Computing the Behavior of Malicious Code
With Function Extraction Technology

Richard Linger
Stacy Prowell
Kirk Sayre

CERT, Software Engineering Institute
Carnegie Mellon University, Pittsburgh, PA
rlinger@sei.cmu.edu, sprowell, sayre@cert.org

1. The Idea of Behavior Computation

Modern society is irreversibly dependent on computer-based infrastructure systems of astonishing scope and complexity. Yet these systems are increasingly vulnerable to disabling intrusions by adversaries equipped with the knowledge and resources to mount sophisticated attacks [1,2]. Technical and organizational preparation for major attacks is essential to national security. Preparation takes time and effort, and when a major attack occurs, it is too late to prepare. Preparation must provide capability for fast and precise analysis of the technical structure of an attack as a basis for fast response to limit damage and deploy countermeasures.

A key element of preparation is a capability for fast and accurate analysis of malicious code to understand intruder objectives and strategies. In today’s state of art, analysis of malicious code behavior can be a difficult and resource-intensive task that requires in-depth knowledge and experience. Intruders do everything possible to obfuscate the code and increase its complexity in an attempt to thwart analysis. As a result, analysis can take substantial time and effort, all the while damage is being done.

The CERT organization of the Software Engineering Institute at Carnegie Mellon University is creating a next-generation approach to malware analysis. The Function Extraction (FX) project is developing theory and implementation for computing the behavior of software, specifically malware, with mathematical precision to the maximum extent possible.

Behavior is defined as the net functional effect of software in all possible circumstances of use. The objective is automated behavior computation at machine speeds, to help analysts achieve fast and precise understanding of malware as attacks unfold [4]. The initial implementation of FX is for programs written in or compiled into Intel assembly language, however, many applications of the technology are possible [5,6,7,8,9].

The technical basis for behavior computation is treatment of programs and their constituent parts as rules for mathematical functions and relations [3]. A theorem defines mappings from procedural representations into non-procedural, as-built specifications of behavior, expressed as conditional concurrent assignments (CCAs) of initial state into final state. Typical databases of computed behavior contain multiple CCAs that define program behavior in varied circumstances of use.

Behavior computation requires as input the functional semantics of the 1100+ op codes supported by the processor as a starting point, and this definitional task is underway. These semantics define the complete net effect of each op code on the state of the processor.

The first step in behavior computation is to express input program instructions in terms of their defined functional semantics. Next, any spaghetti logic in the input program is transformed into function-equivalent structured form expressed in a fixed basis set of control structures, including sequence, ifthenelse, and whiledo, all nested and sequenced in an algebraic structure. This transformation is defined by the constructive proof of a structure theorem.

Behavior computation then proceeds from leaf to root node control structures in a stepwise manner, at each step propagating non-local behavior effects to the next level in a repeating process of composition and rewriting. The computations are mathematically precise; no heuristics or approximations are employed. Research has developed unique methods for making the effects of theoretical limitations on loop computations arbitrarily small.

2. The Vulnerability of Malware

It is surprising at first thought to realize that malware itself has a vulnerability that can be capitalized upon in its analyses. As depicted in the top display of Figure 1, a typical intrusion strategy is to make a malware payload more difficult to detect and analyze through a variety of means that affect the representation, structure, and location of the code.

But in this quest for complexity, the intruder must ensure that the bedrock functionality of the code remains intact; otherwise, the intended effect of the attack will be negated. This is the fundamental vulnerability of malware; because its function must be preserved by the attacker, it is vulnerable to technology such as FX that can compute that function, as illustrated in the bottom display of Figure 1.
3. A Specialized FX System

A special version of the FX system is being developed to interface directly with the popular Ida Pro disassembler used by malware analysts. A key focus of this system, named FX/Ida, is use of behavior computation to detect and eliminate often pervasive no-op code from malware packages. Such code in inserted into malware to make analysis more difficult and time consuming. Its elimination leaves only those instructions that have direct functional effects, which can in turn be computed by the system. This system continues to evolve, and is available for use.

![Figure 1. Malware Functionality as a Vulnerability](image)

4. Examples of Malware Behavior Computation

The presentation that accompanies this abstract will illustrate use of the FX system for computing the behavior of malware in several forms, such as:

- Malware containing no obfuscation.
- Malware with repeated control flow obfuscation created by an intruder tool.
- Computation of the behavior of a malware unpacker, and application of that behavior to automatically unpack its malicious code payload.
- Use of computed behavior to eliminate no-op obfuscation code from malware.

In each case the behavior of the malicious code is revealed. These examples illustrate both the capability of computed behavior to blunt the effectiveness of various obfuscation methods as weapons for intruders, and the value of behavior computation at machine speeds for supporting malware analysts.

5. References


