Static Analysis of Executables for Collaborative Malware Detection on Android

Aubrey-Derrick Schmidt*, Rainer Bye*, Hans-Gunther Schmidt*, Jan Clausen*, Osman Kiraz †, Kamer Ali Yüksel†, Seyit Ahmet Camtepe*, and Sahin Albayrak*
* Technische Universität Berlin - DAI-Labor, email: {aubrey.schmidt, rainer.bye, hans-gunther.schmidt, jan.clausen, ahmet.camtepe, sahin.albayrak}@dai-labor.de
†Sabanci University, Istanbul, email: {osmankiraz, kamer}@su.sabanciuniv.edu

***Abstract*—Smartphones are getting increasingly popular and several malwares appeared targeting these devices. General countermeasures to smartphone malwares are currently limited to signature-based antivirus scanners which efficiently detect *known* malwares, but they have serious shortcomings with *new* and *unknown* malwares creating a window of opportunity for attackers. As smartphones become host for sensitive data and applications, extended malware detection mechanisms are necessary complying with the resource constraints.**

The contribution of this paper is twofold. First, we perform static analysis on the executables to extract their function calls in Android environment using the command readelf. Function call lists are compared with malware executables for classifying them with PART, Prism and Nearest Neighbor Algorithms. Second, we present a collaborative malware detection approach to extend these results. Corresponding simulation results are presented.

I. Introduction

Malwares (e.g. virus, worms and Trojan horses) have been threats to computer systems for many years and it was only a question of time when the first malicious software writers would get interested in increasingly popular mobile platforms, such as Symbian OS. In 2004, the first articles about malware for smartphones [1], [2] appeared saying that the next generation of targets are mobile devices. Since then, the number of malwares increased every month, and variants for various smartphone platforms appeared.

Commercially available countermeasures to smartphone malware suffer from weaknesses since they mostly rely on signatures. This approach leaves users exposed to *new* malware until the signature is available. Builygin [3] showed that in worst case a MMS worm targeting random phone book numbers can infect more than 700000 devices in about three hours. Additionally, Oberheide *et al.* [4] state that the average time required for a signature-based anti-virus engine to become capable of detecting new threats is 48 days. These numbers request extended security measures for smartphones as a malware can seriously damage an infected device within seconds. In this context, the new smartphone platform Android gained special interest among developers. Since it set open source, security tools can be developed even at kernel level. This allows comprehensive security mechanism to be deployed on Android handsets only being limited by the typical resource constraints of mobile devices.

Due to these constraints, we focus on static and light-weight mechanisms for detecting malware presence on Android devices. Our static approach for detecting malware allows us to use simple classifiers which are not very resource consuming and therefore fit very well to mobile needs. Previous approaches, e.g. [5]–[7], mostly rely on external servers for removing computational burden from the mobile device. In our case, the detection can benefit from a server but does not have to rely on it. Thus, for processing heavy-weight learning mechanism, we will benefit from the integration of an remote server.

In this work, we employ collaboration for security approach to extend our Malware detection results. Therefore, a set of entities is enabled to work on a common task without predefined roles in a heterarchical manner. The collaborative scheme is used to interact with other mobile devices in order to exchange detection data and system information. It can be considered as an operation mode whenever a mobile device is relying on the remote server but cannot access it.

The document is structured as follows: in Section II, related work is presented. Section III describes how the data for our approach is collected. Section IV presents our detection approach. Results are used for collaboration scenario in Section V-B. In Section VI we conclude.

II. Basics and Related Work

A. Smartphone Intrusion Detection

Several publications in the field of smartphone intrusion and malware detection have been made in the past years where no specific approach prevailed. Promising battery power-based mechanisms were introduced in [8]–[11] where these approaches depend on the quality and age of the battery. Anomaly and behavior-based approaches were introduced in [6], [7], [12]. These systems suffer from a high computational burden that is moved to an external server in most cases.

