STATIC ANALYSIS TECHNIQUES USED IN ANDROID APPLICATION SECURITY ANALYSIS

Presenter: Suzanna Schmeelk

Course: E6998 **Date:** December 1, 2014

Instructor: Professor Alfred Aho

PRESENTATION OUTLINE

1. Android Security Threats and Concerns
2. Under-the-hood of Android security static analysis:
   - Static Analysis Techniques used from an analysis of over forty research papers published between 2009-2014 by USENIX, ACM, IEEE and more.
   - Can we categorize all the research-based static analysis techniques?
3. Conclusions: Are there security research areas left wide-open?
4. References

ANDROID INFORMATION SECURITY CIA THREATS

- **Confidentiality** – Concealment of information and resources [Bishop, 2009]
  - Privilege Escalation [Zhou and Jiang, 2012]
  - Spyware [Zhou and Jiang, 2012] (Disclosure [Bishop, 2009])
    - Backdoors [Xu et al, 2013]
    - Remote Control [Zhou and Jiang, 2012]
  - Financial Charges (Premium #s) [Zhou and Jiang, 2012]
  - Snooping [Bishop, 2009]
    - Leaking Private Data
- **Integrity** – Origin of data & how well data is protected on current machine [Bishop, 2009]
  - Deception, Spoofing [Bishop, 2009]
  - Man-in-the-Middle
    - Financial Charges (Premium #s) [Zhou and Jiang, 2012]
- **Availability** – User information access without interference [Whitman and Mattord, 2010]
  - Botnets [Xu et al, 2013]
  - Usurpation [Bishop, 2009]
  - Purposeful delay [Bishop, 2009]

ARCHETYPE STATIC ANALYSIS TECHNIQUES

1. Entry Point Analysis – determining where a program may start (particularly challenging with Android because of the use of callbacks and Activities that are by user demand, not simply a main method.)
2. Reachability Analysis – tracks if it is possible to follow a path that lead between two locations.
3. Side-effects Analysis – tracks which parameters of a method might be affected by its execution.
4. Field-initialization Analysis – tracks if field/object has been initialized.
5. Cyclicity-Analysis – tracks if an assignment creates a cycle.
6. Path-Length Analysis – tracks the maximum number of pointer dereferences that can be followed from a program variable.
7. Taint Analysis –
    1. Explicit – Passed in assignments
    2. Implicit – Passed in control flow structures (ICC, Broadcasts, Media)
8. Program slicing - the computation of the set of programs statements, the program slice, that may affect the values at some point of interest.

ARCHETYPE STATIC ANALYSIS TECHNIQUES

7. Pointer Analysis
    1. Alias – tracks which pointers access the same heap memory.
    2. Sharing – tracks potential variables that might ever be bound to overlapping data structures.
    3. Points-to - computes the objects that a pointer variable might refer to at runtime.
    4. Escape – tracks where in a program a pointer can be accessed.
    5. Shape – discovers and verifies properties of linked, dynamically allocated data structures. (captures aliasing, points-to and acyclicity information)
8. Data-Flow Analysis
    1. Context-Sensitive - an interprocedural analysis that considers the calling context when analyzing the target of a function call.
    2. Path-Sensitive - analysis computes different pieces of analysis information dependent on the predicates at conditional branch instructions.
    3. Flow-Sensitive - analysis takes into account the order of statements in a program.
    4. Inter-Procedural – Between procedures (Call Graphs)
    5. Intra-Procedural – Within a procedure

STATIC ANALYSIS PERSPECTIVES

Malware is generally defined as software designed to damage or do other unwanted actions on a computer system.

1. Developer perspective at compile time to guard against security violations
2. Mobile-Sandbox (e.g. behind-the-scenes at Google's Marketplace, designated companies)
3. Research Institutions that are more than a sandbox and actually repackage your application according to suggestions (e.g. Chiromancer tool at Delhi, A5 at Carnegie Mellon University, Anadroid at University of Utah, DAI-Labor in Germany)
4. Building tools to search and destroy (e.g. DARPA's APAC Challenge).

