Attacking DRM subsystem to gain kernel privilege on Chromebooks

Di Shen a.k.a. Retme (@returnsme)

Keen Lab of Tencent

Bio

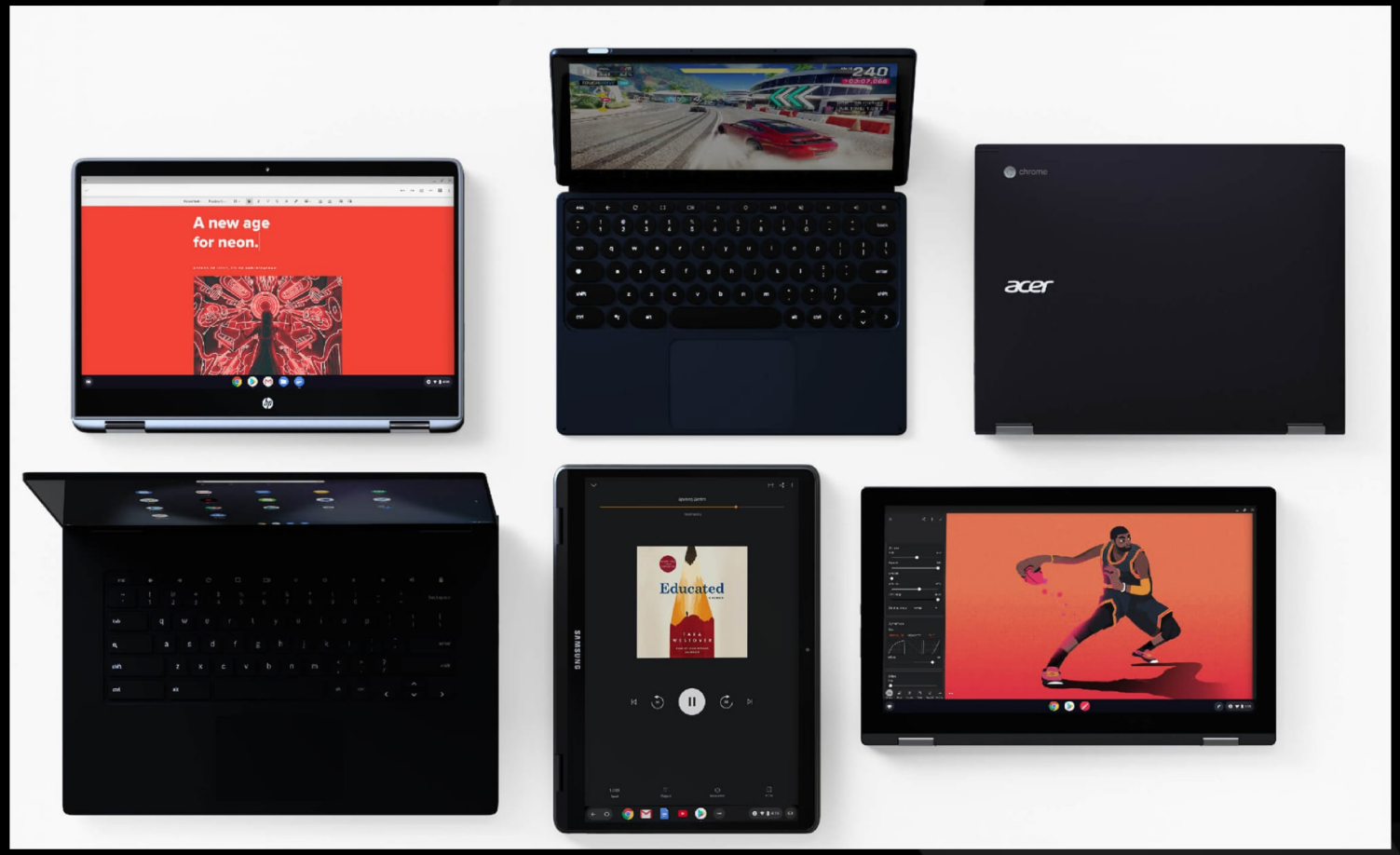
- Di Shen a.k.a. Retme
- Sr. Security Researcher of Keen Lab, Tencent
- Kernel vulnerability hunting and exploitation since 2014
- @returnsme
- github.com/retme7
- Fan of console games, anime and mystery novels(本格推理)

Agenda

- **Chrome OS security: a brief look**
- Choose a better attack surface: for Chrome PWNium
- The Direct Rendering Manager(DRM)
- CVE-2019-16508: analysis and exploitation
- Demonstration

What is a Chromebook

- running the Linux-based Chrome OS
- Different brands and models
  - HP,Dell,ASUS,Lenovo…



Tencent

KEEN security lab

Why Chromebook

- Not for video gamers
- But still good enough to finish daily work
- Easy to use
- Some models can be very cheap
- It is very **SECURE**.

One of the most secure laptop.

- Chrome browser
- Sandbox
  - Both apps and system services are isolated and sandboxed.
- Secure file system / storage
  - Read-only root file system
  - Verified boot and dm-verity
  - Additional Linux security module is stacked ahead of SELinux
- Automatic updates
  - Fix known bugs ASAP
  - Apply security patches to the kernel regularly

Tencent

KEEN security lab

Sandbox

- Use minijail0 to execute system services
- Apply different kind of restrictions:
  - User id
  - Capabilities
  - Namespace
  - Seccomp filter policy
  - SeLinux policy
- More details check the official documents [1] [2]

Verified boot

- Firmware, root disk(kernel),system partition can be verified
- System partition is verified by dm-verity in kernel.
- Details of verified boot [1]

dm-verity

- Used by Chrome OS and Android
- Implemented in Linux kernel.
- Protect system partition from being modified.
- Will calculate the checksum of every block on the disk before I/O operation.
- Will crash the whole system if the checksum is wrong.