Different from these publications, the use of Android allows us to modify the system even at kernel-level. Therefore, up to our knowledge, this is the first time that a light-weight on-device function call analysis is investigated for smartphones.

B. Intrusion Detection by Static Signature Analysis

We present a method of static analysis of executables by disassembly. Essential characteristics like system and library functions are extracted and form the basis for identifying malware. Identification is done by machine learning classifiers. Static analysis of executables is a well explored technique. Zhang and Reeves [13] propose a static analysis to establish a similarity measure between two executables in order to identify metamorphic malware. Kruegel *et al.* describe static disassembly in [14]. Wang, Wu and Hsieh [15] present data mining methods to dicriminate between benign executables and viruses, whose dynamically linked libraries and application programming interfaces are statically extracted. They use support vector machines for feature extraction, training, and classification. Eskin *et al.* [16] apply machine learning methods on a data set of malicious executables.

C. Intrusion Detection in Ad-Hoc Networks

Ad-Hoc networks can be considered as the enabling technology for the realization of collaborative intrusion detection among Android devices. In that scope, new challenges arise from the inherent dynamic characteristics of these networks.

Zhang *et al.* [17] mention that intrusion detection in mobile computing environment may benefit from distributed and cooperative approaches. In this regard, they propose to use anomaly detection models constructed using information available from the routing protocols. Huang *et al.* [18] present a cluster-based detection approach for intrusion detection system and showed that they could maintain the same level of detection performance as an original per-node detection scheme with less host CPU utilization. Sterne *et al.* [19] propose a generalized, cooperative intrusion detection architecture with dynamic topology and *clusterheads*. These cluster-heads are determined according to valuable characteristics, e.g. distance, bandwidth etc. and they perform special tasks like aggregation and analysis of monitoring results. A general overview of intrusion detection in Ad-Hoc Networks is given in [20].

All these approaches target in special security concerns arising in Ad-Hoc networks, whereas our approach is striving for the opportunities Ad-Hoc networks offer. In this context, Bye *et al.* [21] present an overlay framework including an algorithm to find common groups and exchange security related data, e.g. monitoring results.

III. System and Function Call Analysis on Android

The overall system realizes a client-server architecture which can be seen on Figure 1. It basically provides three main functionalities: On-device analysis, Collaboration, and Remote analysis. The client gathers data for on-device, collaboration, or remote analysis. For improving detection, data can be exchanged between two mobile clients in a collaborative manner. This data can consist, e.g. of detection results or anomalous feature vectors. Whenever on-device detection is not feasible, the client can send data to the remote server. In turn, the server can send detection results back to the client. Additionally, it can send commands for reconfiguring the client.

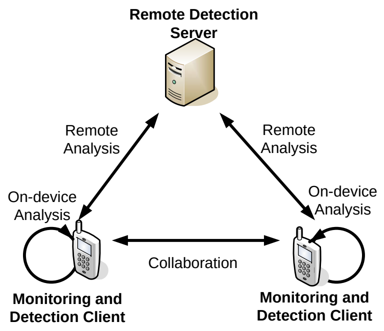

Fig. 1. Overall System Architecture

A. Data Extraction Architecture

The Android Java framework, as of time writing, only offers a restricted set of Java methods in order to access the underlying OS-level, e.g. it is not possible to get a list of all running system processes. In order to extract further information, a mediator is required that collects the desired data on OS-level and delivers it to an upper lying software stack. Responsible for this task is a self-written tool called *Interconnect Daemon*, a Linux server daemon which consists of several modules, e.g. system monitors. Additional module tasks are scanning the filesystem, creating hashes from important files, or waiting for operating system signals to indicate events.

The various modules work on top of Android's system binaries, mostly supported via *toolbox*, an all-in-one statically compiled binary. *Toolbox* offers a number of standard Linux system commands with a limited set of parameters. Additional tools were added: busybox (http://www.busybox.net/) supports a far greater number of Linux commands with appropriate parameters; strace (http://sourceforge.net/projects/strace/) offers debugging and system call tracing capabilities. Further descriptions can be found in [22].

