On Leveraging Stochastic Models for Remote Attestation

Tamleek Ali, Mohammad Nauman and Xinwen Zhang

Institute of Management Sciences, Peshawar, Pakistan
Computer Science Research and Development Unit (CSRDU), Pakistan
Huawei Research Center, USA
tamleek@imsciences.edu.pk, nauman@csrdu.org, xinwen.zhang@huawei.com

Abstract. Remote attestation is an essential feature of Trusted Computing that allows a challenger to verify the trustworthiness of a target platform. Existing approaches towards remote attestation are largely static or too restrictive. In this paper, we present a new paradigm in remote attestation that leverages recent advancements in intrusion detection systems. This new approach allows the modeling of an application's behavior through stochastic models of machine learning. We present the idea of using sequences of system calls as a metric for our stochastic models to predict the trustworthiness of a target application. This new remote attestation technique enables detection of unknown and zero-day malware as opposed to the known-good and known-bad classification currently being used. We provide the details of challenges faced in the implementation of this new paradigm and present empirical evidence supporting the effectiveness of our approach.

1 Introduction

One of the main objectives of Trusted Computing (TC) is to create a system for the establishment of trust on a remote platform. This concept is termed as *remote attestation*. The most popular and well adopted technique for this purpose is the *Integrity Measurement Architecture* (IMA) [1]. It allows a target platform to report the hashes of all loaded executables to an authorized remote challenger. The challenger can verify that all of the loaded executables were in fact benign, by comparing the hashes against known-good ones. However, this technique does not cater to the problems posed by execution of the known-good code in a malicious way. Buffer overflow attacks and, in recent times, return-oriented programming [2] are clear examples of such issues. It has therefore been accepted widely [3–7] that dynamic behavior attestation should be investigated as the means of establishing trust on a platform.

In traditional security research, some of the most successful behavior measurement techniques revolve around the concept of sequences of system calls originating from an application [8–10]. In the context of remote attestation, measurement of behavior through system calls has been studied previously in

[3]. However, in our opinion, there are limitations to this approach that keep it from being deployed in real-world scenarios. For one, it requires the challenger to statically analyze the source code of the target application (or a disassembly of the executable if the source is unavailable). This is a complex process and might lead to a resistance from the mainstream security audience. The magnitude of this problem increases significantly when we consider the large number of applications that typically execute on a platform and interact with each other in intricate patterns. Secondly, this technique requires the reporting of all system calls to the challenger – a rather tall order considering the huge number of calls made by a typical application in a short span of time [11].

We argue that the problem of deciding upon the trustworthiness of an application has been thoroughly studied in related fields such as intrusion detection systems (IDS). Some of the techniques of IDS seem to be useful for remote attestation. However, due to the differences in their protection goals, directly applying intrusion detection techniques for remote attestation is not viable. There are several facets of this argument:

1. In an IDS, analysis is done locally and can thus operate on real-time data. In remote attestation, on the other hand, the analysis has to be performed remotely. This places a restriction on the amount of information that can be transmitted over the network in an acceptable time window and thus used for the purpose of attestation.[1]
2. In order to be able to trust the validation performed by an IDS on the target platform, the IDS will have to be a part of the trusted computing base. This is infeasible due to the size and complexity of typical IDS code.
3. Remote attestation has the added advantage of being supported by cryptographic hardware. This can be leveraged to add strength to the measurement engine. Another way of looking at this is that IDS techniques employ software-based solutions alone. It has been shown that it is not possible to provide tamper-proof security without the use of hardware-based solutions [12].
4. IDS are more concerned with *protecting local resources* while attestation aims at *verifying the integrity* of the remote platform before releasing sensitive data to or initiating communication with it.

In this paper, we present the idea of using the techniques developed for intrusion detection to facilitate dynamic behavior attestation of a remote platform. In particular, we discuss how sequences of system calls can be modeled through stochastic machine learning techniques to decide upon the behavior of a target application executing on a remote platform. The limitations of having to measure the system calls on a remote platform and reporting them to the challenger in an efficient and trustworthy manner are discussed at length. Finally, we present the details of our experimental results to show that the new remote attestation technique is not only feasible but also very effective in identifying malicious behavior on a target platform.

