International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue IX, September 2017- Available at www.ijraset.com

A Study on Stagefright Attack

Arepalli Venkata Sai Krishna[1], Dr.G.Ramakoteswara Rao[2]
[1, 2] *Department of information technology, Velagapudi Ramakrishna Siddhartha Engineering College, Vijayawada, Andhra Pradesh (India) 520007*

Abstract**: *In android Media files are processed without known of user control. There are many possible ways to send yourself a mediafile. Android has a massive security bug in a component known as "Stagefright." Just receiving a malicious MMS message could result in your phone being compromised. It's surprising we haven't seen a worm spreading from phone to phone like worms. Stagefright is part of a larger group of code called Media server, which has issues of its own. Apart from running media, it also has access to Bluetooth, camera, and others.***
Keywords**: *Android, Stagefright, Media file, Media Player***

I. INTRODUCTION

Stagefright 2.0, a set of two vulnerabilities that manifest when processing specially crafted MP3 audio or MP4 video files. The first vulnerability (in libutils) impacts almost every Android device since version 1.0 released in 2008. We found methods to trigger that vulnerability in devices running version 5.0 and up using the second vulnerability (in libstagefright). Google assigned CVE-2015-6602 to vulnerability in libutils. We plan to share CVE information for the second vulnerability as soon as it is available.
Android's Multimedia Framework library written primarily in C++

1) Handles all video and audio files
2) Provides playback facilities - e.g. {Awesome}Player
3) Extracts metadata for the Gallery, etc.
4) Android is very modular
  a) Things run in separate processes
  b) Lots of inter-process communications
5) Libstagefright executes inside "MEDIA SERVER"

The architecture of the media player infrastructure, from the end-user apps to the media codecs that perform the algorithmic magic, is layered with many levels of indirection. The following diagram depicts the high-level architecture.
Android gives you the freedom to implement your own device specifications and drivers. The hardware abstraction layer (HAL) provides a standard method for creating software hooks between the Android platform stack and your hardware. The Android operating system is also open source, so you can contribute your own interfaces and enhancements.
To ensure devices maintain a high level of quality and offer a consistent user experience, each device must pass tests in the compatibility test suite (CTS). The CTS verifies devices meet a quality standard that ensures apps run reliably and users have a good experience.
Before porting Android to your hardware, take a moment to understand the Android system architecture at a high level. Because your drivers and the HAL interact with Android, knowing how Android works can help you navigate the many layers of code in the Android Open Source Project (AOSP) source tree.

A. Application Framework

The application framework is used most often by application developers. As a hardware developer, you should be aware of developer APIs as many map directly to the underlying HAL interfaces and can provide helpful information about implementing drivers.

B. Binder IPC

The Binder Inter-Process Communication (IPC) mechanism allows the application framework to cross process boundaries and call into the Android system services code. This enables high level framework APIs to interact with Android system services. At the application framework level, this communication is hidden from the developer and things appear to "just work."

©IJRASET (UGC Approved Journal): All Rights are Reserved

C. System Services

Functionality exposed by application framework APIs communicates with system services to access the underlying hardware. Services are modular, focused components such as Window Manager, Search Service, or Notification Manager. Android includes two groups of services: system (services such as Window Manager and Notification Manager) and media (services involved in playing and recording media).

D. Hardware Abstraction Layer (HAL)

The hardware abstraction layer (HAL) defines a standard interface for hardware vendors to implement and allows Android to be agnostic about lower-level driver implementations. The HAL allows you to implement functionality without affecting or modifying the higher level system. HAL implementations are packaged into modules (.so) file and loaded by the Android system at the appropriate time.

E. Linux Kernel

Developing your device drivers is similar to developing a typical Linux device driver. Android uses a version of the Linux kernel with a few special additions such as wake locks (a memory management system that is more aggressive in preserving memory), the Binder IPC driver, and other features important for a mobile embedded platform. These additions are primarily for system functionality and do not affect driver development.

You can use any version of the kernel as long as it supports the required features (such as the binder driver). However, we recommend using the latest version of the Android kernel.