Chrome OS Persistence

- ‘Persistence’ means keep your malicious process persistently alive across reboots
- Chrome OS has multiple features to stop persistent attack

Anti Persistence

- dm-verity
- noexec mount for writable data partition
- Linux security module (LSM)
- Have other mitigations for specific persistent exploits in the history

(security/chromiumos/lsm.c)

- A Linux security module
- Reject symlinks that will redirect system's write actions during boot (designed_docs)
- Refuse to mount a path with symlinks
- Hardening against loading module/firmware

Agenda

- Chrome OS security: a brief look
- **Choose a better attack surface: for Chrome PWNium**
- The Direct Rendering Manager(DRM)
- CVE-2019-16508: analysis and exploitation
- Demonstration

Chrome PWNium

- *"We have a standing $150,000 reward for participants that can compromise a Chromebook or Chromebox with device persistence in guest mode (i.e. guest to guest persistence with interim reboot, delivered via a web page)."* – Chrome VRP
- Previously it was $100k, raised to $150k this July.

Tencent

KEEN security lab

PWNium requirements (browser category)

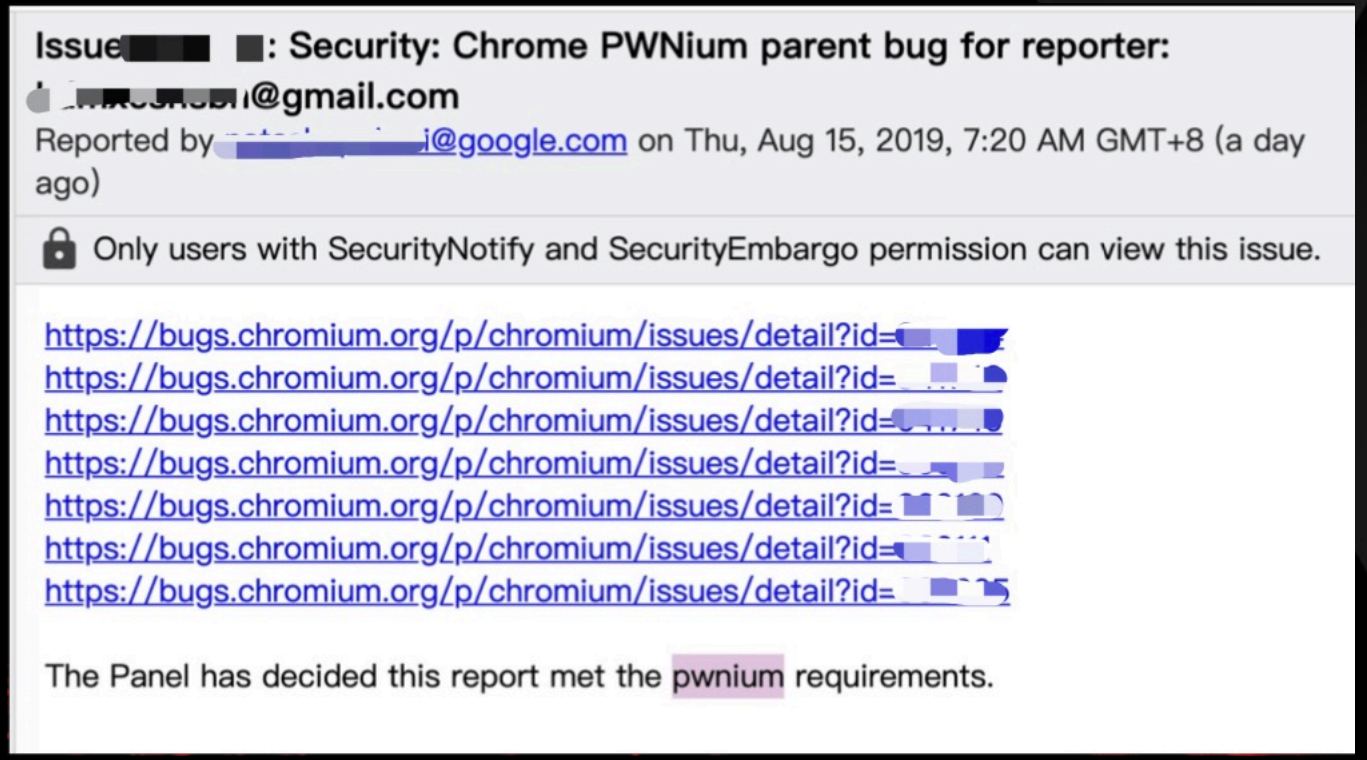
- Chrome browser Renderer RCE (in guest mode)
- Sandbox escape
- OS Persistence

Keenlab's exploit chain

- CVE-2019-5825: Out-of-bounds write in V8
- CVE-2019-5826: Use-after-free in IndexedDB
- CVE-2019-16508: Privilege escalation in kernel mode
- CVE-2019-13690: Privilege escalation in user mode
- CVE-2019-13689: Chrome OS persistence bug

