AndroSimilar : Robust Statistical Feature Signature for Android Malware Detection

Parvez Faruki*
Malaviya National Institute of Technology, Jaipur, India
parvez@mnit.ac.in

Vijay Ganmoor
Malaviya National Institute of Technology, Jaipur, India.
vijay.ganmoor@gmail.com

Vijay Laxmi
Malaviya National Institute of Technology, Jaipur, India.
vlaxmi@mnit.ac.in

M. S. Gaur
Malaviya National Institute of Technology, Jaipur, India.
gaurms@mnit.ac.in

Ammar Bharmal
Malaviya National Institute of Technology, Jaipur, India.
bharmal.ammar@gmail.com

ABSTRACT

Android Smartphone popularity has increased malware threats forcing security researchers and AntiVirus (AV) industry to carve out smart methods to defend Smartphone against malicious apps. Robust signature based solutions to mitigate threats become necessary to protect the Smartphone and confidential user data. In this paper we present AndroSimilar, a robust approach which generates signature by extracting statistically improbable features, to detect malicious Android apps. Proposed method is effective against code obfuscation and repackaging, widely used techniques to evade AV signature and to propagate unseen variants of known malware. AndroSimilar is a syntactic foot-printing mechanism that finds regions of statistical similarity with known malware to detect those unknown, zero day samples. Syntactic file similarity of whole file is considered instead of just opcodes for faster detection compared to known fuzzy hashing approaches. Results demonstrate robust detection of variants of known malware families. Proposed approach can be refined to deploy as Smartphone AV.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection–Invasive Software(Trojan, Worms, Rootkits etc.)

General Terms

Experimentation and Security

Keywords

Android Malware, Improbable Features, Code Obfuscation, Similarity Digest, Statistical features

1. INTRODUCTION

*Corresponding Author

SIN '13, November 26-28, 2013, Aksaray, Turkey.

http://dx.doi.org/10.1145/2523514.2523539

Penetration of Android Smartphone and Tablets has increased manifold due to ease of use and cheap availability. Android developed for low power, memory constrained embedded devices has captured more than half of the total market share. Gartner research predicts sale of one million smartphone devices in 2013-14 with android taking more than 60% of total sales [15]. Android is popular as it is open source, developed by Google and supported by Open Handset Alliance (OHA). Robust application features has increased the popularity of Android. Software distribution on smartphone platforms differs from conventional Personal computer (PC) distribution. Smartphone applications, popularly called *apps*, are hosted on central repository of Google [19] (Official Android Market - Google Play), reputed third party app-stores and local third party markets. Apps are installed directly on user's Smartphone unlike download and execute on Windows based Personal Computer.

Official Android market place and certain popular third party repositories do not distribute executable code and their cryptographic hash as a trusted app source. Increasing popularity of Android OS has attracted malware developers to exploit android ecosystem by inserting code obfuscation, repackaging popular paid apps to distribute them in less monitored third party markets. Repackaging is a process of inserting malicious content inside a benign app using reverse engineering techniques. Official Android market, Google Play, has more than 2,04,000 free apps and many paid apps. Popular third party store Amazon app-store [1] in-house thousands of free and paid Android apps. Online repositories are streamlined with different protection methods to guard against threat(s). Popular response has attracted the malware developers to misuse these centralized stores also by injecting malicious code into them [7]. In 2012 Enck et al. [10] explored 1100 apps from Google Play store to ascertain security perspective of developed apps. Incremental version changes of Android OS is also fast for distributors to release security patches. Frequent updates and patches are difficult to manage as 17 upgrades have been distributed since its launch in 2008.

In this research, we present a robust statistical attribute extraction approach that explores improbable byte features [21] for capturing code homogeneity among variants of known apps. Inter and Intra family homologous features can be determined with proposed approach as variants of a family share common attributes [18] to identify code similarity of an unknown sample and explore its similarity with known malicious family(s). Unknown variants generated with obfuscation techniques are marginally dissimilar but they still defeat AV signature detection approach. Code homogeneity is captured by searching for strong features that is visible within re-

lated files, but has a very low probability of occurrence among two unrelated files. To detect the similarity of unknown samples with existing malicious ones, we propose AndroSimilar, a robust syntactical approach which generates variable length signatures, to detect unseen, zero day samples crafted from known malware. Proposed approach is effective compared to context trigger piecewise hashing technique SSDEEP [18], proposed by Xiang et. al [30] to detect repackaged apps on third party android markets. Proposed approach is tested against 49 malware families consisting 1260 apps, 6779 official android market and 545 third party market apps.

2. BACKGROUND

Google Play consists various categories apps such as Books, Business, Casual, Communication, Security, Media, Lifestyle, Personalization etc., to enable smartphone users to gain smooth access to repository. Google Play maintains repository of android app developer accounts. Apps are self signed and uploaded to the official android marketplace Google Play. Bouncer, a heuristic, dynamic detector guards the repository against malware and malicious attacks [17]. Bouncer verifies runtime behavior of an app before publishing to the official market. Developers may also upload same app(s) on other third-party android stores to strengthen their product reach and earn advertisement revenues. Some apps are available at the Google Play while some are available only with third party markets due to their localized use.

