Towards a Tunable, Sandbox-Independent Approach for Exploring Hidden Behaviour in Malware

James Gatt, Mark Vella, and Mark Micallef

Department of Computer Science, University of Malta
{james.c.gatt.06 | mark.vella | mark.micallef}@um.edu.mt

Introduction: Malware analysis is the reverse engineering of malicious binaries in order to explore and extract their complete behaviour for protecting against attacks and to disinfect affected sites. The dynamic analysis of malware inside sandboxes is useful since it removes the need for the analyst to look into the malware code itself. However, this approach could end up disclosing no behaviour at all if faced with trigger-based malware. The two main categories of trigger-based malware are the environment-specific type (for example, time bombs that only run when the date and time are right) and the externally-stimulated type (a recent case waited for a certain number of mouse clicks by the user before continuing execution[2]). Existing work that investigates how to unlock hidden behaviour in malware takes an execution path exploration approach with the aim of maximizing effectiveness by increasing both the path coverage along with the precision, the latter of which is increased by excluding infeasible paths and executing paths under the correct runtime values. In these terms, Symbolic Execution fits the bill. This technique uses constraint-solving to map out each execution path as a conjunction of all the values needed at each conditional branch in order to reach a certain point in the execution tree.

Issues: However, while encouraging results have been reported in work such as that by Brumley et al.[1], it is also true that due to its constraint-solving approach, symbolic execution is costly in relation to its overall performance, lowering its efficiency, measured in terms of the time elapsed versus the amount of hidden behaviour found. Existing work tries to increase efficiency by introducing heuristics, however, the implications of this in terms of effectiveness remain unspecified. Symbolic execution also has a number of inherent limitations, some of which have been tackled in existing work and others which have not, which degrade its overall effectiveness. Additionally, the majority of existing approaches use System-Wide Instrumentation, as it allows access to both kernel and user-level code at runtime, while also allowing for the creation of system-wide 'snapshots' which can be reverted to when backtracking. While this works to great effect, it also means that the solution is tied to a particular sandbox or full-system emulator, in a world where sandboxes usually come with hefty price tags. Sandbox Independence would require a shift to Process-Level Instrumentation, which introduces a new set of limitations that need to be overcome.

Aim and Objectives: The main aim of this project is to build on existing work in order to provide a solution for automated malware analysis that is capable of uncovering hidden behaviour in malware, but is also tunable in terms of its efficiency versus its effectiveness, while also being Sandbox Independent. Specifically, the objectives set for the scope of this work include: (1) the full exploration of environment-specific trigger-based malware; (2) a move towards Process-Level Instrumentation, which will require overcoming the resultant loss of control of any kernel-level code as well as the possibility of saving the system-wide state for backtracking; (3) a system that can be tuned in terms of the effectiveness-efficiency tradeoff; (4) a system that should be able to map a malware's behaviour to the sequence of system library calls issued by the malware at runtime.

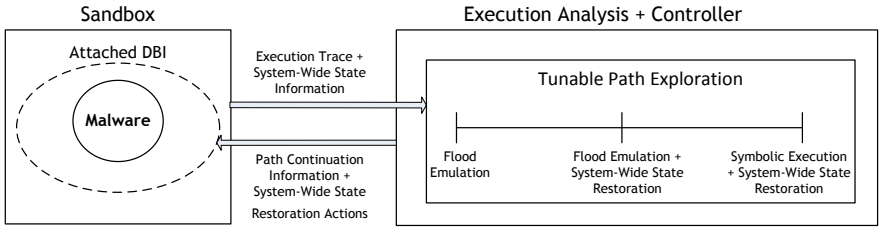

Fig. 1: Proposed Solution

Solution: Our proposed solution is presented in Figure 1 above. In terms of the objectives stated above, the malware will be launched by a Dynamic Binary Instrumentation (DBI), capable of running in any sandbox, and allowing us to control the malware at runtime through its Process Control Interface (PCI), provided by the underlying operating system. This approach allows us to provide system-wide snapshots for backtracking, while also taking away the need to rely on kernel-level code. Tunability will be introduced by also considering a lightweight alternative to Symbolic Execution which can still be used to detect hidden behaviour, at the cost of effectiveness. For this purpose, we have chosen Flood Emulation[3], which forgoes the constraint-solving approach, and resolves path exploration at the code block level. Thus, malware analysis can be tuned between both configurations, across the effectiveness-efficiency tradeoff.

Evaluation: We propose to evaluate our work as follows. The effectiveness of DBI as an equivalent to system-wide emulation will be measured by analysing a custom binary that requires backtracking in order to unlock all its hidden behaviour (for example, one which deletes and recreates files). Evaluating the tunability of the analysis configurations as well as the exploration of trigger-based malware is currently work in progress as there are a number of possible approaches (for example, testing purpose-built synthetic binaries as opposed to different categories of real malware). In each case, the effectiveness of analysis will be determined by the accuracy of the behaviour extracted from the malware.

Conclusion: The framework for this project is currently under construction, with WinAppDbg[1] being used for the DBI and Flood Emulation components, and BAP[2] (Binary Analysis Platform) being used to handle the Symbolic Execution side.

References

1. Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., Yin, H.: Automatically Identifying Trigger-based Behavior in Malware. Springer US, 1st edn. (2008)
2. Hwa, C.R.: Trojan.apt.banechant: In-memory trojan that observes for multiple mouse clicks (Apr 2013), http://www.fireeye.com/blog/technical/malware-research/2013/04/trojan-apt-banechant-in-memory-trojan-that-observes-for-multiple-mouse-clicks.html
3. Wilhelm, J., Chiueh, T.: A forced sampled execution approach to kernel rootkit identification. In: RAID'07 Proceedings of the 10th international conference on Recent advances in intrusion detection. pp. 219–235. Springer-Verlag (2007)

[1] http://winappdbg.sourceforge.net/
[2] http://bap.ece.cmu.edu/