MALWARE MUTATION SOLUTION

STAYING AHEAD OF THE THREAT

SOLUTION VALUE

With the ever-increasing frequency of cyber-attacks across all sectors, next-generation avionics architectures require the ability to adapt and evolve their security posture to keep pace with the rapidly changing threat landscape. The ability to rapidly detect and prevent attacks while adapting security policy in real-time as new threats come online will be critical to maintaining system confidentiality, functionality, and availability.

Technica, in partnership with Vanderbilt University, proposes a framework called the Malware Evolution Engine. This cyber resilience solution consists of two modules:

1. A Binary Lifting Engine that converts executables into an intermediate representation of code that is "analysis friendly," and
2. A Mutation Engine that transforms the malware program based on what is learned from the Binary Lifting Engine to create a different version of the program.

EVASIVE MALWARE

Following the tenets for secure cyber-physical system design and assessment, traditional cyber security experimentation, such as searching for vulnerabilities, often pits one team of cyber security experts against another. The flaw here is a potential mismatch of skillsets for those involved, which may include nation-state-sponsored teams of attackers. Likewise, the current skill set of threat actors does not mean that all threats that can exist already do exist. Instead, it is necessary to explore and evolve malware to remain one step ahead. The need for new malware prevention measures is apparent to the US military, not the least within the US Air Force. Malware can interfere with operations, damage equipment, open security risks, and poses many other risks that must be mitigated.

To build malware detection tools and measure their effectiveness, it is necessary to develop some way to predict new intrusion methods attempts and record them accurately in a comprehensive library of known threats, along with the counter methods to resolve them. The concept of artificial evolution has been applied to numerous real-world applications. Technica IR&D has been researching the application of this concept to computer viruses, and we call this domain “Evolvable Malware”. Technica’s research focuses on Evolutionary Algorithm (EA) techniques to evolve variants of a computer virus that successfully evade popular antivirus scanners. The result is a valid database of malware variants, which is sought by anti-malware scanners, so as to identify the variants before they are released by malware developers.

The security arms race has continually pressured adversaries to develop malware that remains undetected as long as possible once deployed. Categories of evasive malware techniques include:

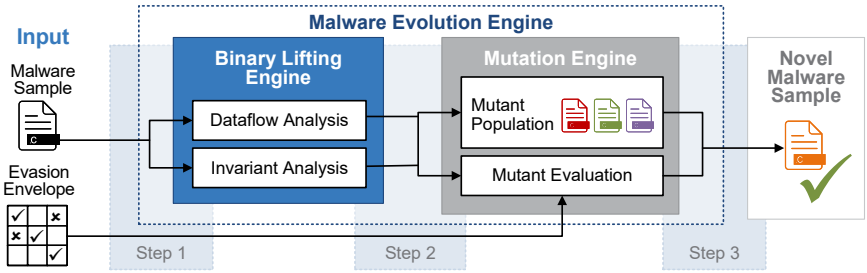
- **Polymorphic and Metamorphic Malware Engines** apply semantics-preserving transformations (e.g., changing instructions to equivalent sequences) primarily to avoid signature-based detection.
- **Packers** encrypt or compress malicious instructions to conceal true behavior. A packed malware sample executes decryption code at runtime to unpack the malicious instructions into dynamic memory, then transfers control to those instructions.

EVASIVE MALWARE (cont.)

As a result, it is not possible to know the behavior of the malware sample without dynamic analysis. Instead, conservative heuristics detect the unpacking code rather than the malicious payload within.

- **Environment Detection** refers to a broad set of runtime checks that malware can execute to deduce the environment in which it executes. For example, a malware analyst may attach a debugger to a malware sample to step through its execution. However, the malware sample can first test if a debugger is attached – if the sample detects a debugger, it can simply abort execution or change its behavior to subvert the analyst.