Taking varied nature of availability, malware writers use available tools to dissect popular apps, insert malicious contents like adware, spyware, IMEI, IMSI stealer, SMS Trojan to sends messages to premium rate numbers, and push them by repackaging, towards less monitored third party markets. Boxer SMS Trojan discussed in [2] employed this approach to extract money by inserting malicious logic in Need For Speed Shift, a popular app on official market. Repackaging also hurts the original app developer reputation and revenues, as malware writers may replace publisher identification of affiliated ad networks with their own to earn money. Repackaged apps pose a serious threat to the centralized app distribution system. Users trust Android app markets assuming that installed apps do not contain illegitimate contents to spy on their activities or steal user information or drain their money.

We present AndroSimilar, a robust syntactic signature approach to detect unknown, obfuscated, repackaged variants of known malware families. We employ statistical similarity digest hashing scheme on the byte stream based on robust statistical malicious features not easily seen among normal apps. Sample signature match with a known malware beyond an experimental threshold (35%), flags sample as potential threat.

Rest of the paper is organized as follows. In section 3 we briefly discuss Android APK file format with a discussion on reverse engineering and app repackaging. AndroSimilar approach and signature is covered in 4. Evaluation of proposed method is discussed in 5 followed by discussion in section 6. Related work is discussed in section 7. Conclusion and future work is discussed in the last section.

3. ANDROID APP OVERVIEW

Android app is written in Java. Java source is compiled as class files (.class), converted into Dalvik Executable using Dx tool [14]. Android app source executes on Dalvik Virtual Machine (DVM), a register based Virtual Machine for embedded devices [14], unlike Java Virtual Machine (JVM), a stack based virtual machine. Executable code of the app is inside Dalvik executable (.dex) file. Typical app structure is shown in figure 1.

3.1 App Structure

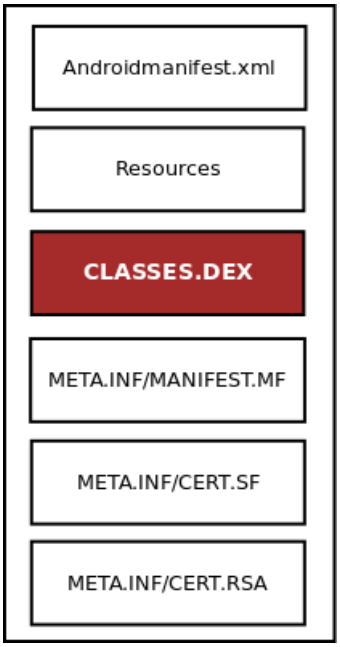

Figure 1: APK File format

Androidmanifest.xml has important information like Package Name, Permissions, Activities, Services, Receivers and Content Providers. Icons, shortcuts, images, string constants, dimension constants etc. are made available through resources.

CLASSES.DEX file contains executable code. META.INF folder has developer certificate information. This content is compiled into a single package Android PacKage (APK). Once the app is developed, developer self signs it with his private key and publishes it on Offical or third party android market. Application development is made easier by eclipse environment with ADT plugin [27].

3.2 App Reversing on Android Platform

Android apps are amenable to decompilation in Java source compared to structural language C. App is reversed to its source code to check as well as perform obfuscation or repackage an known popular app. Decompilation convert existing app to its original Java source with help of tools JDGUI [16] and dex2jar [9]. Dalvik executables can also be converted to smali code by disassembling it to obtain opcodes for investigation.

3.3 App Repackaging

Repackaging methods can be applied directly on an app without disassembling it to create a look alike of the popular paid / free app from app-markets. Repackaged app is transformed into a new one by preserving the actual semantics. File alignment is one repackaging technique, used to realign the existing apk for better utilization of memory. Realignment process modifies the content but preserves the semantics of executable. Zip align [14], available with Android Development Kit, is used for the same purpose by changing the byte boundaries and thus it may change the cryptographic signatures of an app that can evade fixed block analysis in a file.

Developer signature must be added into an app before submitting it on Google Play or a third party market. Google Play approach is based on trust, where assumption is that a developer would care for his image once he is certified by his private key. But apps can be