In Brief

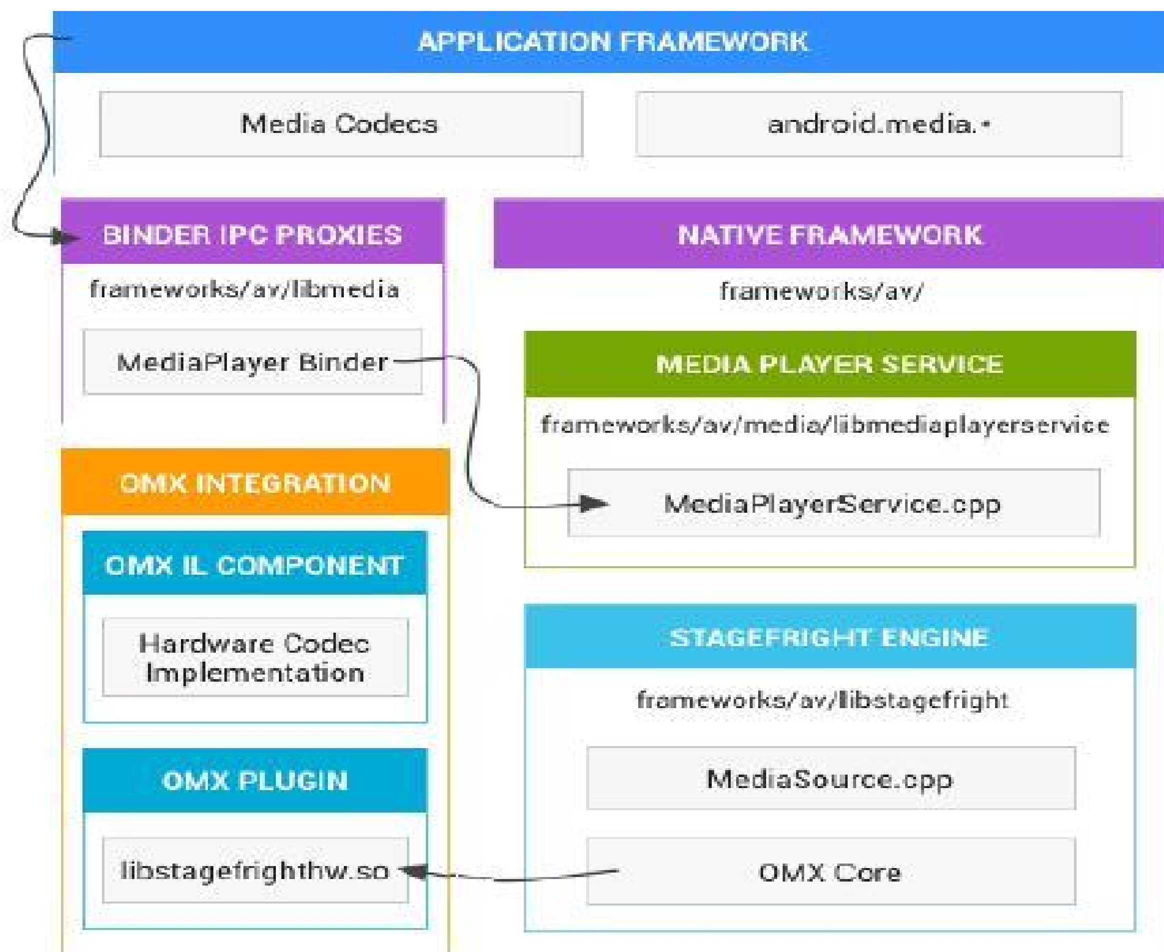

Fig 1: Detailed view of framework

©IJRASET (UGC Approved Journal): All Rights are Reserved

Architecture of the media playback infrastructure (with Stagefright as the underlying media player) My goal is to help you, an interested reader, get a grasp of how things work behind the curtains, and to help you more easily identify the part of the code base you may want to tweak or optimize. Some useful online resources already outline certain bits and pieces of the media player architecture, but the slideshow format of these descriptions omits a number of important details.
The Android vulnerability known as Stagefright is back in the limelight. Stagefright is a bug, or more accurately a series of similar bugs, inan Android programming library called libstagefright. Libstagefright is part of the operating system that handles media files such as movies. If you receive an MMS (Multimedia Messaging Service) message that links to a video, or watch a clip embedded in a web page, or download a video for later, your Android will probably load up and use the libstagefright software to play the file. Researchers at Zimperium found a number of memory corruption bugs in the libstagefright code, and prepared a paper for the recent Black Hat conference to talk about them. Presentations at Black Hat that deal with vulnerabilities in widespread devices often get a lot of publicity before the event,and with Zimperium claiming that some 950,000,000 Androids might be at risk, Stagefright got more publicity that most. Eventually, Zimperium didn't release any actual exploits at BlackHat, giving the Android ecosystem a bit more time to get its patches tested and shipped.

