Improved Cyber System Digital Twinning, Reverse Engineering, and Vulnerability Analysis through RESim, a High Fidelity, Full System Simulator

February 2022

POC: CDR Chad Bollmann, USN, PhD
cabollma@nps.edu
(831) 656-1819

Technical POC: Mike Thompson
mfthomps@nps.edu
(831) 656-7595

This white paper was selected by a DoD panel as the top response to an RFI on Next-Generation Modeling and Simulation, Testing, and Training Tools from members of the DoD University Consortium for Cybersecurity (DoD UC2)

Executive Summary

Simulated computer hardware can be instrumented to facilitate flexible introspection into the behavior of software and firmware that executes on systems of interest. In this context, a “simulated computer” refers to a high fidelity software model of processors and peripherals. The ability to simulate a full system of networked computers extends that capability, allowing the analyst to pause, step, and reverse the entire system of simulated computers, providing detailed insight into complex software interactions without first building a network testbed and custom test harnesses.

The ability to arbitrarily instrument and control complex computer systems provides an entirely new set of solutions for testing and dynamic analysis of software and its interaction with hardware. Barriers to broad use of full system simulation have included availability of high fidelity software models of hardware and the challenges of divining system behavior through external observation. The latter is similar to the “semantic gap” encountered when using virtual machine introspection, e.g., determining process behavior without invoking the kernel. The Naval Postgraduate School (NPS) Center for Cyber Warfare (CCW) is today using high-fidelity, full system simulation to analyze complex embedded systems with RESim, a platform we have built on the Simics full system simulator.

RESim provides dynamic, detailed insight into processes, programs and data flow within networked Linux-based computers; other operating systems are possible but have not yet been explored. RESim supports reverse engineering of systems of software by instrumenting simulated hardware and observing memory references and processor state. IDA Pro plugins support analysis of process and threads in both the forward and backward directions. RESim is also integrated with the AFL fuzzer to aid testing, vulnerability analysis and code coverage by injecting fuzzed data directly into simulated memory.

Simics includes tools to extend and create high fidelity models of processors and devices, providing a clear path to deploying and managing digital twins. The simulations include optional real-world network interfaces to facilitate integration into networks and test ranges. Simics is a COTS product that runs on commodity hardware and is able to execute several parallel instances of complex multi-component systems on a typical engineering workstation or server. RESim is designed to support remote access to servers running the simulations, and its open architecture can be integrated with existing ecosystems and modular frameworks.

1 Problem Statement

Cyber-relevant modeling and simulation environments can benefit testing, analysis and training efforts by decoupling those studies from the availability, provisioning and deployment of physical hardware. Additionally, simulation of computer systems through the use of high fidelity models facilitates analysis and testing of software through the instrumentation of the modeled hardware. This can be particularly valuable when analyzing heterogeneous systems and binary programs that lack source code and complete documentation. The ability to simulate and instrument an entire complex system can greatly reduce the need for test harness development and maintenance because software is analyzed in its native, multi-component environment rather than being extracted and instrumented within a virtual machine or emulated machine.

Potential barriers to obtaining the benefits of full system simulation include fidelity of the models and scalability of the solutions. Additionally, a high fidelity simulation does not aid analysis unless there are functions that can externally observe machine state, (i.e., memory and processor content) and interpret that into useful information about running processes.

The sections below describe how RESim uses Simics to address these barriers. Future work to expand RESim includes:

- Interpreting Windows operating system and application state (currently RESim supports Linux targets.)
- Development of new hardware models to represent additional OT devices.
- Integrating the Ghidra reverse engineering suite with RESim as an alternative to IDA Pro.
- Broadening the community of RESim users, e.g., by taking advantage of Intel's *Simics Simulator Public Release.*
- Integration of RESIm into multi-institution development networks or test ranges.

2 Innovative Approach

RESim is a platform for reverse engineering, experimentation, and vulnerability analysis of software running on networks of heterogeneous computers. Target software executes on simulated hardware which RESim instruments to analyze process execution. The IDA Pro dis-assembler/debugger is integrated with the simulated hardware for interactive analysis of individual processes and threads, and has the ability to skip to selected execution states (e.g., a reference to an input buffer), and "reverse execution" to reach a breakpoint by appearing to run backwards in time. An integrated custom AFL fuzzer [7] identifies new execution paths and potential vulnerabilities which are assessed using automated crash analysis. An analyst can replay any of the AFL-generated inputs, using the

IDA Pro client to view and analyze crashes and locate execution paths not yet taken.

