Poster: System Anomaly Detection with Program Analysis and Machine Learning Assistance

Kui Xu, Danfeng (Daphne) Yao, Barbara Ryder
Department of Computer Science
Virginia Tech
Blacksburg, VA, 24060
{xmenxk,danfeng,ryder}@cs.vt.edu

Detecting anomalous program behaviors is an important approach to protect personal computers and organizational infrastructures from various exploitations and attacks.

We present a novel anomaly detection methodology that efficiently provides quantitative measure to predict the real-time assurance of a system or application. We define assurance as evidence-based confirmation that the system or software is behaving as intended and is not tampered by malicious entities (e.g., malicious software or malicious insider). We define real-time quantified system assurance (QSA) as the capability to compute the likelihood of each system event occurring according to the intended program behaviors.

For the purpose of understanding program anomalies, static and dynamic analyses are two existing paradigms. In the literature, static analysis may be referred to as a white-box technique because the source code or binary is statically analyzed. On the static end of the related-work spectrum, static program analysis (such as [1], [9], [10], [11], [14]) provides insights on the intended control flow, call context, and call dependences of a program.

Prediction through pure static analysis usually incurs significant overhead and may overestimate the possible execution paths while ignore dynamic functionality. Thus dynamic techniques can be combined to yield a hybrid approach such as in [10] to refine the model and better predict the dynamic functionalities such as dynamic linked libraries and indirect calls. Conceivably, advanced program analysis techniques such as blended analysis [2], [3], [5] may be used to capture a more refined and accurate representation of the intended program behaviors. However, the above solutions generate binary predictions (i.e., allowed or not allowed), which is not accurate enough to provide quantitative measurement. For example, mass occurrences of low probability events could be a symptom of DoS attack, but will be considered as normal by these binary predictors.

A few dynamic analysis techniques based on probabilistic machine learning algorithms can quantify the desirable likelihood of occurrence for observed system behaviors. For example, the seminal work on anomaly detection by Forrest and colleagues [7] trained hidden Markov model to classify system call sequences based on their probabilities. Hidden Markov model (HMM) can characterizes a programs normal behaviors with respect to the training traces. In general, dynamic analysis (aka black- or gray-box) techniques build a model of system behaviors by monitoring sample executions. The system behaviors include system call sequence [6], libc call sequence [12], and process memory [4], [8], [13]. These approaches do not require code inspection and thus are referred to as black-box or gray-box techniques (the latter usually requires more execution information beyond system-call observables). However, the overall accuracy of the dynamic models is severely limited by the availability and completeness of the training data; thus, the anomaly detection may be too conservative (causing false alarms).

Nevertheless, a significant advantage of the probabilistic machine learning based black-box techniques (e.g., [15] is the ability to quantify anomalous events with probabilistic reasoning. However, the existing solutions train the machine learning models (e.g., HMM) from scratch, which is unnecessarily naive, since all the useful knowledge that can be obtained through static analysis is completely ignored. This type of practice yields biased models due to possibly incomplete training data, also even the data used for training may contain anomaly which will add false negatives to the classifier model. Our proposed approach (QSA) will fill in the gap between the state-of-the-arts and the vision of real-time quantified system assurance.

Our hypothesis is that knowledge extracted from static program analysis can be used to jump start the training of a hidden Markov model for anomaly detection. Our technical approach is to enhance a learning-based model with control flow dependence information extracted from static program analysis. Specifically, our approach is to initiate and refine probabilistic dynamic learning model (namely hidden Markov model) with static dependence information from control flow graphs and call graphs produced by static program analysis. The use of probabilistic machine learning technique allows one to detect anomalies quantitatively. Compared to existing approaches, our method will have the following two important technical advantages:

- The probabilistic prediction of the likelihood of system event sequence (e.g., with the forward algorithm of HMM)
- The availability of program analysis information for initial model construction.
- The complete and accurate coverage of both static and dynamic program behaviors.

We have conducted extensive experimental evaluation covering:

- both client-side and server-side applications.
- different lengths of trace segments.
- both system call and library call sequences.

