Full Software System Simulations Can Reveal Cyberattack Vulnerabilities That Other Testing Does Not

WNDRVR

EXECUTIVE SUMMARY

Complex embedded systems pose challenges for cyber testing and vulnerability analysis. Testing system interfaces is not sufficient, and dynamic analysis requires instrumentation that makes the process overly complex.

There is an alternative approach: Instrumenting high-fidelity hardware models can offer superior insight into software behavior while also allowing the simulation to be recorded, paused, inspected, and reversed.

TABLE OF CONTENTS

Executive Summary 2
Instrumenting Systems for Cyber Testing: Necessary but Challenging 3
Save Time and Money with Full System Simulation 3
Using RESim for Dynamic Analysis 4
Better Together 4
Simplify When Using Fuzzed Data 5
Case Study: Simulation Workflow for an Example Notional System 6

If not done thoroughly, cybersecurity testing and vulnerability analysis of complex embedded computing software can leave systems open to cyberattack. Simply testing system interfaces, for example, can fail to detect vulnerabilities. Dynamic analysis is an alternative, but it requires instrumentation that can be difficult to deploy, or it may require substantial harnessing. Failure to provide source code or detailed documentation can make it even more difficult to conduct effective cybersecurity testing.

Instrumenting high-fidelity hardware models is a better alternative. This approach gives system designers a better understanding of software behavior, while enabling analysts to record, pause, inspect, and reverse the simulation.

INSTRUMENTING SYSTEMS FOR CYBER TESTING: NECESSARY BUT CHALLENGING

Cybersecurity testing is evolving away from traditional, functional security testing. New testing approaches that involve fuzzing and fault injection tend to be more ad hoc in nature. The goal is to exercise the software interfaces and protocols in an effort to break the software with unexpected data.

With dynamic analysis and cybersecurity testing, teams must observe how software executes under different conditions and determine code coverage. In addition, fuzzing tools require code coverage feedback, so teams know when they're making progress.

Instrumenting embedded systems — adding code to gather data and metrics — is the best way to gain visibility into what's happening in the box. Instrumentation can detect anomalies and see how tests are executing software.

Unfortunately, instrumentation isn't always easy, and it can create side effects. This is particularly true with embedded systems. In these cases, teams usually instrument hardware models of target systems. With this approach, the software runs in a simulation of its native environment.

SAVE TIME AND MONEY WITH FULL SYSTEM SIMULATION

By using full system simulation to create a digital twin of complex systems, teams can automate systems that would be costly and difficult to conduct on physical hardware. Such a digital twin helps identify and address issues before they occur in the real world.

It's possible to simulate hardware systems from chip to system, test in a virtual environment, and replicate system behavior through testing and analysis. This offers advantages over conventional unit testing and static code analysis. Full system simulation improves collaboration and efficiency in embedded development by allowing teams to pause, rewind, and record sessions for future analysis, formal qualification testing, and more.

Wind River® Simics® is a modular simulation system that provides a simulation core, an API, and the ability to dynamically load Simics modules. This architecture supports the creation of additional tools to enhance or create features.

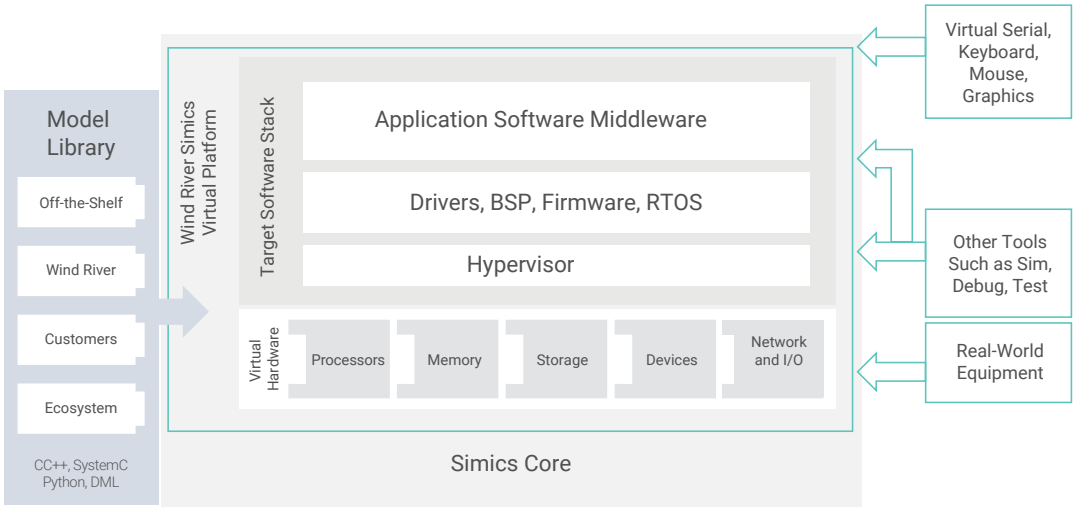

Figure 1. Simics architecture

A Simics model may contain a single device or processor model, or all the models of a system-on-chip (SoC). A model can also include a set of simulator features relevant for a specific use case.