[1] It also leads to privacy concerns but we refrain from commenting on those issues here.

2 Background

2.1 Dynamic Remote Attestation

Several approaches have been proposed in the past to cater to problems caused by the static nature of IMA [1]. Loscocco et al. [4] proposed measuring kernel data structures at runtime for attestation purposes. These structures are to be compared with those generated by debug kernels running in a pristine environment to decide upon the trustworthiness of the target platform. PRIMA [13] considered information flows between the target application and other executables on the system to render some dynamism in the remote attestation paradigm. *Model-based Behavioral Attestation* [6] and subsequent efforts [14] have used the concept of behavior measurement to verify the trustworthiness of a target application based on the stakeholder's policies. Davi et al. [5] have used taint analysis and dynamic tracing to address the issues of static load-time measurement. Similarly, Gu et al. [3, 15] have proposed the use of measuring the *set* of system calls made by a target application at runtime and comparing them with the benchmark. The benchmark is obtained by statically analyzing the calls made by a pristine copy of the application – either through source code analysis or a disassembly process in case the source is not available.

All of these approaches are too inflexible to be operable in heterogeneous environments since even the slightest change to the target environment or application can cause the target to be tagged as being malicious. We believe that the approach presented by Gu et al. [3] has a lot of potential but has several limitations:

1. Reporting all system calls made by a process to a remote platform is a serious bottleneck. The number of calls made by a typical mail server, for example, may easily reach several million within the span of a few days' uptime [11].
2. The sequence of calls made at runtime can be significantly different from the order expected as a result of code analysis. This can lead to a large false positive rate when identifying malicious behavior of an application.
3. The approach presented by Gu et al. does not consider levels of trustworthiness of an application. A target is either trusted or untrusted with no 'gray areas' in between.

We believe that the use of stochastic models developed for intrusion detection systems can provide a foundation for better, more dynamic and flexible remote attestation techniques. Below, we discuss some of the relevant aspects of these systems.

2.2 Stochastic Models for Intrusion Detection Systems

Intrusion detection is a highly mature field of research and has seen many simple to complex techniques. *Intrusion detection systems* (IDS) fall in two major categories: *host-based* and *network-based*. In this paper, we focus on host-based

intrusion detection systems as our focus is on identifying anomalies in target platforms themselves rather than on the attacks made on the systems. Within host-based IDS, two major approaches are *signature-based* and *anomaly-based* [16]. Signature-based systems are akin to IMA and use hashes of executables to detect malicious applications. These systems are easier to build and are very accurate in identifying known malicious code [17]. However, detecting zero-day malware is virtually impossible with these systems [16]. To detect malicious behavior of executables, anomaly-based IDS were proposed. These build profiles of 'good' and 'bad' behavior and judge unknown executables based on their patterns of execution.

One of the most significant efforts at anomaly-based intrusion detection is *Sequence Time Delay Embedding* or STIDE [8]. It defines the behavior of an application based on the sequence of system calls it makes. Subsets of these sequences are defined using predetermined, fixed-width windows. Calls within individual windows are used as patterns for behavior matching. Unknown code is measured against these windows to decide whether it is acting suspiciously or not. Since the inception and successful demonstration of STIDE, several techniques [9, 10, 18–20] have built on this concept to enhance anomaly-based intrusion detection.

For our purposes, we use the most recent (and empirically successful) work proposed by Mehdi et al. [9] We explain our rationale for this choice in Section 3. Here, we cover the basics of this technique as it is at the heart of our proposed architecture. Mehdi et al. propose the concept of a *hypergram*, which is a structure capable of storing system calls made by an executable. It captures information such as the *sequence* in which the calls were made as well as the *distance* (spatial difference, as opposed to temporal) between elements of the sequence. Formally, a hypergram is a point in a hyperspace of n dimensions, each representing a specific system call. The occurrence of a system call causes the hypergram to *move* along the dimension associated with that call. The exact semantics of the move are defined as follows. For each program, the hypergram is *initialized* to lie at the origin of the hypercube of n dimensions, each representing a *critical system call* that needs to be measured. Afterwards, on the occurrence of each system call i:

1. The hypergram is *diminished* along all dimensions by a factor of δ_i. The new position of the hypergram γ along dimension i at time instance t is given by: $\gamma_i^t = \gamma_i^{t-1} * \delta_i$. A small value of δ for a system call ensures that calls made in the distant past eventually lose their ability to define the behavior. Calls made in the recent past have more effect on the behavior.
2. The hypergram is *moved* along the dimension i. The position is updated as: $\gamma_i^{t'} = \gamma_i^t + \alpha_i * \frac{\beta_i}{\beta_i + \gamma_i^t}$ where α is the *addition factor* and β is the *sloping factor*. The value of the addition factor defines the effect that each occurrence of the system call has on the hypergram and β ensures that extremely large values of γ along one dimension do not skew the data unnecessarily if a particular system call is made extensively by a program.

Finally, after each move, the values of hypergrams, along all dimensions, are rounded off to the nearest integer. In our experiments, we did not see a reason

to enforce this last step (cf. Section 4). Using these semantics, a hypergram is capable of storing the *complete history* of the system call sequence leading up to it.

We argue that the naïve approach of using this technique directly for remote attestation is not a feasible solution. Our reasons are three-fold: (1) The number of updations to the hypergram is the same as the number of system calls made by an application. As has been argued before, this is a rather large number and reporting the log of such a huge number of updations is simply not possible in the context of remote attestation. (2) This approach uses an *in-execution classifier* that produces results in real-time. Remote attestation, by nature, requires a *post-execution* analysis and thus would require a difficult classification approach. (3) In the work of Mehdi et al. [9], there is no mechanism of reporting the hypergrams to a remote party in a trustworthy manner. We therefore believe that in order to leverage this model for the purposes of remote attestation, we would need to come up with an architecture that can address all three of these issues.

3 Leveraging Stochastic Models for Remote Attestation

In order to cater to the problems presented in the previous section, we propose the introduction of stochastic models in the area of remote attestation to enable detection of unknown and zero-day malware as opposed to the known-good and known-bad classification currently being used. Specifically, we use the constructs of intrusion detection through hypergrams in collaboration with the reporting facilities provided by the Trusted Platform Module (TPM) [21]. Our reasons for choosing this specific intrusion detection technique are as follows:

1. Hypergrams allow the *aggregation* of a large number of system calls, capturing sequences of these calls and the displacement between each call;
2. they have been shown to be highly accurate in the area of local IDS; and
3. they provide a flexible model that can be tailored to a specific scenario by tweaking the parameters of the model.

Figure 1 shows the proposed architecture in detail. There are two aspects of this architecture: 1) Trusted measurement and reporting mechanisms at the target platform and 2) verification mechanisms at the challenger end.

3.1 Hypergram Measurement and Reporting

Chain-of-trust: In order to establish the chain of trust from the Trusted Platform Module (TPM) to our logging mechanism, we use the constructs of TCG. The chain-of-trust up to the bootloader is established as per TCG specifications [21]. Moreover, the trustworthiness of the kernel is established through *trusted grub* – a trusted boot loader that measures the kernel and extends its hash into a PCR before passing control onto it. The kernel itself does not have to have IMA enabled for our architecture to work. In-kernel behavior measurement is done through a different approach described below.