Technica's malware mutation solution rapidly and effectively devises novel malware samples that evade detection and subvert analysis across a swath of architectures and safety-critical systems. Increasingly sophisticated detection mechanisms are generated by adapting previously successful automated program repair techniques. Figure 1 illustrates our approach at a high level. We begin with a malware sample we wish to evolve to become more evasive and an evasion envelope consisting of detection techniques or mechanisms we wish to evade.



High-level Architecture of Technica's Malware Mutation Solution

In Step 1, the input malware sample is fed to a binary lifting engine that recovers abstract information about the sample, including dataflow analysis to create a Control Flow Graph (CFG) and invariant analysis to identify inputs and outputs of functions within the sample.

In Step 2, the lifted representations are fed into a mutation engine that produces a population of many candidate mutants that are each evaluated against the evasion envelope.

Finally, in Step 3, one or more mutants form a novel malware sample that evades a target fraction of the evasion envelope.

> We can treat the fraction of the evasion envelope that does not detect the mutant as malicious to be a notion of fitness to guide the mutation process. We continue evolving populations of mutants and validating each mutant against the evasion envelope until we find at least one mutant that evades a target fraction of the evasion envelope. That mutant is our desired output sample that has evolved to gain evasive properties while maintaining its original malicious behavior.

The two critical aspects of the solution are thus (1) the binary lifting engine and associated intermediate representations and (2) the mutation engine and integration with the evasion envelope.

BINARY LIFTING ENGINE

Once the user identifies a starting sample to evolve, the binary lifting engine derives key abstract information about the sample:

1. Using tools such as Ghidra, IDA Pro, objdump, or radare2, a CFG is computed for each function defined in the input sample. This CFG is the key structure used to mutate the input sample to defeat signature-based detection mechanisms.
2. Using tools such as DIG or KLEE, a set of invariants is derived for each function defined in the input sample. These invariants represent critical, logical predicates that provide insights into the sample's desired behavior.

While CFGs are rapidly computed from input binaries, invariant detection can take several hours on mid-range workstations. This is because it may require simulating execution or concolic execution of functions within the input. That said, invariant detection is readily parallelized in the cloud or cluster environments.

MUTATION ENGINE

Together, the CFG and invariants for an input sample are fed to the mutation engine to generate novel binary mutants. Given the CFG, a list of candidate transformation sites is generated: branch instructions, comparison instructions, function prologues and epilogues, and loop heads. A fixed number of sites within the code are selected (configurable but expected to be between 500 - 1,000) and then each transformation described below is enumerated to create a new candidate mutant.

- **Inverting Branch Conditions**
  By identifying branch and comparison instructions, we can invert the boolean condition associated with each branch outcome and reorder the branch targets. This will shift the topology of the CFG by changing the order in which branch outcomes are considered at runtime.
- **Adding Superfluous Conditions and Branches**
  Typical binary and malware detection techniques analyze the topology of the CFG. Introducing new branches that have no outcome in the underlying program behavior will add new edges and nodes to the CFG that will preclude many existing detection mechanisms.
- **Adding Superfluous Loads and Stores**
  Binary program code includes interactions with the runtime stack to load and store information computed within a function and to pass parameters. By adding additional push, pop, load, and store instructions to a program, we can influence and disrupt how program analyzers and malware detectors recover function prototype information and call graph edges.
- **Copying CFG Subgraphs**
  Traditional automated program repair involves copying portions of program source code from one site to another. Instead we copy subgraphs of the CFG from one site to another. While this transformation is a higher risk — it will more likely produce nonfunctional mutants, but those functional mutants contain more novel CFG topologies.

SUMMARY

Malicious software often employs a variety of methods to increase analysis time. Unfortunately, such stealthy malware is typically produced by a small set of human-written transformations, making it difficult to analyze by conventional means and for defenders to reconstruct. Technica's Malware Mutation Solution captures the approaches that the user seeks to have when evolving a new and more robust malware sample. This ability to predict future malware can help augment future defensive techniques.