Stumping the Mobile Chipset

Adam Donenfeld & Yaniv Mordekhay
Agenda

- Chipsets and security
- Kernel vulnerabilities and exploitation
  - ASHmenian Devil
  - Qualaroot
  - Syncockaroot
  - Kangaroot
- Disclosure process
- Conclusions
Adam Donenfeld
• Years of experience in research
• Vulnerability assessment
• Vulnerability exploitation
• In meiner Freizeit, lerne ich Deutsch gern 😊

Yaniv Mordekhay
• Veteran developer and researcher
• Specialization in behavioral analysis
• Specialization in statistical analysis
• Avid hiker

Special thanks to Avi Bashan, Daniel Brodie and Pavel Berengoltz for helping with the research
Android Architecture

Linux Kernel → Android Open Source Project → Chipset → OEMs
Qualcomm Chipsets

The dangers of chipset vulnerabilities

- Qualcomm: 65%
- Samsung: 12%
- Other: 23%

* ABI Research, February 2016
Qualcomm’s chipset subsystems

- GPU
- IPC Router
- Performance
- QSEECOM
- Thermal
- Audio
Welcome to Qualand

- GPU
- Thermal
- Performance
- IPC Router

Ashmem
ASHmenian Devil (ashmem vulnerability)

CVE-2016-5340

- Ashmem – Android’s propriety memory allocation subsystem
- Qualcomm devices uses a modified version
  - Simplifies access to *ashmem* by Qualcomm modules
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)

CVE-2016-5340

- Obtain a file struct from file descriptor
- Compare file operation handlers to expected handler struct
  - If it matches → file type is valid
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 : 1; /* Oh my god */
}
ASHmenian Devil (ashmem vulnerability)

CVE-2016-5340

- Exploitation requires –
  - Creation of file named “ashmem” on root mount point (“/”)
- / is read-only 😞
ASHmenian Devil - POC
CVE-2016-5340

- Opaque Binary Blob
  - APK Expansion File
  - Support APKs > 100MB
  - Deprecated (still works!)
- A mountable file system
ASHmenian Devil - POC
CVE-2016-5340

• Create an OBB
• Create “ashmem” in it’s root directory
• Mount the OBB
• Map “ashmem” memory to the GPU
  – Pass a fd to the fake ashmem file
Qualaroot (IPC Router vulnerability)
CVE-2016-2059

• Qualcomm’s IPC router
• Special socket family
  – AF_MSM_IPC (27)
• Unique features
  – Whitelist specific endpoints
  – Everyone gets an “address” for communication
  – Creation/destruction can be monitored by anyone
• Requires no permission 😊
Qualaroot (IPC Router vulnerability)
CVE-2016-2059

- **AF_MSM_IPC** socket types
  - **CLIENT_PORT**
  - **CONTROL_PORT**
  - **IRSC_PORT**
  - **SERVER_PORT**

- Each new socket is a **CLIENT_PORT** socket
static int msm_ipc_router_ioctl(
    struct socket *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;

    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);
    return 0;
}
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);

Control list
```c
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 (IPC Router vulnerability)
CVE-2016-2059

- `control_ports` list is modified without a lock
- Deleting 2 objects from `control_ports` simultaneously!
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; 
}
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
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
CONTEXT SWITCH
```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;
}
```

```c
entry = B
next = C
prev = control_ports
C->prev = control_ports
```
```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;
}
```

```c
entry = B
next = C
prev = control_ports
C->prev = control_ports
```
```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
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;
}
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;
}
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
B->prev = B->next = POISON
CONTEXT SWITCH
```
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;
}
```
```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
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;
}
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;
}
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`
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(&temp_pkt->list, &UAF_OBJECT->port_rx_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;
}
• **wake_up** function
  – Macros to **__wake_up_common**

```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 - Implementation

• *wake_up* function
  – Macros to `__wake_up_common`

• New primitive!
  – A call to function with first controllable param

• *Not good enough for commit_creds*
• Upgrade primitives
• Find a function that can call an arbitrary function with address-controlled parameters
• `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);
}
```
• Chaining 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;
  }
}
```
I DON'T OFTEN CALL FUNCTIONS

BUT WHEN I DO, I GET ROOT
Create UAF situation using the vulnerability
Spray unix_dgrams to catch the UAF
Spray unix_dgrams to catch the UAF
Qualaroot – Exploitation Flow

- `__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=CAP_FULL_SET)
PERMISSIVE=1,UID=0

NOT BAD
Demo Time!
IDR mechanism

- ID to pointer translation service
- Handle to kernel objects from user mode without using pointers
User Mode

Create Object Request

Kernel Mode

create_object()

