Android安全机制

罗升阳

http://weibo.com/shengyangluo

http://blog.csdn.net/luoshengyang

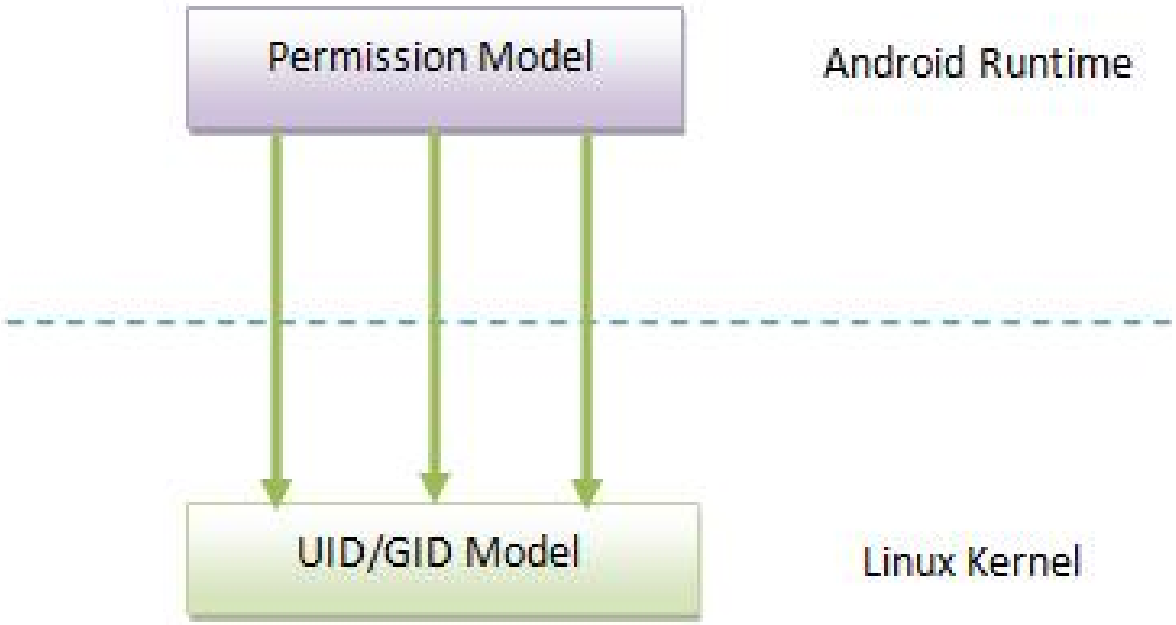
About Me

- 《老罗的Android之旅》博客作者
- 《Android系统源代码情景分析》书籍作者
- 博客：http://blog.csdn.net/Luoshengyang
- 微博：http://weibo.com/shengyangluo

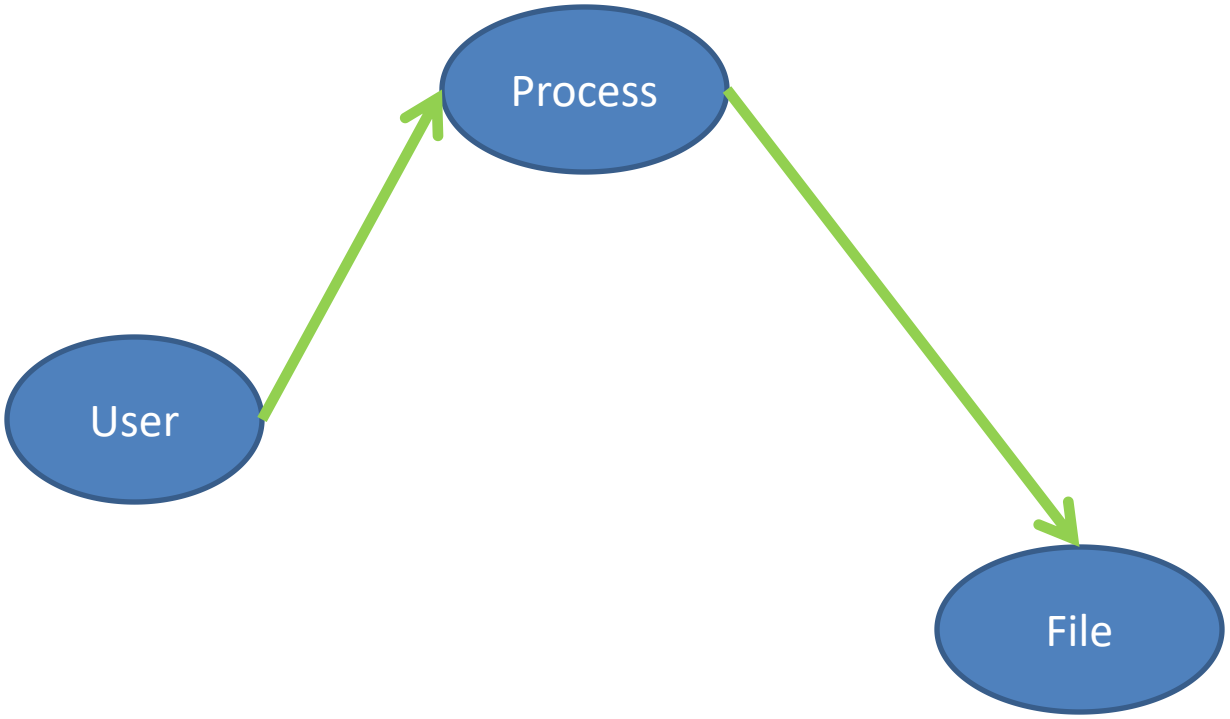
Agenda

- Android安全模型
- SO注入技术
- SO加壳技术
- C/C++函数拦截技术
- DEX注入技术
- DEX加壳技术
- Java函数拦截技术

Android安全模型



Android安全模型



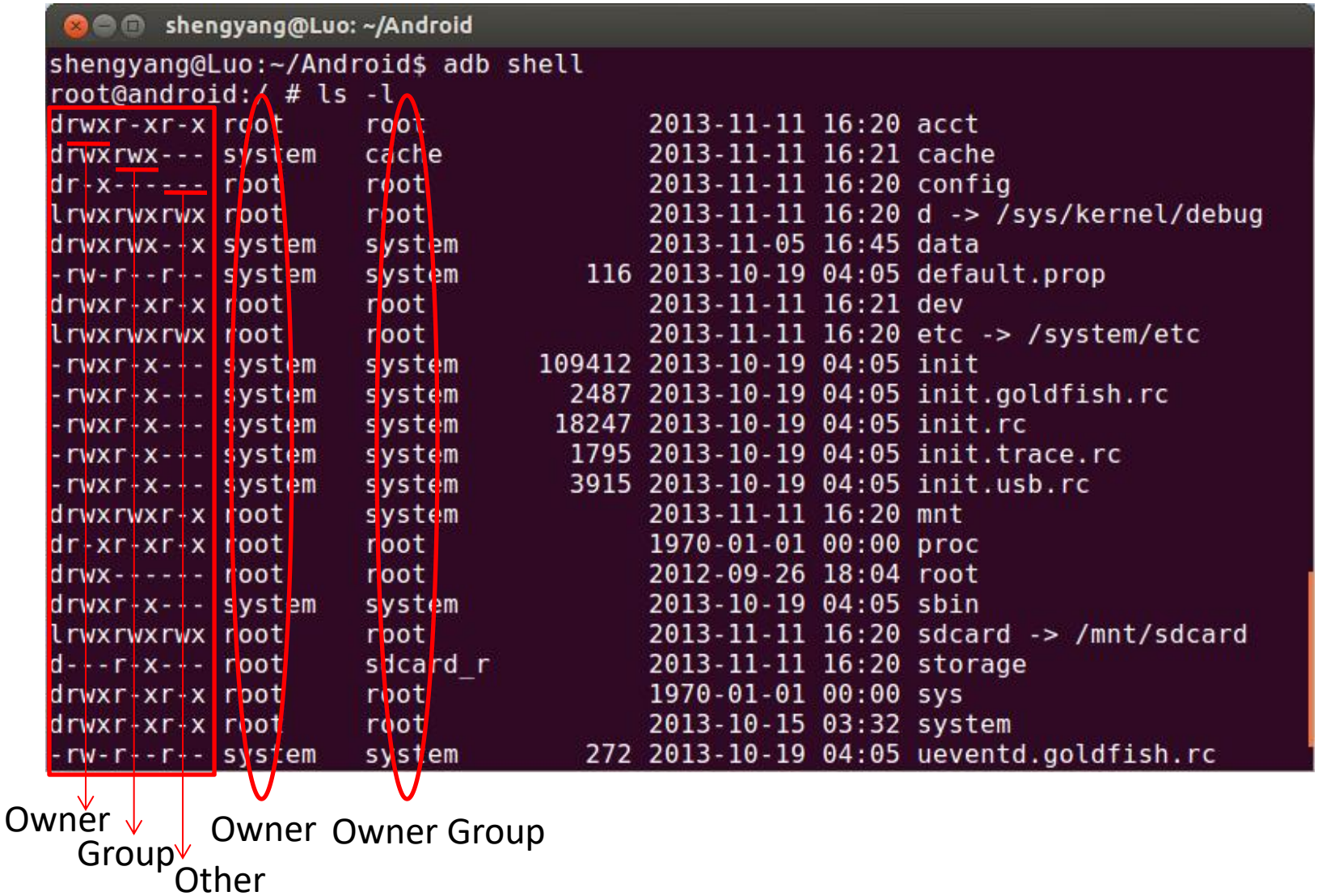
Android安全模型

- 用户
  - 系统中可以存在多个用户，每一个用户都具有一个UID
  - 用户按组划分形成用户组，每一个用户组都具有一个GID
  - 一个用户可以属于多个用户组

