualaroot - implementation

```c
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```
Qualaroot - implementation

control_ports

A

B

C

POISON

entry = A
next = B
prev = control_ports

B->prev = control_ports

static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
Qualaroot - implementation

```
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```

entry = A
Next = B
Prev = control_ports
B->prev = control_ports
Qualaroot - implementation

control_ports

A
B
C

entry = B
Next = C
Prev = control_ports

C->prev = control_ports
Qualaroot - implementation

```
static inline void list_del(  
    struct list_head *entry)  
{
    next = entry->next;  
    prev = entry->prev;  
    next->prev = prev;  
    prev->next = next;  
    entry->next = LIST_POISON1;  
    entry->prev = LIST_POISON2;  
}
```

- **entry = B**
- **Next = C**
- **Prev = control_ports**
- **C->prev = control_ports**
Qualaroot - implementation

```
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```

entry = B
Next = C
Prev = control_ports

control_ports->next = C
Qualaroot - implementation

control_ports

A

B

C

POISON

entry = B
Next = C
Prev = control_ports

control_ports->next = C

static inline void list_del(
struct list_head *entry)
{
next = entry->next;
prev = entry->prev;
next->prev = prev;
prev->next = next;
entry->next = LIST_POISON1;
entry->prev = LIST_POISON2;
}
Qualaroot - implementation

```
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```

entry = B
Next = C
Prev = control_ports

entry is freed
next = prev = LIST_POISON

control_ports
A
B
C
POISON
Qualaroot - implementation

```c
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```

- **entry = B**
- **Next = C**
- **Prev = control_ports**

- **entry is freed**
- **next = prev = LIST_POISON**
Qualaroot - implementation

CONTEXT SWITCH
Qualaroot - implementation

- control_ports
- A
- B
- POISON
- C

entry = A
Next = B
Prev = control_ports

control_ports->next = B

static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
Qualaroot - implementation

entry = A
Next = B
Prev = control_ports
control_ports->next = B
Qualaroot - implementation

control_ports

A

B

C

POISON

static inline void list_del(
struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}

entry = A
Next = B
Prev = control_ports

entry is freed
next = prev = LIST_POISON
Qualaroot - implementation

```c
static inline void list_del(
    struct list_head *entry)
{
    next = entry->next;
    prev = entry->prev;
    next->prev = prev;
    prev->next = next;
    entry->next = LIST_POISON1;
    entry->prev = LIST_POISON2;
}
```

entry = A
Next = B
Prev = control_ports

entry is freed
next = prev = LIST_POISON

control_ports

A
B
POISON
C
Qualaroot - implementation

- Two following objects are deleted
  - Simultaneously!

- `control_ports` points to a **FREE** data
  - `LIST_POISON` worked
    - No longer mappable
    - Spraying `af_unix_dgram` works

- Iterations on `control_ports`?
  - Just close a client_port!
    - Notification to all `control_ports` with `post_pkt_to_port`
Qualaroot - implementation

```c
static int post_pkt_to_port(struct msm_ipc_port *UAF OBJECT,
                              struct rr_packet *pkt, int clone)
{
    struct rr_packet *temp_pkt = pkt;
    void (*notify)(unsigned event, void *oob_data,
                   size_t oob_data_len, void *priv);
    void (*data_ready)(struct sock * sk, int bytes) = NULL;
    struct sock *sk;

    mutex_lock(&UAF OBJECT->port_rx_q_lock_lhc3);
    __pm_stay_awake(UAF OBJECT->port_rx_ws);
    list_add_tail(&temp_pkt->list, &UAF OBJECT->port_rx_q);
    wake_up(UAF OBJECT->port_rx_wait_q);
    notify = UAF OBJECT->notify;
    sk = (struct sock *)UAF OBJECT->endpoint;
    if (sk) {
        read_lock(&sk->sk_callback_lock);
        data_ready = sk->sk_data_ready;
        read_unlock(&sk->sk_callback_lock);
    }
    mutex_unlock(&UAF OBJECT->port_rx_q_lock_lhc3);
    if (notify)
        notify(pkt->hdr.type, NULL, 0, UAF OBJECT->priv);
    else if (sk && data_ready)
        data_ready(sk, pkt->hdr.size);