RESim simulates networks of computers through use of the Simics[1] platform's high fidelity models of processors, peripheral devices (e.g., network interface cards), and memory. The networked simulated computers load and run firmware and software from images extracted from the physical systems being modeled. Instrumenting the simulated hardware allows RESim to observe software behavior from the other side of the hardware, i.e., without affecting its execution. Figure 1 represents the structure of RESim, illustrating how its dynamic analysis is external to the system being analyzed, i.e., the mechanisms do not interact with the hardware interfaces employed by the software being analyzed.

The platform has three primary intended uses:

1. Reverse engineering software on networks of Linux-based systems to learn their protocols and operations,

2. Discovering novel vulnerabilities through human-directed dynamic analysis and/or fuzzing, and

3. Understanding root vulnerabilities exposed by inputs (e.g. the results of fuzzing) that can lead to exploits or crashes. This analysis may occur on networks of systems having complex dependencies affecting the ability to exploit a vulnerability.

RESim and its underlying Simics platform can also be used to create and manage digital twins and to provide an ability to integrate these twins into networks and test ranges.

RESim derives from the *CGC Monitor*, which was developed to vet and analyze software within the US Department of Defense Advanced Research Projects Agency (DARPA) Cyber Grand Challenge (CGC) [3]. The work presented here extends the analysis functions of the CGC Monitor to cover applications running on full Linux systems rather than the minimal API provided by the CGC execution environment. The original work was also extended to analyze ARM platforms in addition to x86 based systems. The new work builds on the analysis features of the original CGC Monitor work.

3 Institution Team Qualifications

This RFI response was coordinated and is endorsed by the NPS NCAE-C point of contact.

The NPS development team created the original CGC Monitor for the DARPA CGC, and evolved that tool into RESim. We have years of experience using Simics products and using RESim to simulate real complex embedded systems.

[1]Simics is a full system simulator sold by Wind River, which holds all relevant trademarks.

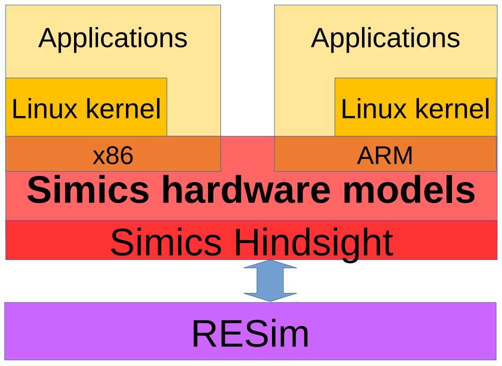

Figure 1: RESim architecture

The Center for Cyber Warfare (CCW) at NPS manages the "Engineering Enclave for Maritime Systems" lab, which replicates computer systems and networks installed on many commercial and military vessels. The CCW has years of experience with testing, analysis, and reverse engineering of real world systems and their communications protocols, and synthesizing this hardware into RESim simulations. The CCW has historically partnered with DoD agencies to investigate specific components and systems of interest. We are interested in partnering with other CAE institutions to expand the capabilities of RESim.

Team members:

- CDR Chad Bollmann – CDR Bollmann is an assistant professor of Electrical Computer Engineering. As Director of the NPS Center for Cyber Warfare he oversees and conducts research into the software and hardware reverse engineering of embedded systems. His other research interests include next generation wireless network security.

- Michael Thompson – Mr. Thompson was a member of the DARPA CGC competition infrastructure development team, and he was responsible for implementing the CGC Monitor to create a digital twin of the game infrastructure as a means of ensuring the integrity of the infrastructure. He has years of experience working with Simics, developing the RESim product and using it to analyze software systems.

- Research teams typically include 1 – 3 active duty duty officers (Navy, Marine Corps, and Army) with information warfare or communications backgrounds.

- Depending on the level of effort and type of investigation being conducted, other CCW faculty with backgrounds in reverse engineering, formal verification, cryptology, and communications serve as Co-PIs or key members of research projects.

4 Background and preliminary work

DARPA created the CGC to push the boundary of technology in "autonomous cyber defense capabilities", able to "...prove the existence of flaws in networked applications, and formulate effective defenses. The performance of these automated systems will be evaluated through head-to-head tournament style competition." [1]. The challenge was framed around vulnerabilities in binary software. As described in [3], the CGC Monitor was used to vet competitor software introduced into the competition infrastructure to help ensure the integrity of the competition. Additionally, the DARPA CGC competition infrastructure team used the CGC Monitor to analyze all successful exploits from the competition, concluding that half of the vulnerabilities were not exploited as intended by the authors of the deliberately flawed services [4]. The CGC Monitor also helped analyze defenses deployed by competitors, demonstrating that defenses which defeated otherwise successful exploits were generic in nature, rather than mitigating specific vulnerabilities.

RESim builds on the core instrumentation functions of the CGC Monitor to reverse engineer and analyze networks of computers. One goal is to aid analysts in understanding the behavior of systems which lack source code and/or detailed documentation. Our approach replaces the physical hardware with simulated hardware that can be liberally instrumented.