Android安全模型

- 文件
  - 每一个文件都具有三种权限
    - Read、Write、Execute
  - 文件权限按用户属性分为三组
    - Owner、Group、Other

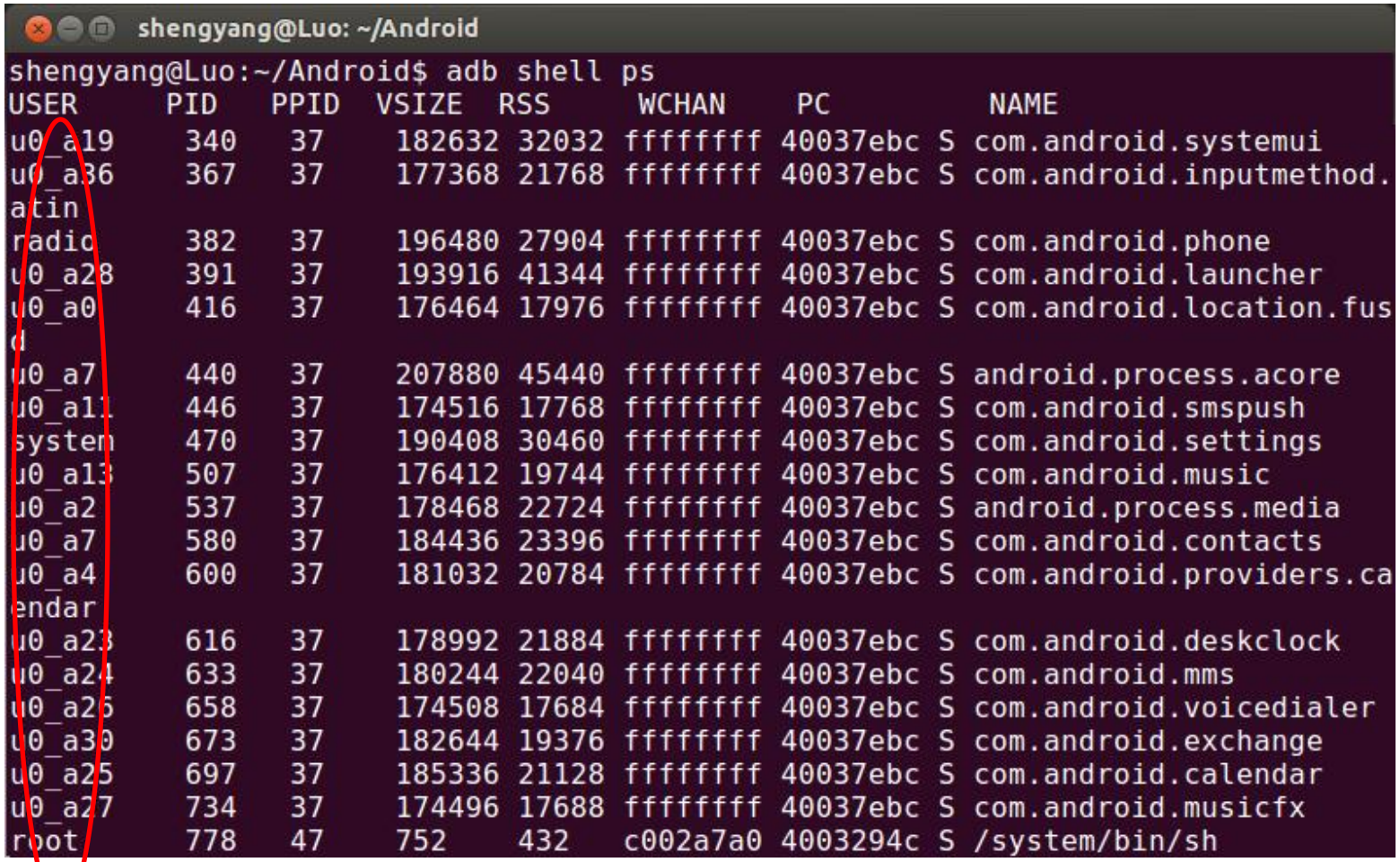
Android安全模型



Android安全模型

- 进程
  - UID -- setuid
  - GID -- setgid
  - Supplementary GIDS – setgroups
  - Capabilities -- capset

Android安全模型

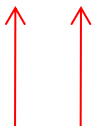


UID

Android安全模型

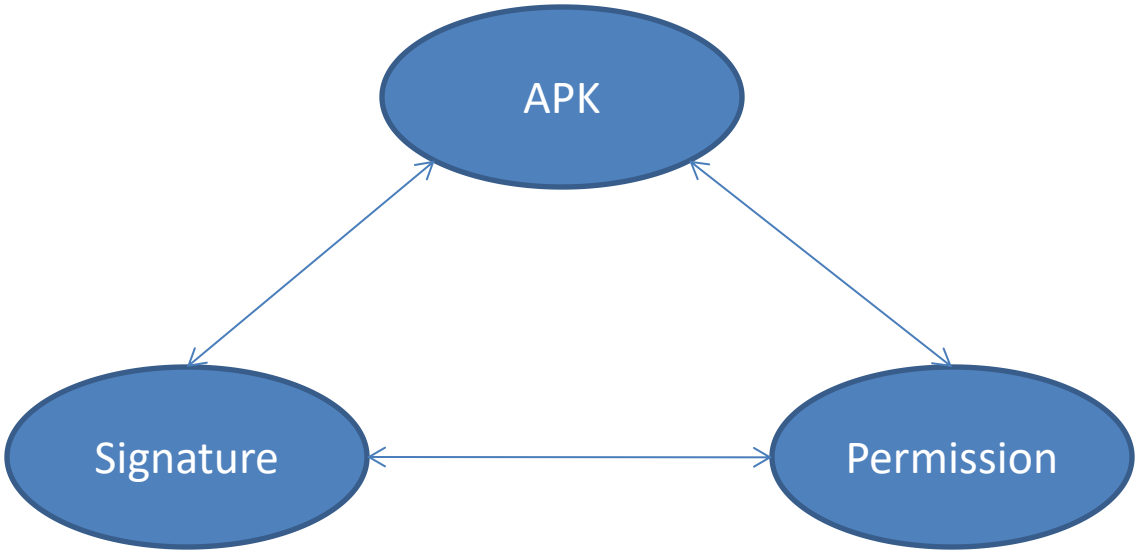
- 系统中的第一个进程Init的UID是root
- 子进程的UID默认与父进程相同，但可以通过setuid进行修改
- 子进程被fork之后exec了一个设置了SUID位的bin文件，那么子进程的UID变为该bin文件的Ower UID

```
shengyang@Luo:~/Android$ adb shell ls -l /system/xbin/su
-rwsr-sr-x root     root        95912 2013-11-12 10:49 su
```



SUID SGID

Android安全模型



Android安全模型

- 每一个APK在安装的时候，PMS都会给它分配一个唯一的UID和GID
  - 如果两个APK具有相同的签名，那么可以通过android:sharedUserId申请分配相同的UID和GID
  - 如果一个APK具有平台签名，那么可以通过android:sharedUserId=“android.uid.system”获得System UID

Android安全模型

- 每一个APK都可以通过<uses-permission android:name="android.permission.XXX"/>申请若干个Permission
  - 有些Permission需要具有平台签名才可以申请，如INSTALL_PACKAGES
- 每一个Permission都对应于一个Supplementary GID，因此，给APK分配Permission即为APK分配Supplementary GID