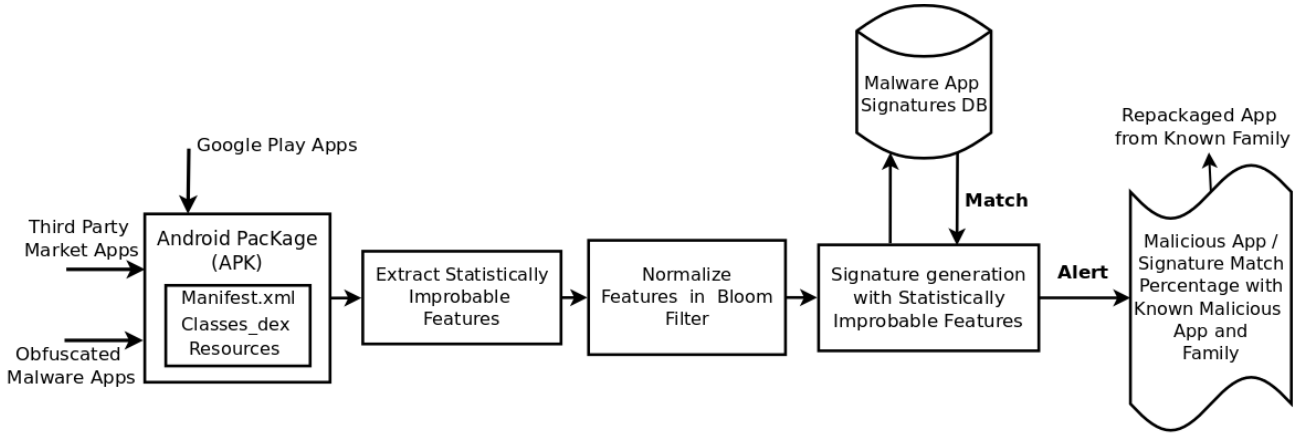

Figure 2: Proposed Methodology : AndroSimilar

re-signed multiple times using different reverse engineering tools and thus developer certificate can be changed. Existing app can be ripped-off using apktool [3] and dex code can be disassembled using tools like smali/baksmali [4]. After making some changes in it, same can be repackaged using apktool. Our approach is resistant to changes being made at start or end blocks as we have considered statistical file features unlike context triggered piecewise hashing scheme which calculates variable size block based on context trigger.

Proposed approach is also resistant against the code obfuscation techniques as malicious code insertions into the executable at different locations does not change the statistical features of the file. Code insertions at different blocks change the structural aspect of the app, still variable size signature is robust for searching similarity. Legitimate apps when developed in incremental versions, share similar code with each other.

4. PROPOSED METHODOLOGY

AndroSimilar approach is shown in the above diagram in figure 2. Algorithm of the proposed approach is discussed in figure 3. We have generated statistical signatures of known malware families in our database. Score threshold is the similarity index value beyond threshold (experimental threshold 35%) match with malware database, app under observation is flagged malicious with signature match of known malware.

Completely unrelated files have lower probability of having common features. When two unrelated objects share same features, they become weak leading to false positives (different files sharing common features). This attribute is uncommon files, unrelated data sources [20]. We have investigated similarity digest signature of android apps. Feature classification is performed with byte stream features normalized entropy for feature extraction where high entropy value leads to low clarity in classification and low entropy shows definite judgement towards a clear information. Diagrammatic representation proposed approach is discussed below.

- Collect Google Play and third party market apps as input to AndroSimilar.
- Generate entropy features for byte blocks, normalized them for feature representation.
- Select the normalized features from the blocks to select statistically strong attributes.
- Select the most popular features according to similarity digest scheme as representative attributes for the data block.
- Most popular 128 features are stored in bloom filter for signature generation, adding bloom till signature is generated.
- Generate signature database for the malicious apps.
- Compare the collected apps with signatures to detect match with known signature beyond a given threshold
- Identify percentage similarity with the known malicious sample to detect code obfuscated, repackaged malicious sample.

Algorithm:	AndroSimilar – Malicious apps detection.
Input:	*mal_sign_db* – Improbable features signature database of known malicious apps. *appset* – Apps to be checked. score_threshold – Similarity threshold.
Output:	*malicious_set – Set of tuples as (appName, familyName).*

```
1: for app in appset do
2:     score_results: an empty array.
3:     app_sign = generate_improbable_features_signature(app)
4:     for malicious_sign in mal_sign_db do
5:         score = compare_signatures(app_sign, malicious_sign)
6:         score_results.add(score)
7:     best_similarity_score = find_best_score_match(score_results)
8:     if best_similarity_score >= threshold then
9:         tuple = (app.name, best_similarity_score.familyName)
10:        malicious_set.add(tuple)
```

Figure 3: AndroSimilar Algorithm

Detecting repackaged apps requires an effective and efficient environment. We have incorporated varied apps from the official market to consider similar functionality performed by the legitimate .apk files. Malicious functionality is inserted into a legitimate program by modifying the assembly (smali) code. Malicious function of known malware is modified and its corresponding smali code is inserted into a legitimate application to achieve malicious intention like sending SMS to premium rate numbers, collecting IMEI

or IMSI of user smartphone, sending user information to some remote location without user knowledge. Malware may perform normal functions along with some malicious class spreading viciousness. When such malicious code is inserted into the legitimate applications, malware developers ensure that they do not make drastic changes to the existing app, so as to evade the detection approaches. Moreover google play also runs an dynamic emulated environment Bouncer [17] on Google Play cloud to ascertain and verify the behavior of apps uploaded by the developers.
Entropy of an file is given as:

$$H(S) = \sum_{i=0}^{255} P(X_i) \cdot \left(\frac{1}{\log(P(X_i))} \right)$$

where $(X_1 ... X_{255})$ are ASCII characters. Entropy value is calculated for each file block of size 64 bytes. It is then normalized into a range 0 to 1000 using:

$$H_{normal} = \left\lfloor 1000 \cdot \frac{H(S)}{\log_2 W} \right\rfloor$$

where B is the byte block for normalization. Data sequence in close regions share strong entropy estimate as Byte sequences have common components in nearby regions. ASCII data features repeat themselves by some difference (0-255). When entropy is measured for smaller blocks, high variation may not seem to occur leading to weak or ambiguous features [21].
Whenever a byte sequence consists of 64 different characters, particular block will have maximum entropy, but such cases are highly unlikely as closely connect byte content is stored nearby. Certain low, high threshold values also assist in discarding weak / repetitive but unimportant features. Considering ambiguous byte sequences increases false positives. Higher occurrence of false positives generates incorrect class assignment. To calculate unique and improbable features we calculate observed distribution of entropy by considering most popular features occurring at regular intervals, and discard certain features though occurring repetitively but ambiguous.

4.1 Robust feature Signature generation

To calculate robust features that remain persistent among related samples, normalized entropy feature are assigned Precedence Rank by associating normalized bytes based on empirical observations. This rank is a measure of occurrence of features obtained by reading the byte content. A feature whose likelihood of occurrence is lowest gets a low rank. Certain features having very high / too low score are given null score to avoid introduction of weak features during attribute selection to ensure header content is excluded for obtaining strong features.

Rprec	**882**	**807**	**852**	**849**	**834**	**852**	**866**	**866**	875	882	859	849	
Rpop		1											
Rprec	882	**807**	**852**	**849**	**834**	**852**	**866**	**866**	**875**	882	859	849	
Rpop		2											
Rprec	882	807	**852**	**849**	**834**	**852**	**866**	**866**	**875**	**882**	859	849	
Rpop		2			1								
Rprec	882	807	852	**849**	**834**	**852**	**866**	**866**	**875**	**882**	**859**	849	
Rpop		2			2								
Rprec	882	807	852	849	**834**	**852**	**866**	**866**	**875**	**882**	**859**	**849**	
Rpop		2			3								

Figure 4: Robust feature extraction

As shown in figure 4 the precedence values of corresponding normalized entropy values are shown in each row. Each feature is 64 bytes. Window W rolls over consecutive precedence features $R_{precedence}$ and selects lowest ranked left most from the window. In row one $R_{precedence} = 807$, popular feature is incremented and Window rolls over the next consecutive features. Row 2 picks up $R_{precedence} = 807$ and increments it by one. Procedure continues for the file till feature block of B bytes is investigated and window of size W moves over consecutive features, feature with lowest Precedence Rank is searched and rank is associated with the feature. The popularity score keeps a tab on the feature occurrence in feature window. The ranked features are selected based on a threshold value to exclude weak ones. Threshold value is considered for the popular features to remove ambiguous features to enable collection of strong improbable features. For the above block if threshold value 2 considers 807 and 834 as representatives of the file block.
In some files some content may be filled with zeroes in large number of blocks. Such attributes creates false positives, but they are only 2-3% of the file size [22]. Removing ambiguous features helps containing false positive rates and it does not have an effect on feature selection. Feature filtering can remove features below 100 and above 990 to eliminate weak attributes [21]. Maximum entropy value appears due to header content being common among the files. The similarity digest hashing technique[21] uses these features to generate a signature for each app. We created a set of signatures for malicious samples using this approach, which would be input along with samples to test. Then, for each app we create a signature, compare it with each malicious signature to get a similarity score. From the resulting matches, we find a best similarity score. If this score is higher than experimental threshold(>=35), then app is identified as malicious sample matching a particular family and classify it as a variant of the known family. As improbable features were selected, any app under test shows similarity with signature of existing malware if it is beyond a given threshold.

5. EVALUATION

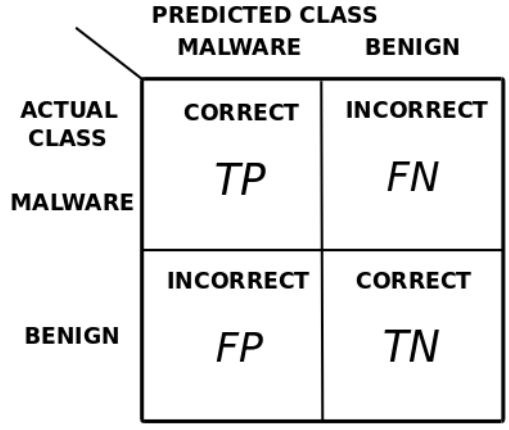

Figure 5: Evaluation Model

Proposed approach AndroSimilar is implemented in Linux. Our approach can be used in real time as the samples are scanned in average one second. Unlike DroidMOSS where disassembled dalvik opcode is considered, we rely on statistically strong file features of an app based on normalized entropy. Without depending on tools for analysis, we have generated a variable length signature from

