SANER-45: Comparison by Tools

I. Comparison and Evaluation

We find that there are a wide variety of repackaging detection techniques and corresponding systems. In this situation, it will be significant to conduct a complete comparison for these systems or tools, which can help us pick a suitable tool/system for a specific purpose of interest. In this section, we will provide a comparison and evaluation summary of the repackaging detection techniques in the form of a taxonomy with different perspectives.

A. Overall Comparison of the Detection Techniques

Table I provides different systems how to filter the public libs. As we can see, most systems use the whitelist to filter the common libraries. There is a drawback using the whitelist to filter public libs,the whitlist may not includes all public whitelist. Besides, some public libraries may be obfuscated by some tools like DexGuard. We cannot find them by using package name.

A few tools try to use the clustering algorithm to find the public libs. Because the same libs have the same features,using machine learning based method they may cluster in a large scale. Based on this idea, Wukong [40] and PiggAPP [41] use clustering-based method to find the common libs. This paper [39] points out the ad libs have the some special APIs features or UI etc., using these features we can find the ad libs.ResDroid use pagerank algorithm to find the primary code.

B. Evaluation of clone Detection Techniques

Before conducting the actual experimental evaluation of different techniques, we give a high level comparison of different techniques. We first refer to corresponding literature and summarize a comparison experimental results form the literature based on different techniques.

Table II shows these three opcode-based systems evaluation Experiment from the literature. As can be seen from the Table II, we can find the sample size of these existing detecting systems is not very large, they all smaller than 100,000.DroidKin just use 8182 apps to evaluate their system. Opcode-based is an early detecting technique of repackaging detection. The latest detecting system appeared in 2014. We can find the accuracy of opcode-based is good but the false negative is high.

Table III shows the evaluation experimental result on AST-based technique. There are only one literature employs the AST-based technique. This system also came out very early in 2012. It analyzed 158,000 apps from Google Play and find about 29.4% of the test samples are more likely to be plagiarized. This system has good scalability and low false positive rate. It is resilient to method renaming obfuscation attack and random method insertion obfuscation.

Table IV shows the evaluation experimental result on token-based technique. According to this tavle, we can find the test set is obviously enlarged. The false positive rate is also very low.

Table V show the evaluation results on the three API-based systems. Their published time relatively are late. The precision and robustness is also good.

Table VI shows the system evaluation results from the literature. We also use these typical methods to evaluation the effectiveness of PDG-based approaches and give a detailed report. Compared with opcode-based method, PDG-based technique is more efficiency and have better scalability. The average sample size is larger than opcode-based method. We also can find from the Table VI, the false negatives decreased, the false positives are also very low.

Table VII shows the system evaluation results from the literature. We also use these typical methods to evaluation the effectiveness of CFG-based approaches and give a detailed report. We can find CFG-based method has low false positive rate and false negative rate. They also have high accuracy. Especially the 3D-CFG that employs the centriod algorithm, it can detect repackaged apps in a short time.

Table VIII provides evaluation results on view-based tools. Except the dynamic method [26], the scalability is not very perfect, other methods all have wonderful scalability. Especially the MassVet which can find the malware in 10 seconds on the scale of Google Play. There are about 1.6 million apps on Android official App Market. Based on these data, we can find view-based method has high precision and recall. The robustness is also extraordinary.

Table IX provides evaluation results on system call based tools.

Table X provides the traffic-based system evaluation result from the literature. This approach not depends on the code feature, it uses the HTTP traffic as the birthmark. Therefore, it can defend against the code obfuscation and encryption. It use the VPT metric to find the repackaged apps, which can dramatically reduce the searching time. This system has good scalability and robustness.

Table XI gives evaluation results on three Resources-based tools. These three literature were published in or after 2014. We can find all of them have great scalability and can defend against code obfuscation. Their performances are better than opcode-based method. The sample size in their respective experiment are very large. The FPR and FNR are also very low, they have good precision and robustness.