Android安全模型

- APK进程是由UID为root的Zygote进程fork出来的，fork之后：

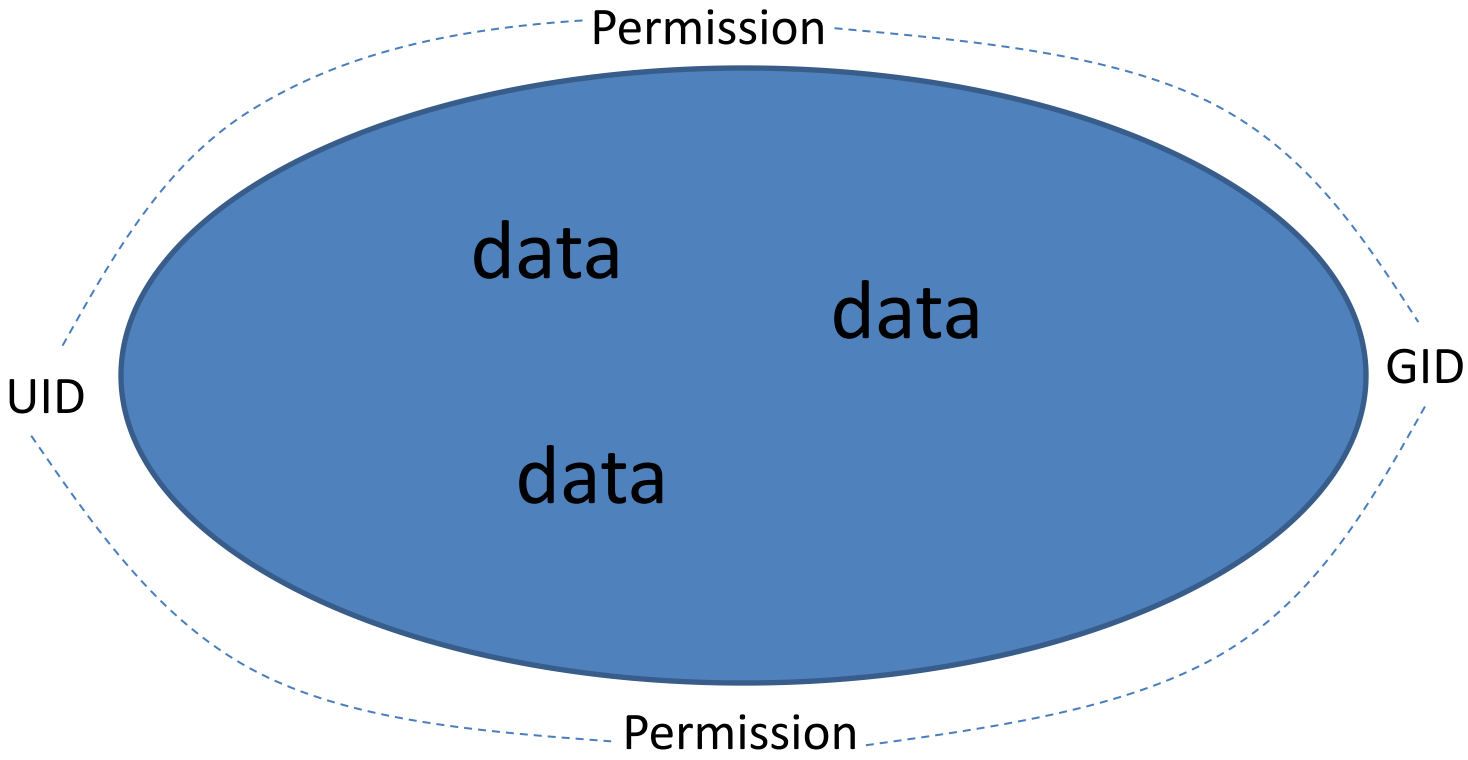
```
static pid_t forkAndSpecializeCommon(const u4* args, bool isSystemServer)
{
    pid_t pid;

    uid_t uid = (uid_t) args[0];
    gid_t gid = (gid_t) args[1];
    ArrayObject* gids = (ArrayObject *)args[2];
    u4 debugFlags = args[3];
    ArrayObject *rlimits = (ArrayObject *)args[4];
    int64_t permittedCapabilities, effectiveCapabilities;

    if (isSystemServer) {
        permittedCapabilities = args[5] | (int64_t) args[6] << 32;
        effectiveCapabilities = args[7] | (int64_t) args[8] << 32;
    } else {
        permittedCapabilities = effectiveCapabilities = 0;
    }
    ......

    pid = fork();

    if (pid == 0) {
        ......
        err = setgroupsIntarray(gids);
        ......
        err = setrlimitsFromArray(rlimits);
        ......
        err = setgid(gid);
        ......
        err = setuid(uid);
        ......
        err = setCapabilities(permittedCapabilities, effectiveCapabilities);
        ......
    }

    return pid;
}
```

Android安全模型

- PMS记录有每一个APK所申请的Permission，当APK调用敏感API时，相应的模块就会通过PMS会验证调用APK是否申请有相应的Permission

```
public class PackageManagerService extends IPackageManager.Stub {
    ......

    public void installPackageWithVerificationAndEncryption(Uri packageURI,
            IPackageInstallObserver observer, int flags, String installerPackage
Name,
            VerificationParams verificationParams, ContainerEncryptionParams enc
ryptionParams) {
        mContext.enforceCallingOrSelfPermission(android.Manifest.permission.INST
ALL_PACKAGES,
                null);
        ......
    }

    ......
}
```

Android安全模型



Application Sandbox

Android安全模型

- 突破沙箱：创建其它APK也能访问的文件

public abstract FileOutputStream **openFileOutput** (String name, int mode) Added in API level 1

Open a private file associated with this Context's application package for writing. Creates the file if it doesn't already exist.

Parameters

name	The name of the file to open; can not contain path separators.
mode	Operating mode. Use 0 or `MODE_PRIVATE` for the default operation, `MODE_APPEND` to append to an existing file, `MODE_WORLD_READABLE` and `MODE_WORLD_WRITEABLE` to control permissions.

Android安全模型

- 突破沙箱： Binder IPC

```
public class PackageManagerService extends IPackageManager.Stub {
    ......

    public void installPackageWithVerificationAndEncryption(Uri packageURI,
            IPackageInstallObserver observer, int flags, String installerPackageN
ame,
            VerificationParams verificationParams, ContainerEncryptionParams encr
yptionParams) {
        mContext.enforceCallingOrSelfPermission(android.Manifest.permission.INSTA
LL_PACKAGES,
                null);

        final int uid = Binder.getCallingUid();
        ......
        final int filteredFlags;

        if (uid == Process.SHELL_UID || uid == 0) {
            ......
            filteredFlags = flags | PackageManager.INSTALL_FROM_ADB;
        } else {
            filteredFlags = flags & ~PackageManager.INSTALL_FROM_ADB;
        }

        ......
```

Android安全模型

- 突破沙箱： Content Provider

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.android.providers.contacts"
        android:sharedUserId="android.uid.shared"
        android:sharedUserLabel="@string/sharedUserLabel">

    <application android:process="android.process.acore"
        android:label="@string/app_label"
        android:icon="@drawable/app_icon"
        android:allowBackup="false">

        <provider android:name="ContactsProvider2"
            android:authorities="contacts;com.android.contacts"
            android:label="@string/provider_label"
            android:multiprocess="false"
            android:exported="true"
            android:readPermission="android.permission.READ_CONTACTS"
            android:writePermission="android.permission.WRITE_CONTACTS">
```

Android安全模型

- 突破沙箱：黑客技术
  - SO注入
  - C/C++函数拦截
  - DEX注入
  - Java函数拦截
  - ……

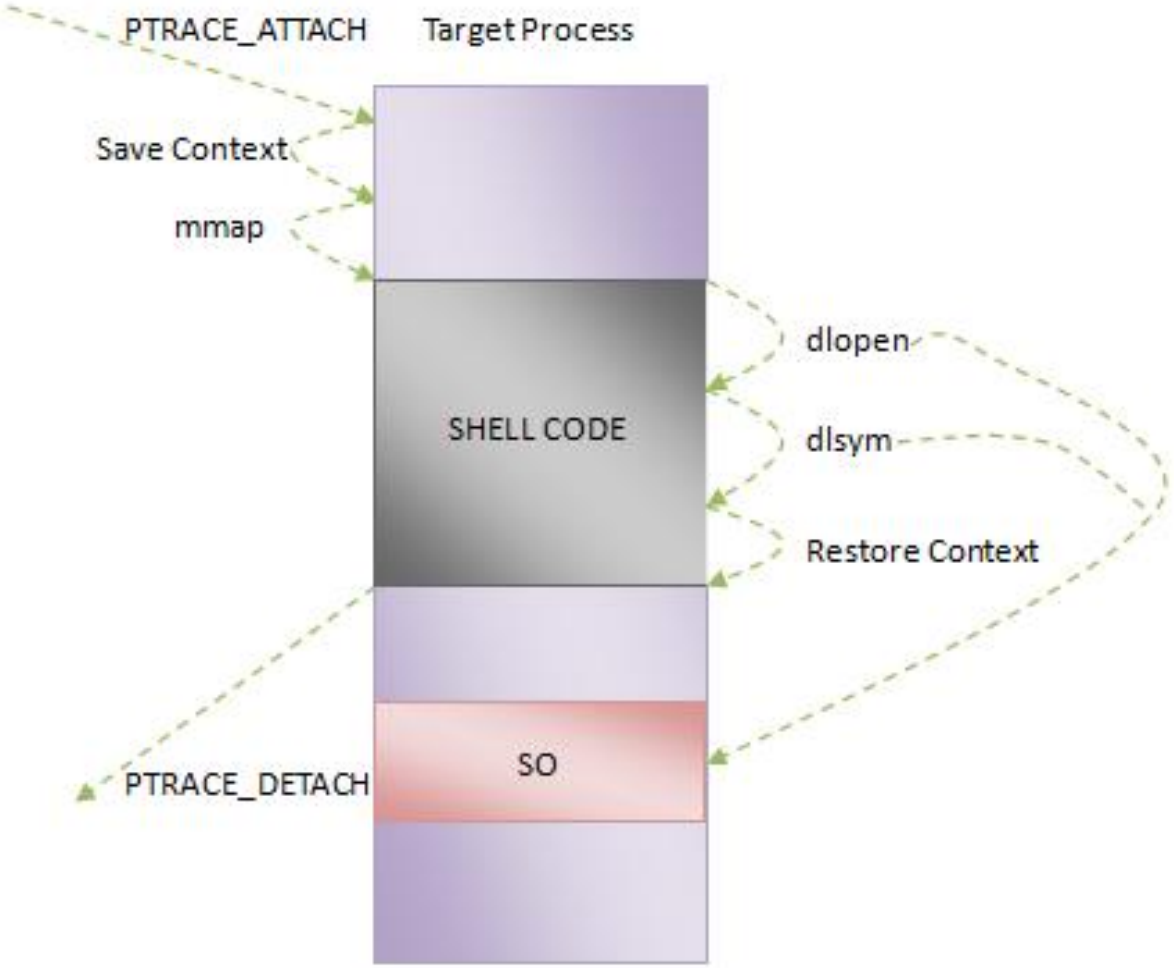
SO注入技术

- ptrace

long ptrace(enum __ptrace_request *request*, **pid_t** *pid*, **void ***addr*, **void ***data*);**

  - http://man7.org/linux/man-pages/man2/ptrace.2.html

SO注入技术


PTRACE_ATTACH
Target Process
Save Context
mmap
SHELL CODE
dlopen
dlsym
Restore Context
SO
PTRACE_DETACH

SO注入技术

- Step 1: PTRACE_ATTACH到目标进程，并且让目标进程发生PTRACE_SYSCALL时停止
- Step 2: PTRACE_GETREGS保存目标进程的上下文
- Step 3: PTRACE_SETREGS改写目标进程的PC寄存器，使得它指向函数mmap的地址
- Step 4: PTRACE_CONT让目标进程恢复执行，这时候将会执行函数mmap
- Step 5: PTRACE_GETREGS获得目标进程的R0寄存器值，即为函数mmap的返回值，指向在目标进程地址空间分配的一块内存

SO注入技术

- Step 6: PTRACE_POKETEXT往在目标进程分配的地址写入以下一段SHELL CODE