- Credit goes to Gengming Liu, Jianyu Chen, Zhen Feng, Jessica Liu and Retme at Tencent Keenlab

The chain met PWNium requirements



Issue [redacted]: Security: Chrome PWNium parent bug for reporter: [redacted]@gmail.com

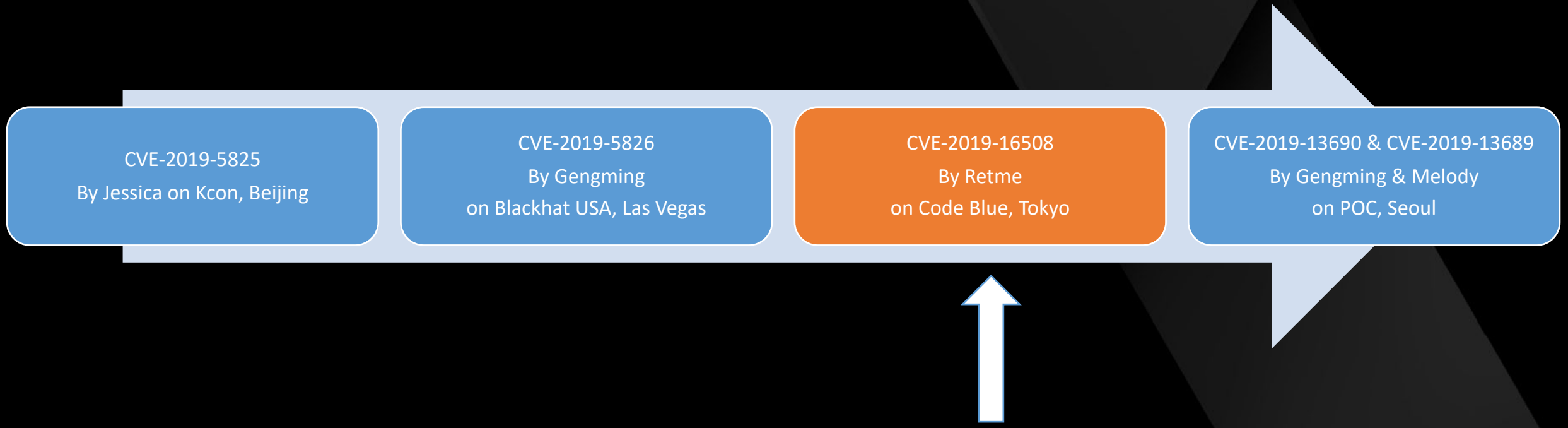
Reported by [redacted]@google.com on Thu, Aug 15, 2019, 7:20 AM GMT+8 (a day ago)

Only users with SecurityNotify and SecurityEmbargo permission can view this issue.

https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]
https://bugs.chromium.org/p/chromium/issues/detail?id=[redacted]

The Panel has decided this report met the pwnium requirements.

Disclosure process



My contribution to this chain

- The end of March,2019
- We already had V8 bug for RCE and sandbox escape
- But no privilege escalation and Persistence
- We need ROOT privilege to trigger the persistent exploitation

Attack surface : my choice

- I didn’t need to find a Linux kernel vulnerability that affect all Linux distribution.
- A bug targeted at official Chrome OS hardware will be fine.
- Need to make it within 30 days or so
- Let’s find a vendor bug on specific Chromebook.

Which one?

Release	OEM	Model	Code name	Board name(s)	Base board	User ABI	Kernel	Kernel ABI	Platform	Form Factor	First Release
排序	排序	排序	排序	排序	排序	排序	排序	排序	排序	排序	排序
2018年8月8日	ASUS	ASUS Chromebook C223	babymega	coral	coral	x86_64	4.4	x86_64	ApolloLake	Chromebook	R67
2018年8月31日	Lenovo	Lenovo Chromebook C330	maple	hana	oak	arm	3.18	aarch64	MT8173	Convertible	R67
2018年9月14日	ASUS	ASUS Chromebook C423	rabbid	coral	coral	x86_64	4.4	x86_64	ApolloLake	Chromebook	R68
2018年9月17日	Lenovo	Lenovo Chromebook S330	maple14	hana	oak	arm	3.18	aarch64	MT8173	Chromebook	M67
2018年10月8日	ASUS	ASUS Chromebook C523	babytiger	coral	coral	x86_64	4.4	x86_64	ApolloLake	Chromebook	R69
2018年10月15日	Acer		Epaulette	coral	coral	x86_64	4.4	x86_64	ApolloLake	Chromebook	R69
2018年10月26日	Dell	Dell Inspiron 14 2-in-1 Model 7486	vayne	vayne	nami	x86_64	4.4	x86_64	Kabylake-U/R	Chromebook	R69
2018年10月26日	Lenovo	Yoga Chromebook C630	pantheon	pantheon	nami	x86_64	4.4	x86_64	Kabylake-U/R	Convertible	R69

Obviously the Mediatek-based one

- Mediatek, the paradise for bug hunter…
- Lenovo S330
- Based on MT8173 platform
- Board name ”hana” (花?)

Easy to exploit?

Mitigation	Enabled？
PXN *	Yes
PAN	Yes
UAO	No
KASLR	No
Any others	No

* PXN : Privileged eXecute Never
  PAN : Privileged Access Never
  UAO : User Access Override

What is next?