TABLE I: A brief comparison of different detection approaches how to filter the third-party libraries

Method	Tools
No mentioned	[1], [2], [3], [4], [5], [6] , [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]
Whitelist	[21], [22], [23], [24], [25], [26], [27], [28], [29], [11], [30], [31], [32], [33], [34], [35], [35], [36],[34],[37], [33]
pre-comparison callback	SimiDroid [38]
comparing the UI,publisher identities or API calls	[39]
clustering-based	[40], [41]
page rank	[42]

TABLE II: Opcode-based Systems Evaluation Experiment from literature

No.	**tool**	**year**	**App Markets**	**sample size**	**outcome**
1	DroidMOSS	2012	slideme(3108),freewarelovers(3188), eoemarket(8261),goapk(4334), softportal(2305),proandroid(1710), official Android Market(68187)	84,767	third-party market repackaging rates range from 5% to 13%; false negative rate of 10.7%
2	Juxtapp	2013	official Android Market(30,000 apps); third-party Chinese market Anzhi(28,159 apps); Contagio malware(72); a set of 95,000 Android apps from the official Android Market to evaluate the performance of Juxtapp	58,231 / 95,000	1)463 applications with reuse of vulnerable code 2) 34 known malware 3) pirated variants of apps
3	DroidKin	2014	Malware Genome Project Virus Total Virus Share Google Play	8182	Acc(99.6%-99.8%)

TABLE III: AST-based System Evaluation Experiment from literature

No.	**tool**	**year**	**App Markets**	**sample size**	**outcome**	
1	[43]	2012	Pesudo-Market	7,600	false positive rate	0.5%

TABLE IV: Token-based System Evaluation Experiment from literature

No.	**tool**	**year**	**App Markets**	**sample size**	**outcome**
1	Wukong [40]	2015	anzhi(14,047) eoemarket(40,1334) gfan(23,673) baidu(13,672) myapp(20,833)	105,299	false positive(0)

TABLE V: API-based Systems Evaluation Experiment from literature

No.	**tool**	**year**	**App Markets**	**sample size**	**outcome**			
1	MIGDroid	2014	Genome Malware samples	1260	detection rate	95.94%		
2	DR-Droid	2016	Malware Genome VirusShare database	7,542	false negative rate	0.35%		
					false positive rate	2.97%		
3	Kim Dayoung	2016	Google Play	350 app pairs	false negative rate	15%		
					false positive rate	15%		
4	PiggyApp [41]	2013	slideme(3108) freewarelovers(3188) eoemarket(8261) goapk(4334) softportal(2305) proandroid(1710) Official Android Market(68,187)	84,767	piggybacked detection rate			
					0.97% to 2.7%			
					false negative rate	5.2%		
5	RepDetector	2016	apps from [24]	1,000	μ	FP	FN	ACC
					0.7	9	0	99.1%
					0.75	6	0	99.4%
					0.8	0	8	99.2%
					0.85	0	19	98.1%

TABLE VI: PDG-based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome
1	DNADroid [31]	2012	official Android Market, Amazon appstore, Androidonline market, appchina, Eoemarket, freeware lovers market, Goapk, Handango, m360	75,000	false positive rate%(0.0%),at least 191 repackaged pairs in 9,400 pairs
2	AnDarwin [33]	2013	Google Play(224,108), SlideME(16,479), m360(15,248),Brothersoft(14,749), Android Online(10,381), Gfan(7,229), 1Mobile(9,777), Eoemarket(5,515), GoApk(3,243), Freeware Lovers(1,428), AndAppStrore(1301), SoftPortal(1017), Androidsoft(613), AppChina(404), ProAndroid(370), AndroidDownloadz(245), PocketGear(227)	265,359	false positive rate(3.72%)

TABLE VII: CFG-based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome	
1	3D-CFG[28]	2014	American Market (Pandaapp, SlideME) Chinese Market (Anzhi,Dangle) European Market(opera)	150,145	Accuracy in Method Level	
					false positive rate	(0.38%)
					false negative rate	(0.4%-0.7%)
					Accuracy in Detecting App Clones false positive rate (0.0%)	
2	DroidSim	2014	Genome Project appsapk	827 706 for variant detection 25 repackaged app pairs	false negative rate	0.00%
					false positive rate	0.83%
					detection rate	96.60%