Simics can be integrated with other software modeling frameworks. It can simulate software data or signals to generate complex feedback loops within a virtual system that would be difficult to execute in a real-world environment.

> "Simics has been used by the likes of NASA and the U.S. Navy, as well as aerospace and defense industry leaders. It allows you to create simulation models that save time and money for complex missions and commercial products."
>
> *—Hans Weggeman, Wind River*

Using RESim for Dynamic Analysis

RESim is a reverse-engineering platform that supports dynamic analysis of simulated systems. An open source reverse-engineering platform developed by the Naval Postgraduate School, RESim is a powerful tool that allows users to perform dynamic analysis on simulated systems, including monitoring and modifying system states, tracing execution, and injecting faults.

This platform was originally developed as part of the DARPA Cyber Grand Challenge, and it has since become an essential tool for teams looking to test system behaviors under various conditions that are impossible to replicate with conventional tools.

RESim is designed to meet four objectives:

1. Identify and analyze software that executes on embedded systems.
2. Reverse-engineer protocols used by services on those systems.
3. Discover and analyze potential vulnerabilities.
4. Understand what's happening on unknown, "black box" systems and test them.

Better Together

By using Simics and RESim together, teams can create real-world cyber testing scenarios that accurately represent system behavior.

Simics' full system simulation environment allows teams to model an entire system, including the hardware, firmware, and software. RESim extends Simics, providing enhanced capabilities for reverse-engineering and cyber testing.

When Simics and RESim are combined, it is possible to identify and address potential system vulnerabilities and weaknesses, simulate edge cases, and perform cyber testing efficiently and cost-effectively. Multiple scenarios can be tested in parallel, which requires fewer testing resources than conventional test methods.

For embedded systems, software is run on hardware models, and RESim observes what is happening by instrumenting the hardware. It observes simulated memory and the processor from the other side of the hardware.

This methodology gives teams unique insights into the behavior of the software:

- Full system execution traces build a complete picture of the processes and software programs that are running, as well as the network connections that are made.
- If the system connects to a network or binds to a port, all interprocess communication is visible, such as establishing shared memory or pipes and IPC mechanisms.
- It is possible to perform dynamic analysis of individual processes. RESim's IDA Pro or Ghidra plugins transform Simics into a debugging server that controls execution and provides the ability to run forward or backward to system events.
- Teams can track ingested data and every reference that the application makes to that data. Each reference is a bookmark that analysts can use to jump to the context in the execution environment.
- Data can be injected directly into the application memory, thanks to the integration of the AFL fuzzer into RESim. This enables analysts to run the target to the desired starting state, rather than building test harnesses.

Simics is available from Wind River and RESim can be downloaded from a GitHub page.

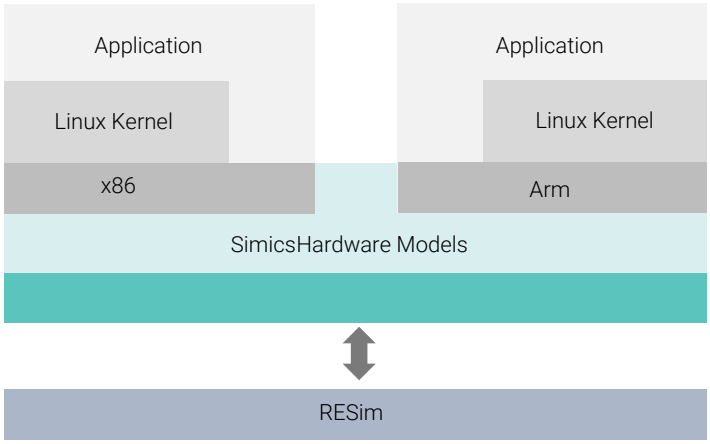

Figure 2. Simics and RESim create a view from the other side of the hardware

Simplify When Using Fuzzed Data

When using fuzzed data, full system simulation eliminates the need to set up harnesses. When teams use fuzzing tools, they often want to monitor the target application's consumption of fuzzed data. However, real applications can be complex; they are often not just a simple service. In addition, analysts may need to interact with the target application to bring it to the desired state for fuzzing.

Simics simplifies these challenging real-world conditions. A simulation can be run exactly to the point where fuzzing should begin, and fuzzed data can be directly injected.