B. Creating a Training Set With Readelf

For this paper, a specific module within the *Interconnect Daemon* was responsible for identifying and extracting all Linux system executables, to be precise, all ELF (Executable and Linking Format) object files (excluding shared libraries). These executables (mostly in */bin*) hold static information which can be read out with the appropriate reader, in our case *readelf*. The *readelf* command delivers detailed information on relocation and symbol tables of each ELF object file. Most interestingly, it outputs the static list of referenced function calls for each system command. The following example shows the first lines of the output of readelf running on a system command (/bin/ls):

```
Symbol table '.dynsym' contains 104 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000   622 FUNC    GLOBAL DEFAULT  UND abort@GLIBC_2.0 (2)
     2: 00000000    29 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.0 (2)
     3: 00000000    84 FUNC    GLOBAL DEFAULT  UND sigemptyset@GLIBC_2.0 (2)
     4: 00000000    52 FUNC    GLOBAL DEFAULT  UND sprintf@GLIBC_2.0 (2)
     5: 00000000   433 FUNC    GLOBAL DEFAULT  UND localeconv@GLIBC_2.2 (3)
     6: 00000000    10 FUNC    GLOBAL DEFAULT  UND dirfd@GLIBC_2.0 (2)
     7: 00000000    87 FUNC    GLOBAL DEFAULT  UND __cxa_atexit@GLIBC_2.1.3 (4)
    [...]
```

We identified a number of Linux system commands within Google Android (less than 100). After extracting those, inspecting them with *readelf*, and extracting the lists of function calls, this data formed our benign training set. In order to build a set of malicious training examples, we selected approximately 240 different malwares, found via Google Search, and extracted the static lists of function calls with the same method as described above. The malware set consisted of virus, worms, and Trojans specifically designed for Linux (not specifically designed for Android's ARM-architecture). A few malwares have been successfully compiled for ARM-architecture and compared with its i386-counterpart. The results showed only very minor differences leading us to the conclusion that using this set as preliminary malicious training set was a valid approach. The combination of both benign and malicious data set formed our final training set which has been used for further analysis.

IV. Classify Executables by Static Analysis

The executables (ELF) can be fairly well identified as normal and malicious respectively by looking only at the names of the functions and calls appearing at the output of *readelf*. In the sequel, we will call these names simply *attributes*, which are grouped in *relocation* and *dynamic* attributes due to their appearance at the readelf output. The *combined* attribute sets is a union of the relocation and dynamic attribute set. The set of attributes is further split: an attribute is in the set of *mutual attributes* if there is at least one malware ELF and at least one normal ELF whose readelf output contains it, whereas attribute is in the set of *all attributes* if it is contained in the readelf output for at least one ELF, no matter if malicious or normal. Eventually, six attribute classes are gained by the just mentioned discrimination, the sizes of which are presented in table I. An attribute class will be denoted by $\aleph$. The attribute class which is, for instance, both dynamic and mutual have the shape $\aleph = \{$ abort, _errno_location, sigemptyset, $\ldots\}$.

TABLE I
Sizes of the attribute classes

	relocation	dynamic	combined
mutual attributes	174	145	189
all attributes	1662	2284	2816

The question arises whether these attribute sets have the potential to distinguish normal from malicious executables. By applying several state-of-the-art classifiers, it turned out this is the case for most of them. The table below indicates accuracy parameters, i.e. correctly classified instances rate (CC), detection rate (DR), and false positive rate (FP), for each attribute set and each applied classifier due to our data set. To check the generalizing ability of the trained classifiers, stratified ten fold cross validation is used, where each fold is constructed randomly. The data mining package *weka* (http://www.cs.waikato.ac.nz/ml/weka/) served as test environment.