TABLE VIII: View-based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome			
1	ViewDroid	2014	Google Play	10,311	false positive rate		false negative rate	
					129/573872		1.3%	
2	ResDroid	2014	10 app market(Google Play and 9 third-party markets)	169,352	clustering approach		NNS-based approach	
					FPR	FNR	FPR	FNR
					3%	0.0%	5.0%	1.0%
3	Massvet	2015	33 app markets over 400,000 from Google Play 596,437 from 28 China app stores, 61,866 from European stores, 27,047 from US stores	10,311	false positive rate		false positive rate	
					4.73% - 9.46%		1.0%	
4	DroidEagle	2015	Google Play(500),appchina(34,989) appfun(12,427),hiapk(5287) anzhi(18,736),android.d.cn(4064) jimi168(23723),Baidu(200),Huawei(300)	100,126	Market		Visual similarity	Malware
					Third-party Market		1159(1.6%)	10
					Cloud Storage(Baidu)		50(10.0%)	0
					Cloud Storage(Huawei)		89(17.8%)	15
5	[26]	2015	Google Play(105) Anruan(100) Appsapk(167) Pandaapp(12)	521	Market	# of repackaged apps	detection rate	
					Google Play	8	7.6%	
					Anruan	8	8%	
					Appsapk	7	4.8%	
					Pandaapp	12	7.1%	
6	[44]	2016	Google Play(14323) CoolAPK(5,666) 163 (24,069) 1mobile(24,173) mumayi (29,990) anzhi (36,202) slideMe (19,730) android.d.cn (4635)	158,449	Average analysis Time			
					7min			
					Analysis Time of Google Play Two weeks			
					false positive Rate 15%			
					false negative rate 15%			
7	RepDroid	2017	Wandoujia	98 app repackaged pairs(Set1)	FNR		0%	
			F-Droid	125 apps(set2)	FPR		0%	
					encrypted APPs		FPR	0%

TABLE IX: System call based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome	
1	SCSDroid	2013	Malware Genome	1156	ACC	96.97%
					FNR	2.04%
					FPR	2%
2	Zimin Lin [8]	2016	Malware Genome other markets	770 malware 1250 benign apps	ACC	96.3%
					Precision	96.8%
					Recall	95.2%
3	PICARD	2015	Google Play	8	[22]	

TABLE X: Traffic-based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome	
1	[27]	2015	Anzhi(1286) Hiapk(828) Gfan(715) AppChina(601) ZOL(31) Google Play(95)	7,619	Accuracy	97%
					false negative rate	3.85%
					repackaged detection rate	3.40%
					processing time	5538.293s(7619 apps)

TABLE XI: Resource-based Systems Evaluation Experiment from literature

No.	tool	year	App Markets	sample size	outcome			
1	ResDroid	2014	10 app market(Google Play and 9 third-party markets)	169,352	clustering approach		NNS-based approach	
					FPR	FNR	FPR	FNR
					3%	0.0%	5.0%	1.0%
2	FSquaDRA	2014	Google Play(13,223) androidbest(1,662), androidrawer(2,857) anruan(4232) androidlife(1678) appsapk(2679) pandaapp(14,143) slideMe(15,305)	55,779	market	# repackaged pairs	detecting rate	
					androidbest	10	713194(3.24%)	
					androiddrawer	14	6097437(16.14%)	
					androidlife	44	1143400(5.15%)	
					anruan	97	3347895(5.982%)	
					appsapk	86	2094716(5.91%)	
					Google Play	1301	8985401(10.27%)	
					pandaapp	381	10726743(5.73%)	
					SlideMe	579	9481029(4.68%)	
3	ImageStruct	2015	Android Drawer(985) Freeware Lovers(1438) eoemarket(3347) Anzhi(3009) Malware Genome(1246) Google Play(39,492)	49,517	can find more repackaged apps than Androguard 60s can handle 1409 images			

Table XII provides evaluation results on existing Hybird-based tools.