```
inject_start_s:
    @ debug loop
3:
    @sub r1, r1, #0
    @B 3b

    @ dlopen
    ldr r1, _dlopen_param2_s          void* handle  = dlopen(_dlopen_param1_s,
    ldr r0, _dlopen_param1_s          _dlopen_param2_s);
    ldr r3, _dlopen_addr_s
    blx r3
    subs r4, r0, #0
    beq 2f

    @dlsym
    ldr r1, _dlsym_param2_s           void* func  = dlsym(handle, _dlsym_param2_s);
    ldr r3, _dlsym_addr_s
    blx r3
    subs r3, r0, #0
    beq 1f

    @call our function
    ldr r0, _inject_function_param_s  int ret = func(_inject_function_param_s);
    blx r3
    subs r0, r0, #0
    beq 2f

1:
    @dlclose
    mov r0, r4
    ldr r3, _dlclose_addr_s           dlclose(handle);
    blx r3

2:
    @restore context
    ldr r1, _saved_cpsr_s             restore context;
    msr cpsr_cf, r1
    ldr sp, _saved_r0_pc_s
    ldmfd sp, {r0-pc}
                                                          20,1          36%
```

SO注入技术

- Step 7: PTRACE_SETREGS改写目标进程的PC寄存器，使得它指向上述SHELL CODE的起始地址_inject_start_s
- Step 8: PTRACE_DETTACH目标进程，目标进程恢复执行后，就会执行注入的
- Step 9: 注入的SHELL CODE在目标进程加载一个SO，并且找到这个SO的指定入口函数，进行调用

SO加壳技术

- 系统中的SO文件由一个叫Linker的加载器负责加载，即调用dlopen函数进行加载：

  void *dlopen(const char *_filename_, int _flag_);

  - 如果能将第一个参数改为一个内存地址，就可以实现从内存加载SO的功能，进而可以对该内存进行加密处理

SO加壳技术

- dlopen

```
void *dlopen(const char *filename, int flag)
{
    soinfo *ret;

    pthread_mutex_lock(&dl_lock);
    ret = find_library(filename);
    if (unlikely(ret == NULL)) {
        set_dlerror(DL_ERR_CANNOT_LOAD_LIBRARY);
    } else {
        soinfo_call_constructors(ret);
        ret->refcount++;
    }
    pthread_mutex_unlock(&dl_lock);
    return ret;
}
```

SO加壳技术

- find_library

```
soinfo *find_library(const char *name)
{
    soinfo *si;
    ......

    si = find_loaded_library(name);
    if (si != NULL) {
        ......
        if(si->flags & FLAG_LINKED) return si;
        DL_ERR("OOPS: recursive link to \"%s\"", si->name);
        return NULL;
    }

    si = load_library(name);
    ......

    return init_library(si);
}
```

SO加壳技术

- load_library

```
static soinfo* load_library(const char* name)
{
    // Open the file.
    scoped_fd fd;
    fd.fd = open_library(name);
    ......
    // Read the ELF header.
    Elf32_Ehdr header[1];
    int ret = TEMP_FAILURE_RETRY(read(fd.fd, (void*)header, sizeof(header)));
    ......
    // Read the program header table.
    const Elf32_Phdr* phdr_table;
    phdr_ptr phdr_holder;
    ret = phdr_table_load(fd.fd, header->e_phoff, header->e_phnum,
                          &phdr_holder.phdr_mmap, &phdr_holder.phdr_size, &phdr_table);
    ......
    // Reserve address space for all loadable segments.
    void* load_start = NULL;
    Elf32_Addr load_size = 0;
    Elf32_Addr load_bias = 0;
    ret = phdr_table_reserve_memory(phdr_table,
                                    phdr_count,
                                    &load_start,
                                    &load_size,
                                    &load_bias);
    ......
    /* Map all the segments in our address space with default protections */
    ret = phdr_table_load_segments(phdr_table,
                                   phdr_count,
                                   load_bias,
                                   fd.fd);
    ......
    soinfo_ptr si(name);
    si.ptr->base = (Elf32_Addr) load_start;
    si.ptr->size = load_size;
    si.ptr->load_bias = load_bias;
    ......
    return si.release();
}
-- INSERT --                                                    739,1         37%
```

SO加壳技术

- Read Elf32_Ehdr

```
Elf32_Ehdr header[1];
read(fd.fd,  (void*)header, sizeof(header))
```

SO加壳技术

- Read Elf32_Phdr

```
int phdr_table_load(int                 fd,
                    Elf32_Addr          phdr_offset,
                    Elf32_Half          phdr_num,
                    void**              phdr_mmap,
                    Elf32_Addr*         phdr_size,
                    const Elf32_Phdr**  phdr_table)
{
    Elf32_Addr  page_min, page_max, page_offset;
    void*       mmap_result;
    ......

    page_min = PAGE_START(phdr_offset);
    page_max = PAGE_END(phdr_offset + phdr_num*sizeof(Elf32_Phdr));
    page_offset = PAGE_OFFSET(phdr_offset);

    mmap_result = mmap(NULL,
                       page_max - page_min,
                       PROT_READ,
                       MAP_PRIVATE,
                       fd,
                       page_min);
    ......

    *phdr_mmap = mmap_result;
    *phdr_size = page_max - page_min;
    *phdr_table = (Elf32_Phdr*)((char*)mmap_result + page_offset);

    return 0;
}
-- INSERT --                                              131,1         21%
```

SO加壳技术

- Reserve Enough Memory

```
int
phdr_table_reserve_memory(const Elf32_Phdr* phdr_table,
                          size_t phdr_count,
                          void** load_start,
                          Elf32_Addr* load_size,
                          Elf32_Addr* load_bias)
{
    Elf32_Addr size = phdr_table_get_load_size(phdr_table, phdr_count);
    if (size == 0) {
        errno = EINVAL;
        return -1;
    }

    int mmap_flags = MAP_PRIVATE | MAP_ANONYMOUS;
    void* start = mmap(NULL, size, PROT_NONE, mmap_flags, -1, 0);
    if (start == MAP_FAILED) {
        return -1;
    }

    *load_start = start;
    *load_size  = size;
    *load_bias  = 0;

    for (size_t i = 0; i < phdr_count; ++i) {
        const Elf32_Phdr* phdr = &phdr_table[i];
        if (phdr->p_type == PT_LOAD) {
            *load_bias = (Elf32_Addr)start - PAGE_START(phdr->p_vaddr);
            break;
        }
    }
    return 0;
}
```

SO加壳技术

- Load Segments

```
int
phdr_table_load_segments(const Elf32_Phdr* phdr_table,
                         int               phdr_count,
                         Elf32_Addr        load_bias,
                         int               fd)
{
    int nn;

    for (nn = 0; nn < phdr_count; nn++) {
        const Elf32_Phdr* phdr = &phdr_table[nn];
        void* seg_addr;

        /* Segment addresses in memory */
        Elf32_Addr seg_start = phdr->p_vaddr + load_bias;
        Elf32_Addr seg_end   = seg_start + phdr->p_memsz;

        Elf32_Addr seg_page_start = PAGE_START(seg_start);
        Elf32_Addr seg_page_end   = PAGE_END(seg_end);

        Elf32_Addr seg_file_end   = seg_start + phdr->p_filesz;

        /* File offsets */
        Elf32_Addr file_start = phdr->p_offset;
        Elf32_Addr file_end   = file_start + phdr->p_filesz;

        Elf32_Addr file_page_start = PAGE_START(file_start);
        Elf32_Addr file_page_end   = PAGE_END(file_end);

        seg_addr = mmap((void*)seg_page_start,
                        file_end - file_page_start,
                        PFLAGS_TO_PROT(phdr->p_flags),
                        MAP_FIXED|MAP_PRIVATE,
                        fd,
                        file_page_start);

        ......
    }
    return 0;
}
```

SO加壳技术

- What data we need?
  - Elf32_Ehdr
  - Elf32_Phdr
  - Segments
- How to fill above data?

```
mmap_result = mmap(NULL,
                   page_max - page_min,
                   PROT_READ | PROT_WRITE,
                   MAP_PRIVATE | MAP_ANONYMOUS,
                   -1,
                   0);

memcpy((char*)mmap_result + page_offset, addr, size);
```

SO加壳技术

- struct elfinfo

```
struct elfinfo {
    char name[SOINFO_NAME_LEN];
    void* elf32_ehdr;
    size_t elf32_ehdr_size;
    void* elf32_phdr;
    size_t elf32_phdr_size;
    seginfo* segs;
    size_t seg_count;
};
```

SO加壳技术

- Open file and create elfinfo

```
int fd = open(path, O_RDONLY);

if(fd == -1) {
    return NULL;
}

elfinfo* ei = (elfinfo*)malloc(sizeof(elfinfo));
memset(ei, 0, sizeof(elfinfo));

//name
strlcpy((char*)ei->name, path, strlen(path));
```

SO加壳技术

- Read Elf32_Ehdr

```
//elf32_ehdr
Elf32_Ehdr* ehdr = (Elf32_Ehdr*)malloc(sizeof(Elf32_Ehdr));

read(fd, (void*)ehdr, sizeof(Elf32_Ehdr));

ei->elf32_ehdr = ehdr;
ei->elf32_ehdr_size = sizeof(Elf32_Ehdr);
```

SO加壳技术

- Read Elf32_Phdr

```
//elf32_phdr
Elf32_Phdr* phdr = (Elf32_Phdr*)malloc(sizeof(Elf32_Phdr) * ehdr->e_phnum);


lseek(fd, ehdr->e_phoff, SEEK_SET);
read(fd, (void*)phdr, sizeof(Elf32_Phdr) * ehdr->e_phnum);

ei->elf32_phdr = phdr;
ei->elf32_phdr_size = sizeof(Elf32_Phdr) * ehdr->e_phnum;
```

SO加壳技术

- Read segments