- Bought Lenovo S330 from Amazon UK…
- Then wait 2 weeks for the global delivery
- Think about which part of code may be vulnerable during my wait
- It ‘s Direct Rendering Manager

Agenda

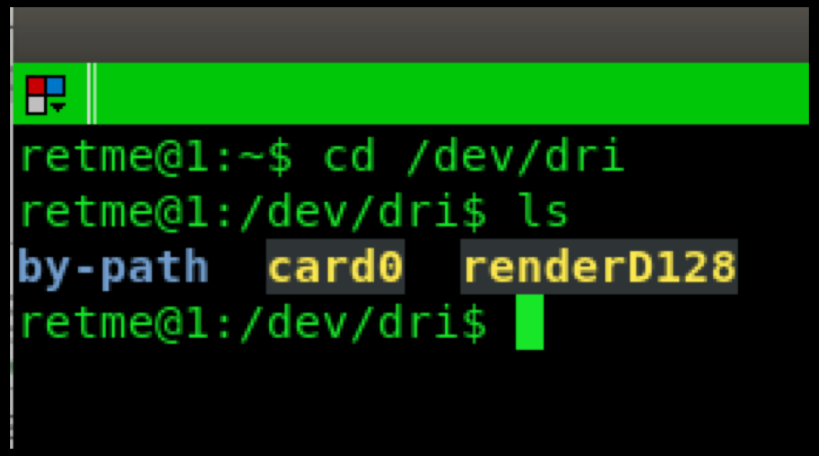
- Chrome OS security: a brief look
- Choose a better attack surface: for Chrome pwnium
- **The Direct Rendering Manager(DRM)**
- CVE-2019-16508: analysis and exploitation
- Demonstration

Direct Rendering Manager

- a subsystem of the Linux kernel
- drivers/gpu/drm/
- responsible for interfacing with GPUs
- MT8173 is using PowerVR GPU by Imagination

Provide accessible /dev path

- There will be card0,card1,card2… , if you have multiple video card
- It is accessible for normal user (chronos) on Chrom OS.
- It exposes IOCTL interface for users



Public command handled by drm_ioctl()

- We have no permission to request the commands which is set these flags:
  - DRM_ROOT_ONLY
  - DRM_MASTER
  - DRM_AUTH
- Few commands are OK to request
- Not a big attack surface

ers / gpu / drm / drm_ioctl.c — Search Identifier

```
/** Ioctl table */
static const struct drm_ioctl_desc drm_ioctls[] = {
        DRM_IOCTL_DEF(DRM_IOCTL_VERSION, drm_version, DRM_UNLOCKED|DRM_RENDER_ALLOW),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_UNIQUE, drm_getunique, 0),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_MAGIC, drm_getmagic, 0),
        DRM_IOCTL_DEF(DRM_IOCTL_IRQ_BUSID, drm_irq_by_busid, DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_MAP, drm_getmap, DRM_UNLOCKED),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_CLIENT, drm_getclient, DRM_UNLOCKED),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_STATS, drm_getstats, DRM_UNLOCKED),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_CAP, drm_getcap, DRM_UNLOCKED|DRM_RENDER_ALLOW),
        DRM_IOCTL_DEF(DRM_IOCTL_SET_CLIENT_CAP, drm_setclientcap, 0),
        DRM_IOCTL_DEF(DRM_IOCTL_SET_VERSION, drm_setversion, DRM_MASTER),

        DRM_IOCTL_DEF(DRM_IOCTL_SET_UNIQUE, drm_setunique, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_BLOCK, drm_noop, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_UNBLOCK, drm_noop, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_AUTH_MAGIC, drm_authmagic, DRM_AUTH|DRM_MASTER),

        DRM_IOCTL_DEF(DRM_IOCTL_ADD_MAP, drm_legacy_addmap_ioctl, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_RM_MAP, drm_legacy_rmmap_ioctl, DRM_AUTH),

        DRM_IOCTL_DEF(DRM_IOCTL_SET_SAREA_CTX, drm_legacy_setsareactx, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_SAREA_CTX, drm_legacy_getsareactx, DRM_AUTH),

        DRM_IOCTL_DEF(DRM_IOCTL_SET_MASTER, drm_setmaster_ioctl, DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_DROP_MASTER, drm_dropmaster_ioctl, DRM_ROOT_ONLY),

        DRM_IOCTL_DEF(DRM_IOCTL_ADD_CTX, drm_legacy_addctx, DRM_AUTH|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_RM_CTX, drm_legacy_rmctx, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_MOD_CTX, drm_noop, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_GET_CTX, drm_legacy_getctx, DRM_AUTH),
        DRM_IOCTL_DEF(DRM_IOCTL_SWITCH_CTX, drm_legacy_switchctx, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_NEW_CTX, drm_legacy_newctx, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_RES_CTX, drm_legacy_resctx, DRM_AUTH),

        DRM_IOCTL_DEF(DRM_IOCTL_ADD_DRAW, drm_noop, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),
        DRM_IOCTL_DEF(DRM_IOCTL_RM_DRAW, drm_noop, DRM_AUTH|DRM_MASTER|DRM_ROOT_ONLY),

        DRM_IOCTL_DEF(DRM_IOCTL_LOCK, drm_legacy_lock, DRM_AUTH),
        DRM_IOCTL_DEF(DRM_IOCTL_UNLOCK, drm_legacy_unlock, DRM_AUTH),
```