strong features to represent a malicious signature. MNIT Android malware dataset is labelled consisting 49 families with 1768 malware downloaded from Contagiodump [8] and allied user agencies. We have matched the samples with family classification by Zhou et al. [26], Dissecting Android Malware, Characterization and Evolution.
Figure 5 displays the evaluation model of the approach. True Positive (TP) are the instances correctly classified as malicious. False Negative (FN) instances are Malicious ones incorrectly identified normal apps. True Negative (TN) class identifies normal apps correctly. False Positive (FP) misjudges good apps as malicious.
Proposed method effectively identifies normal apps with high detection due to its robust signature generation approach. The statistical signature considers strong malicious features which are not easily available in every normal app. High value in correct class prediction and lower mis classification-classification is a good sign of effective detector. Signature matching is performed with an experimentally determined threshold value. A variant may also be created from two different base samples, the signature percentage is able to identify malware family the unknown sample belongs to.

5.1 Results of Proposed Model

Category	Files Analyzed	Correct Detection	Suspicious (FP)	% Accuracy
Google Play apps	6779	6739	40	99.4
Third Party apps	545	539	06	98.89

Table 1: Testing benign apps against 1260 malicious signatures.

Table 1 shows test result for Google Play and Third party market apps with AndroSimilar. Proposed model had 1159 signature in the database for comparison with the test set. Out of 6779 apps scanned with AndroSimilar, We were able to find 40 official market apps matching suspicious signatures with our malware database and 6 third party suspicious apps. They are given in the above table as false positive. Manual analysis of third party apps revealed the robustness of our approach. All the third party market apps are repackaged versions of existing known apps with malicious permission and advertisement signatures. We also verified the similarity of these samples using Androguard [6], a semantic similarity matching research tool. 34 samples of Google Play also has 40% and higher match with our signature database. Though the signature database is limited, AndroSimilar is robust against some known code obfuscation techniques.
Table 2 displays detection of unknown and known code obfuscated malicious samples. 101 unseen android malware were given input for validation. 73 samples from the 101 unknown were matches correctly with AndroSimilar beyond the threshold value. 23 samples did not match our signature threshold. Same samples were also given to well known AV on VirusTotal for detection. Except top three AV AndroSimilar performed better.
Part two in table 2 shows results for code obfuscated malicious samples. We applied obfuscation to known malware using code obfuscation techniques like method renaming, Junk method insertion, Control flow obfuscation using GOTO and string encryption to change their known signature. These code obfuscated samples were given input to AndroSimilar to check its effectiveness against various code obfuscation techniques as obfuscation techniques have changed the signature of the known malware. Out of 120 samples obfuscated using method renaming technique, 75 were correctly identified with the existing malware signature of AndroSimilar. Though there is a false negative, still more than 63 percentage samples showed signature similarity with original malware above AndroSimilar similarity threshold. In case of junk code insertion out of 120 samples, 76 samples were correctly matched with original malware in AndroSimilar database.
AndroSimilar signature performed well against control flow obfuscation where 75 of the 118 samples were identified to have similarity beyond the threshold value. On an average, proposed method is able to identify more than 60% samples with the existing signatures. This means lesser similarity digest signatures are needed for variant detection of known malware families.

Category	Files Analyzed	Correct Detection	False Negative	% Accuracy
Unknown Malware	101	73	28	72.27
Code Obfuscated				
Method Renaming	120	75	45	62.5
Junk Method Insertion	120	76	44	63.33
Goto Obfuscation	118	75	43	63.56
String Encryption	68	44	24	64.7
Total	426	270	156	63.3

Table 2: Results of Unknown (test Samples) and Code Obfuscated Samples with Proposed Approach

5.2 Detection of unknown Malware variants

While evaluating scan result match with our signatures database in Table 1, 40 false positives above threshold signature were detected as suspicious from Google Play and six suspicious samples were reported from third party markets. Manual analysis of third party market apps against their matching malware signature of our database is shown in table 3, revealed malicious samples are repackaged apps of six known benign apps on third party markets. Manual analysis of apps flagged malicious by AndroSimilar revealed malicious insertions of permissions, launchers, and main activities as shown in below. Signature of our malware database detected 43% similarity with repackaged malware. To verify this we gave the third party apps and our matching apps to Androguard [6], a semantic NCD based tool to verify similarity percentage for existing third party apps. We were able verify that the similarity percentage given by AndroSimilar is comparable to Androguard. Official android market apps detected suspicious are under manual analysis to identify malicious behavior.
We are in process of manual analysis 40 suspicious samples of Google Play to identify if these samples resemble the existing signatures below threshold or they are unknown repackaged samples.