F. Big picture of a Stagefright

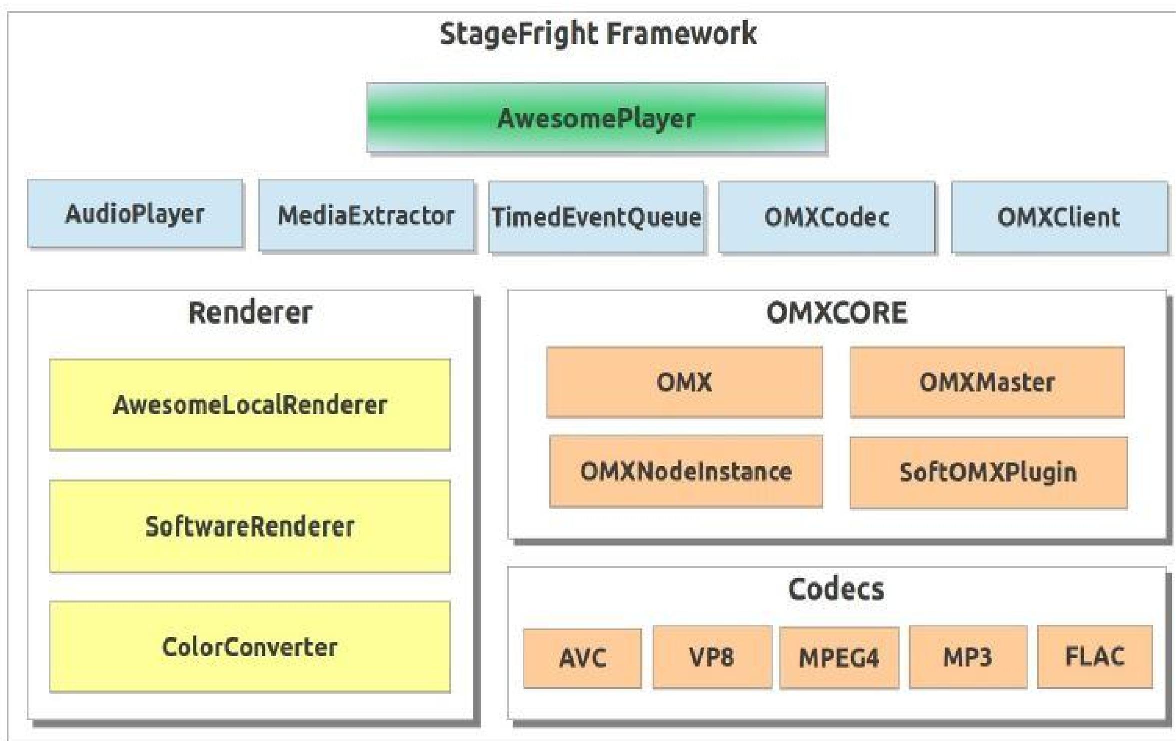

Fig2: Big picture of a Stagefright:

II. WORK FLOW OF ANDROID MEDIA PLAYER ARCHITECTURE

Architecture are the user apps that leverage media playback functionality, such as playing audio streams, ringtones, or video clips. These apps use the components from the Application Framework that implement the high-level interfaces for accessing and manipulating media content. Below this layer, things get more complicated as the Application Framework components communicate with native C++ components through JNI interfaces. These native components are essentially thin clients that forward media requests from Java code to the underlying Media Player Service via IPC calls. The Media Player Service selects the appropriate media player (in our case Stagefright), which then instantiates an appropriate media decoder, as well as fetches and manipulates the media files, manages buffers, etc. While even this high-level architecture is relatively complex, the devil is still in the details, so I will now guide you through the different subsystems can be shown in fig 3.4

