Towards Automated Forensic Event Reconstruction of Malicious Code (Poster Abstract)

Ahmed F. Shosha, Joshua I. James, Chen-Ching Liu, and Pavel Gladyshev

University College Dublin, Ireland
Ahmed.Shosha@ucdconnect.ie,
{Joshua.James,Liu,Pavel.Gladyshev}@ucd.ie

Abstract. A call for formalizing digital forensic investigations has been proposed by academics and practitioners alike [1, 2]. Many currently proposed methods of malware analysis for forensic investigation purposes, however, are derived based on the investigators' practical experience. This paper presents a formal approach for reconstructing the activities of a malicious executable found in a victim's system during a post-mortem analysis. The behavior of a suspect executable is modeled as a finite state automaton where each state represents behavior that results in an observable modification to the victim's system. The derived model of the malicious code allows for accurate reasoning and deduction of the occurrence of malicious activities even when anti-forensic methods are employed to disrupt the investigation process.

Keywords: Formal Models, Event Reconstruction, Model Checking and Automated Static Malware Analysis.

Introduction: This work introduces a formal model for automated malware investigation based on the modeling of malicious executables. In the proposed approach, malicious code is analyzed using automated static analysis methods [3-5]. The malicious code's control flow graph is then formally modeled as a finite state automaton (FSA). The formalized model of the malicious code behavior is processed by an extension of the event reconstruction algorithms proposed in [2, 6], which computes the set of all possible explanations for the state of the victim's system in the context of the malicious code where the observed state of the victim's system and malware trace creation states intersect. The result is a reduced state-space where malicious actions agree with the observed state of the system. Furthermore, the modeled FSA allows for the inference of the occurrence of actions that do not leave an observable trace.

***Modeling Investigated Malicious Code*:** Malicious executable *IE* is formally defined as a sequence of instructions$\langle I_1, I_2 \ldots . I_n\rangle$. The behavior of *IE* is represented in a finite state automata $M = (Q, \sum, \delta, q)$, where Q is a finite set of all possible instructions in *IE* and δ represents transition function that determines the next instruction I_m for every possible combination of event and instruction state I_q, such that, $\delta: \sum \times Q \rightarrow Q$. A transition is the process of instruction execution. An execution path $p = (s_0, s_1 \ldots s_q)$ is a run of finite computations consisting of a sequence of instructions that lead executable *IE* to the final state q.

***Malicious Events Reconstruction*:** is the process of determining all possible execution paths that are consistent with observable evidence. In this approach, we extend and improve a formal model for automated reasoning of evidential statements and reconstruction of digital events proposed in [2]. The extended formal model is based on back-tracing execution paths that hold the observation O_x. The proposed back-tracing technique over all possible execution paths is based on the finite computation $c_j = \langle c_j^{\sigma}, I_j^{q} \rangle$, where, $c_j^{\sigma} \in \Sigma$ is an event and $I_j^{q} \in Q$ is a state. Any two instructions I_k and I_{k-1} are related via the transition function for a given instruction $I_j^{q} = \delta\left(c_{k-1}^{\sigma}, I_{k-1}^{q}\right)$. The notation of back-tracing an execution path is formalized in Equation 1, where ψ^{-1} traces back all finite computations representing the execution paths in the malicious executable *IE*.

$$\psi^{-1}(Q) = \bigcup^{\forall I \in Q} \psi^{-1}(I) \quad (1) \quad O = (P, min, opt, pr_c) \quad (2) \quad AG\ AF\ \mu \Rightarrow \psi^{-1}(p) \quad (3)$$

***Formalizing Malicious Code Observations*:** Evidence is described as an observable property, *O,* of a victim's system that denotes the execution of a malicious payload. The formalization of an observation is defined in Equation 2, where *P* is a set of all instructions in *IE* that have the observed property *pr*. *min* and *opt* are positive integers specifying the duration of the observation and pr_c is the set of characteristics of the observed property *pr*. An execution path *p* is said to contribute to *O* if a set of sequence of instructions in *p* possesses the observed property *pr.*

***Observation Consistency Checking*:** Anti-forensic techniques are formally encoded in a CTL specification model [7] μ. Using the proposed model checking algorithm, the model of a suspect executable *IE* is checked against the encoded techniques μ in the context of malicious code execution to identify tampered observations. The model checking algorithm takes a formula μ and executable model *IE* and verifies all states $s \in IE$ where μ holds. The notation of the model checking algorithm is formalized in Equation 3, where *A* is a quantifier over all paths *p* that contribute to the observation *o*, and *G/F* are a path specific quantifiers that check if μ holds over all states *s* and possess *o*.

References

1. Stephenson, P.: Using a Formalized Approach to Digital Investigation. In: Computer Fraud & Security (2003)
2. Gladyshev, P., Patel, A.: Finite state machine approach to digital event reconstruction. Digital Investigation (2004)
3. Christodorescu, M., Jha, S.: Static analysis of executables to detect malicious patterns. In: USENIX Security Symposium (2003)
4. Christodorescu, M., Jha, S., Kruegel, C.: Mining specifications of malicious behavior. In: ESEC-FSE (2007)
5. Kinder, J., Katzenbeisser, S., Schallhart, C., Veith, H.: Detecting Malicious Code by Model Checking. In: Julisch, K., Kruegel, C. (eds.) DIMVA 2005. LNCS, vol. 3548, pp. 174–187. Springer, Heidelberg (2005)
6. James, J., et al.: Analysis of Evidence Using Formal Event Reconstruction. Digital Forensics and Cyber Crime (2010)
7. Emerson, E.A.: Temporal and modal logic. In: van Jan, L. (ed.) Handbook of Theoretical Computer Science, vol. B (1990)