Discovering Designer Intent through Dynamic Binary Analysis of Obfuscated Malcode

Keith Jarrin (RE), Bertrand Sobesto (ENSIB, RE), Michel Cukier (RE, ISR, ECE, CS)

Objective

- Dynamic binary analysis (DBA) of malcode countering encryption, obfuscation transformations and anti-reverse engineering techniques
- Transformation of control/data flow graphs (CFG/DFG) into semantically tagged instruction sequences related to lifecycle behaviors of self-replicating malcode – so-called behavior templates
- Model identification of behavior templates allowing for structural taxonomy of malware
- Model generalization of behavior templates allowing for behavior motifs and malcode phylogenetic comparisons

Functional Components

- **Dynamic Binary Analysis Environment**
  Develop a unique dynamic binary analysis environment for gathering instruction sequences for self-replicating Windows malcode. This environment will be based on a new lightweight emulation technology, which will allow for the execution of Windows malcode on Unix platforms for high fidelity observation. This consists of Free Running Execution (FRE), Conditional Branch Execution (CBE) and Hybrid Static Analysis (HSA).
- **Semantic Threads Database**
  Develop new algorithms to derive semantic threads from instruction traces. Create a database containing semantic threads from malcode execution emulations.
- **Behavior Template Model Specification**
  Develop the means to abstractly specify lifecycle behaviors as sets of semantically-tagged instructions and operations, which occur between them.
- **Behavior Template Model Identification**
  Develop new algorithms using both the behavior templates and the semantic threads database.
- **Behavior Motif Generation**
  Develop the means to extend semantic thread groupings to behavior motifs.

Architecture

The architecture will support workflows and processes to build malcode structural databases from dynamic binary analysis and then mine the databases for behavior templates and behavior motifs.

During the dynamic binary analysis both the self-modifying behavior such as unpacking or unencrypting and the unpacked/unencrypted malcode will be captured. During the mining of the databases, the semantically-tagged instructions and their control/data dependencies will normalize all observed instruction sequences and eliminate all control/data paths involved in obfuscation. Lastly, the lightweight emulation does not invoke any debug registers and cannot therefore be detected by anti-reverse engineering mechanisms.

The architecture allows for dynamic binary analysis to create coarse grain (semantic thread graph) and fine grain features (semantically-tagged instruction sequences and abstract syntax trees). The coarse grain features allow for the identification of high level lifecycle components found in self-replicating malcode.

The fine grain features allow for understanding of the algorithms and data flows each lifecycle component uses. Both coarse and fine grain features can assist in understanding both the unique and common design features of malcode. They represent “templates” which can be applied generically across many classes of malcode. The further development of these coarse grain and fine grain features will lead to a practical means to describe malcode as “templated” behaviors. These “behavior template models” can be used for pattern matching pertinent to unique features when comparing malcode sets. Common features found in collected malcode sets, on the other hand, can be identified through the generation of “motifs” of coarse and fine grain features.

Validation

- Specification of behavior template models based on observed behaviors
- Model identification and matching
- Identify behavior templates and motifs in a wide variety of self-replicating malcode
- Applied to > 50,000 malcode samples