©IJRASET (UGC Approved Journal): All Rights are Reserved

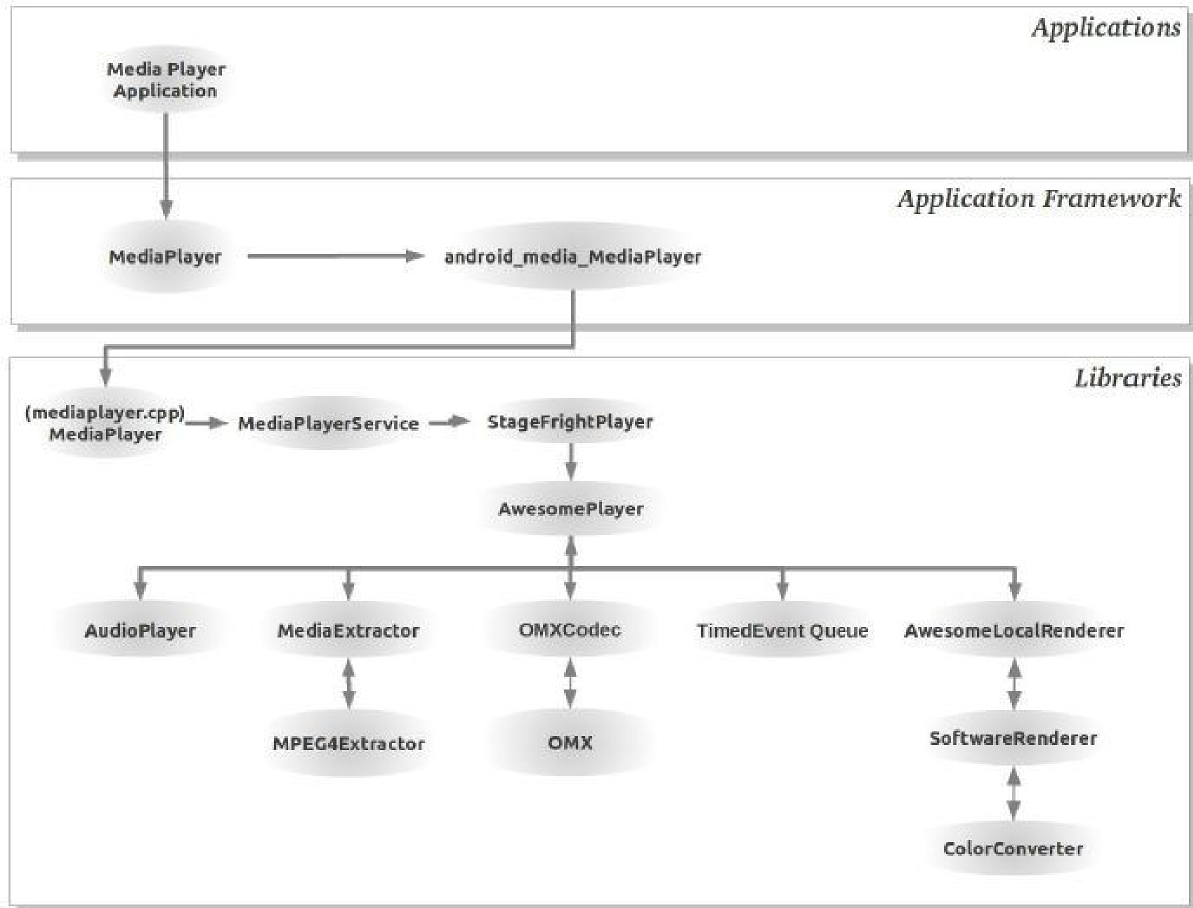

Fig3: Work Flow Of Android Media Player Architecture

AwesomePlayerleverages the OMXCodec component (implemented by the static methods of frameworks/av/media/libstagefright/OMXCodec.cpp) to set up the decoders to use for each data source (audio, video, and captions, naturally, utilize separate codecs). The decoder functionality resides in the OMX subsystem (Android's implementation of OpenMAX, the API for media library portability), where handling of memory buffers, translation into raw format, and similar low-level operations are performed. The subsystem, implemented primarily with the classes located in the frameworks/av/media/libstagefright/omx and frameworks/av/media/libstagefright/codecs folders, is complex on its own and will not be covered in this article. Stagefright Media Player components communicate with OMX via IPC invocations. The implicit client for these invocations is the MediaSource/OMXCodec object created by OMXCodec component and returned toAwesomePlayer.AwesomePlayer finally handles playing, pausing, stopping, and restarting media playback, while doing so in a different manner depending on the type of media. For audio, AwesomePlayer instantiates and invokes anAudioPlayer component that is used as a wrapper for any audio content. For example, in case only audio is played, AwesomePlayer simply invokes AudioPlayer::start () and remains idle until the audio track finishes or a user submits a new command. During the playback, AudioPlayer uses the MediaSource/OMXCodec object to communicate with the underlying OMX subsystem.