C. A high-level Comparison

In this part, we list some of parameters we use for comparing different tools/techniques:

- ***Precision***: Precision is the ration of correctly predicted positive samples to the total predicted samples. $Precision = \frac{TP}{TP+FP}$ The system should be should find repackaged apps with high precision. The system should be sound enough so that it can detect false positives as less as possible.
- ***Recall***:Recall is the ratio of correctly predicted positive samples to the all samples in the actual class. The system should be robust enough so that it can detect false negative as less as possible.
- ***Scalability***: The system should can handle numerous apps from different markets. The tool should be capable of identifying the similar code from large code bank with high efficiency.
- ***Robustness***: The system should can find different types of repackaging problem and can defend some attacks(code modification and obfuscation)
- ***Efficiency***: They system should find the repackaged apps in a reasonable time.

According to the above Tables which summarize the evaluation results about different kinds of repackaging tools, Table XIII provides a specific comparison of different detection tools in terms of the number of data set, scalability, whether them resilient to code obfuscation, robustness, and efficiency. Scalability want to show whether these systems can deal with large scale apps from various app markets. Robustness want to check whether these systems can detect different kinds of repackaged apps and defend against different attack models.

Table XIII provide a specific information about every detection tools. According to the Table XIII, we can summarize a high level comparison of different techniques.

The Table XV give a high level comparison of different detection techniques in terms of the precision,recall, scalability, robustness.

We can find that opcode-based method have to face billions of opcode problem. It is not good for scalability and cannot deal with code modification or obfuscation. The public obfuscator can easily change the syntactic structure of code which

TABLE XII: Hybird-based Systems Evaluation Experiment from literature

No.	**tool**	**year**	**App Markets**	**sample size**	**outcome**	
1	Massvet	2015	33 app markets over 400,000 from Google Play 596,437 from 28 China app stores, 61,866 from European stores, 27,047 from US stores	10,311	false positive rate	false positive rate
					4.73% - 9.46%	1.0%

TABLE XIII: A comparison of different detection tools

Techniques	Tool	citation	dataset	scalability	resilient to code obfuscation	Robustness	Efficiency
opcode-based	Juxtapp	[32]	58,231	medium (need to cloud computing)	NO	poor	poor
	DroidMOSS	[21]	84,767	poor	NO	poor	poor
		[14]	36	poor	NO	poor	poor
	DroidKin	[11]	8,182	poor	NO	poor	medium
AST-based	Rahul Potharaju	[43]	7,600	poor	Yes	good	medium
PDG-based	DNADroid	[31]	75,000	medium	Yes	good	medium
	Andarwin	[33]	265,359	good	Yes	good	good
CFG-based	3D-CFG	[28]	150,145	good	Yes	good	excellent
	Wukong	[40]	105,299	good	Yes	good	good
	DroidSim	[35]	827	poor	Yes	good	medium
	RepDetector	[6]	1000	poor	Yes	good	poor
UI-based	ViewDroid	[29]	10,311	medium	Yes	medium	medium
	DroidEagle	[25]	100,126	depend on the using situation	Yes	medium	medium
		[26]	521	poor	Yes	good	medium
		[44]	158,449	medium	Yes	good	medium+
	RepDroid	[5]	125	medium	Yes	good	medium
Resources-based	FSquaDRA	[34]	55,779	good	Yes	medium	good
		[30]	30,625	Good	Yes	good	good
	ImageStruct	[9]	49,517	good	Yes	medium	good
resource + code feature	ResDroid	[42]	169,352	good	Yes	good	medium
UI + code feature	Massvet	[24]	10,311	excellent	Yes	good	Excellent
method and class level	MIGDroid	[36]	1260	poor	?	medium	poor
	DRDroid	[45]	7,542	medium	Yes	medium	good(they just consider the core code)
	CLANDroid	[7]	14,450	good	Yes	medium	good
		[39]	600	poor	Yes	good	general
HTTP traffic		[27]	7,619	good	Yes	good	good
State Flow Chart symbolic execution	RepDetector	[6]	1,000	good	Yes	good	poor

can make these syntax-based methods fail.