0xFF6DE000

IDR mechanism

Return Safe ID

1

1
Synccockaroot (syncsource vulnerability)

CVE-2016-2503

- **SyncSource objects**
  - Used to synchronize activity between the GPU and the application
- Can be created using IOCTLs to the GPU
  - IOCTL_KGSL_SYNCSOURCE_CREATE
  - IOCTL_KGSL_SYNCSOURCE_DESTROY
- Referenced with the IDR mechanism
```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);
    if (!syncsource)
        goto done;
    /* put reference from syncsource creation */
    kgsl_syncsource_put(syncsource);
    /* put reference from getting the syncsource above */
    kgsl_syncsource_put(syncsource);

done:
    return 0;
}
```
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);
    if (!syncsource)
        goto done;
    /* put reference from syncsource creation */
    kgsl_syncsource_put(syncsource);
    /* put reference from getting the syncsource above */
    kgsl_syncsource_put(syncsource);
    done:
    return 0;
Synccockaroot (syncsource vulnerability)
CVE-2016-2503

Thread A
```
syncsource = kgsl_syncsource_get(id);
...
kgsl_syncsource_put(syncsource);
...
kgsl_syncsource_put(syncsource);
```

Thread B
```
syncsource = kgsl_syncsource_get(id);
...
kgsl_syncsource_put(syncsource);
...
kgsl_syncsource_put(syncsource);
```

free, sprayable data
• 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, object can be sprayed

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
long kgsl_ioctl_gpumem_free_id(
  struct kgsl_device_private *dev_priv,
  unsigned int cmd, void *data)
{
  struct kgsl_gpumem_free_id *param = data;
  struct kgsl_mem_entry *entry = NULL;

  entry = kgsl_sharedmem_find_id(private, param->id);

  if (!entry) {
    return -EINVAL;
  }

  return _sharedmem_free_entry(entry);
}
static long _sharedmem_free_entry(
    struct kgsl_mem_entry *entry)
{
    bool should_free = atomic_compare_exchange(
        entry->pending_free,
        0, 0, /* if pending_free == 0 */
        true, /* then set pending_free = 1 */
    kgsl_mem_entry_put(entry);
    if(should_free)
        kgsl_mem_entry_put(entry);

    return 0;
}
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);
    if (ret)
        kgsl_mem_entry_detach_process(entry);
    return ret;
}
Thread A - allocator

```c
entry = kgsl_mem_entry_create();
...
... id = idr_alloc(..., entry, ...);
...
... initialize_entry(entry);
```

Thread B - releaser

```c
entry = kgsl_sharedmem_find_id(id);
...
... if(!entry)
    return -EINVAL;
...
... sharedmem_safe_free_entry(entry);
```
Thread A - allocator

```c
entry = kgsl_mem_entry_create();
...
... id = idr_alloc(..., entry, ...);
... initialize_entry(entry);
```

Thread B - releaser

```c
entry = kgsl_sharedmem_find_id(id);
...
... if(!entry)
    return -EINVAL;
...
... sharedmem_safe_free_entry(entry);
```
KanGaroot - POC
CVE-2016-2504

• Map memory
• Save the IDR
  – Always get the first free IDR – predictable
• Another thread frees the IDR
  – Before the first thread returns from the IOCTL

\textit{UAF in kgsl\_mem\_entry\_attach\_process on ‘entry’ parameter}
Syncockaroot (CVE-2016-2503)

- **4th April, 2016**: Vulnerability disclosure to Qualcomm
- **2nd May, 2016**: Qualcomm confirmed the vulnerability
- **6th July, 2016**: Qualcomm released a public patch
- **6th July**: Google deployed the patch to their Android devices
Kangaroot (CVE-2016-2504)

4th April, 2016
Vulnerability disclosure to Qualcomm

2nd May, 2016
Qualcomm confirmed the vulnerability

6th July, 2016
Qualcomm released a public patch

1st August, 2016
Google deployed the patch to their Android devices
ASHmenian Devil (CVE-2016-5340)

10th April, 2016
Vulnerability disclosure to Qualcomm

02nd May, 2016
Qualcomm confirmed the vulnerability

28th July, 2016
Qualcomm released a public patch

Google deployed the patch to their Android devices
Qualaroot (CVE-2016-2059)

2nd February, 2016
Vulnerability disclosure to Qualcomm

10th February, 2016
Qualcomm confirmed the vulnerability

29th April, 2016
Qualcomm released a public patch

TBD
Google deployed the patch to their Android devices
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
Am I Vulnerable?

QuadRooter By Check Point

Are you vulnerable?

Tap to Scan

Get ZoneAlarm Protection
Protect Your Enterprise

By tapping 'Scan', you agree with the terms of the licensing agreement

Google Play
QuadRooter Scanner
Thank You!

Yaniv Mordekhay
yanivmo@checkpoint.com

Adam Donenfeld
doadam@gmail.com