TABLE II
Accuracy values of classifiers according to attribute sets

	relocation			dynamic			combined		
	mutual attributes								
Accur.	CC	DR	FP	CC	DR	FP	CC	DR	FP
Prism	0.78	0.70	0.00	n.V.	n.V	n.V.	0.78	0.70	0.00
PART	0.94	0.99	0.15	0.97	1.00	0.12	0.97	1.00	0.12
n. Nb	0.92	0.98	0.21	0.90	0.92	0.13	0.96	0.98	0.11
	all attributes								
Prism	0.81	0.76	0.00	0.83	0.76	0.00	0.83	0.77	0.00
PART	0.95	1.00	0.16	0.97	1.00	0.12	0.97	1.00	0.12
nNb	0.94	0.99	0.12	0.96	0.99	0.10	0.96	0.99	0.10

Three classifiers of different kinds are applied to our data. The classifier *PART* extracts decision rules from the decision tree learner C4.5. [23]. *Prism* is a simple rule inducer which covers the whole set by pure rules [24]. Both take into account the interdependencies of attributes and are – once learned – efficient classifiers. The computational costs of learning could be shifted to a server, then mobile devices will be provided by rules. Prism produces in all cases we tested no false positive whereas it performs less well in detection malware, and a higher set of rules (from 10 to 30) are usually induced than with PART, which is satisfied with 2 to 12 rules. Our third classifier is the *nearest Neighbor* algorithm (nNb). We used the following light-weight version of this standard classifier: Let $\mathbb{M}$, $\mathbb{N}$ be the sets of malicious and normal ELFs respectively, and let $\aleph$ be an attribute set and ρ a metric on $\{0,1\}^{|\aleph|}$. An ELF is represented by $x = (x_i)_{i\in\aleph}$ where a component x_i is equal to 1 if this ELF has attribute i and equal to 0 if not. Here the simple metric

$$\rho(x,y) = \sum_{i\in\aleph} |x_i - y_i|, \qquad \text{for } x,y \in \{0,1\}^{|\aleph|}.$$

is applied. By $d(x,K) = \inf_{k\in K} \rho(x,k)$ the distance of x to a subset $K \subset \{0,1\}^{|\aleph|}$ is denoted. The classifier φ maps a formated readelf output of an ELF x to the state space $\{\text{malicious, normal}\}$,

$$\varphi(x) = \begin{cases} \text{malicious,} & \text{if } d(x,\mathbb{M}) < d(x,\mathbb{N}), \\ \text{normal,} & \text{else.} \end{cases} \tag{1}$$

The computational complexity of the detection by φ is of acceptable order, namely $O(|\aleph|\cdot(|\mathbb{M}|+|\mathbb{N}|))$. Nearest neighbor detection has the advantage that no server is required for training and that it behaved in our test stable w.r.t. thinning of the attribute set: if the most distinctive attributes due to our rules are omitted, the accuracy parameters of nNb do not vary significantly. A drawback is that the attributes of each single malicious or normal ELF has to be stored, which is acceptable in our case but might become inconvenient with a growing data basis.

The detection map in (1) makes binary decisions whereas it delivers no statement on the certainty of judgment. A simple solution is the strictly increasing function $\tilde{\varphi} : \{0,1\}^{|\aleph|} \to [0,1]$ which is derived from (1) by affine linear transformation, where an output of 0 means that we definitely deal with a normal ELF and 1 that the ELF is malicious with probability one. The values in between stand for the level of maliciousness. In

$$\tilde{\varphi}(x) = \begin{cases} \frac{1}{2} r(x), & \text{if } r(x) \in [0,1], \\ \frac{1/2 - 1/|\aleph|}{|\aleph|-1} r(x) + \frac{|\aleph|/2 + 1/|\aleph| - 1}{|\aleph|-1}, & \text{if } r(x) \in (1,\infty), \\ 1, & \text{if } d(x,\mathbb{M}) = 0. \end{cases}$$

the ratio of the distance to normal set and the distance to malware set is abbreviated by $r(x) = d(x,\mathbb{N})/d(x,\mathbb{M})$. If the output of $\tilde{\varphi}$ overcomes a *threshold* $\theta \in (0,1)$ it might be concluded that the ELF is malicious. For the threshold of $1/2$