To some degree,the precision of syntax-based method is high, and usually use the opcode as the birthmark and create the comparison feature by using hash algorithms(fuzzy hash, feature hash). These methods just can detect the lazy attack.

CFG-based, PDG-based and API-based techniques consider the semantic information, they can handle some code obfuscation. CFG-based and PDG-based methods are static analysis, some API-based techniques are dynamic analysis. Different techniques have different advantages and disadvantages. According to the using demand, we can try different tools.

View-based and Resources-based method fully consider the characteristics of Android apps. However, some of systems use the subgraph isomorphism algorithm to identify repackaged apps, the comparison algorithm is not good for scalability and time-consuming. We can get the UI feature and Resource feature by using dynamic analysis and static analysis. Hybrid method combines the method level information and UI structure information, which can handle most repackaging problems.

In order to give the readers more deep understanding about these system, we also list some characteristics about these tools. Table XVI mainly shows the purpose these tools want to solve and their novelty.

TABLE XIV: Higher level comparison of detection Techniques

Technique	Precision	Recall	Scalability	Robustness	Efficiency
Opcode-based	Depends on the comparison algorithms	Depends on the comparison algorithms	poor	poor	time-consuming, billions of opcode
PDG-based	High, can handle the structural and semantic similarity	Medium, cannot detect all repackaged apps	depends on the comparison algorithm. graph matching is costly	medium	medium
CFG-based	high, considers the structural and semantic info	Medium, cannot detect all repackaged apps	depends on the comparison algorithm. Graph matching is not good for scalability	medium	medium
API-based	high, consider the semantic information	medium, depend on the number of API they use and which API they choose to detect.	depend on the detecting method. Static analysis usually is better than dynamic analysis. Dynamic analysis cannot get the whole traces, while static analysis cannot defend against some code obfuscation	medium, depends on the techniques	good
view-based	high, consider the UI structure and code	Medium	low, subgraph isomorphism algorithm is costly	medium, cannot handle all repackaged apps	medium
resource-based	high	medium,depends on the technique, some just consider the UI info	high, the feature is good	high, can detect most repackaged apps	depends on the techniques they distill the features.

TABLE XV: Higher level comparison of detection Techniques

Technique	Precision	Recall	Scalability	Robustness	Efficiency
traffic-based	high, the false positive is low	high, the false negative is low	high, existing method use the VPT metric to find repackaged apps, the time complexity is reasonable	good,it can defend code obfuscation	high
UI + CFG	high	high, consider the UI info and code info	high	High, can handle most repackaged apps	High
State Flow Chart	high,the performance is better than UI-based and opcode-based method	high,the performance is better than UI-based and opcode-based method	medium, symbolic execution is not good for scalability	good, it can tolerate some code noise insertion and code obfuscation	poor, symbolic execution is time-consuming

REFERENCES

[1] T. Ke, D. D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged android malware with code-heterogeneity features," in *TDSC*, 2017.

[2] L. Li, T. F. Bissyandé, A. Bartel, J. Klein, and Y. L. Traon, "The multi-generation repackaging hypothesis," in *ICSE-C*, 2017.

[3] L. Li, L. DaoYuan, B. T. F., J. Klein, H. Cai, D. Lo, and Y. L. Traon, "On locating malicious code in piggybacked android apps," *Journal of Computer Science & Technology*, 2017.

[4] L. Li, L. Daoyuan, B. T. F., K. Jacques, C. Haipeng, L. David, and T. Y. Le, "Automatically locating malicious packages in piggybacked android apps," in *MobileSoft*, 2017.

[5] S. Yue., W. Feng., J. Ma, Y. Jiang, X. Tao, C. Xu, and J. Lu, "Repdroid: An automated tool for android application repackaging detection," in *ICPC*, 2017.

[6] Q. Guan, H. Huang, W. Luo, and S. Zhu, "Semantics-based repackaging detection for mobile apps," in *ESSoS*, 2016.

[7] M. Linares-Vasquez, A. Holtzhauer, and D. Poshyvanyk, "On automatically detecting similar android apps," in *ICPC*, 2016.