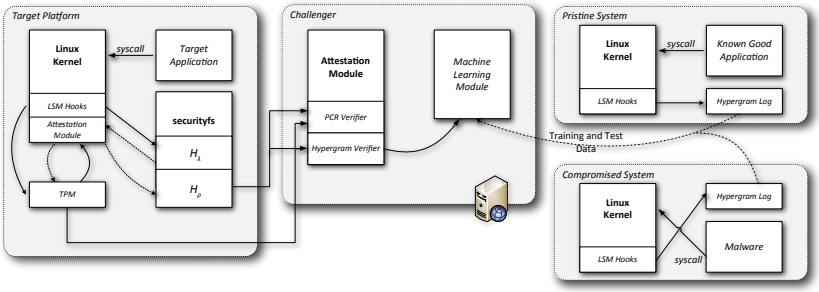

Fig. 1. Behavior Attestation through Stochastic Models

Hypergram Measurement: Capturing critical system calls made by an executable is a fairly trivial task. We implemented this logging mechanism in a fashion similar to IMA by creating a custom Linux Security Module (LSM) – labeled `scbm` (for *system call behavior monitor*). An LSM [22] allows the execution of custom functions in the kernel space on the execution of certain kernel operations. The purpose is to allow a pluggable architecture for defining access control models and policies. The LSM is notified whenever system calls are made. In our implementation, however, we are only interested in logging certain system calls that we identified as being critical (see Section 4.1). For example, the system call for mapping files to memory, `mmap`, is recorded through the `file_mmap` hook, file operations such as `open`, `close`, `read` and `write` through `inode_permission` and `file_permission`, socket operations through `socket_connect` etc.[2]

In each of these routines, we call a measurement function that calculates the values of the hypergram associated with the originating executable (as defined in Section 2.2). The new hypergram value can be retrieved from `scbm/scbm_hypergram_log_lv` file in the `securityfs`. Moreover, the hash computed over this new hypergram value is extended into PCR-11.

Hypergram Reporting: The task of reporting the measured hypergrams is far from trivial. As mentioned in Section 2.1, the typical syscall trace can easily reach up to millions of lines. Transmitting such a huge log over a network readily becomes a bottleneck for any remote attestation technique. On the other hand, if the complete log is not reported, the value of the PCR cannot be verified simply from the PCR quote structure. This is a problem not only for our architecture but also for any technique that aims to report the syscall trace to a challenger. However, the novelty of our approach is that it utilizes hypergrams that, by definition, incorporate the *complete history of the syscall trace* leading up to them. Therefore, in our scenario, it is possible to come up with a solution to this dilemma.

For this purpose, we introduce the concept of *local verification*. When an attestation request is received, a piece of trusted code within kernel space performs

[2] We refer the reader to the source of `security_operations` structure in the linux kernel at `include/linux/security.h` for the details of these and other hooks.

local verification of the hypergram log created during the measurement process. We call this log *hypergram log for local verification* (H_λ). Hashes of entries in H_λ are aggregated in the same manner as the `pcr_extend` operation of the TPM. The aggregated value is compared with the expected value retrieved from the `pcr_quote` operation over PCR-11 to ensure that none of the entries in H_λ has been tampered with. This process is termed as local verification of H_λ.

Upon successful verification, the final values of hypergrams associated with each process are stored in another log file termed as *hypergram log for remote verification* (H_ρ). With each entry in H_ρ, PCR-12 is extended with the hash of the entry. Thus, H_ρ contains one entry per process thus significantly reducing the size of the log. Moreover, since the final hypergram value is a function of the complete history of the system calls made by that process, no information loss occurs as a result of this operation.

Finally, H_ρ and `pcr_quote` over PCR-12 is returned to the challenger where it can be verified and validated as described below.

3.2 Verification through Stochastic Models

Once the hypergram log (H_ρ) and PCR quote are received at the challenger end, they must be verified in order to establish the trustworthiness of the target application(s). The PCR verification procedure is straight-forward. The aggregate *expected* value of the PCR is compared with the value returned as `pcr_quote`. If the values match, H_ρ can be considered trustworthy and can be used to measure the trustworthiness of the target application.

For the establishment of the trustworthiness of the behavior, we utilize stochastic models of machine learning techniques. For this purpose, as per the general practice, the known-good and known-bad hypergram values must first be collected. This data is called *training data* and was collected using the same syscall logging mechanism described in the previous section. We collected *benign hypergrams* from applications running in a pristine, sandboxed environment. The collection of *malicious hypergrams* is a little more cumbersome as the environment has to be reset to a pristine state after the dataset for each malicious application has been collected. Several malicious executables can be found on popular virus databases such as VX Heavens [23]. Moreover, a *test set* has to be collected in the same manner in order to test the effectiveness of the stochastic model being used. Once the malicious and benign behaviors have been modeled, the hypergrams received from the target platform can be classified as either benign or malicious based on the model (or *classifier*) being used for the classification.