ANATOMY OF AN ANDROID APP [46]

- **Componen**t – App components are the essential building blocks of an Android app. Each component (there are four types) is a different point through which the system can enter your app.
  - **Activities** - An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent
  - **Services** - A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. t of the others.
  - **Content providers** - A content provider manages a shared set of app data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access.
  - **Broadcast receivers** - A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

ANATOMY OF AN ANDROID APP: ACTIVATING COMPONENTS [46]

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or another.
- An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component—an intent can be either explicit or implicit, respectively.
- There are separate methods for activating each type of component. E.g.:
  - You can start an activity (or give it something new to do) by passing an Intent to startActivity() or startActivityForResult() (when you want the activity to return a result).
  - You can start a service (or give new instructions to an ongoing service) by passing an Intent to startService(). Or you can bind to the service by passing an Intent to bindService().
  - You can initiate a broadcast by passing an Intent to methods like sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
  - You can perform a query to a content provider by calling query() on a ContentResolver.

ANATOMY OF AN ANDROID APP EXAMPLE [46]

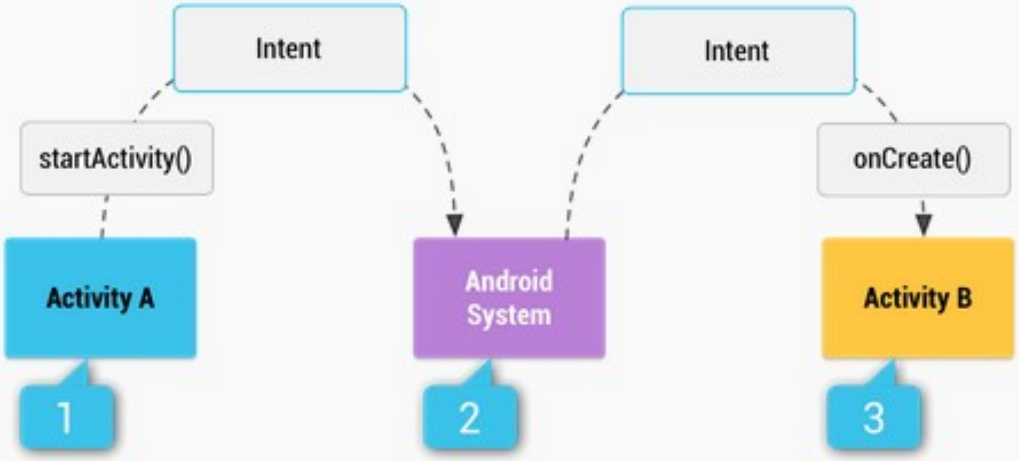


Figure 1. Illustration of how an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an `Intent` with an action description and passes it to `startActivity()`. **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its `onCreate()` method and passing it the `Intent`.

CONFIDENTIALITY ATTACK MITIGATION

STATIC ANALYSIS FOR CONFIDENTIALITY THREATS

- At lest 60% papers consider Permissions and Information Leaking issue.
- Most use some form of Data Flow Analysis with Taint Analysis
  - Sources: Location Data, Unique IDs, Call State, Authentication Data and Contact/Calendar Data
  - Sinks: SMS Communications, File Output, Network Communication, Intents, Content Resolver
  - Examples: TaintDroid (2010), AndroidLeaks (2012)
- From SandBox Perspective:
  - MobileSandbox (2013):
    - Extract App and create a Hash, submit sample to VirusTotal for a reading
    - Obfuscation typically hinders static analysis from a SandBox perspective (Mobile-Sandbox, 2013).
  - Apposcopy (2014) expands taint analysis to inter-component call graph and responds well to obfuscation.
    - Application matches signature based on control-flow (e.g. Launch a broadcast) and data-flow (e.g. reads private data and sends through designated sink).
  - Ways for malware to get around dynamic analyses (IMEI and other variables not set). [23]

STATIC ANALYSIS FOR CONFIDENTIALITY THREATS

- AppProfiler (2013)
  - Data Flow Analysis - technique for gathering information about the possible set of values calculated at various points in a program.
  - Derive rules from examining a few apps
  - Decompiled apps with DEX
  - Use Fortify Static Code Analyzer to perform analysis of the above rules over all apps (33K)
- Amandroid (2014)
  - Flow-sensitive, context-sensitive data flow analysis to calculate object points-to information with building inter-procedural control flow graph (ICFG)
  - Inter-component data flow graph (IDFG)
- Information Flows as Permission Mechanism [6] (2014)
  - Information flows, entry point discovery as it crawled API and discovered 1.738K callback methods that can serve as entry points.
  - Taint analysis, neither path or context sensitive
  - classical forward, backward, intraprocedural and interprocedural based on reachability