DRM supports private commands by vendors

- Will call vendor's private IOCTL handler if *nr* > *DRM_COMMAND_BASE*

```
long drm_ioctl(struct file *filp,
               unsigned int cmd, unsigned long arg)
{
        struct drm_file *file_priv = filp->private_data;
        struct drm_device *dev;
        const struct drm_ioctl_desc *ioctl = NULL;
        drm_ioctl_t *func;
        unsigned int nr = DRM_IOCTL_NR(cmd);
        int retcode = -EINVAL;
        char stack_kdata[128];
        char *kdata = NULL;
        unsigned int usize, asize;

        dev = file_priv->minor->dev;

        if (drm_device_is_unplugged(dev))
                return -ENODEV;

        if ((nr >= DRM_CORE_IOCTL_COUNT) &&
            ((nr < DRM_COMMAND_BASE) || (nr >= DRM_COMMAND_END)))
                goto err_i1;
        if ((nr >= DRM_COMMAND_BASE) && (nr < DRM_COMMAND_END) &&
            (nr < DRM_COMMAND_BASE + dev->driver->num_ioctls)) {
                u32 drv_size;
                ioctl = &dev->driver->ioctls[nr - DRM_COMMAND_BASE];
                drv_size = _IOC_SIZE(ioctl->cmd_drv);
                usize = asize = _IOC_SIZE(cmd);
                if (drv_size > asize)
                        asize = drv_size;
                cmd = ioctl->cmd_drv;
        }
```

Private handler on S330

- /dev/dri/card0
  - mkt_ioctls()
- /dev/dri/card1
  - pvr_drm_ioctls()

```
static struct drm_driver mtk_drm_driver = {
    .driver_features = DRIVER_MODESET | DRIVER_GEM | DRIVER_PRIME |
            DRIVER_ATOMIC | DRIVER_RENDER,

    .get_vblank_counter = drm_vblank_count,
    .enable_vblank = mtk_drm_crtc_enable_vblank,
    .disable_vblank = mtk_drm_crtc_disable_vblank,

    .gem_free_object = mtk_drm_gem_free_object,
    .gem_vm_ops = &drm_gem_cma_vm_ops,
    .dumb_create = mtk_drm_gem_dumb_create,
    .dumb_map_offset = mtk_drm_gem_dumb_map_offset,
    .dumb_destroy = drm_gem_dumb_destroy,

    .prime_handle_to_fd = drm_gem_prime_handle_to_fd,
    .prime_fd_to_handle = drm_gem_prime_fd_to_handle,
    .gem_prime_export = drm_gem_prime_export,
    .gem_prime_import = drm_gem_prime_import,
    .gem_prime_get_sg_table = mtk_gem_prime_get_sg_table,
    .gem_prime_import_sg_table = mtk_gem_prime_import_sg_table,
    .ioctls = mtk_ioctls,
    .num_ioctls = ARRAY_SIZE(mtk_ioctls),

    .name = DRIVER_NAME,
    .desc = DRIVER_DESC,
    .date = DRIVER_DATE,
    .major = DRIVER_MAJOR,
    .minor = DRIVER_MINOR,
};
```

```
const struct drm_driver pvr_drm_generic_driver = {
    .driver_features    = DRIVER_MODESET | DRIVER_RENDER,

    .dev_priv_size      = 0,
#if (LINUX_VERSION_CODE >= KERNEL_VERSION(3, 18, 0))
    .load               = NULL,
    .unload             = NULL,
#else
    .load               = pvr_drm_load,
    .unload             = pvr_drm_unload,
#endif
    .open               = pvr_drm_open,
    .postclose          = pvr_drm_release,

    .ioctls             = pvr_drm_ioctls,
    .num_ioctls         = ARRAY_SIZE(pvr_drm_ioctls),
    .fops               = &pvr_drm_fops,

    .name               = PVR_DRM_DRIVER_NAME,
    .desc               = PVR_DRM_DRIVER_DESC,
    .date               = PVR_DRM_DRIVER_DATE,
    .major              = PVRVERSION_MAJ,
    .minor              = PVRVERSION_MIN,
    .patchlevel         = PVRVERSION_BUILD,
};
```

pvr_drm_ioctl()