Our proposed architecture is model-neutral in that no model-specific transformation is performed at the target platform. All model-specific operations are carried out at the challenger side. Therefore, the classifier used by each challenger can be different and can be upgraded without having to modify the target architecture. Notice that no machine learning algorithms have been utilized on the target side. Only the hypergram-based IDS technique is used for measuring the hypergrams. The machine learning part of our proposed architecture operates on the challenger end.

Currently, we have investigated several models of machine learning for the purpose of this classification. These include *naïve bayes* [24], *bayesian networks* [25], *J48* [26], and *0R*. Results achieved by these models are discussed in Section 4.

Here, we would like to risk belaboring the obvious by mentioning why this classification cannot be carried out at the target platform thus removing the need for trustworthy reporting of the hypergrams. If such an architecture is used, the complete classification system would have to reside within each of the target platforms. This would not only increase the size of the trusted computing base significantly, but also add excessively to the administrative overhead of deploying and maintaining the attestation system.

4 Evaluation

4.1 Experiment Setup

In order to ensure that our results can be reproduced, we have utilized the traces of system calls available in the datasets provided by UNM [11]. For the sake of simplicity, we created a tool – *HypergramGenerator (hGen)* – for generating hypergrams from the traces of these system calls. hGen is capable of taking STIDE [8] input format system call sequences and a mapping file and generating hypergrams. It allows the user to specify the critical system calls as well as to set the δ, α and β parameters . The hypergrams produced by hGen are used to train the stochastic model based on different machine learning techniques [24–27].

The hGen tool was used to generate training data including both malicious and benign behaviors from the UNM datasets for `inetd`, `ps` and `login`. As our workbench, we used the popular Wikato Environment for Knowledge Analysis (WEKA) tool [28]. Approximately 20% of the data was used for training and the rest was used as a testset. Similar to [9], we found out that including the scarcely used system calls in the dataset significantly reduced the accuracy of the model. We therefore used the hGen tool to find the most-oft used system calls and then identified the critical calls from among these.

The experiments were run using manually configured values of the three model parameters δ, α and β as we were unable to use the reasoning proposed by [9] to automate the process. We then utilized the different machine learning algorithms to come up with the best possible stochastic model for our datasets.

4.2 Evaluation Criteria

To measure the effectiveness of our approach, we use the widely-adopted concept of *Receiver Operating Characteristics* (ROC) curves [29]. An ROC curve is a line graph of true positives (on the y-axis) against false positives (on the x-axis) for different threshold values. The *Area Under the Curve* (AUC) of the ROC is a measure of how well the model performs. The higher the AUC, the more accurate is the model.[3] We calculated the AUC for each of the models we applied on the hypergrams. The results of our experiments follow.

[3] See [30] for the latest details on the ROC curve-based model evaluation.

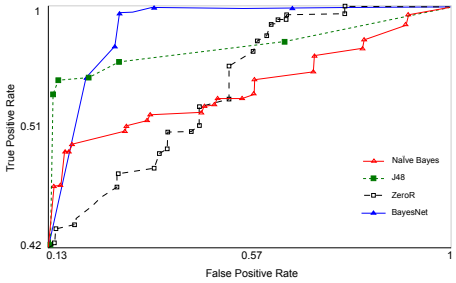

Fig. 2. ROC curves for different learning models

Classifier	**AUC**
Naïve Bayes	59.14
0R	62.19
J48	78.65
BayesNet (with Genetic Search)	85.72
In-execution classifier [9]	87.85

Table 1. Comparison of AUCs for different learning models

4.3 Results

Figure 2 shows the ROC curves for the different machine learning algorithms we used for attestation purposes i.e. naïve bayes, bayesian networks, 0R and J48. Table 1 shows the AUC values for the different classifiers. Naïve bayes and 0R performed quiet poorly in this scenario. J48 (the WEKA implementation of C4.5 algorithm) provided better results than these two. Using BayesNet, we were able to achieve the AUC of 85.72. Note that we do not have access to the in-execution classifier developed by Mehdi et al. [9] and the values under that column are taken from their original paper. Note also that we have not been able to reproduce the AUC level achieved by Mehdi et al. However, we are operating in a different scenario under different constraints. Also, we believe that this is the start of a new approach in remote attestation and is likely to get better as more and more efforts are put in this area.