INTEGRITY ATTACK MITIGATION

STATIC ANALYSIS FOR INTEGRITY ATTACKS

- DroidRay (2014):
  - Pre-installed apps have more permissions than apps a user downloads.
  - Analyzed 250 Android firmware strands and 24K pre-installed apps.
  - Control-Flow Analysis for Privilege Escalation looking for malicious apps.
  - Examined shared UID from AndroidManifest.xml
- CHEX (2012):
  - *Vetting* Apps for component hijacking vulnerabilities using dataflow analysis
  - Low-overhead reachability tests on customized dependency graphs.
  - Introduced concept of App Splitting to model asynchronous execution
  - Flow and context sensitive
  - Based on Dalysis comprise of DexLib for byte code parsing, IBMs WALA for IR

AVAILABILITY ATTACK MITIGATION

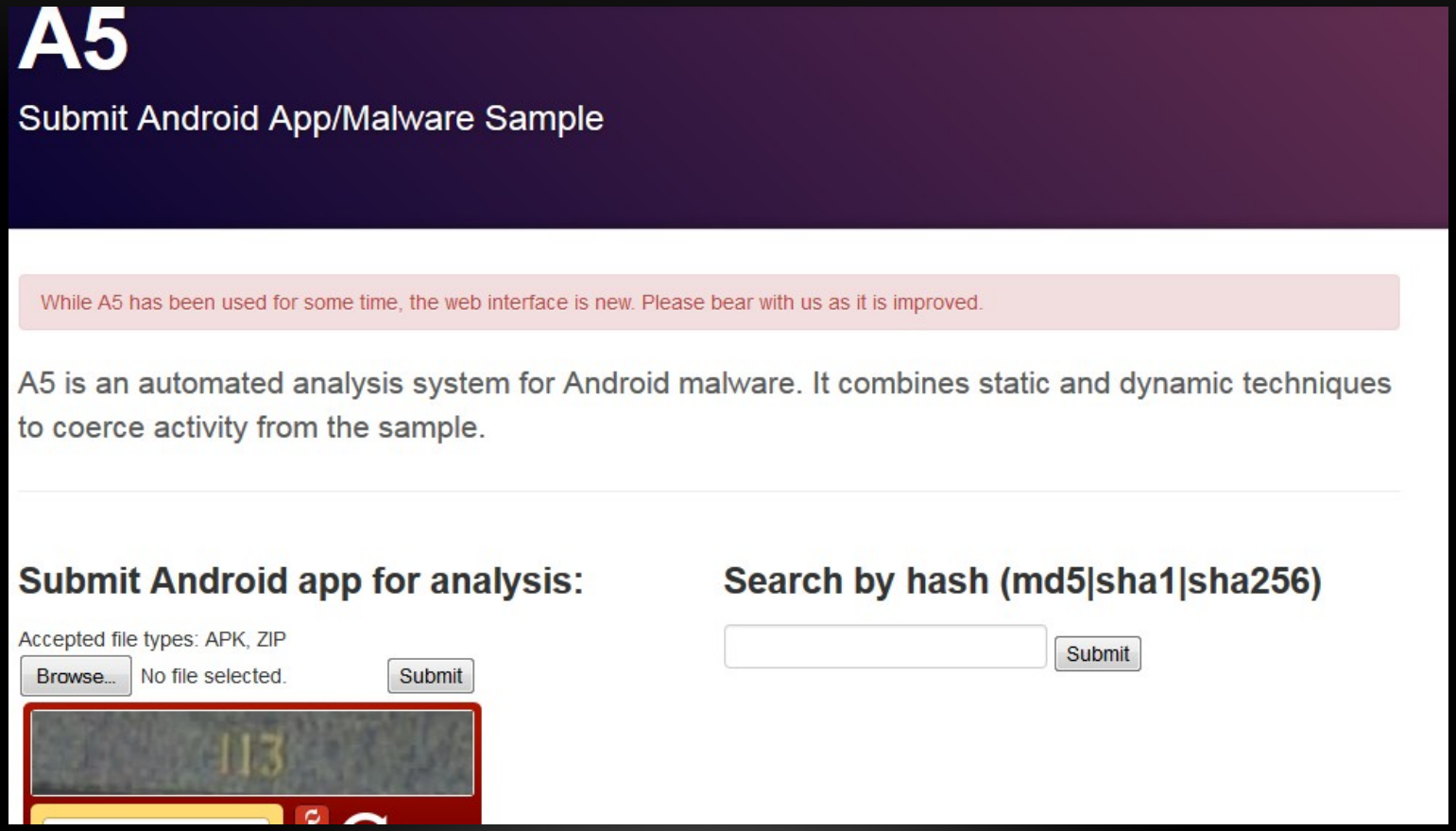
STATIC ANALYSIS FOR AVAILABILITY THREATS