    return 0;
}
```
Qualaroot - implementation

- `wake_up` function
  - Macros to `__wake_up_common`
static void __wake_up_common(
    wait_queue_head_t *q,
    ........)
{
    wait_queue_t *curr, *next;

    list_for_each_entry_safe(curr, next, &q->task_list, task_list) {
        ...
        if (curr->func(curr, mode,
                    wake_flags, key))
            break;
    }
}
Qualaroot - implementation

● `wake_up` function
  ➢ Macros to `__wake_up_common`

● New primitive!
  ➢ Call to function with first controllable param!
  ➢ We can’t control the `address` though 😞

● Not good enough for `commit_creds`…
Qualaroot - implementation

- Upgrade primitives
- Find a function that can call an arbitrary function with *address-controlled* parameters
Qualaroot - implementation

- `usb_read_done_work_fn` receives a function pointer and a function argument!

```c
static void usb_read_done_work_fn(
    struct work_struct *work)
{
    struct diag_request *req = NULL;
    struct diag_usb_info *ch = container_of(
        work, struct diag_usb_info,
        read_done_work);
    ...
    req = ch->read_ptr;
    ...
    ch->ops->read_done(req->buf,
        req->actual,
        ch->ctxt);
}
```
Qualaroot - implementation

I DON'T OFTEN CALL FUNCTIONS

BUT WHEN I DO, I GET ROOT
Qualaroot - implementation

- Chain function calls
  - __wake_up_common
  - usb_read_done_work_fn
  - any function

```c
static void __wake_up_common(
    wait_queue_head_t *q,
    ........)
{
    wait_queue_t *curr, *next;

    list_for_each_entry_safe(curr, next,
        &q->task_list, task_list) {
        ...
        if (curr->func(curr, mode,
            wake_flags, key))
            break;
    }
}
```
Qualaroot – Exploitation flow

Create UAF situation using the vulnerability
Qualaroot – Exploitation flow

Spray af_unix_dgrams to catch the UAF
Qualaroot – Exploitation flow

Spray `af_unix_dgrams` to catch the UAF

LIST_POISON

Spray `af_unix_dgrams` iteratively to catch the UAF
Qualaroot – Exploitation flow

Trigger list iteration

sprayed

__wake_up_common

UAF->port_rx_wait_q->task_list

usb_read_work_done_fn

qdisc_list_del

control_ports is empty

usb_read_work_done_fn

enforcing_setup

SELinux is permissive

usb_read_work_done_fn

commit_creds

UID=0;
cap = FULL_CAP_SET
Qualaroot

PERMISSIVE=1, UID=0

NOT BAD
Disclosure
Hey,

Attached is a full exploit for Nexus 6 devices running Android Marshmallow, build MRA58K. The binary itself should be run from an application context, i.e. from an APK, (otherwise SELinux prevents the exploitation) and no extra privileges are required in order to successfully exploit the vulnerability.

Please note that on other Qualcomm based devices or versions it might still cause a kernel panic, however the current exploitation requires a modification for each device.

The exploit currently sets SELinux to permissive mode, grants root privileges to the process, modifies the system partition to read-write and writes a suid file there named "zugang" (full path /system/zugang). The payload can easily be changed in the function do_root, file qualroot.c.

If you wish to test the exploit without creating an extra APK for that, let me know, and I will supply you with an APK.

To build, extract the NDK using the make-standalone-toolchain.sh to the same directory with the qualroot exploit, and run build_and_strip.sh.

---

qualroot.tar.gz
8.6 KB  Download

---

Project Member  #4 qua...@google.com

Thanks Adam.

Qualcomm also notified us that they received this report as well and they have assigned an ID for it: QPSIIR-170.

Quan

---

Project Member  #5 qua...@google.com

Hello,

Thank you for submitting this vulnerability report. The engineering team has reviewed the issue and set the severity to Low.

For reference, the severity classification is documented here: https://source.android.com/security/overview/updates-resources.html

Quan

---

Labels: Severity-Low Triaged-yes
Hey,

Attached is a full exploit for Nexus 6 devices running Android Marshmallow, build MRA58K. The binary itself should be run from an application context, i.e. from an APK, (otherwise SELinux prevents the exploitation) and no extra privileges are required in order to successfully exploit the vulnerability. Please note that on other Qualcomm based devices or versions it might still cause a kernel panic, however the current exploitation requires a modification for each device.

The exploit currently sets SELinux to permissive mode, grants root privileges to the process, modifies the system partition to read-write and writes a suid file there named "zugang" (full path /system/zugang). The payload can easily be changed in the function do_root, file qualroot.c.

If you wish to test the exploit without creating an extra APK for that, let me know, and I will supply you with an APK.

To build, extract the NDK using the make-standalone-toolchain.sh to the same directory with the qualroot exploit, and run build_and_strip.sh.

---

Project Member  #4 qua...@google.com

Thanks Adam.

Qualcomm also notified us that they received this report as well and they have assigned an ID for it: QPSIIR-170.

Quan

---

Project Member  #5 qua...@google.com

Hello,

Thank you for submitting this vulnerability report. The engineering team has reviewed the issue and set the severity to Low.

For reference, the severity classification is documented here: https://source.android.com/security/overview/updates-resources.html

Quan

Labels: Severity-Low Triaged-yes
Syncockaroot (syncsource vulnerability) CVE-2016-2503

- SyncSource objects are used to synchronize the activity between the GPU and the application.

- Can be created using IOCTLS to the GPU
  - IOCTL_KGSL_SYNCSOURCE_CREATE
  - IOCTL_KGSL_SYNCSOURCE_DESTROY

- Referenced further with the “idr” mechanism
Syncockaroot (syncsource vulnerability)