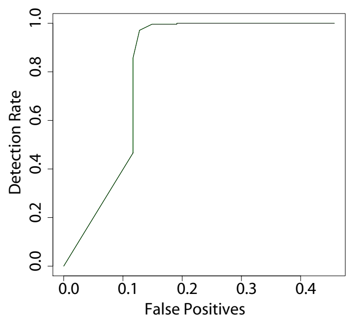

Fig. 2. ROC graph for nNb with varying threshold and detection function $\tilde{\varphi}$

$\tilde{\varphi}$ will lead to the same decision as φ. If a lower false positive rates is desired, increase threshold θ. Note that this will be accompanied by a worse detection rate, recall the ROC graph in Figure 2.

V. Collaborative Intrusion Detection

A. Approach

The collaboration module is triggered when a specific *event* takes place. Subsequently, communication is established with neighboring nodes for the assistance. A *request* takes place for support, e.g. computation or available information. Next, responses are collected and an action is taken after *evaluating* them. Figure 3 gives an illustrative example of the collaboration scheme in the context of the *Collaborative Malware Detection* scenario presented in the following.

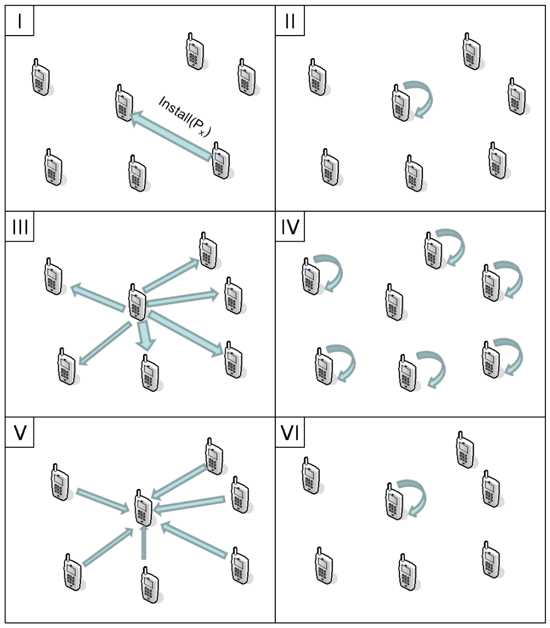

Fig. 3. **Collaborative Malware Detection:** *I- An infected node tries to install a malicious program to the target device. II- The detection status is determined. In the case, the status is within the uncertainty interval, following steps are triggered. III- A request is sent out to the neighboring nodes with a feature vector containing output of static ELF analysis of the program. IV- Each neighbor nodes determines the detection status of the application according to its trained classifier. V- The initiating node is informed about the results. VI- Evaluation of results; if joint status still falls in the uncertainty interval or below, the node become infected. Otherwise, it is removed from the set of susceptible nodes.*

Based on the approach presented in Section IV, we extend the on-device detection with a collaborative approach. We introduce an uncertainty interval $[\theta - x, \theta]$, triggering the collaboration mechanism. Hence, the neighboring nodes are requested to determine the detection status according to their classifier. The initiating nodes collects the responses and builds the arithmetic mean. If the average of the responses is still below θ, the executable is defined as benign, otherwise as malicious. We conducted simulations for this specific scenario.

B. Simulation

We set up a simulation environment reflecting the characteristics of the Ad-Hoc network scenario. 100 nodes are used in a simulated area of 1500 x 1500 units with transmission range of 200 units. A unit is an abstract term for a distance measure, e.g. meter. In each round, a node is able to communicate to one or several present nodes in his neighborhood determined by his transmission range. The conduction of the algorithm lasts four rounds: worm tries to infect, request for collaboration, response, evaluation. We performed 100 runs with 100 rounds per run. Nodes are mobile and move every round according to a random walk model with a maximum of (+/-)5 units in each dimension. The attack vector is based on a worm propagation, e.g. the Cabir worm (http://www.f-secure.com/v-descs/cabir.shtml). Initially, a device is selected randomly to be infected. Than, the worm tries to infect all devices in transmission range. We apply the aforementioned collaborative detection scheme. If a new device becomes infected, the worm propagates further. If the worm is classified as malicious by a device, this device is removed from the set of susceptible devices.