- AsDroid (2014)
  - Check if UI mismatches with Program Function Calls and Behavior
    - Static Analysis to find function calls of interest and associate behavior
    - Text Analysis to analyze intent described by the corresponding interface artifacts
  - Reachability Analysis
    - Build a Control Flow Graph to propagate intents from the API to top-level functions
    - Based on IBM's WALA, CFG, SSA and May-Points-to-analysis
- SmartDroid (2012)
  - Tries to elicit behavior by using Static analysis to find behavior and, then, Dynamic Analysis to check behavior
  - Function Call Graph (FCG)
  - Activity Call Graph (ACG)
  - Iterative data-flow algorithm until reaches fixed state

STATIC ANALYSIS FOR AVAILABILITY THREATS

- SandBox Approach
  - A5 from CMU (2014) – Automated Analysis of Adversarial Android Applications Framework
    - Client -> CMU Server -> A5 -> Returns analysis results
    - Androguard to extract information from Manifest file
    - DED Decompiler to create Java Classes
    - SOOT – IR in Jimple to build CFG
    - Components of CFG flush-out as AST to examine each element of a statement
  - Chiromancer (2014) from New Delhi
    - Client -> Delhi Server -> Examines -> Gives user options -> changes App -> resigns App -> Returns to client as new APK file
    - Optimizations include skipping adware, removing SMS to premium numbers, removed GPS update frequency & removing a too long wake lock.
    - Transforms to Jimple via SOOT then does a data flow analysis

A5 SANDBOX: HTTP://DOGO.ECE.CMU.EDU/A5/



SECURITY-BASED MODEL CHECKING

- Lu and Mukhopadhyay Framework (2012):
  - Verify the permissions of the Android APIs invoked in the Java source code based on the Manifest.xml. The results are used to modify the models of the APIs.
  - Generate abstract collecting semantics constraints form the Java source code.
  - Import models of un-interpreted methods and objects as assertions into the already generated constraints. This may need interaction with a programmer for annotations.
  - Generate an analyzer by adding appropriate analysis 'aspect' constraints.
    - Security properties are represented as constraints
  - Analyze by solving security constraints
    - Using Yices Satisfiability Modulo Theories (SMT) Libraries
    - Model checking involves finding a counterexample – a model instance that violates a particular assertion.
- Gives three example:
  - Found that an expectant array passed back from function could contain the wrong values
  - Found a hardcoded password
  - Found more permissions than required.

BEST DEVELOPER PRACTICE TECHNIQUES DETECTABLE DURING A STATIC ANALYSIS

- Daswani et al, 2007:
  - Specifying Error Handling Requirements
    - Bad error handling
    - Nokia's Ping of Death
  - Including validation and fraud checks
    - Check length of file, etc.
  - Develop Security Requirements
    - Access control (don't hard code passwords)
    - Logs (take care in what is logged)
    - Confidentiality (encryption)

IS THERE ANY OPEN TERRITORY?

- Broad Cast receiving functionality is talked about in literature but not fully described.
  - Apps could call suspicious activities upon receiving notification
  - Botnets
- Covert Channels have not been addressed at all in literature
  - Covert channels are always labeled outside the scope of research
  - Static analysis for certain patterns might prove useful since SD Card and log activity is somewhat shared between Apps.
- Lint a relatively recent addition to the Android Development Kit
  - What is the current state?
- Static analysis for techniques to attack availability have not been done
  - Causing smartphone to vibrate until battery dead
  - Causing application to dereference a null and terminate
  - Alarm manager which sends intents even when App is paused