Use of a full system simulator simplifies analysis of multi-computer typologies having potentially complex interactions. The simulator lets us checkpoint and return to an entire system state, and lets us reverse the system to a desired event. Use of a full system simulator obviates some challenges associated with creating a test harness for non-trivial applications. Instead of custom software intended to provide target applications with suitable execution environments, we can simply run the entire system to reach the desired state.

4.1 Scaling

RESim and Simics run on commodity hardware platforms. Simics optimisations let us simulate several networked computer systems on a typical engineering workstation or server. Entire system state can be captured in checkpoints and shared with other Simics installations, allowing many instances of the identical system and environment to be shared.

The AFL fuzzing functions integrated with RESim can be scaled to operate on many servers in parallel, with all newly discovered execution paths shared with each server.

4.2 RESim Capabilities

This section describes functions and features of the tool intended to help analysts understand software behavior and identify and analyze potential vulnerabilities.

Dynamic observation of simulated memory and processor state allows RESim to identify which processes are running, what programs they execute, and the data they consume and emit. Interactive functions allow the analyst to progress the simulation forward to selected system calls, e.g., a network connection. And execution can proceed either forwards or backwards to reach memory breakpoints.

The tool focuses on application program behavior, rather than kernel behavior. As discussed in section 4.6, the implementation of RESim does not rely on detailed knowledge of kernel implementations or data structures.

4.2.1 Profiling systems

System traces record each kernel system call and the corresponding parameters, including dereferenced indirect values. Traces can be generated for all processes on all computers within a simulation, or for a selected process and its threads. The system keeps a dynamic tally of inter-process communication mechanisms and network connections and bindings, providing a convenient view of potential communication between processes and systems. RESim also keeps a dynamic process map, identifying process lineage. The CGC Monitor trace function was limited to the seven system calls of the DECREE execution environment; RESim traces all Linux system calls for 32-bit and 64-bit kernels.

The system trace feature can be enabled at initial boot, or at any time during the simulation. Since most Linux system boots involve tens of thousands of file opens, the analyst may choose to progress the simulation until some event (e.g., the creation of a login daemon) before enabling tracing.

When tracing a single process (and its threads), RESim generates a map of the shared object library usage. These shared object maps and other artifacts can be saved along with the Simics system state for reloading at a later time.

A suite of scripts perform post-processing on trace files to create summary reports of process and network activity.

4.2.2 Interactive analysis

A notional reverse engineering workflow starts with generation of system traces which are then assessed to identify processes of interest, e.g., based on network activity. The analyst can direct a simulation to proceed until a named program is loaded, and then to proceed further to other selected events, such as binding to a particular network address. The granularity of event selection criteria ranges from requesting a break on the next system call to the *n*th read operation on a given file descriptor.

Once a process of interest is loaded, the IDA Pro dis-assembler/debugger can attach to a GDB server within Simics. RESim plugins enable IDA to control the simulation, including skipping to selected execution points and reversing execution to breakpoints. The functions described in this section can be controlled either via IDA Pro, or using extensions to the Simics command line.

4.2.3 Data tracking

An input tracking feature records each instance of input using a given file descriptor. For each input system call referencing the given file descriptor, RESim records the address range of the input buffer and then records each memory reference to those buffers. Common Linux memory copy functions are recognized, and references to resulting copies of the input are also tracked. The resulting list of data references is available to aid human analysis of the associated protocols. The analyst can directly modify memory values to observe alternate execution paths, or, for more iteration-intensive investigations, a function is available to reverse to a specified IO event and replace the read buffer content with data from a file. Thus, the *record and replay* feature of RESim allows the analyst to dynamically modify data consumed during iterative replays, and also allows the simulation to continue forward past the end of the current recording.

4.2.4 Reverse data tracking

The providence of data within a memory address or register can be tracked backwards to identify transfers between memory and registers, potentially leading back to the point where the data was ingested. RESim records each transfer point, allowing the analyst to skip to any of those execution states. While RESim cannot unwind complex data manipulations, it does allow the analyst to obverse those points and manually select a new data location for continued reverse tracing.

4.2.5 Output tracking

Data written by target applications to selected files or devices can be captured for review. For example, standard output of a service might be redirected to the null device within the target system, but if directed, RESim will capture a copy to a file on the computer hosting RESim.

4.3 Dynamic modifications

RESim includes functions that dynamically modify memory or system configurations, triggered by system events. These *DMOD* functions trigger on the reading or writing of a specified regular expression via the `write` or `read` system calls (or related calls such as `recv`. An example DMOD file is below:

```
sub_replace read
#
# match
# was
# becomes
root:x:0:0:root
root:x:
root::
```

This DMOD would cause strings from all subsequent read system calls to be compared to the first line in the directive. If a match is found, the string in the second line is replaced by the string in the third line. Thus, an `su` command in a shell would yield root access without knowing the root password.