```
//segments
seginfo* si = (seginfo*)malloc(sizeof(seginfo) * ehdr->e_phnum);

ei->segs = si;
ei->seg_count = ehdr->e_phnum;

for(size_t i = 0; i < ehdr->e_phnum; ++i) {
    Elf32_Phdr* iphdr = &phdr[i];
    seginfo* isi = &si[i];

    isi->addr = malloc(iphdr->p_filesz);
    isi->size = iphdr->p_filesz;
    isi->file_offset = iphdr->p_offset;

    lseek(fd, iphdr->p_offset, SEEK_SET);
    read(fd, (void*)isi->addr, iphdr->p_filesz);
}
```

C/C++函数拦截技术

- Got Hook
- VTable Hook
- Inline Hook

Got Hook

```
void DisplayDevice::swapBuffers(HWComposer& hwc) const {
    EGLBoolean success = EGL_TRUE;
    if (hwc.initCheck() != NO_ERROR) {
        // no HWC, we call eglSwapBuffers()
        success = eglSwapBuffers(mDisplay, mSurface);
    } else {
        // We have a valid HWC, but not all displays can use it, in particular
        // the virtual displays are on their own.
        // TODO: HWC 1.2 will allow virtual displays
```

```
PUSH            {R0-R2,R4,R5,LR}
MOV             R4, R0
MOV             R0, R1
MOV             R5, R1
BL              _ZNK7android10HWComposer9initCheckEv ; android::HWComposer::initCheck(void)
CBNZ            R0, loc_1D486
LDR             R3, [R4,#0x44]
CMP             R3, #1
```

```
.plt:0001B9D0 eglSwapBuffers                            ; CODE XR
.plt:0001B9D0                   ADRL            R12, 0x339D8
.plt:0001B9D8                   LDR             PC, [R12,#(eglSwa
```

```
loc_1D486                               ; CODE XREF: andrc
                                        ; android::Displa
                LDR             R0, [R4,#0x5C]
                LDR             R1, [R4,#0x60]
                BLX             eglSwapBuffers
                CBNZ            R0, locret_1D4B6
                BLX             eglGetError
```

```
got:000339D0
got:000339D0
got:000339D4 _GLOBAL_OFFSET_TABLE_ DCD 0
got:000339D4
got:000339D8                    DCD 0
got:000339DC                    DCD 0
```

Got Hook

- Step 1: Find the address of eglSwapBuffers

```
void * handle = dlopen(“/system/lib/libEGL.so”, RTLD_NOW)
void* addr = dlsym(handle, “eglSwapBuffers”);
```

Got Hook

- Step 2: Find the .got section

```
uint32_t got_plt_section_addr = 0;
uint32_t got_plt_section_size = 0;

for(int i = 0; i < sh_num; ++i) {
    read(fd, &shdr, sh_ent_size);
    if(shdr.sh_type == SHT_PROGBITS) {
        int name_idx = shdr.sh_name;
        if(strcmp(&(string_table[name_idx]), ".got") == 0) {
            got_plt_section_addr = (uint32_t)base_addr + shdr.sh_addr;
            got_plt_section_size = shdr.sh_size;
            break;
        }
    }
}
```

Got Hook

- Step 3: Find the address of eglSwapBuffers in .got section

```
void* got_addr = NULL;

for(uint32_t j = 0; j < got_plt_section_size; j += 4) {
    uint32_t got_item = *(uint32_t*)(got_plt_section_addr + j);
    if(got_item == (uint32_t)func_addr) {
        got_addr = (void*)(got_plt_section_addr + j);
        break;
    }
}
```

Got Hook

- Step 4: Replace it

```
static void* hook_addr(void* addr, void* val)
{
    uint32_t old_val = *(uint32_t*)addr;

    mprotect((void*)PAGE_START((uint32_t)addr), PAGE_SIZE, PROT_READ | PROT_WRIT
E);

    *(uint32_t*)addr = (uint32_t)val;

    mprotect((void*)PAGE_START((uint32_t)addr), PAGE_SIZE, PROT_READ);

    return (void*)old_val;
}
```

VTable Hook

```
static void nativeUnlockCanvasAndPost(JNIEnv* env, j
anvasObj) {
    ......

    // unlock surface
    status_t err = surface->unlockAndPost();
    if (err < 0) {
        doThrowIAE(env);
    }
}
```

```
478         status_t err = surface->unlockAndPost();
2: x/10i 0x40167294
   0x40167294 <android::nativeUnlockCanvasAndPost(JNI
:       ldr     r2, [r3, #100]  ; 0x64
   0x40167296 <android::nativeUnlockCanvasAndPost(JNI
:       blx     r2
```

```
.data.rel.ro:000344F0 ; `vtable for'android::Surface
.data.rel.ro:000344F0 _ZTVN7android7SurfaceE DCD 0, 0, _ZN7android7SurfaceD2Ev+1, _ZN7
.data.rel.ro:000344F0                                         ; DATA XREF: android::Su
.data.rel.ro:000344F0                                         ; .text:off_27704↑o ...
.data.rel.ro:000344F0                 DCD __imp__ZN7android7RefBase10onFirstRefEv, __i
.data.rel.ro:000344F0                 DCD __imp__ZN7android7RefBase20onIncStrongAttemp
.data.rel.ro:000344F0                 DCD __imp__ZN7android7RefBase13onLastWeakRefEPKv
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient12cancelBuf
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient11queueBuff
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient7performEiS
.data.rel.ro:000344F0                 DCD _ZNK7android7Surface5queryEiPi+1, _ZN7androi
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient21lockBuffe
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient7connectEi+
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient14setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient20setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient24setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient16setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient14setScalin
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient19setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient19setBuffer
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient7setCropEPK
.data.rel.ro:000344F0                 DCD _ZN7android20SurfaceTextureClient8setUsageEj
.data.rel.ro:000344F0                 DCD _ZN7android7Surface13unlockAndPostEv+1
```

VTable Hook

- Step 1: Find the address of Surface::unlockAndPost

```
void * handle = dlopen("/system/lib/libgui.so", RTLD_NOW)
void* addr = dlsym(handle,
                   "_ZN7android7Surface13unlockAndPostEv");
```

VTable Hook

- Step 2: Find the .data.rel.ro section

```
uint32_t data_rel_ro_section_addr = 0;
uint32_t data_rel_ro_section_size = 0;

for(int i = 0; i < sh_num; ++i) {
    read(fd, &shdr, sh_ent_size);
    if(shdr.sh_type == SHT_PROGBITS) {
        int name_idx = shdr.sh_name;
        if(strcmp(&(string_table[name_idx]), ".data.rel.ro") == 0) {
            data_rel_ro_section_addr = (uint32_t)base_addr + shdr.sh_addr;
            data_rel_ro_section_size = shdr.sh_size;
            break;
        }
    }
}
```

VTable Hook

- Step 3: Find the address of Surface::unlockAndPost in .data.rel.ro section

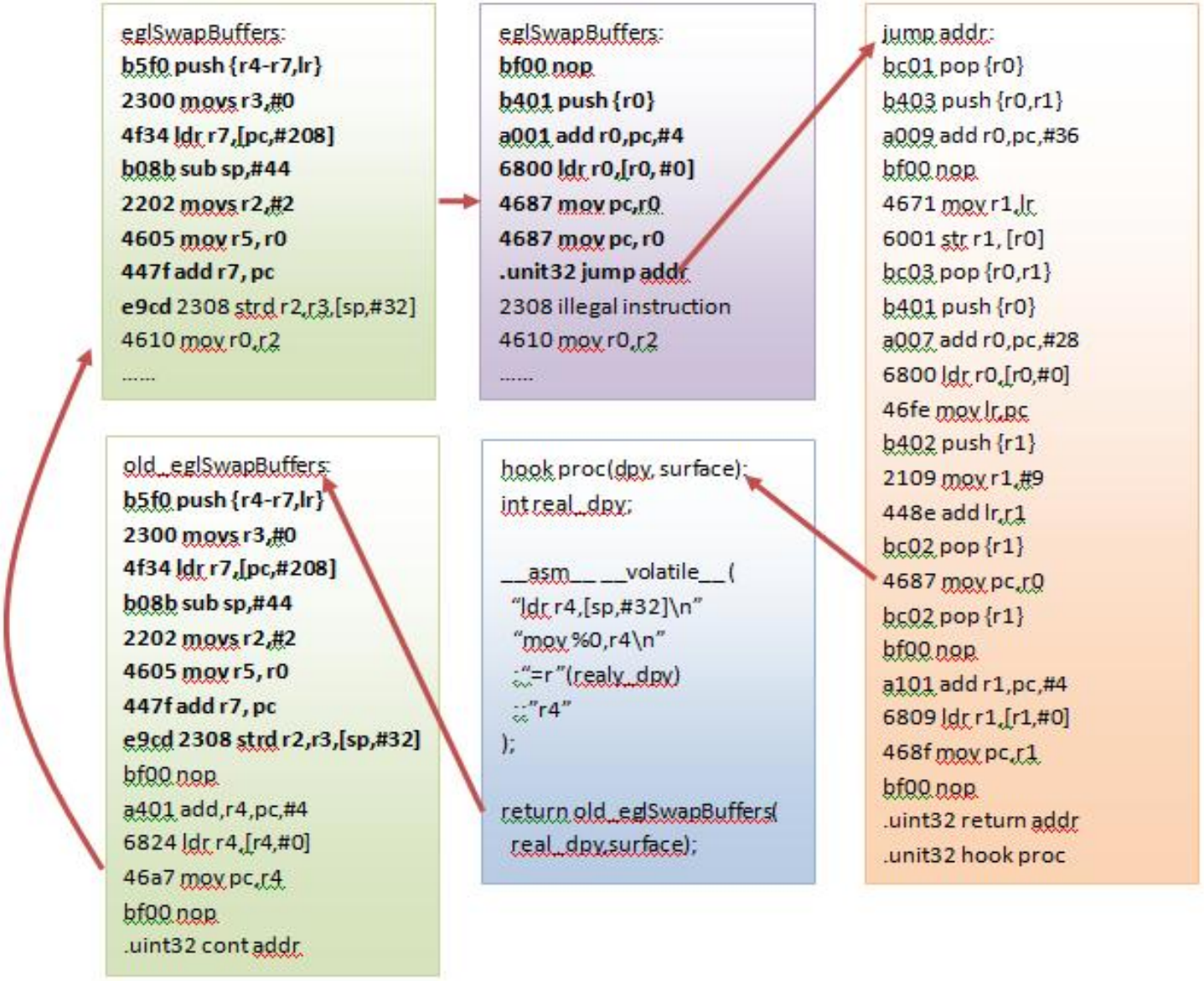
```
void* data_rel_ro_addr = NULL;