REFERENCES #1

1. AsDroid: Detecting Stealthy Behaviors in Android Applications by User Interface and Program Behavior Contradiction (2014) Huang, Jianjun and Zhang, Xiangyu and Tan, Lin and Wang, Peng and Liang, Bin. Proceedings of the 36th International Conference on Software Engineering.
2. Mobile-sandbox: Having a Deeper Look into Android Applications (2013) Spreitzenbarth, Michael and Freiling, Felix and Echtler, Florian and Schreck, Thomas and Hoffmann, Johannes. Proceedings of the 28th Annual ACM Symposium on Applied Computing.
3. A Framework for Static Detection of Privacy Leaks in Android Applications. (2012). Mann, Christopher and Starostin, Artem. Proceedings of the 27th Annual ACM Symposium on Applied Computing.
4. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. (2014). Wei, Fengguo and Roy, Sankardas and Ou, Xinming and Robby. Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.
5. Apposcopy: Semantics-based Detection of Android Malware Through Static Analysis. (2014). Feng, Yu and Anand, Saswat and Dillig, Isil and Aiken, Alex. Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering.
6. Information Flows As a Permission Mechanism. (2014). Shen, Feng and Vishnubhotla, Namita and Todarka, Chirag and Arora, Mohit and Dhandapani, Babu and Lehner, Eric John and Ko, Steven Y. and Ziarek, Lukasz. Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering.
7. DroidRay: A Security Evaluation System for Customized Android Firmwares. (2014). Zheng, Min and Sun, Mingshen and Lui, John C.S. Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security.

REFERENCES #2

8. DroidAlarm: An All-sided Static Analysis Tool for Android Privilege-escalation Malware. (2013). Zhongyang, Yibing and Xin, Zhi and Mao, Bing and Xie, Li. Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security.
9. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. (2014). Arzt, Steven and Rasthofer, Siegfried and Fritz, Christian and Bodden, Eric and Bartel, Alexandre and Klein, Jacques and Le Traon, Yves and Octeau, Damien and McDaniel, Patrick. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation.
10. CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. (2012). Lu, Long and Li, Zhichun and Wu, Zhenyu and Lee, Wenke and Jiang, Guofei. Proceedings of the 2012 ACM Conference on Computer and Communications Security.
11. AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users. (2013). Rosen, Sanae and Qian, Zhiyun and Mao, Z. Morely. Proceedings of the Third ACM Conference on Data and Application Security and Privacy.
12. PScout: Analyzing the Android Permission Specification. (2012). Au, Kathy Wain Yee and Zhou, Yi Fan and Huang, Zhen and Lie, David. Proceedings of the 2012 ACM Conference on Computer and Communications Security.
13. Static Reference Analysis for GUI Objects in Android Software. (2014). Rountev, Atanas and Yan, Dacong. Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization.
14. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications. (2012). Zheng, Cong and Zhu, Shixiong and Dai, Shuaifu and Gu, Guofei and Gong, Xiaorui and Han, Xinhui and Zou, Wei. Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices.
15. Chiromancer: A Tool for Boosting Android Application Performance. (2014). Anwer, Samit and Aggarwal, Aniya and Purandare, Rahul and Naik, Vinayak. MOBILESoft 2014.

REFERENCES #3

16. A5: Automated Analysis of Adversarial Android Applications. 2014. Vidas, Timothy and Tan, Jiaqi and Nahata, Jay and Tan, Chaur Lih and Christin, Nicolas and Tague, Patrick. Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices.
17. Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soott. (2012). Bartel, Alexandre and Klein, Jacques and Le Traon, Yves and Monperrus, Martin. Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis.
18. Sound and Precise Malware Analysis for Android via Pushdown Reachability and Entry-point Saturation. (2013). Liang, Shuying and Keep, Andrew W. and Might, Matthew and Lyde, Steven and Gilray, Thomas and Aldous, Petey and Van Horn, David. Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices.
19. Analysis of Android Applications' Permissions. (2012). Johnson, R. and Zhaohui Wang and Gagnon, C. and Stavrou, A. Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on.
20. STAAF: Scaling Android Application Analysis with a Modular Framework. (2012). Smith, R.W. and Pridgen, A. System Science (HICSS), 2012 45th Hawaii International Conference on.
21. A Study of Android Application Security. (2011). Enck, William and Octeau, Damien and McDaniel, Patrick and Chaudhuri, Swarat. Proceedings of the 20th USENIX Conference on Security.
22. Rage Against the Virtual Machine: Hindering Dynamic Analysis of Android Malware. (2014). Petsas, Thanasis and Voyatzis, Giannis and Athanasopoulos, Elias and Polychronakis, Michalis and Ioannidis, Sotiris. Proceedings of the Seventh European Workshop on System Security.
23. Detecting Control Flow in Smarphones: Combining Static and Dynamic Analyses. (2012). Graa, Mariem and Cuppens-Boulahia, Nora and Cuppens, F. and Cavalli, Ana. Proceedings of the 4th International Conference on Cyberspace Safety and Security.
24. LeakMiner: Detect Information Leakage on Android with Static Taint Analysis. (2012). Zhemin Yang and Min Yang. Software Engineering (WCSE), 2012 Third World Congress on.