We define the threshold for detection as $\theta = 0.5$. If the return value is higher, the installed application is considered malicious and removed from the set of susceptible nodes. If the return value is lower, in case of non-collaborative scenario the node becomes infected. For the simulation, we have two varying input variables. On the one hand, the uncertainty interval $[\theta - x, \theta]$, where we use as x values from 0 to 0.5 in steps of 0.1.

The second input variable is the distribution function for the initial detection values. These are assigned according to normal distribution with a varying mean μ and a standard deviation σ of 1. All resulting values in the interval $[-2\mu$, $2\mu]$ are normalized to the interval [0, 1]. Afterwards, for each applied distribution the mean is shifted by continuously adding 0.1. In the case, a value becomes bigger than one, it is set to one.

C. Results and Discussion

The results of the simulation are depicted in Figure 4. The chart shows the resulting number of infected devices with respect to the initial detection value distribution and varying uncertainty intervals. The first observation is that an increasing uncertainty interval reduces the false negative rate. In other words, if the collaborative scheme is executed more frequently, this results into less infections. On the other hand, it can be seen that the higher the detection value is, the more the collaborative scheme becomes effective. In the first distribution ($\mu = 0.5$), the fraction of the "Interval 0.40 - 0.5" approach to the non collaborative approach is 80 per cent whereas with the third distribution ($\mu = 0.6$) this fraction decreases to 32 percent. The most costly combination in terms of communication is $\mu = 0.5$ and the uncertainty interval $x = 0.5$. Here, the averaged maximum of communication acts took place in round 2 with 0.4 per node. Although we focused on decreasing false negative rate, we assume false positives can be reduced by this approach similar as it is shown by Luther *et al.* in [25]. A collaborative scheme is susceptible to attacks. An extensive use of resources, e.g. to drain the battery, can be prevented by defining an explicit counter to serve only a maximum number of requests per time. In addition, related work given in Section II-C shows further mechanisms.

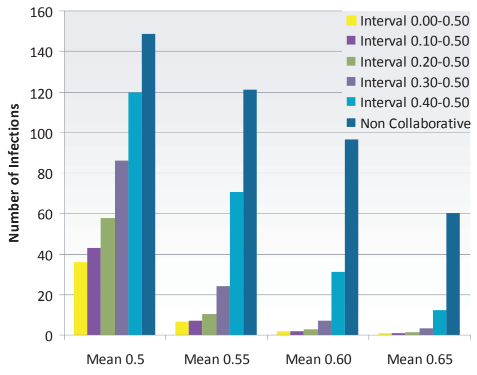

Fig. 4. Simulation results of collaborative scheme

VI. Conclusion

Using static ELF analysis turned out to be an efficient way to detect malware on Android using simple classifiers. These results can be improved when applying collaborative measures which can reduce the false-negative rate. Further investigations are needed in order to evaluate our findings using real Android hardware and malware, as soon as available. Real resource consumption will be a significant indicator whether this system can be extended to more complex tasks, e.g. adding more semantical information to the collaborative approach or using more complex classifiers.

References

[1] D. Dagon, T. Martin, and T. Starner, "Mobile phones as computing devices: The viruses are coming!" *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 11–15, 2004.

[2] M. Piercy, "Embedded devices next on the virus target list," *IEE Electronics Systems and Software*, vol. 2, pp. 42–43, Dec.-Jan. 2004.

[3] Y. Bulygin, "Epidemics of mobile worms," in *Proceedings of the 26th IEEE International Performance Computing and Communications Conference, IPCCC 2007, April 11-13, 2007, New Orleans, Louisiana, USA*. IEEE Computer Society, 2007, pp. 475–478.