for(uint32_t j = 0; j < data_rel_ro_section_size; j += 4) {
    uint32_t data_rel_ro_item = *(uint32_t*)(data_rel_ro_section_addr + j);
    if(data_rel_ro_item == (uint32_t)func_addr) {
        data_rel_ro_addr = (void*)(data_rel_ro_section_addr + j);
        break;
    }
}
```

Got Hook

- Step 4: Replace it

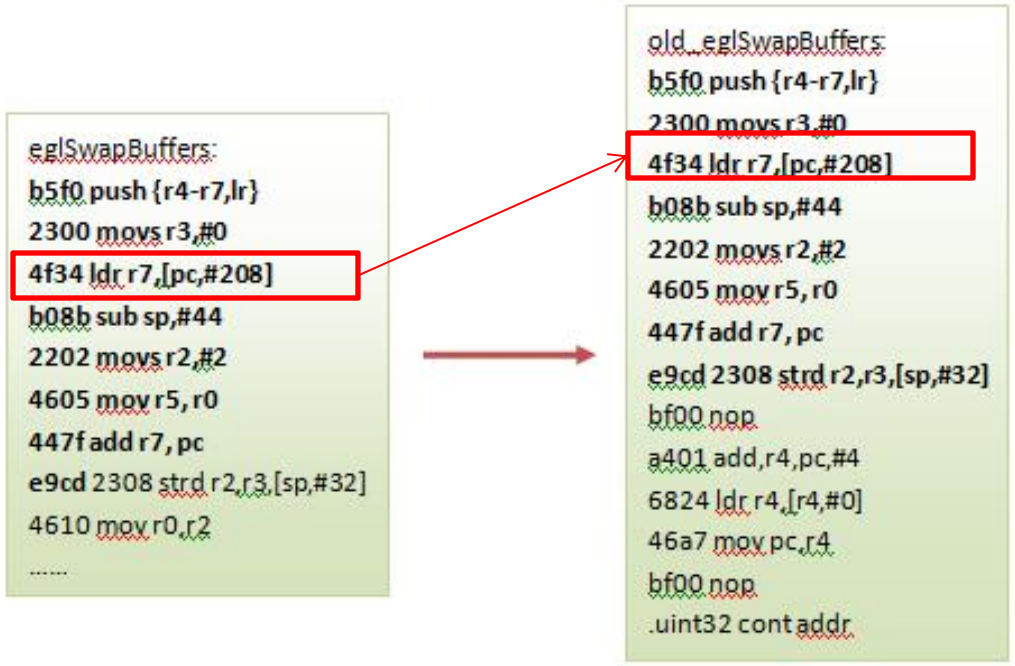
```
static void* hook_addr(void* addr, void* val)
{
    uint32_t old_val = *(uint32_t*)addr;

    mprotect((void*)PAGE_START((uint32_t)addr), PAGE_SIZE, PROT_READ | PROT_WRIT
E);

    *(uint32_t*)addr = (uint32_t)val;

    mprotect((void*)PAGE_START((uint32_t)addr), PAGE_SIZE, PROT_READ);

    return (void*)old_val;
}
```

Inline Hook


eglSwapBuffers:
b5f0 push {r4-r7,lr}
2300 movs r3,#0
4f34 ldr r7,[pc,#208]
b08b sub sp,#44
2202 movs r2,#2
4605 mov r5, r0
447f add r7, pc
e9cd 2308 strd r2,r3,[sp,#32]
4610 mov r0,r2
.....
eglSwapBuffers:
bf00 nop
b401 push {r0}
a001 add r0,pc,#4
6800 ldr r0,[r0, #0]
4687 mov pc,r0
4687 mov pc, r0
.unit32 jump addr
2308 illegal instruction
4610 mov r0,r2
.....
jump addr:
bc01 pop {r0}
b403 push {r0,r1}
a009 add r0,pc,#36
bf00 nop
4671 mov r1,lr
6001 str r1, [r0]
bc03 pop {r0,r1}
b401 push {r0}
a007 add r0,pc,#28
6800 ldr r0,[r0,#0]
46fe mov lr,pc
b402 push {r1}
2109 mov r1,#9
448e add lr,r1
bc02 pop {r1}
4687 mov pc,r0
bc02 pop {r1}
bf00 nop
a101 add r1,pc,#4
6809 ldr r1,[r1,#0]
468f mov pc,r1
bf00 nop
.uint32 return addr
.unit32 hook proc
old_eglSwapBuffers:
b5f0 push {r4-r7,lr}
2300 movs r3,#0
4f34 ldr r7,[pc,#208]
b08b sub sp,#44
2202 movs r2,#2
4605 mov r5, r0
447f add r7, pc
e9cd 2308 strd r2,r3,[sp,#32]
bf00 nop
a401 add,r4,pc,#4
6824 ldr r4,[r4,#0]
46a7 mov pc,r4
bf00 nop
.uint32 cont addr
hook proc(dpy, surface):
int real_dpy;
__asm__ __volatile__(
"ldr r4,[sp,#32]\n"
"mov %0,r4\n"
:"=r"(realy_dpy)
:"r4"
);
return old_eglSwapBuffers(
real_dpy,surface);

Inline Hook

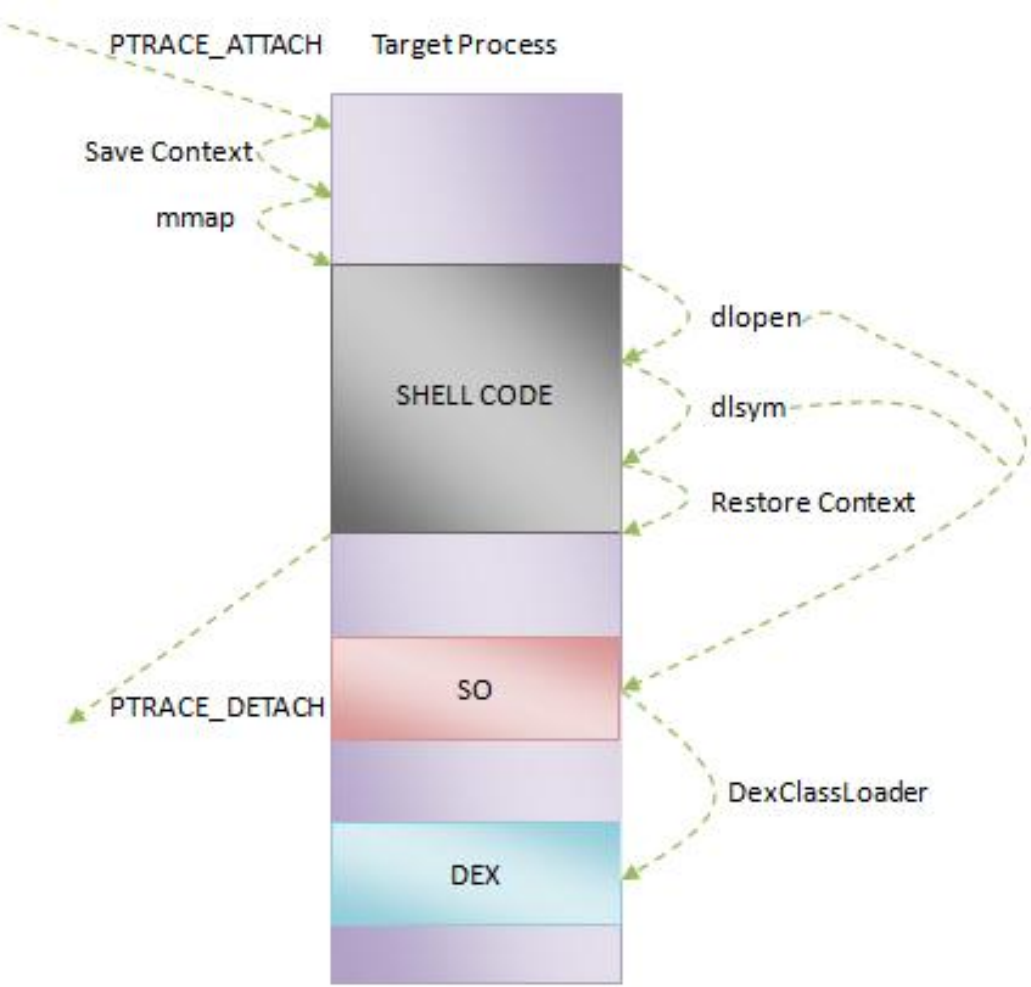
- Problem



Inline Hook

- 移动的指令可能包含：
  - 普通指令
  - PC相对寻址指令
  - 跳转指令
- 对于PC相对寻址和跳转指令：
  - 需要进行重定位
- 因此，实现Inline Hook要求：
  - 动态的指令解析
  - 动态的指令重定位

DEX注入技术


PTRACE_ATTACH
Target Process
Save Context
mmap
SHELL CODE
dlopen
dlsym
Restore Context
PTRACE_DETACH
SO
DexClassLoader
DEX

DEX注入技术