[8] Z. Lin, J. X. qi, Z. Shengzhi, and W. Chuankun, "Analyzing android repackaged malware by decoupling their event behaviors," in *IWSEC*, 2016.

[9] J. Sibei, C. Yao, Y. Lingyun, S. Purui, and F. Dengguo, "Erratum: A rapid and scalable method for android application repackaging detection," *Information Security Practice and Experience. Lecture Notes in Computer Science, vol 9065. Springer, Cham*, 2015.

[10] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *ICSME*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2015.7332476

[11] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Droidkin: Lightweight detection of android apps similarity," in *Proc. SecureComm*. Springer, 2014.

[12] H. Moon, J. YongSung, and K. JeongYeo, "Reducing the impact of repackaged app on android," in *ICTC*, 2014.

[13] I. J. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A large-scale empirical study on software reuse in mobile apps," *IEEE software*, vol. 31, no. 2, 2014.

[14] H. Shahriar and V. Clincy, "Detection of repackaged android malware." in *ICITST*, 2014.

[15] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi, "Adrob: examining the landscape and impact of android application plagiarism," in *Proc. ACM MobiSys*, 2013.

[16] ——, "Characterizing android application plagiarism and its impact on developers," in *MobiSys*, 2013.

[17] H. Huang, S. Zhu, P. Liu, and D. Wu, "A framework for evaluating mobile app repackaging detection algorithms," in *Proc. TRUST*, 2013.

[18] Y.-D. Lin, Y.-C. Lai, C.-H. Chen, and H.-C. Tsai, "Identifying android malicious repackaged applications by thread-grained system call sequences," *Comput. Secur.*, vol. 39, Nov. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.cose.2013.08.010

[19] A. Desnos, "Android: Static analysis using similarity distance," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, 2012.

TABLE XVI: Higher level comparison of detection Techniques

System	Citation	Language	Approach	Background	Purpose	Novelty
DroidMOSS	[21]	C	opcode-based	Academic	Repackaged apps detection	fuzzy hashing to speed up detection
	[20]		AST-based	Academic	Plagiarizing detection	can defend two-level obfuscations
Androguard	[19]	python	NCD	Academic and Industry	static analysis(including similar detection)malware detection,extract injected malware	can identify the similarity and differences of apps
DNADroid	[31]	Java	PDG-based	Academic	detect clone apps	a early research by using PDG and Vf2 to identify the repackaged apps
AppInk	[46]	Java and shell script	customized DVM	Academic and Industry	generate watermark	can defend against app repackaging attacks, distortive,subtractive and additive attacks
PiggyApp	[41]		metirc-based PDG and VPT and NNS	Academic	piggybacked apps detection	using PDG to filter the non-primary parts
SCSdroid	[18]		thread-grained system call	Academic	repackaged apps detection	without needing the original apps
Juxtapp	[32]	6400 lines of C++ 1600 lines of Java 600 lines of script	opcode-based	Academic	repackaged apps detection	Feature hashing
Andarwin	[33]		PDG-based	Academic	finding clone apps or ”rebranded” apps and new malware and malicious apps	LSH clustering
FSquaDRA	[34]		resources-based	Academic	repackaged apps detection	Using the resource files to identify the repackaged apps
MIGDroid	[36]		method-invocation graph	Academic	repackaging malware	
DroidSim	[35]		CFG-based	Academic	repackaged apps detection	using the component-based control flow graph to detect repackaged apps
ResDroid	[42]	Python,Java and C	Resources-based,UI-based	Academic	repackaged apps detection	can handle hardened apps
	[14]		opcode-based	Academic	repackaged malware detection	Using the Kullback-Leibler Divergence metric
Droidmarking	[47]		dynamic software watermarking techniques	Academic	impeding Android apps repackaging, real-time app repackaging detection	non-stealthy, enable normal users to recover and verify the watermark copyright
ViewDroid	[29]	python	UI-based	Academic	repackaged app detection	the first system use UI to find repackaged apps

Continued on Next Page...

[20] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, “Plagiarizing smartphone applications: attack strategies and defense techniques,” in *ESSoS*, 2012.