- Handles PVR_SRVKM_CMD request
- Argument
  - bridge_id [0-24]
  - bridge_func_id [0-N]

```
struct drm_pvr_srvkm_cmd {
        __u32 bridge_id;
        __u32 bridge_func_id;
        __u64 in_data_ptr;
        __u64 out_data_ptr;
        __u32 in_data_size;
        __u32 out_data_size;
};
```

```
/*   0: Default handler */
#define PVRSRV_BRIDGE_DEFAULT                   0UL
#define PVRSRV_BRIDGE_DEFAULT_DISPATCH_FIRST    0UL
#define PVRSRV_BRIDGE_DEFAULT_DISPATCH_LAST     (PVRSRV_BRIDGE_DEFAULT_DISPATCH_FIRST)
/*   1: CORE functions  */
#define PVRSRV_BRIDGE_SRVCORE                   1UL
#define PVRSRV_BRIDGE_SRVCORE_DISPATCH_FIRST    (PVRSRV_BRIDGE_DEFAULT_DISPATCH_LAST+1)
#define PVRSRV_BRIDGE_SRVCORE_DISPATCH_LAST     (PVRSRV_BRIDGE_SRVCORE_DISPATCH_FIRST +

/*   2: SYNC functions  */
#define PVRSRV_BRIDGE_SYNC                  2UL
#define PVRSRV_BRIDGE_SYNC_DISPATCH_FIRST   (PVRSRV_BRIDGE_SRVCORE_DISPATCH_LAST + 1)
#define PVRSRV_BRIDGE_SYNC_DISPATCH_LAST    (PVRSRV_BRIDGE_SYNC_DISPATCH_FIRST + PVRSRV_

/*   3: SYNCEXPORT functions  */
#define PVRSRV_BRIDGE_SYNCEXPORT            3UL
#if defined(SUPPORT_INSECURE_EXPORT) && defined(SUPPORT_SERVER_SYNC)
#define PVRSRV_BRIDGE_SYNCEXPORT_DISPATCH_FIRST (PVRSRV_BRIDGE_SYNC_DISPATCH_LAST + 1)
#define PVRSRV_BRIDGE_SYNCEXPORT_DISPATCH_LAST  (PVRSRV_BRIDGE_SYNCEXPORT_DISPATCH_FIRST
#else
#define PVRSRV_BRIDGE_SYNCEXPORT_DISPATCH_FIRST 0
#define PVRSRV_BRIDGE_SYNCEXPORT_DISPATCH_LAST  (PVRSRV_BRIDGE_SYNC_DISPATCH_LAST)
#endif
```

pvr_drm_ioctl()

- Handles PVR_SRVKM_CMD request
- Argument
  - bridge_id [0-24]
  - bridge_func_id [0-N]

```
#define PVRSRV_BRIDGE_SYNC_CMD_FIRST                0
#define PVRSRV_BRIDGE_SYNC_ALLOCSYNCPRIMITIVEBLOCK          PVRSRV_BRIDGE_SYNC_CMD_FIRST+0
#define PVRSRV_BRIDGE_SYNC_FREESYNCPRIMITIVEBLOCK           PVRSRV_BRIDGE_SYNC_CMD_FIRST+1
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMSET          PVRSRV_BRIDGE_SYNC_CMD_FIRST+2
#define PVRSRV_BRIDGE_SYNC_SERVERSYNCPRIMSET            PVRSRV_BRIDGE_SYNC_CMD_FIRST+3
#define PVRSRV_BRIDGE_SYNC_SERVERSYNCALLOC          PVRSRV_BRIDGE_SYNC_CMD_FIRST+4
#define PVRSRV_BRIDGE_SYNC_SERVERSYNCFREE           PVRSRV_BRIDGE_SYNC_CMD_FIRST+5
#define PVRSRV_BRIDGE_SYNC_SERVERSYNCQUEUEHWOP          PVRSRV_BRIDGE_SYNC_CMD_FIRST+6
#define PVRSRV_BRIDGE_SYNC_SERVERSYNCGETSTATUS          PVRSRV_BRIDGE_SYNC_CMD_FIRST+7
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPCREATE         PVRSRV_BRIDGE_SYNC_CMD_FIRST+8
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPTAKE           PVRSRV_BRIDGE_SYNC_CMD_FIRST+9
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPREADY          PVRSRV_BRIDGE_SYNC_CMD_FIRST+10
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPCOMPLETE           PVRSRV_BRIDGE_SYNC_CMD_FIRST+11
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPDESTROY            PVRSRV_BRIDGE_SYNC_CMD_FIRST+12
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMPDUMP            PVRSRV_BRIDGE_SYNC_CMD_FIRST+13
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMPDUMPVALUE           PVRSRV_BRIDGE_SYNC_CMD_FIRST+14
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMPDUMPPOL         PVRSRV_BRIDGE_SYNC_CMD_FIRST+15
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMOPPDUMPPOL           PVRSRV_BRIDGE_SYNC_CMD_FIRST+16
#define PVRSRV_BRIDGE_SYNC_SYNCPRIMPDUMPCBP         PVRSRV_BRIDGE_SYNC_CMD_FIRST+17
#define PVRSRV_BRIDGE_SYNC_SYNCALLOCEVENT           PVRSRV_BRIDGE_SYNC_CMD_FIRST+18
#define PVRSRV_BRIDGE_SYNC_SYNCFREEEVENT            PVRSRV_BRIDGE_SYNC_CMD_FIRST+19
#define PVRSRV_BRIDGE_SYNC_CMD_LAST         (PVRSRV_BRIDGE_SYNC_CMD_FIRST+19)
```

```
struct drm_pvr_srvkm_cmd {
    __u32 bridge_id;
    __u32 bridge_func_id;
    __u64 in_data_ptr;
    __u64 out_data_ptr;
    __u32 in_data_size;
    __u32 out_data_size;
};
```

pvr_drm_ioctl()

- (bridge_id*bridge_func_id) interfaces can be accessed by user mode
- Exactly the attack surface I want!

```
struct drm_pvr_srvkm_cmd {
    __u32 bridge_id;
    __u32 bridge_func_id;
    __u64 in_data_ptr;
    __u64 out_data_ptr;
    __u32 in_data_size;
    __u32 out_data_size;
};
```

Time to build a fuzzer

- Recompile the kernel and flash it to Chromebook [1][2]
- Define fuzzing rules targeting PowerVR interface PVR_SRVKM_CMD
- Run the fuzzer via SSH and wait for one night
- Maybe play some games on PS4 during my wait …