```
static jclass load_class_from_library(JNIEnv* env, const char* lib_path, const char* opt_path, const char* class_name)
{
    // Get system class loader object
    jclass class_loader_class = env->FindClass("java/lang/ClassLoader");
    jmethodID get_system_class_loader_method_id = env->GetStaticMethodID(class_loader_class,
                    "getSystemClassLoader", "()Ljava/lang/ClassLoader;");

    jobject system_class_loader_object = env->CallStaticObjectMethod(class_loader_class,
                    get_system_class_loader_method_id);

    // Create DexClassLoader
    jclass dex_class_loader_class = env->FindClass("dalvik/system/DexClassLoader");
    jmethodID dex_class_loader_constructor_id = env->GetMethodID(dex_class_loader_class,
                    "<init>", "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;Ljava/lang/ClassLoader;)V");
    jstring lib_path_string = env->NewStringUTF(lib_path);
    jstring opt_path_string = env->NewStringUTF(opt_path);
    jobject dex_class_loader_object = env->NewObject(dex_class_loader_class, dex_class_loader_constructor_id,
                    lib_path_string, opt_path_string, NULL, system_class_loader_object);

    // Use DexClassLoader to laod target class
    jmethodID load_class_method_id = env->GetMethodID(dex_class_loader_class, "loadClass",
                    "(Ljava/lang/String;)Ljava/lang/Class;");
    jstring class_name_string = env->NewStringUTF(class_name);
    jclass target_class = (jclass)env->CallObjectMethod(dex_class_loader_object,
                    load_class_method_id, class_name_string);

    // Release local ref
    env->DeleteLocalRef(class_loader_class);
    env->DeleteLocalRef(system_class_loader_object);
    env->DeleteLocalRef(dex_class_loader_class);
    env->DeleteLocalRef(lib_path_string);
    env->DeleteLocalRef(opt_path_string);
    env->DeleteLocalRef(dex_class_loader_object);
    env->DeleteLocalRef(class_name_string);

    return target_class;
}
```

DEX加壳技术

- 通过DexClassLoader可以动态地加载DEX文件，但是它在加载DEX文件的过程会生成一个ODEX文件，给别人提供了静态逆向的可能
- 通过分析DexClassLoader的实现可以知道，它是通过DexFile来实现动态加载DEX文件的

DEX加壳技术

- 进一步分析DexFile的实现，发现它提供了两个隐藏接口来实现加载内存DEX文件

```
public final class DexFile {
    ......

    private native static Class defineClass(String name, ClassLoader loader, int cookie);


    /*
     * Open a DEX file based on a {@code byte[]}. The value returned
     * is a magic VM cookie. On failure, a RuntimeException is thrown.
     */
    native private static int openDexFile(byte[] fileContents);

    ......
}
```

DEX加壳技术

```
static jclass load_class_from_memory(JNIEnv* env, void* addr, int len, const char* class_name)
{
    jclass dex_file_class = env->FindClass("dalvik/system/DexFile");
    if(dex_file_class == NULL) {
        INFO("Failed to find dalvik.system.DexFile");
        return NULL;
    }

    jmethodID open_dex_file_method_id = env->GetStaticMethodID(dex_file_class, "openDexFile", "([B)I");
    if(open_dex_file_method_id == NULL) {
        INFO("Failed to get openDexFile(byte[]) member of dalvik.system.DexFile");
        return NULL;
    }

    jbyteArray dex_file_content = env->NewByteArray(len);
    env->SetByteArrayRegion(dex_file_content, 0, len, (jbyte*)addr);

    jint cookie = env->CallStaticIntMethod(dex_file_class, open_dex_file_method_id, dex_file_content);
    if(cookie == 0) {
        dvmLogExceptionStackTrace();
        env->ExceptionClear();
        return NULL;
    }

    jmethodID define_class_method_id = env->GetStaticMethodID(dex_file_class, "defineClass",
                "(Ljava/lang/String;Ljava/lang/ClassLoader;I)Ljava/lang/Class;");
    if(define_class_method_id == NULL) {
        INFO("Failed to get defineClass member of dalvik.system.DexFile");
        return NULL;
    }

    // Get system class loader object
    jclass class_loader_class = env->FindClass("java/lang/ClassLoader");
    jmethodID get_system_class_loader_method_id = env->GetStaticMethodID(class_loader_class,
                    "getSystemClassLoader", "()Ljava/lang/ClassLoader;");

    jobject system_class_loader_object = env->CallStaticObjectMethod(class_loader_class,
                    get_system_class_loader_method_id);

    jstring target_class_name_string = env->NewStringUTF(class_name);

    jclass target_class = (jclass)env->CallStaticObjectMethod(dex_file_class, define_class_method_id,
            target_class_name_string, system_class_loader_object, cookie);

    return target_class;
}
```

DEX加壳技术

- 但是，DexFile从Android 4.0开始才支持加载内存DEX文件，如何支持Android 4.0以下的版本呢？
- 通过分析DexFile加载内存DEX文件的实现可以发现，里面用到的关键函数都可以从libdex.a和libdvm.so获得
- 于是，可以模仿Android 4.0，实现DexFile加载内存DEX文件的功能

DEX加壳技术

- 辅助数据结构和函数

```
/*
 * Internal struct for managing DexFile.
 */
struct DexOrJar {
    char*       fileName;
    bool        isDex;
    bool        okayToFree;
    RawDexFile* pRawDexFile;
    JarFile*    pJarFile;
    u1*         pDexMemory; // malloc()ed memory, if any
};

/*
 * (This is a dvmHashTableLookup compare func.)
 *
 * Args are DexOrJar*.
 */
static int hash_cmp_dex_or_jar(const void* tableVal, const void* newVal)
{
    return (int) newVal - (int) tableVal;
}
```

DEX加壳技术

- Step 1: custome_load_class_from_memory

```
static jclass custom_load_class_from_memory(JNIEnv* env, void* addr, int len, const char* class_name)
{
    INFO("custom load class from memory");

    jint cookie = open_dex_file_from_memory(addr, len);

    jclass dex_file_class = env->FindClass("dalvik/system/DexFile");
    if(dex_file_class == NULL) {
        INFO("Failed to find dalvik.system.DexFile");
        return NULL;
    }

    jmethodID define_class_method_id = env->GetStaticMethodID(dex_file_class, "defineClass",
                  "(Ljava/lang/String;Ljava/lang/ClassLoader;I)Ljava/lang/Class;");
    if(define_class_method_id == NULL) {
        INFO("Failed to get defineClass member of dalvik.system.DexFile");
        return NULL;
    }

    // Get system class loader object
    jclass class_loader_class = env->FindClass("java/lang/ClassLoader");
    jmethodID get_system_class_loader_method_id = env->GetStaticMethodID(class_loader_class,
                    "getSystemClassLoader", "()Ljava/lang/ClassLoader;");

    jobject system_class_loader_object = env->CallStaticObjectMethod(class_loader_class,
                    get_system_class_loader_method_id);

    jstring target_class_name_string = env->NewStringUTF(class_name);

    jclass target_class = (jclass)env->CallStaticObjectMethod(dex_file_class, define_class_method_id,
            target_class_name_string, system_class_loader_object, cookie);

    return target_class;
}
```

DEX加壳技术

- Step 2: open_dex_from_memory

```
static int open_dex_file_from_memory(void* addr, int len)
{
    INFO("open dex file from memory");

    u1* pBytes;
    RawDexFile* pRawDexFile;
    DexOrJar* pDexOrJar = NULL;

    pBytes = (u1*) malloc(len);
    if(pBytes == NULL) {
        INFO("Unable to allocate DEX memory");
        return 0;
    }

    memcpy(pBytes, addr, len);

    if(raw_dex_file_open_array(pBytes, len, &pRawDexFile) != 0) {
        INFO("Unable to open in-memory DEX file");
        free(pBytes);
        return 0;
    }

    pDexOrJar = (DexOrJar*) malloc(sizeof(DexOrJar));
    pDexOrJar->isDex = true;
    pDexOrJar->pRawDexFile = pRawDexFile;
    pDexOrJar->pDexMemory = pBytes;
    pDexOrJar->fileName = strdup("<memory>"); // Needs to be free()able.

    add_to_dex_file_table(pDexOrJar);

    return (int)pDexOrJar;
}
```

DEX加壳技术

- Step 3: raw_dex_file_open_array

```
int raw_dex_file_open_array(u1* pBytes, u4 length, RawDexFile** ppRawDexFile)
{
    INFO("raw dex file open array");

    DvmDex* pDvmDex = NULL;

    if(!prepare_dex_in_memory(pBytes, length, &pDvmDex)) {
        INFO("Unable to open raw DEX from array");
        return -1;
    }

    *ppRawDexFile = (RawDexFile*) calloc(1, sizeof(RawDexFile));
    (*ppRawDexFile)->pDvmDex = pDvmDex;

    return 0;
}
```

DEX加壳技术

- Step 4: prepare_dex_in_memory

```
bool prepare_dex_in_memory(u1* addr, size_t len, DvmDex** ppDvmDex)
{
    INFO("prepare dex in memory");

    DexClassLookup* pClassLookup = NULL;

    if(!rewrite_dex(addr, len, &pClassLookup, ppDvmDex)) {
        return false;
    }

    (*ppDvmDex)->pDexFile->pClassLookup = pClassLookup;

    return true;
}
```

DEX加壳技术

- Step 5: rewrite_dex

```
static bool rewrite_dex(u1* addr, int len, DexClassLookup** ppClassLookup, DvmDex** ppDvmDex)
{
    DexClassLookup* pClassLookup = NULL;
    DvmDex* pDvmDex = NULL;
    bool result = false;
    const char* msgStr = "???";