5 Discussion

One of the most important points to note in our technique is that we have provided a rather limited training set to our machine learning models. One might argue that, as with traditional remote attestation techniques, we are only providing known-good and known-bad hypergram values to our attestation module. The number of possible hypergrams that might be generated as a result of execution on the target platform is potentially infinite. Despite this fact, the results obtained are promising. The reason is exactly our rationale for choosing machine

learning techniques for attestation. Machine learning models are capable of operating on limited and noisy data [27, Chapter 2]. They are frequently used to predict unknown information based on *incomplete* input knowledge. Here, we have leveraged this strength of machine learning models to cater to the fact that it is impossible to predict all paths of execution for any target application. Of course, the more accurate information we have about known-good and known-bad hypergrams (i.e. the larger the training and test data), the more accurate our model will be.

Secondly, in this paper, we have only discussed the behavior of a *single application* and, by extension, define the trustworthiness of the target *platform*. The trustworthiness of the platform is assumed to be synonymous with the trustworthiness of all applications executing on the platform. We have not modeled the behavior of the platform as a whole. This might form a potentially useful future direction in this line of research.

Third, as discussed in Section 2.1, none of the existing attestation techniques have the ability to predict *levels of trustworthiness*. A target is either trustworthy or it is not. The technique proposed in this paper, in its current form has the same limitation. However, this is due to the fact that typical machine learning algorithms apply a threshold for *classification*. By removing this threshold application step, we may be able to output the *level* of trustworthiness rather than the *class*. This would require the development of a new 'classifier', which forms part of our future work.

Finally, we would like to point out an issue that this paper has not addressed. The avid TC developer may have noticed that the local verification step, detailed in Section 3, breaks the chain-of-trust. A malicious application may somehow alter the local verification procedure and simply report a 'known-good' hypergram to the challenger. We are currently working on addressing this issue by exploring the semantics of executing the local verification mechanism in a secure execution environment such as one proposed by Flicker [31]. While it might pose some implementation difficulties, the procedure is trivial at the conceptual level. Such an implementation would restore the chain-of-trust and provide the challenger assurance that the local verification mechanism was indeed correct.

6 Conclusions

In this paper, we have presented a new remote attestation technique that utilizes the constructs of machine learning for the purpose of remote attestation. The end result is a flexible attestation mechanism that can measure the dynamic behavior of a target application and, by extension, the target platform. We have presented the details of the implementation, the challenges faced and the empirical results in the favour of the proposed system. We believe that this is a new avenue in remote attestation research and will lead to several new and improved techniques of remotely verifying the behavior of a platform in a flexible and scalable manner.