Agenda

- Chrome OS security: a brief look
- Choose a better attack surface: for Chrome pwnium
- The Direct Rendering Manager(DRM)
- **CVE-2019-16508: analysis and exploitation**
- Demonstration

Tencent

KEEN security lab

CVE-2019-16508

- Stack trace

```
[  119.409544] [<ffffffc00089bf0c>] PVRSRVBridgeSyncPrimOpCreate+0x53c/0x678
[  119.409567] [<ffffffc0008c4d6c>] BridgedDispatchKM+0x404/0x5d4
[  119.409589] [<ffffffc00091a1c8>] PVRSRV_BridgeDispatchKM+0x1a0/0x1ec
[  119.409612] [<ffffffc00033570c>] drm_ioctl+0x5d0/0x6ac
[  119.409633] [<ffffffc0002ac2ac>] SyS_ioctl+0x848/0x8bc
```

- PVRSRVBridgeSyncPrimOpCreate()
- bridge_id = 2; bridge_func_id = 8

PVRSRVBridgeSyncPrimOpCreate()

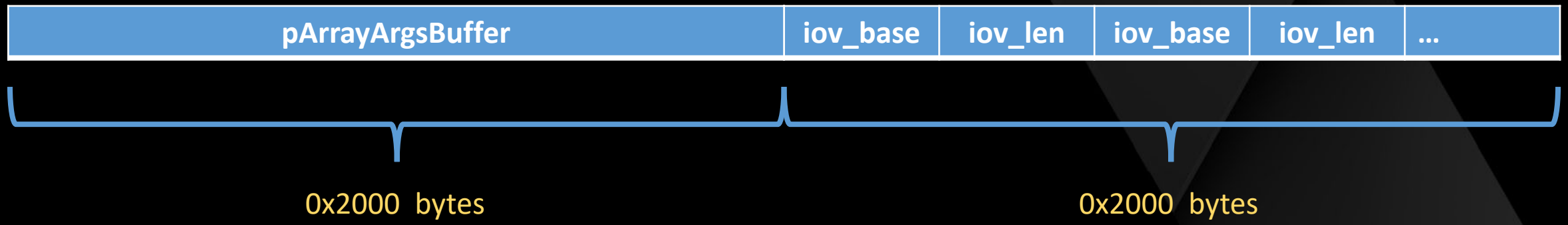
- Miss checking buffer size coming from user input *psSyncPrimOpCreateIN*, which may lead to an integer overflow.
- It’s able to convert this bug to a heap out-of-bound write, then achieve arbitrary memory overwriting.

- *ui32BufferSize* overflowed
- A smaller buffer *pArrayArgsBuffer*
- Heap OOB Write

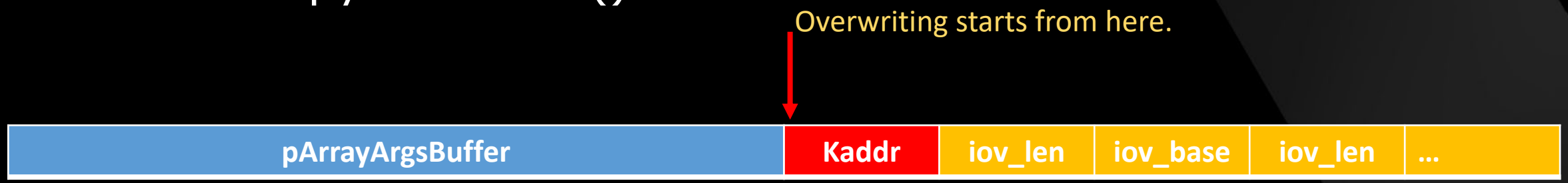
```
static IMG_INT
PVRSRVBridgeSyncPrimOpCreate(IMG_UINT32 ui32DispatchTableEntry,
                             PVRSRV_BRIDGE_IN_SYNCPRIMOPCREATE *psSyncPrimOpCreateIN,
                             PVRSRV_BRIDGE_OUT_SYNCPRIMOPCREATE *psSyncPrimOpCreateOUT,
                            CONNECTION_DATA *psConnection)
{
    ...
    /* 1. ui32BufferSize overflowed here*/
    IMG_UINT32 ui32BufferSize =
            (psSyncPrimOpCreateIN->ui32SyncBlockCount * sizeof(SYNC_PRIMITIVE_BLOCK *)) +
            (psSyncPrimOpCreateIN->ui32SyncBlockCount * sizeof(IMG_HANDLE)) +
            (psSyncPrimOpCreateIN->ui32ClientSyncCount * sizeof(IMG_UINT32)) +
            (psSyncPrimOpCreateIN->ui32ClientSyncCount * sizeof(IMG_UINT32)) +
            (psSyncPrimOpCreateIN->ui32ServerSyncCount * sizeof(SERVER_SYNC_PRIMITIVE *)) +
            (psSyncPrimOpCreateIN->ui32ServerSyncCount * sizeof(IMG_HANDLE)) +
            0;
    if (ui32BufferSize != 0)
    {
        ...
            /* 2. Allocate a buffer much smaller than expectation */
            pArrayArgsBuffer = OSAllocMemNoStats(ui32BufferSize);
        ...
    }
    ...
            /* Copy the data over */
            if (psSyncPrimOpCreateIN->ui32SyncBlockCount * sizeof(IMG_HANDLE) > 0)
            {
                /* 3. OSCopyFromUser  trigger a  heap OOB write*/
                if ( OSCopyFromUser(NULL,
                    hBlockListInt2,
                    (const void __user *) psSyncPrimOpCreateIN->phBlockList,
                    psSyncPrimOpCreateIN->ui32SyncBlockCount * sizeof(IMG_HANDLE)) != PVRSRV_OK )
                {
                    psSyncPrimOpCreateOUT->eError = PVRSRV_ERROR_INVALID_PARAMS;
```