AwesomePlayer finally handles playing, pausing, stopping, and restarting media playback, while doing so in a different manner depending on the type of media. For audio, AwesomePlayer instantiates and invokes anAudioPlayer component that is used as a wrapper for any audio content.Forexample, in case only audio is played, AwesomePlayer simply invokes AudioPlayer::start () and remains idle until the audio track finishes or a user submits a new command. During the playback, AudioPlayer uses the MediaSource/OMXCodec object to communicate with the underlying OMX subsystem.

For video, AwesomePlayer invokes AwesomeRenderer's video rendering capabilities, while also directly communicating with the OMX subsystem through MediaSource/OMXCodec object (there is no proxy such asAudioPlayer in the case of video playback). In addition, AwesomePlayer is in charge of audio and video synchronization. For this reason, AwesomePlayer employs a timed

queuing mechanism (TimedEventQueue) that continuously schedules rendering of buffered video segments. When a queued timed event's deadline is reached,TimedEventQueue invokes AwesomePlayer's callback functions that perform bookkeeping and make sure that everything is running properly and that audio and video are in sync (AudioPlayer is invoked to check the state and timing of audio playback). This AwesomePlayer's functionality is implemented in theAwesomePlayer::onVideoEvent ()invokesAwesomePlayer::postVideoEvent_l() to schedule the next video segment. Similar functions are implemented in other callback functions such as onBufferingUpdate, onCheckAudioStatus, onPrepareAsyncEvent, andonStreamDone, which are invoked by TimedEventQueue when processing media playback.

AwesomePlayer implements the executive functionality, which includes connecting video, audio and video caption sources with the appropriate decoders, playing the media, and synchronizing video with the audio and captions. At initialization, AwesomePlayer's data source is set up (setDataSource command), which internally requires communication with the MediaExtractor component. MediaExtractor invokes the appropriate data parsers in accordance to the media type (e.g., frameworks/av/media/libstagefright/MP3Extractor.cppfor MPEG/MP3 audio). The returned memory reference to the data obtained in this manner is then used for media playback.

The Synchronization Formula of StageFright

$$mPositionTimeRealsUs=\left(\frac{mNumFramesPlayed+\frac{sizedone}{mFrameSize}}{mSampleRate}\right)\times 1000000$$

$$mTimeSourceDeltaUs=mPositionTimeRealUs-mPositionTimeMediaUs$$

$$nowUs=RealTimeUs-mTimeSourceDeltaUs$$

$$latenessUs=nowUs-timeUs$$

III. METHODS TO TRIGGER STAGEFRIGHT VULNERABILITY

A. *Sending exploit via MMS message*

Here the hacker only needs your mobile number, using which he will first send you a specially crafted MMS (multimedia message) containing possibly an .MP4 file. And when the MMS containing the .MP4 file gets downloaded, The hacker will then be able to remotely execute malicious code on your smartphone that can lead to data loss or compromise your sensitive data, depending on the purpose. The very fact that you get a preview of any message received over the air on all the latest versions of Android, this means that the attached media file (.MP4 file) also gets downloaded automatically. This makes this vulnerability very dangerous as it doesn't require you to take any action to be exploited. The hacker can just send the MMS, trigger the code, and wipe the MMS signs (even delete the message before you see it) while you sleep at night. Next morning you will continue using your affected smartphone without knowing that it is compromised.

B. *Embed Exploit in Android Application*

In the previous hacking method, the hacker had to know the mobile phone number for triggering exploit via MMS. With that approach if the hacker wanted to compromise large number of smartphone, he should be able to gather large number of Mobile numbers first and then invest money in sending out MMS messages to these victims.

In this Method the hacker need not know phone numbers to carry out mass hacking. He could just embed the exploit in Apps and play the infected mp4 file to trigger the exploit.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue IX, September 2017- Available at www.ijraset.com

C. HTML exploit to Target Website visitors

The hacker simply embeds the same malformed MP4 file into a HTML web page and publishes the page online. When the visitor comes in, he is presented with a video that auto plays triggering the exploit taking advantage of "Stagefright" vulnerability. This kind of attack method is very dangerous since you can lure victims to websites easily than asking them to install apps.