[4] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, San Jose, CA, July 2008.

[5] D. Samfat and R. Molva, "IDAMN: An Intrusion Detection Architecture for Mobile Networks," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 7, pp. 1373–1380, Sep. 1997.

[6] A.-D. Schmidt, F. Peters, F. Lamour, and S. Albayrak, "Monitoring smartphones for anomaly detection," in *MOBILWARE 2008, International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, Innsbruck, Austria, 2008.

[7] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in *Proceeding of the 6th international conference on Mobile systems, applications, and services*. Breckenridge, CO, USA: ACM, 2008, pp. 225–238.

[8] D. C. Nash, T. L. Martin, D. S. Ha, and M. S. Hsiao, "Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices," in *PERCOMW '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 141–145.

[9] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2008, pp. 239–252.

[10] G. Jacoby and N. Davis, "Battery-based intrusion detection," in *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, vol. 4, 2004, pp. 2250–2255.

[11] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2008, p. 296.

[12] M. Miettinen, P. Halonen, and K. Hätönen, "Host-Based Intrusion Detection for Advanced Mobile Devices," in *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 2 (AINA'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 72–76.

[13] Q. Zhang and D. S. Reeves, "Metaaware: Identifying metamorphic malware," in *ACSAC*, 2007, pp. 411–420.

[14] F. V. Christopher Kruegel, William Robertson and G. Vigna, "Static disassembly of obfuscated binaries," *USENIX Security Symposium*, vol. Volume 13, pp. 18 – 18, 2004.

[15] C. H. T. Wang, C. Wu, "A virus prevention model based on static analysis and data mining methods," in *Computer and Information Technology Workshops*, 2008, pp. 288–293.

[16] S. J. S. Eleazar Eskin, Matthew G. Schultz and E. Zadok, "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[17] Y. Zhang, W. Lee, and Y.-A. Huang, "Intrusion detection techniques for mobile wireless networks," *Wireless Networks*, vol. 9, no. 5, pp. 545–556, 2003.

[18] Y. an Huang, "A cooperative intrusion detection system for ad hoc networks," in *In SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, 2003, pp. 135—147. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.8768

[19] D. Sterne, P. Balasubramanyam, D. Carman, B. Wilson, R. Talpade, C. Ko, R. Balupari, C.-Y. Tseng, and T. Bowen, "A general cooperative intrusion detection architecture for manets," in *Proceedings of the Third IEEE International Workshop on Information Assurance*, March 2005, pp. 57–70.

[20] J. W. Tiranuch Anantvalee, *Wireless Network Security*. Springer, 2007, ch. A Survey on Intrusion Detection in Mobile Ad Hoc Networks, pp. 159–180.

[21] R. Bye and S. Albayrak, "CIMD- Collaborative Intrusion and Malware Detection," Technische Universität Berlin - DAI-Labor, Tech. Rep. TUB-DAI 08/08-01, Aug. 2008, http://www.dai-labor.de.

[22] A.-D. Schmidt, R. Bye, H.-G. Schmidt, K. A. Yüksel, O. Kiraz, J. Clausen, K. Raddatz, A. Camtepe, and S. Albayrak, "Monitoring android for collaborative anomaly detection: A first architectural draft," Technische Universität Berlin - DAI-Labor, Tech. Rep. TUB-DAI 08/08-02, Aug. 2008, http://www.dai-labor.de.

[23] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Shavlik, J., ed., Machine Learning*, vol. Proceedings of the Fifteenth International Conference, 1998.

[24] J. Cendrowska, "Prism: An algorithm for inducing modular rules," *International Journal of Man-Machine Studies*, vol. 27, No. 4, pp. 349–370, 1987.

[25] K. Luther, R. Bye, T. Alpcan, S. Albayrak, and A. Müller, "A Cooperative AIS Framework for Intrusion Detection," in *Proceedings of the IEEE International Conference on Communications (ICC 2007)*, 2007.