    /* if the DEX is in the wrong byte order, swap it now */
    if (dexSwapAndVerify(addr, len) != 0)
        goto bail;

    /*
     * Now that the DEX file can be read directly, create a DexFile struct
     * for it.
     */
    if (dex_file_open_partial(addr, len, &pDvmDex) != 0) {
        ALOGE("Unable to create DexFile");
        goto bail;
    }

    /*
     * Create the class lookup table.  This will eventually be appended
     * to the end of the .odex.
     *
     * We create a temporary link from the DexFile for the benefit of
     * class loading, below.
     */
    pClassLookup = dexCreateClassLookup(pDvmDex->pDexFile);
    if (pClassLookup == NULL)
        goto bail;
    pDvmDex->pDexFile->pClassLookup = pClassLookup;

    result = true;

bail:
    ......
    return result;
}
```

DEX加壳技术

- Step 6: dex_file_open_partial

```
int dex_file_open_partial(const void* addr, int len, DvmDex** ppDvmDex)
{
    DvmDex* pDvmDex;
    DexFile* pDexFile;
    int parseFlags = kDexParseDefault;
    int result = -1;

    /* -- file is incomplete, new checksum has not yet been calculated
    if (gDvm.verifyDexChecksum)
        parseFlags |= kDexParseVerifyChecksum;
    */

    pDexFile = dexFileParse((u1*)addr, len, parseFlags);
    if (pDexFile == NULL) {
        ALOGE("DEX parse failed");
        goto bail;
    }
    pDvmDex = allocate_aux_structures(pDexFile);
    if (pDvmDex == NULL) {
        dexFileFree(pDexFile);
        goto bail;
    }

    pDvmDex->isMappedReadOnly = false;
    *ppDvmDex = pDvmDex;
    result = 0;

bail:
    return result;
}
```

DEX加壳技术

- Step 7: allocate_aux_structures

```
static DvmDex* allocate_aux_structures(DexFile* pDexFile)
{
    DvmDex* pDvmDex;
    const DexHeader* pHeader;
    u4 stringSize, classSize, methodSize, fieldSize;

    pHeader = pDexFile->pHeader;

    stringSize = pHeader->stringIdsSize * sizeof(struct StringObject*);
    classSize  = pHeader->typeIdsSize * sizeof(struct ClassObject*);
    methodSize = pHeader->methodIdsSize * sizeof(struct Method*);
    fieldSize  = pHeader->fieldIdsSize * sizeof(struct Field*);

    u4 totalSize = sizeof(DvmDex) +
                   stringSize + classSize + methodSize + fieldSize;

    u1 *blob = (u1 *)dvmAllocRegion(totalSize,
                            PROT_READ | PROT_WRITE, "dalvik-aux-structure");
    if ((void *)blob == MAP_FAILED)
        return NULL;

    pDvmDex = (DvmDex*)blob;
    blob += sizeof(DvmDex);

    pDvmDex->pDexFile = pDexFile;

    pDvmDex->pHeader = pHeader;

    pDvmDex->pResStrings = (struct StringObject**)blob;
    blob += stringSize;
    pDvmDex->pResClasses = (struct ClassObject**)blob;
    blob += classSize;
    pDvmDex->pResMethods = (struct Method**)blob;
    blob += methodSize;
    pDvmDex->pResFields = (struct Field**)blob;

    pDvmDex->pInterfaceCache = dvmAllocAtomicCache(DEX_INTERFACE_CACHE_SIZE);

    return pDvmDex;
}
```

DEX加壳技术

- Step 8: add_to_dex_file_table

```
static void add_to_dex_file_table(DexOrJar* pDexOrJar) {
    /*
     * Later on, we will receive this pointer as an argument and need
     * to find it in the hash table without knowing if it's valid or
     * not, which means we can't compute a hash value from anything
     * inside DexOrJar. We don't share DexOrJar structs when the same
     * file is opened multiple times, so we can just use the low 32
     * bits of the pointer as the hash.
     */
    u4 hash = (u4) pDexOrJar;
    void* result;

    dvmHashTableLock(gDvm.userDexFiles);
    result = dvmHashTableLookup(gDvm.userDexFiles, hash, pDexOrJar,
            hash_cmp_dex_or_jar, true);
    dvmHashTableUnlock(gDvm.userDexFiles);

    if (result != pDexOrJar) {
        INFO("Pointer has already been added?");
        return;
    }

    pDexOrJar->okayToFree = true;
}
```

DEX加壳技术

- 上述过程用到的关键函数均可从libdex.a和libdvm.so获得：
  - dexSwapAndVeriry
  - dexCreateClassLookup
  - dexFileParse
  - dvmAllocRegion
  - dvmAllocAtomicCache
  - dvmHashTableLock
  - dvmHashTableLookup
  - dvmHashTableUnlock

Java函数拦截技术

- 在Dalvik虚拟机中，无论是Java函数，还是Native函数，都是通过Method结构体来描述的

```
struct Method {
    /* the class we are a part of */
    ClassObject*    clazz;

    /* access flags; low 16 bits are defined by spec (could be u2?) */
    u4              accessFlags;
    ......

    /* the actual code */
    const u2*       insns;          /* instructions, in memory-mapped .dex */
    ......

    /*
     * Native method ptr; could be actual function or a JNI bridge.  We
     * don't currently discriminate between DalvikBridgeFunc and
     * DalvikNativeFunc; the former takes an argument superset (i.e. two
     * extra args) which will be ignored.  If necessary we can use
     * insns==NULL to detect JNI bridge vs. internal native.
     */
    DalvikBridgeFunc nativeFunc;
    ......

};
```

Java函数拦截技术

- Dalvik虚拟机通过dvmIsNativeMethod判断一个函数是Java函数还是Native函数

```
INLINE bool dvmIsNativeMethod(const Method* method) {
    return (method->accessFlags & ACC_NATIVE) != 0;
}
```

Java函数拦截技术

- Dalvik虚拟调用一个函数之前，首先判断它是Java函数还是Native函数

```
GOTO_TARGET(invokeMethod, bool methodCallRange, const Method* _methodToCall,
    u2 count, u2 regs)
    {
        STUB_HACK(vsrc1 = count; vdst = regs; methodToCall = _methodToCall;);
        StackSaveArea* newSaveArea;
        u4* newFp;
        ......
        newFp = (u4*) SAVEAREA_FROM_FP(fp) - methodToCall->registersSize;
        newSaveArea = SAVEAREA_FROM_FP(newFp);
        ......
        newSaveArea->prevFrame = fp;
        newSaveArea->savedPc = pc;
        ......
        if (!dvmIsNativeMethod(methodToCall)) {
            curMethod = methodToCall;
            methodClassDex = curMethod->clazz->pDvmDex;
            pc = methodToCall->insns;
            fp = self->curFrame = newFp;
            ......
            FINISH(0);                                  // jump to method start
        } else {
            self->curFrame = newFp;
            ......
            (*methodToCall->nativeFunc)(newFp, &retval, methodToCall, self);
            ......
        }
        ......
    }
GOTO_TARGET_END
```

Java函数拦截技术

- Java函数拦截技术原理分析
  - 对于Java函数，Davik虚拟机使用解释器来执行
  - 对于Native函数， Davik虚拟机找到它的函数指针nativeFunc，进行直接调用
  - 如果我们能把一个Java函数修改为Native函数，并且将nativeFunc指针设置为自定义的函数，那么就可以实现拦截了
  - 拦截完成之后，根据情况决定是否需要调用原来的Java函数，即可完成整个拦截过程

Java函数拦截技术

- libdvm导出了两个函数dvmDecodeIndirectRef和dvmSlotToMethod，如果我们知道一个Java函数在它所属的Class里面的位置Slot，那么就可以通过它们获得该Java函数在Dalvik虚拟内部所对应的Method结构体：

```
// Find the internal representation of the method
ClassObject* declared_class = (ClassObject*)dvmDecodeIndirectRef(dvmThreadSelf(), declared_class_indirect);
Method* method = dvmSlotToMethod(declared_class, slot);
if(method == NULL) {
    dvmThrowNoSuchMethodError("Failed to get internal representation for method");
    return;
}
```

Java函数拦截技术

- 得到一个Java函数在Dalvik虚拟内部所对应的Method结构体之后，就可以将它设置为Native函数：

```
// Replace method with our own code
SET_METHOD_FLAG(method, ACC_NATIVE);
method->nativeFunc = &java_method_call_interceptor;
method->registersSize = method->insSize;
method->outsSize = 0;
```

Java函数拦截技术

- 如何获得一个Java函数所属的Class，以及它在该Class的位置Slot呢？
- 假设我们知道：
  - Java函数的名称—methodName
  - Java函数的原型—prototype
  - Java函数的类名称—className
  - 用来加载该Java类的ClassLoader--classLoader

Java函数拦截技术

- Step 1: 获得Class对象

```
Class<?> clazz =
classLoader.loadClass(className);
```

Java函数拦截技术

- Step 2: 获得Method对象

```
Method method =
clazz.getDeclaredMethod(methodName,
prototype);
```

Java函数拦截技术

- Step 3: 获得clazz的Slot域描述

```
Field field =
clazz.getDeclaredField(“Slot”);
```

Java函数拦截技术

- Step 4: 获得method的slot

```
int slot= field.getInt(method);
```

Java函数拦截技术

- Step 5: 将clazz和slot通过JNI传递到C/C++层，调用dvmDecodeIndirectRef和dvmSlotToMethod

```
ClassObject* declared_classs =
(ClassObject*)
dvmDecodeIndirectRef(dvmThreadSelf(),
clazz);
Method* method =
dvmSlotToMethod(declared_class, slot);
```

Q&A

Thank You