6. DISCUSSION

We applied the fuzzy hashing approach by Andrew Tridgell to verify the robustness of signature. When the test-set of Official android apps was input to fuzzy hash, more than 10% samples showed similarity with existing malicious signatures although being clean files. We validated by uploading the samples on VirusTotal to verify be-

Table 3: Repackaged third party market apps flagged suspicious with Proposed approach

Third Party Store App	Malicious App	Comment
SHA-1:2ea5f6dc465cf89caab438ae8bcb5b42de3e444f MD-5: ee8be1b7f2761f42957bb22b2a0f6414 Package Name: com.requiem.armoredStrike	SHA-1: bdd581d2d57ad71b0bf6401e76c8120cd5dc0173 MD5: 5d27c7d0c5630f4c7a8b7a8f45512f09 Package Name:com.requiem.armoredStrike **Malware Family : Geinimi**	**Disguised as updated version** Original Main launcher receiver changed **Dangerous Permissions added:**ACCESS_FINE_LOCATION, ACCESS_GPS, CALL_PHONE, READ/WRITE CONTACTS READ/SEND_SMS, READ/WRITE HISTORY_BOOKMARKS.
SHA-1: fc438c6d0cabc2190c26b41e9dc5d3d3843274eb MD5: ec3b45dc3ebda87cc420722f1895e75c Package Name: com.camelgames.hyperjump	SHA-1: 00983aad12700be0a440296c6173b18a829e9369 MD5: 513971a8cde07e145a85a8707f83e4b5 Package Name: com.camelgames.hyperjump **Malware Family : Pjapps**	**Service added:**Intent receivers for SMS_RECEIVED, WAP_PUSH_RECEIVED. **Dangerous Permissions added:**RECEIVE_SMS, RECEIVE_MMS, SEND_SMS, NEW_OUTGOING_CALL, READ/WRITE_HISTORY_BOOKMARKS, INSTALL_PACKAGES.
SHA-1: 97bc745f247da451995a0be635150eb71a3c0f2f MD5: 07e640792506d889b67e4a7061a9aff7 Package Name: jp.hudson.android.militarymadness	SHA-1: 1f522d9ab07fc716e7f201fad12ccea396987a83 MD5: 6ea15fdda8ba208b19e5c7131e9d413f Package Name: jp.hudson.android.militarymadnes **Malware Family: GoldDream**	**Very minor change in package name and activities added.** Intent receiver for BOOT_COMPLETED, SMS_RECEIVE, PHONE_STATE, NEW_OUTGOING_CALL added. **Dangerous Permissions added:** RECEIVE_SMS, READ_SMS, INSTALL_PACKAGES, RECEIVE_BOOT_COMPLETED, HISTORY_BOOKMARKS.
SHA-1: 2b11e7d3bc8421da143deab57acaea03e0a83b1c MD5: 21d81b064951e9ed7beff98d6879e55a Package Name: jpcom.wuxi.GoldMiner.domob	SHA-1: b9d992b88ef1a4a75362f8f5d069716ea7a3321e MD5: 025a55c1bcbd3be2ca03aa314ce9a4c2 Package Name: jpcom.handcn.GoldMiner.free **Family: Geinimi**	**Original Main launcher receiver changed.** Advertisement Publisher ID changed, along with the ad-library **Dangerous Permissions added:**CALL_PHONE, INSTALL_SHORTCUT, READ/WRITE_CONTACTS, SEND/READ_SMS, READ/WRITE_HISTORY_BOOKMARKS, ACCESS_GPS, ACCESS_LOCATION.
SHA-1: 9ef6fc25d9e599ce6bfa7c3fb4d21a775f5cf98f MD5: c9caebc6cb727d720e04cb77ce1e042e Package Name: com.camelgames.mxmotorlite	SHA-1: ef140ab1ad04bd9e52c8c5f2fb6440f3a9ebe8ea MD5: e5b7b76bd7154dea167f108daa0488fc Package Name: com.camelgames.mxmotor **Family: AnserverBot**	**Many new activities and services added.** Receiver for SMS_RECEIVED, BOOT_COMPLETED, PICK_WIFI_WORK added. **Dangerous Permissions added:**READ/RECEIVE/WRITE_SMS, RECEIVE_BOOT_COMPLETED, READ/WRITE_CONTACTS, CALL_PHONE, READ_PHONE_STATE.
SHA-1: 218af28eeb168dd16df6d0faacb3f77f51fc66df MD5: 4b0c93b1a14b5fdad60715a8a6b1987e Package Name: com.moregame.drakula	SHA-1: b9891ab782cf643c81f0f1c130ea119384dbefe1 MD5: 8498984d8f9b7260fd032d6f0a2534aa Package Name: com.moregame.drakula **Family: Geinimi**	**Original Main launcher receiver changed.** **Receiver for BOOT_COMPLETED event added.** **Dangerous Permissions added:**ACCESS_FINE_LOCATION, CALL_PHONE, READ/WRITE_CONTACTS, READ/SEND_SMS, READ/WRITE_HISTORY_BOOKMARKS, ACCESS_GPS, ACCESS_LOCATION added.