```c
long kgsl_ioctl_syncsource_destroy(
    struct kgsl_device_private *dev_priv,
    unsigned int cmd, void *data)
{

    struct kgsl_syncsource_destroy *param = data;
    struct kgsl_syncsource *syncsource = NULL;

    syncsource = kgsl_syncsource_get(
        dev_priv->process_priv,
        param->id);

    /* put reference from syncsource creation */
    kgsl_syncsource_put(syncsource);
    /* put reference from getting the syncsource above */
    kgsl_syncsource_put(syncsource);
    return 0;
}
```

Any lock on “to-be-destroyed” object?
Create a syncsource object
- A predictable idr number is allocated

Create 2 threads constantly destroying the same idr number

Ref-count will be reduced to -1
- Right after getting to zero, we can spray it

Use After Free 😊
KanGaroot (KGsl vulnerability) CVE-2016-2504

- GPU main module (kgsl-3d0)
- Map user memory to the GPU
  - IOCTL_KGSL_MAP_USER_MEM
  - IOCTL_KGSL_GPUMEM_FREE_ID
- Referenced by a **predictable** ID
  - IDR mechanism
KanGaroot (KGsl vulnerability)

```c
static int
kgsl_mem_entry_attach_process(  
    struct kgsl_mem_entry *entry,  
    struct kgsl_device_private *dev_priv)
{
    ...
    id = idr_alloc(&process->mem_idr,  
        entry, 1, 0, GFP_NOWAIT);
    ...
    ret = kgsl_mem_entry_track_gpuaddr(  
        process, entry);
    ...

    ret = kgsl_mmu_map(pagetable,  
        &entry->memdesc);
    if (ret)
        kgsl_mem_entry_detach_process(entry);
    return ret;
}
```

Should it already be accessible here?
KanGaroot (KGsl vulnerability)

- GPU main module (kgsl-3d0)

- Map user memory to the GPU
  - IOCTL_KGSL_MAP_USER_MEM
  - IOCTL_KGSL_GPUMEM_FREE_ID

- Referenced by a **predictable** ID
  - IDR mechanism

- No locks!
  - Free can be called before map ends
KanGaroot - PoC

- Map memory

- Save the IDR
  - We always get the first free IDR -- predictable

- Another thread frees the object with IDR
  - *Before the first thread returns from the IOCTL

_UAF in kgsl_mem_entry_attach_process on ‘entry’ parameter_
commit_creds for always being there for me

Absense of kASLR,
for not breaking me and commit_creds apart

SELinux, for being liberal,
letting anyone access mechanisms like Qualcomm’s IPC
Thank You!