[21] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “Detecting repackaged smartphone applications in third-party android marketplaces,” in *Proc. ACM CODASPY*, 2012.

[22] A. Aldini, F. Martinelli, A. Saracino, and D. Sgandurra, “Detection of repackaged mobile applications through a collaborative approach,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 11, 2015.

[23] J. Chen, M. H. Alalfi, T. R. Dean, and Y. Zou, “Detecting android malware using clone detection,” *Journal of Computer Science and Technology*, vol. 30, no. 5, 2015.

[24] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, “Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale,” in *Proc. USENIX*, 2015.

[25] M. Sun, M. Li, and J. C. S. Lui, “Droideagle: Seamless detection of visually similar android apps,” in *Wisec*, 2015.

[26] C. Soh, H. B. K. Tan, Y. L. Arnatovich, and L. Wang, “Detecting clones in android applications through analyzing user interfaces,” in *proc. ICPC*, 2015.

[27] X. Wu, D. Zhang, X. Su, and W. Li, “Detect repackaged android application based on http traffic similarity,” *Sec. and Commun. Netw.*, vol. 8, no. 13, Sep. 2015. [Online]. Available: http://dx.doi.org/10.1002/sec.1170

[28] K. Chen, P. Liu, and Y. Zhang, “Achieving accuracy and scalability simultaneously in detecting application clones on android markets,” in *Proc. ICSE*, 2014.

[29] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, “Viewdroid: Towards obfuscation-resilient mobile application repackaging detection,” in *Proc.*

TABLE XVII: Table XVI-Continued from previous page

System	Citation	Language	Approach	Background	Purpose	Novelty
	[13]		hybird	Academic	software resue	can detect different types of reuse, such as, code reuse, framework reuse and class reuse.
	[12]			Academic	protect the Android app from repackaging	verify the security of apps
	[30]		text-based retrieval methods and content-based image retrieval methods	Academic	find camouflaged applications	based on the text similarity and image similarity
DroidKin	[11]		opcode-based	Academic	find similar apps	set the filter to reduce the comparison
3D-CFG	[28]	7000 lines of C++ code and 500 lines of python	CFG-based	Academic	repackaged app detection	using the centroid algorithm to find the repackaged apps
Wukong	[40]	6912 lines of C++, 3300 lines of python, 780 lines of shell script	CFG-based	Academic	repackaged app detection	introduce a automated clustering-based method to find public libraries
	[27]		HTTP traffic	Academic	repackaged app detection	using HTTP traffic to identify the similar APPs
	[26]		UI-based	Academic	repackaged app detection	Using runtime UI information to identify repackaged apps
DroidEagle	[25]		UI-based	Academic	repackaged app detection	can detect the apps from the apps market or on the smartphone
MassVet	[24]	C and python	hybrid mathod	Academic and industry	repackaged malware and zero-day malware detection	using the centroid algorithm to find the UI similarity and methods differences
ImageStruct	[9]		resources-based(image)	Academic	repackaged app detection	using the image resources to detect repackaged apps
	[48]		String Offset Order	Academic	repackaged app detection	using a novel feature - the String Offset Order
	[23]	Java	TXL-based	Academic	identify similar code	detect different granularities(classes,functions,blocks,statements)
PICARD	[22]	C	system call and execution traces	Academic	repackaged app detection	Using the execution path to identify repackaged apps
	[8]	Java and C	execution dependency graph	Academic	repackaged malware detection	Using dynamic approach to analyze and detect repackaged malware, get the behaviors and system call behaviors for native code
	[44]		dynamic UI-based	Academic	repackaged app detection	Using the visual impersonation to detect repackaged app, dynamic trigger the App and get the screenshot
CLANdroid	[7]		Using semantic anchors (APIs,permissions, intent ,sensors,identifier)	Academic	detect similar app	?
	[49]		code graph	Academic	Piggybacked app detection	automating the localization of malicious code snippets
	[39]	C	API-based	Academic	Detecting Plagiarized Mobile Apps	Using the dynamic API birthmark extraction
DRegion	[45]	Python	class dependency graph and method dependency graph	Academic	repackaged app detection	make use of the dependency of classes and methods, we can speed up the detection
RepDetector	[6]		comparing the input-output states (state flow graphs)	Academic	repackaged app detection	it a semantic-based approach to detect repackaged apps, using the state flow graph can defend against the code obfuscation
RepDroid	[5]		UI-based	Academic	repackaged app detection	it a dynamic repackaging detection tool
	[50]		UI-based	Academic	detecting mobile app spoofing attacks	introduce a new concept deception rate to identify the spoofing apps
HookRanker	[4]		package dependency graph	Academic	locate malicious packaged of piggybacked app	can automatically rank potentially malicious packages based on the behaviour triggered modes.