Exploiting Heap OOB

- Compromise pipe subsystem (check my talk on CodeBlue 17)



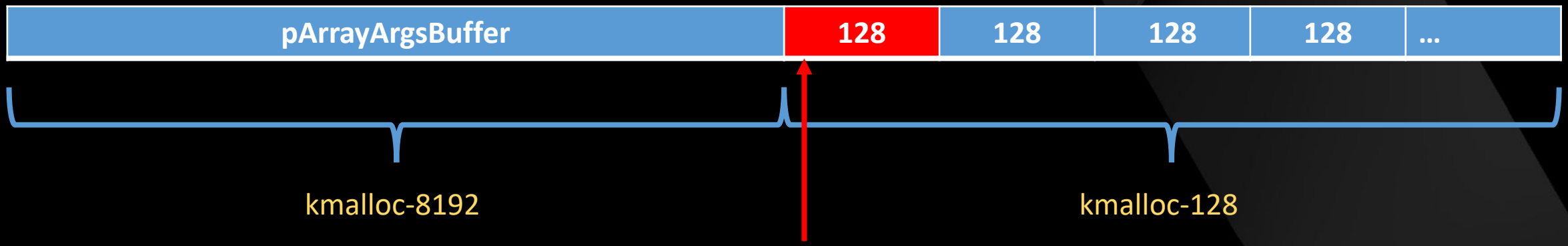
- After OSCopyFromUser()



- Write pipe_fd to copy buffer to ***kaddr*** (arbitrary kernel address)

Reliability

- Kernel may crash in 50% by “HARDENED_USERCOPY”
- That happens when my heap spray failed, the copy_from_user() overwrite some small heap slots.



copy_from_user(ptr-object-128,buffer-8192,8192)

KERNEL PANIC BY HARDENED_USERCOPY

HARDENDED_USERCOPY

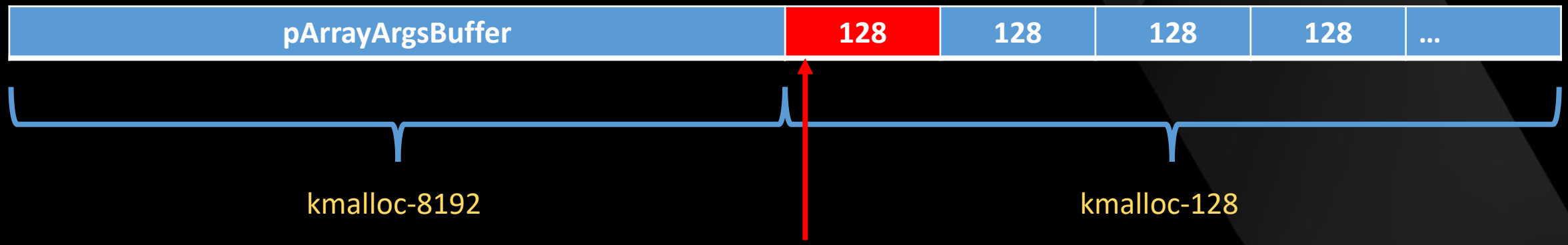
- Seems was back ported to Linux 3.18 by chromium
- Boundary check in copy_from/to_user() by instrumentation
- check_copy_size()

```
static __always_inline unsigned long __must_check
copy_from_user(void *to, const void __user *from, unsigned long n)
{
        if (likely(check_copy_size(to, n, false)))
                n = _copy_from_user(to, from, n);
        return n;
}
```

```
/*
 * Validates that the given object is:
 * - not bogus address
 * - fully contained by stack (or stack frame, when available)
 * - fully within SLAB object (or object whitelist area, when available)
 * - not in kernel text
 */
void __check_object_size(const void *ptr, unsigned long n, bool to_user)
{
        if (static_branch_unlikely(&bypass_usercopy_checks))
                return;

        /* Skip all tests if size is zero. */
        if (!n)
                return;

        /* Check for invalid addresses. */
        check_bogus_address((const unsigned long)ptr, n, to_user);

        /* Check for bad stack object. */
        switch (check_stack_object(ptr, n)) {
        case NOT_STACK:
                /* Object is not touching the current process stack. */
                break;
        case GOOD_FRAME:
        case GOOD_STACK:
                /*
```

How to bypass it?

- I copied 8192 bytes to object-128 previously
- Actually copying 0x20 bytes is good enough to overwrite iov_base.
- The copied content is {0, 0, kADDR, iov_len}



copy_from_user(ptr-object-128,buffer-8192,0x20)
It will **NOT** crash when my spray failed any more.

Agenda

- Chrome OS security: a brief look
- Choose a better attack surface: for Chrome pwnium
- The Direct Rendering Manager(DRM)
- CVE-2019-16508: analysis and exploitation
- **Demonstration**

Tencent

KEEN security lab

Thank you

@returnsme

retme7@gmail.com

github.com/retme7

Tencent

KEEN security lab

Tencent

KEEN
security
lab