References

1. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium. (2004)
2. Shacham, H.: The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). In: Proceedings of the 14th ACM conference on Computer and Communications Security (CCS'08), ACM New York, NY, USA (2007) 552–561
3. Gu, L., Ding, X., Deng, R., Xie, B., Mei, H.: Remote Attestation on Program Execution. In: STC '08: Proceedings of the 2008 ACM Workshop on Scalable Trusted Computing, New York, NY, USA, ACM (2008)
4. Loscocco, P.A., Wilson, P.W., Pendergrass, J.A., McDonell, C.D.: Linux Kernel Integrity Measurement Using Contextual Inspection. In: STC '07: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, New York, NY, USA, ACM (2007) 21–29
5. Davi, L., Sadeghi, A., Winandy, M.: Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In: Proceedings of the 2009 ACM workshop on Scalable trusted computing, ACM (2009) 49–54
6. Alam, M., Zhang, X., Nauman, M., Ali, T., Seifert, J.P.: Model-based Behavioral Attestation. In: SACMAT '08: Proceedings of the thirteenth ACM symposium on Access control models and technologies., New York, NY, USA, ACM Press (2008)
7. Nauman, M., Khan, S., Zhang, X.: Beyond Kernel-level Integrity Measurement: Enabling Remote Attestation for the Android Platform. In: Trust'10: Proceedings of the Third International Conference on Trust and Trustworthy Computing, Springer (2010)
8. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for unix processes. In: 1996 IEEE Symposium on Security and Privacy, 1996. Proceedings. (1996) 120–128
9. Mehdi, B., Ahmed, F., Khayyam, S., Farooq, M.: Towards a Theory of Generalizing System Call Representation For In-Execution Malware Detection. In: ICC'10: Proceedings of the IEEE International Conference on Communications. (2010)
10. Mutz, D., Robertson, W., Vigna, G., Kemmerer, R.: Exploiting execution context for the detection of anomalous system calls. In: Recent Advances in Intrusion Detection (RAID'07), Springer (2007) 1–20
11. University of New Mexico: Computer Immune Systems – Datasets (Accessed May, 2010) Available at: `http://www.cs.unm.edu/~immsec/systemcalls.htm`.
12. Pearson, S.: Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR, Upper Saddle River, NJ, USA (2002)
13. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: Policy-Reduced Integrity Measurement Architecture. In: SACMAT '06: Proceedings of the eleventh ACM Symposium on Access Control Models and Technologies, New York, NY, USA, ACM Press (2006) 19–28
14. Nauman, M., Alam, M., Ali, T., Zhang, X.: Remote Attestation of Attribute Updates and Information Flows in a UCON System. In: Trust'09: Proceedings of the Second International Conference on Technical and Socio-Economic Aspects of Trusted Computing, Springer (2009) 63–80
15. Gu, L., Cheng, Y., Ding, X., Deng, R., Guo, Y., Shao, W.: Remote Attestation on Function Execution. In: InTrust'09: Proceedings of the 2009 International Conference on Trusted Systems. (2009)

16. Axelsson, S.: Intrusion detection systems: A survey and taxonomy. Technical report, Department of Computer Engineering, Chalmers University (2000)
17. Kruegel, C., Toth, T.: Using decision trees to improve signature-based intrusion detection. In: Recent Advances in Intrusion Detection, Springer (2003) 173–191
18. Hofmeyr, S., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. Journal of Computer Security **6**(3) (1998) 151–180
19. Hofmeyr, S., Forrest, S.: Architecture for an artificial immune system. Evolutionary Computation **8**(4) (2000) 443–473
20. Wilson, W., Feyereisl, J., Aickelin, U.: Detecting Motifs in System Call Sequences. In: 8th international workshop on Information security applications, Springer (2007) 157
21. TCG: TCG Specification Architecture Overview v1.2, page 11-12. Technical report, Trusted Computing Group (April 2004)
22. Wright, C., Cowan, C., Morris, J., Smalley, S., Kroah-Hartman, G.: Linux security module framework. In: Ottawa Linux Symposium, Citeseer (2002)
23. VX Heavens: Information and hosting for computer viruses Available at: `http://vx.netlux.org/`. Accessed June 02, 2010.
24. Bayes, T.: Learning Bayesian networks: The combination of knowledge and statistical data. Philosophical Transactions of Royal Society of London **53** (1763) 370–418
25. Heckerman, D., Geiger, D., Chickering, D.: Learning Bayesian networks: The combination of knowledge and statistical data. Machine learning **20**(3) (1995) 197–243
26. Quinlan, J.: C4.5: programs for machine learning. Morgan Kaufmann (1993)
27. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann Pub (2005)
28. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. ACM SIGKDD Explorations Newsletter **11**(1) (2009) 10–18
29. Lippmann, R., Fried, D., Graf, I., Haines, J., Kendall, K., McClung, D., Weber, D., Webster, S., Wyschogrod, D., Cunningham, R., et al.: Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In: DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings. Volume 2. (2000)
30. Ali, M., Khan, H., Sajjad, A., Khayam, S.: On achieving good operating points on an ROC plane using stochastic anomaly score prediction. In: Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09), ACM (2009) 314–323
31. McCune, J., Parno, B., Perrig, A., Reiter, M., Isozaki, H.: Flicker: An execution infrastructure for TCB minimization. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008, ACM (2008) 315–328