nign nature using updated and leading AV products. Fuzzy hashing scheme is prone to False Positive as the signature a fixed signature is generated instead of considering any statistical feature.
Experimental results of AndroSimilar demonstrate its effectiveness in detecting unknown, unseen suspicious samples from official and third party market as discussed in result section. Here, we discuss the limitations of our approach and discuss further improvements for sound detection of variants using family signatures instead of signature for malicious apps. Our initial assumption was Google Play is better guarded than other markets do they only have native apps. Androsimilar malware signatures database shows 40% similarity with official market apps. DroidMOSS [30] assumed all Google Play apps to as benign ones and thus their model only looked at repackaged apps on third party markets. Our manual analysis for the google play apps is still going on for reporting, but third party malicious apps are reported and discussed in results section. In case of same application, original app may be available on a reliable third party market and its repacked version may have been pushed to the official app market. AndroSimilar is robust enough to still find the matching signature and percentage matching similarity with an existing malware. Proposed approach is still in evolution phase and we are yet to determine a single / multiple family signatures to detect samples of the whole family with a single signature. AndroSimilar is still a robust technique to determine code obfuscated malicious samples based on their signatures.
Our crawled dataset is still nascent as it contains 20,000 google play apps and 4000 third party market apps that are available for free. We have not considered popular paid apps from both the markets. We are ascertaining whether popular paid apps are available freely on third party markets to match with our malware signature database. AndroSimilar malware database still consists of very less malware signatures in comparison to commercial AV solutions. There is a still a possibility that there may be more repackaged apps on both markets that are unknown or unseen zero day samples that need to be reported. AndroSimilar works at file level instead of opcode, so manipulation of shared library in repackaging remains a point of investigation.

7. RELATED WORK

Android malware outbreak has reached from 3 families (50 samples) in 2010 to 102 families consisting thousands of sophisticated samples capable of causing considerable harm to the security of Android OS. Software similarity is a measurement to detect plagiarism at file level to detect similar content. Our approach works at file level to search strong statistical features that can differentiate between a malware and normal Android app. Features extracted from variants of known malware families are useful in detecting unknown malware of known malware families. Proposed approach is different from a known DroidMOSS [30] as it is based on fuzzy hashing technique known as Context Trigger Piecewise based on SSDEEP fuzzy hashing scheme, generating a fixed 80 byte signature to detect repackaged malware. Investigation of Permissions in Android manifest, permission escalations was focussed during 2011. Our Proposed model generates normalized entropy features that are selected based on their popularity in a file block with reference to its popularity of occurrence.
Our Improbable feature generation technique is based on Vassil Roussev's similarity digest hashing algorithm for ascertaining similarity among different applications [21]. The above approach proposed to detect file level similarity has been extended for document matching, images and compressed file analysis for similarity measurement. Roussev et al. [22] applied the concept in digital forensics to show similarity of robust features in file forensics. In [23] the authors have considered entropy features of byte blocks starting from 500 upto 10,000 incrementally to determine Packed Windows binary. Their approach is interesting but depends on how the software packer insert packer code in the executable. Initial or last byte consideration may fail as packer code can be inserted randomly.
DroidAnalytics [28] proposed a static analysis framework for effective detection of obfuscated malware Android malware. Signature generation to detect code obfuscated, repackaged malware is done at method level, class level and app level in three categories to detect repackaged malware. The approach seems effective against simple obfuscation techniques like method renaming, control flow obfuscation using goto and string encryption. This framework for static analysis does not discuss the time taken for analyzing the malicious apps.
David Berra et al. [5] proposed a mechanism to detect malicious code based on permission distribution in Android. Adrian Portel Felt et al. [11] also investigated issues in permission escalations to detect the malicious content from official as well as reliable third party markets. Permissions in Android are categorized public, dangerous and system permissions. Woodpecker tool [12] analyzed the apps to identify the dangerous permissions that may compromise the OS.
Xiang et al. [26] performed an extensive study of their malware repository by characterizing them according to the functionality. They were able to identify 211 real malware. DroidRanger approach applied footprint mechanism at the opcode level to find out malicious family logic. Although the approach is interesting it has an overhead of disassembly and opcode analysis leading to less preference in real time scanning. Similarly they also developed [30] fuzzy hashing approach to detect repackaged apps at opcode (dex) level. The approach seems time consuming for real-time detection as there is a disassembly overhead and comparison of opcode using similarity scores. As we have already noted context triggered fuzzy hashing approach mis classifies 10% app official android market apps malicious. Moreover, Context triggered approach generates a fixed 80 bytes string with fuzzy hashing scheme, the approach becomes less effective against large size content.
RiskRanker [13] analyzes the behavior of an app to identify if it has dangerous logic to compromise the security. The approach relies on class path to identify any mutation, but remains ineffective against obfuscation techniques applied for obfuscation. Our proposed approach overcomes some problems of already known repackaged app detection by using a robust signature based approach to identify the malicious features. If the sample consists of match with signature beyond 35%, it is flagged suspicious for further analysis. Size of signature remains compact enough to be effective in real time detection.
TaintDroid [25] is a dynamic analysis tool used that monitors executing apps to find out privacy leakage caused by an executing app. In [24] authors identify malicious behavior based on frequency of dangerous app permissions requested. Based on the specific permissions the apps are declared unsafe for users.
DroidRanger [29] applies a combination of static and dynamic analysis for malicious code detection. Complementary approach helps in observing certain features that may not be possible with a single approach.