The Security researcher who discovered this vulnerability will be presenting his full findings, including six additional attack methods to exploit the vulnerability, at Black Hat security conference in Las Vegas on August 5 and DEF CON 23 on August 7, where he is scheduled to deliver a talk.

D. Where is the code under attack?

Once you have your environment set up, finding the MPEG4 attack surface is relative straight-forward.

1) Attach debugger to media server process
2) Place breakpoint on open
3) Open an MPEG4 video file
4) Sift through breakpoint hits until r0 points at your file
5) Look at the backtrace
6) Dig in and read the surrounding code

E. What do you find?

1) Open("/sdcard/Music/playing.mp4",...) called from:
2) Open (pathname=<value optimized out>, flags=0) at bionic/libc/unistd/open.c:#1 0x40b345e8 in File Source (this=0x479038, filename=0x478d08 "/sdcard/Music/playing.#2 0x40b332fe in android::DataSource::CreateFromURI (uri=0x478d08 "/sdcard/Music/#3 0x40b2ef50 in android::AwesomePlayer::finishSetDataSource_l (this=0x478058) at frameworks/base/media/libstagefright/AwesomePlayer.cpp:2085)
3) 0x40b2efb2 in android::AwesomePlayer::onPrepareAsyncEvent (this=<value optimized #5 0x40b2c990 in android::AwesomeEvent::fire (this=<value optimized out>, queue=#6 0x40b50c28 in android::TimedEventQueue::threadEntry (this=0x47806c) at frameworks/#7 0x40b50c6c in android::TimedEventQueue::ThreadWrapper (me=0x47806c) at frameworks/#8 0x400e8c50 in __thread_entry (func=0x40b50c59 <android::TimedEventQueue::ThreadWrapper(#9 0x400e87a4 in pthread_create (thread_out=<value optimized out>, attr=0xbe81ea28

IV. CONCLUSIONS

Ultimate goal: Find out how to get attacker controlled media files in android library files. Try all possible ways to send yourself a media file! Depends on knowledge of "all possible ways". As far we have seen several attacks, risk factors involved and some basic prevention steps. Android's code base needs more attention to check fuzz, test, and submit to the Android VRP especially in situations where multiple attempts are possible. Android devices also include an application sandbox designed to protect user data and other applications on the device.

REFERENCES

[1] StageFright - Zimperium Blog: https://blog.zimperium.com/experts-found-a-unicorn-in- the-heart-of-android/
[2] StageFright Vulnerability Details: https://blog.zimperium.com/stagefright-vulnerability-details-stagefright-detector-tool-released/
[3] Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection, and Mitigation by Xing Jin, Xunchao Hu, Kailiang Ying, Wenliang Du, Heng Yin and Gautam Nagesh Peri.
[4] Android's StageFright Media Player Architecture: https://quandarypeak.com/2013/08/androids-stagefright-media-player-architecture/
[5] An innovative model approach to prevent malformed pro sql tuples in general DB model by G. Subhasree, Dr. G. Ramakoteswara rao, Dr. P.Vidya Sagar
[6] Android 5.0 Integer Overflow Fix: https://android.googlesource.com/platform/frameworks/av/+/f106b19%5E!/
[7] Prevention of CSRF and XSS Security Attacks in Web-Based Applications by D.Kavitha in March 2016 in IJIRSET.
[8] Vulnerabilities Of SmaRTPHONES by Shakuntala P. Kulkarni, Prof Sachin Bojewar in Dec-2015 in IRJET.
[9] An Android Multimedia Framework based on GStreamer by Hai Wang, Fei Hao, Chunsheng Zhu.
[10] WEBVIEW VULNERABILITIES IN ANDROID by SANDEEP JAIN, PREETI KHANNA in MAY 2014 in INTERNATIONAL JOURNAL OF SCIENCE, ENGINEERING, AND TECHNOLOGY(IJSET).
[11] Cross-Site Request Forgeries: Exploitation and Prevention by William Zeller and Edward W. Felten.
[12] ANALYSIS OF ANDROID VULNERABILITIES AND MODERN EXPLOITATION TECHNIQUES by Himanshu Shewale, Sameer Patil, Vaibhav Deshmukh and Pragya Singh in MARCH 2014 in ICTACT JOURNAL ON COMMUNICATION TECHNOLOGY.

©IJRASET (UGC Approved Journal): All Rights Reserved