Continued on Next Page...

ACM WiSec, 2014.
[30] S. M. Kywe, Y. Li, R. H. Deng, and J. Hong, "Detecting camouflaged applications on mobile application markets," in *International Conference on Information Security and Cryptology*. Springer, 2014.
[31] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in *Porc. ESORICS*, 2012.
[32] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, "Juxtapp: a scalable system for detecting code reuse among android applications," in *Proc. DIMVA*, 2012.
[33] J. Crussell, C. Gibler, and H.Chen, "Andarwin: Scalable detection of semantically similar android applications," in *Proc. ESORICS*, 2013.
[34] Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, and E. Moser, "Fsquadra: fast detection of repackaged applications," in *IFIP DBSec*, 2014.
[35] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, "Detecting code reuse in android applications using component-based control flow graph," in *IFIP*, 2014.
[36] W. Hu, J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han, "Migdroid: Detecting app-repackaging android malware via method invocation graph," in *ICCCN*, 2014.
[37] J. Crussell, C. Gibler, and H. Chen, "Scalable semantics-based detection of similar android applications." in *ESORICS*, 2013.
[38] L. Li, B. T. F., and J. Klein, "Simidroid: Identifying and explaining similarities in android apps," in *TrustCom*, 2017.
[39] D. Kim, A. Gokhale, V. Ganapathy, and A. Srivastava, "Detecting plagiarized mobile apps using api birthmarks," *Automated Software Engg.*, no. 4, Dec. 2016. [Online]. Available: http://dx.doi.org/10.1007/s10515-015-0182-6
[40] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Wukong: A scalable and accurate two-phase approach to android app clone detection," in *Proc. ISSTA*, 2015.
[41] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou, "Fast, scalable detection of piggybacked mobile applications," in *Proc. ACM CODASPY*, 2013.
[42] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang, "Towards a scalable resource-driven approach for detecting repackaged android applications," in *Proc. ACSAC*, 2014.
[43] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, "Plagiarizing smartphone applications: Attack strategies and defense techniques," *Engineering Secure Software and Systems*, 2012.
[44] L. Malisa, K. Kostiainen, M. Och, and S. Capkun, "Mobile application impersonation detection using dynamic user interface extraction," in *ESORICS*, 2016.
[45] T. Ke, D. Yao, B. G. Ryder, and T. Gang, "Analysis of code heterogeneity for high-precision classification of repackaged malware," in *SPW*, 2016.
[46] W. Zhou, X. Zhang, and X. Jiang, "Appink: watermarking android apps for repackaging deterrence," in *Proc. ACM ASIACCS*, 2013.
[47] C. Ren, K. Chen, and P. Liu, "Droidmarking: Resilient software watermarking for impeding android application repackaging," in *Proc. of the 29th ASE*, 2014.
[48] H. Gonzalez, A. A. Kadir, N. Stakhanova, A. J. Alzahrani, and A. A. Ghorbani, "Exploring reverse engineering symptoms in android apps," in *Proc. EuroSec*, 2015.
[49] L. Li, L. Daoyuan, B. T. F., K. Jacques, C. Haipeng, L. David, and T. Y. Le, "Ungrafting malicious code from piggybacked android apps," Technical report, SnT, Tech. Rep., 2016.
[50] L. Malisa, K. Kostiainen, and S. Capkun, "Detecting mobile application spoofing attacks by leveraging user visual similarity perception," in *CODASPY*, ser. CODASPY '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3029806.3029819