> "Imagine getting a memory corruption error at some point in a program. You can set a breakpoint on the memory that was corrupted, execute backwards, and find the most recent write to that corrupted memory. It makes debugging so much easier."
>
> *—Mike Thompson, Naval Postgraduate School*

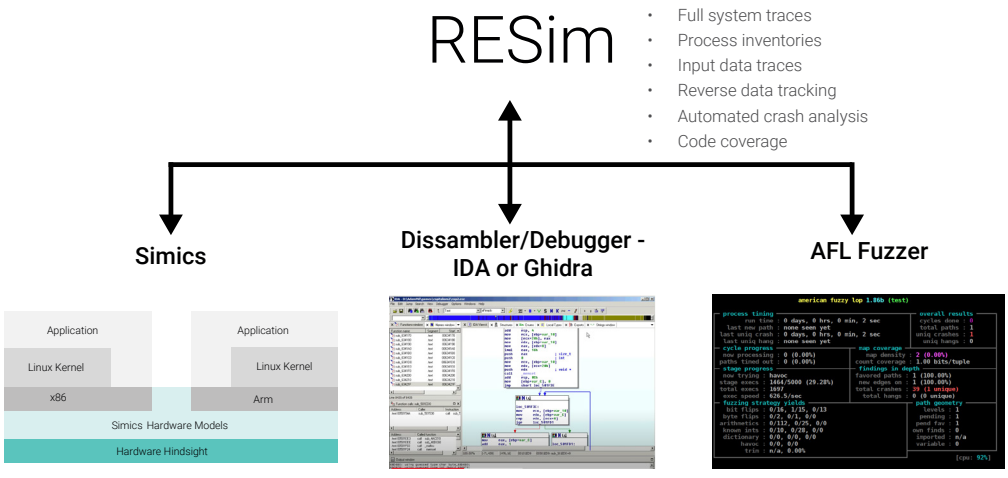

Figure 3. Human-in-the-loop fuzzing

7 STEPS TO INJECT FUZZED DATA

To directly inject fuzzed data into a Simics simulation:

1. Snapshot when the application has returned from a **recv** system call.
2. Instrument the desired basic blocks.
3. Inject fuzzed data into the application buffer and adjust returned size.
4. Unpause the simulation and let the application consume the data.
5. Jump over subsequent read calls and inject more data.
6. Feed instrumentation results back to AFL.
7. Restore snapshot and repeat, using Simics in-memory snapshots.

CASE STUDY: SIMULATION WORKFLOW FOR AN EXAMPLE NOTIONAL SYSTEM

Using RESim to reverse-engineer a system typically requires nine steps:

1. Extract the system software from the targets. Identify the Simics model that represents the hardware or customize a model to host the software. Define a set of virtual network interfaces and set up a driver computer that will interact with the targets.
2. Run a full system trace to identify processes of interest. Usually these are processes with network interfaces that may listen on ports.
3. Direct RESim to debug the selected target process. RESim runs the system forward until the desired process loads, and then the simulation stops. RESim tracks the data read in through a port and determines what the application is doing with that data. Through a driver device, the analyst sends test data to the target port and watches as the application references the data.
4. Use RESim IDA Pro or Ghidra plugins to manage the debug session. The plugins create bookmarks with input data reference points that analysts can use to reverse-track data sources and conduct a first-level analysis of the protocol.
5. Start an AFL fuzzing session using RESim. Based on the results of the initial analysis, the analyst can create input data "seeds." RESim injects the fuzzed data from AFL into simulated RAM and instruments target process basic blocks.
6. Commence AFL sessions at the precise point of data read. The snapshot creation process is fully automated. This eliminates harnessing challenges associated with fuzzing because the software and applications can be run in their native environment.
7. Review the results of the AFL fuzzing session. RESim provides an automated crash analysis that identifies the corruption source. It also replays all "good" AFL results, populates a database with that information, and generates a report on "branches not taken" (BNT).
8. Direct RESim to find inputs for a BNT and load that session. RESim can optionally identify BNT blocks that reference inputs and analyze execution to determine whether the BNT can be reached.
9. Add newly found inputs to fuzzing seeds and repeat the process.

Simics and RESim simulations don't require supercomputer power. The barrier to entry is low. A basic engineering workstation is sufficient to simulate multi-computer networks.

Additionally, it's not necessary to have perfect hardware models. The generic platform models of Simics are often adequate to execute most target software. Before investing time in customizing models, consider the cost-benefit trade-off.

Additional Resources

8 Factors for Effectively Using Simulation at the Intelligent Edge

Full System Simulation: Why Hardware-Based Testing Won't Cut It for Today's DevOps Teams

WNDRVR