REFERENCES #4

26. Model-Based Static Source Code Analysis of Java Programs with Applications to Android Security. (2012). Zheng Lu and Mukhopadhyay, S. Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual.
27. Android Taint Flow Analysis for App Sets. (2014). Klieber, William and Flynn, Lori and Bhosale, Amar and Jia, Limin and Bauer, Lujo. Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis.
28. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. (2012). Gibler, Clint and Crussell, Jonathan and Erickson, Jeremy and Chen, Hao. Proceedings of the 5th International Conference on Trust and Trustworthy Computing.
29. A case study in open source software security and privacy: Android adware. (2012). Erturk, E. Internet Security (WorldCIS), 2012 World Congress on.
30. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. (2011). Batyuk, L. and Herpich, M. and Camtepe, S.A. and Raddatz, K. and Schmidt, A.-D. and Albayrak, S. Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on.
31. Static Analysis of Executables for Collaborative Malware Detection on Android. (2009). Schmidt, A.-D. and Bye, R. and Schmidt, H.-G. and Clausen, J. and Kiraz, O. and Yuksel, K.A. and Camtepe, S.A. and Albayrak, S. Communications, 2009. ICC '09. IEEE International Conference on.
32. Repeat of 24
33. Static Analysis for Extracting Permission Checks of a Large Scale Framework: The Challenges and Solutions for Analyzing Android. (2014). Bartel, A. and Klein, J. and Monperrus, M. and Le Traon, Y. Software Engineering, IEEE Transactions on.
34. Hybrid Static-Runtime Information Flow and Declassification Enforcement. (2013). Rocha, B.P.S. and Conti, M. and Etalle, S. and Crispo, B. Information Forensics and Security, IEEE Transactions on.
35. Exposing software security and availability risks for commercial mobile devices. (2013). Johnson, R. and Zhaohui Wang and Stavrou, A. and Voas, J. Reliability and Maintainability Symposium (RAMS), 2013 Proceedings - Annual

REFERENCES #5

36. MobSafe: cloud computing based forensic analysis for massive mobile applications using data mining. (2013). Xu, J. and Yu, Y. and Chen, Z. and Cao, B. and Dong, W. and Guo, Y. and Cao, J. Tsinghua Science and Technology.
37. An Operational Semantics for Android Activities. (2014). Payet, Etienne and Spoto, Fausto. Proceedings of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation.
38. Enabling BYOD Through Secure Meta-market. (2014). Armando, Alessandro and Costa, Gabriele and Merlo, Alessio and Verderame, Luca. Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks.
39. On the Effectiveness of API-level Access Control Using Bytecode Rewriting in Android. (2013). Hao, Hao and Singh, Vicky and Du, Wenliang. Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security.
40. Reachability Analysis of Program Variables. (2013). Nikolic, D. and Spoto, F. ACM Trans. Program. Lang. Syst.
41. Dissecting Android Malware: Characterization and Evolution. (2012). Zhou, Yajin and Jiang, Xuxian. Proceedings of the 2012 IEEE Symposium on Security and Privacy.
42. Android Security Attacks and Defenses. (2013). Dubey, A. Misra. Anmol. CRC Press.
43. Introduction to Computer Security. (2004). Bishop, M. Addison-Wesley Professional
44. Roadmap to Information Security: For IT and Infosec Managers. (2011). Whitman, Michael E. and Mattord, Herbert J. Delmar Learning.
45. Formal Modeling and Reasoning about the Android Security Framework. (n.d.) Armando, Costa and Merlo.
46. http://developer.android.com/guide/components/fundamentals.html