8. CONCLUSION

In this paper, we have considered detection of Android malware, code obfuscated and repackaged samples from existing market place and present AndroSimilar prototype to detect malicious Android apps with a robust statistical signature. Proposed method utilizes statistically improbable feature selection using similarity digest hash-

ing mechanism, a variable length signature generation approach. Proposed method effectively detects existing, code obfuscated malware with techniques like string encryption, method renaming, junk method addition and control flow obfuscation. Signature generated with the proposed approach is robust enough to detect unknown samples obfuscated with various code obfuscation techniques that are not detected by the updated well known Antivirus products. Suspicious samples detected by the approach were manually analyzed to verify their similarity with existing Android malware signatures. Six apps from third party market were verified as repackaged apps with malicious content. Proposed signature approach is robust against repackaged apps in comparison to the DroidMOSS Fuzzy hashing approach. Our results ascertain the need to have a strong monitoring and regulation to protect third party app markets. In future we would port the approach to constrained memory and develop a strong family signature to detect variants with family representative signatures.

9. REFERENCES

[1] Amazon Inc. Amazon Appstore for Android. http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011, Online; Accessed June 27 2013.

[2] G. Andre and P. Ramos. BOXER SMS trojan. Technical report, ESET Latin American Lab, 2013.

[3] APKTool. Reverse Engineering with apktool. https://code.google.com/android/apk-tool, Online; Accessed March 20 2013.

[4] BakSmali. Reverse Engineering with smali/baksmali. https://code.google.com/smali, Online; Accessed March 20 2013.

[5] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for Empirical analysis of Permission-Based Security models and its application to Android. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[6] BlackHat. Reverse Engineering with androguard. https://code.google.com/androguard, Online; Accessed March 29 2013.

[7] C. A. Castillo. Android Malware Past, Present, and Future. Technical report, MSWC, 2012.

[8] Contagiodump. Contagio Malware Dump. http://contagiodump.blogspot.in/, Online; Accessed March 1 2013.

[9] Dex2Jar. Adroid Decompiling with Dex2jar. http://code.google.com/p/dex2jar/, Online;Accessed May 15 2013.

[10] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application Security. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[11] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[12] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, Feb. 2012.

[13] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. RiskRanker: Scalable Accurate zero-day Android Malware detection. In *10th international conference on Mobile systems, applications, and services*, MobiSys, pages 281–294, NY, USA, 2012. ACM.

[14] A. Inc. Class to Dex Conversion with Dx. http://developer.android.com/tools/help/index.html, Online; Accessed March 5 2013.

[15] G. Inc. Android Smartphone Sales Report, 2013. http://www.gartner.com/newsroom/id/2482816, Online; Accessed June 17.

[16] JD-GUI. Android Decompiling with JD-GUI. http://java.decompiler.free.fr/?q=jdgui, Online; Accessed May 15 2013.

[17] Jon Oberhide. Dissecting The Android Bouncer. http://jon.oberheide.org/blog/2012/06/21/âĂŐ, Online; Accessed June 1 2012.

[18] J. Kornblum. Identifying Identical files using Context Triggered Piecewise Hashing. *Digital Investigation.*, 3:91–97, Sept. 2006.

[19] G. Play. Official Android Market, google play. https://market.android.com/., Online; Accessed June 17 2013.

[20] V. Roussev. Classprints: Class-aware similarity hashes. *Advances in Digital Forensics.*, 2008.

[21] V. Roussev. An evaluation of forensic similarity hashes. *Digital Investigations*, 8:S34–S41, Aug. 2011.

[22] V. Roussev. Data fingerprinting with similarity hashes. *Advances in Digital Forensics.*, 2011.

[23] L. Vijay, G. Manoj Singh, F. Parvez, and N. Smita. PEAL-Packed Executable AnaLysis. In *Proceedings of International Conference on Advanced Computing Networking and Security*, ADCONS '11. SPRINGER, 2011.

[24] E. William, O. Damien, M. Patrick, and C. Swarat. A study of android application security. In *USENIX Security '11*, San Francisco, ca, 2011. USENIX.

[25] E. William, G. Peter, C. Byunggon, and C. Landon. TaintDroid : An information flow tracking system for realtime privacy monitoring on smartphones. In *USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2011.

[26] Z. Yajin and J. Xuxian. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, Oakland 2012. IEEE, 2012.

[27] M. Zheng, P. P. C. Lee, and J. C. S. Lui. ADAM: An Automatic and Extensible Platform to Stress Test Android Anti-virus Systems. In *DIMVA*, pages 82–101, 2012.

[28] M. Zheng, M. Sun, and J. C. S. Lui. DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware. *CoRR*, 2013.

[29] W. Zhou, Y. Zhou, and X. Jiang. Hey, you get off my market: Detecting malicious apps in official and third party android markets. In *Annual Network and Distributed Security Symposium*, New York, NY, USA, 2012. NDSS.

[30] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces. In *Proceedings of Second ACM conference on Data Application Security and Privacy*, CODASPY '12, pages 317–326, New York, USA, 2012.