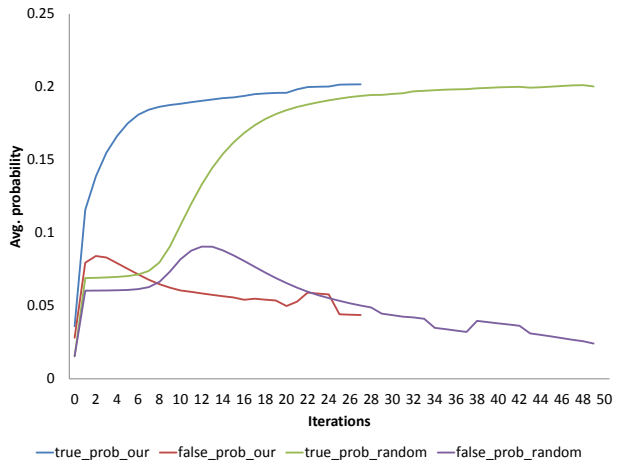

Fig. 1. Compariosn of average probabilities on both true and false sequences, between our model and randomized model (tested app: gzip).

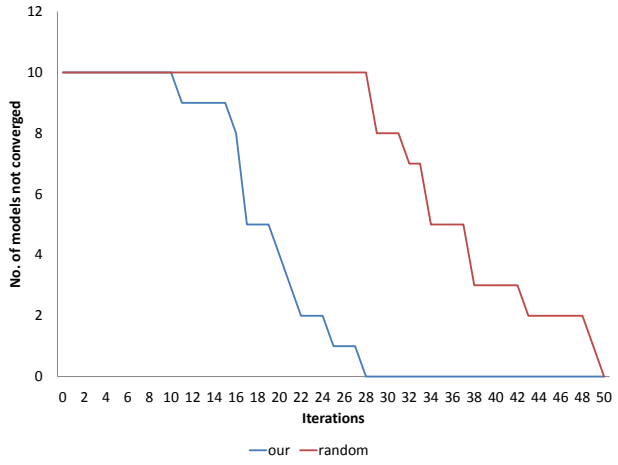

Fig. 2. Comparison of number of models not converged for a 10-fold cross validation, between our model and randomized model (tested app: gzip).

Figure 1 shows the comparison between our model and random model, about average probabilities on both true and false sequences. Figure 2 gives the number of models that have not converged during a 10-fold cross validation after iterations of training. Both figures are regarding program $gzip$.

Through the experiment results, we found that:

- Models initialized with our approach achieve complete coverage of program behaviors, and give quantitative prediction after dynamic model training.
- Models initialized with our approach start at good initial positions and need less time for training procedure to converge.
- Using longer trace segments for training and monitoring achieves better accuracy, but also costs more time for model training. Our approach significantly reduces the amount of time for convergence, thus enables the utilization of more accurate model.

REFERENCES

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 340–353, New York, NY, USA, 2005. ACM.

[2] B. Dufour, B. G. Ryder, and G. Sevitsky. Blended analysis for performance understanding of framework-based applications. In *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, pages 118–128, New York, NY, USA, 2007. ACM.

[3] B. Dufour, B. G. Ryder, and G. Sevitsky. A scalable technique for characterizing the usage of temporaries in framework-intensive java applications. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 59–70, New York, NY, USA, 2008. ACM.

[4] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 62–, Washington, DC, USA, 2003. IEEE Computer Society.

[5] M. Fisher, B. Dufour, S. Basu, and B. G. Ryder. Exploring the impact of context sensitivity on blended analysis. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[6] S. Forrest, S. Hofmeyr, and A. Somayaji. The evolution of system-call monitoring. In *Proceedings of the 2008 Annual Computer Security Applications Conference*, ACSAC '08, pages 418–430, Washington, DC, USA, 2008. IEEE Computer Society.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 120–, Washington, DC, USA, 1996. IEEE Computer Society.

[8] D. Gao, M. K. Reiter, and D. Song. Gray-box extraction of execution graphs for anomaly detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 318–329, New York, NY, USA, 2004. ACM.

[9] J. T. Giffin, S. Jha, and B. P. Miller. Detecting manipulated remote call streams. In *Proceedings of the 11th USENIX Security Symposium*, pages 61–79, Berkeley, CA, USA, 2002. USENIX Association.

[10] J. T. Giffin, S. Jha, and B. P. Miller. Efficient context-sensitive intrusion detection. In *NDSS*, 2004.

[11] R. Gopalakrishna, E. H. Spafford, and J. Vitek. Efficient intrusion detection using automaton inlining. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pages 18–31, Washington, DC, USA, 2005. IEEE Computer Society.

[12] A. Jones and Y. Lin. Application intrusion detection using language library calls. In *Proceedings of the 17th Annual Computer Security Applications Conference*, ACSAC '01, pages 442–, Washington, DC, USA, 2001. IEEE Computer Society.

[13] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 144–, Washington, DC, USA, 2001. IEEE Computer Society.

[14] D. Wagner and D. Dean. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 156–, Washington, DC, USA, 2001. IEEE Computer Society.

[15] C. Warrender, S. Forrest, and B. A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.