DMOD functions can also dynamically modify the simulated configuration through use of Simics CLI commands [5]. Consider a computer with two Ethernet interfaces, configured such that the assignment of the names `eth0` and `eth1` are non-deterministic. The analyst desires a deterministic assignment of the interfaces to specific simulated switches. To achieve this, a DMOD function could observe the assignment of MAC addresses to interfaces, and reconfigure connections to the network switches as needed.

4.4 Modeling Considerations

The current version of RESim supports 32-bit and 64-bit X86 and 32-bit ARM. It can also be extended to support alternate architectures, e.g., 64-bit ARM, that are supported by Simics processor models [2]. RESim is currently limited to single-processor (single core) models. Simics supports multi-processor simulations at reduced performance, and this was an option in the CGC Monitor. Multi-processor support has not yet included in RESim, primarily because behavior of most programs is not functionally affected by concurrency.

The user defines simulated systems within RESim configuration files that assign values to predefined Simics scripts to incorporate processors and interface devices (e.g., network cards, disks and system consoles into the simulation) [2]. RESim currently includes:

- A general-purpose X86 platform
- A generic 64-bit ARM platform whose kernel runs the processor in 32 bit mode.
- The *Integrator* ARM platform with a 32-bit processor.

Other platforms can be modeled via Simics and introduced into RESim [6]. Once a system is modeled and referenced by a RESim configuration file, a RESim script is run, naming the configuration file.

Requirements for fidelity of platform models relative to targeted hardware vary by application. We have found that generic platforms can be quite adequate when modeling some real embedded systems. On the other hand, if a target includes kernels with custom drivers for peripherals, then generic platforms

[2] A summary of Simics device models is at: https://www.windriver.com/products/simics/simics-supported-targets.html

would need to be replaced by customized platforms. For ARM systems, this may require models reflecting detailed knowledge of the target device configuration as defined in the target device tree.

4.5 Fuzzing

RESim includes a customized implementation of the AFL fuzzer, in which RESim injects fuzzed input data generated by AFL directly into simulated memory. The user creates a snapshot of the target system at the point at which the application is about to consume data read from a file or network. RESim instruments the target application using Simics breakpoints and provides AFL with a record of basic blocks entered after each iteration, then recommences execution from the snapshot with new fuzzed data from AFL. Use of memory-based snapshots (an unadvertised feature of Simics), enables repeated iterations to insert new inputs without the overhead of reversing execution.

Depending on the nature of the application, RESim injects data into either application memory or into a kernel buffer. The latter is useful for applications that treat the kernel as a buffered input stream, e.g., making system calls to read one byte at a time. RESim automates identification and recording of buffer locations when creating the snapshot. To locate a kernel buffer, RESim employs its reverse data tracing feature to find the source of data read into application memory resulting from the selected system call. This lets us inject data into a kernel buffer without any knowledge of the kernel memory layout.

When injecting data into application memory, RESim detects and skips over subsequent read system calls and dynamically alters the returned count. This enables fuzzing sessions to consume multiple UDP packets.

The use of snapshots and data injection allows us to exclude network connect setup and data transmission from the fuzzing sessions. It also lets us fuzz complex state-dependent systems without creating special purpose test harnesses. We can simply run the full suite of software (and potentially multiple computers) to the desired state and then create a snapshot.

Automated functions allow the analyst eliminate some code paths from the instrumentation, e.g., to avoid tight loops would otherwise slow down the sessions.

Native AFL features for parallel fuzzing allow multiple instances of RESim-AFL pairs to share newly discovered code paths.

4.6 Implementation

The structure of RESim and its dependence on Simics Hindsight [5] breakpoints and callbacks is similar to the CGC Monitor, and is covered extensively in [3]. We subsequently extended that design approach to encompass both 32-bit and 64-bit Linux, and the addition of the ARM instruction set. To summarize, Simics APIs allow the setting of breakpoints on virtual or physical memory addresses. Callback functions associated with each breakpoint execute when the breakpoint is reached. The breakpoints and the callbacks do not affect the target systems, i.e., code within the target is not replaced by interrupt-generating instructions. Where to set breakpoints, when to set them, and what to do when they are hit is, in short, controlled by RESim.

Simics offers similar functionality, known as *Operating System Awareness* within their *Analyzer* package. While that product offers a rich debugging environment for code that you know, we found that reverse engineering of unknown binaries benefits from alternate, extensible strategies for tracking process behavior [3].

The RESim implementation philosophy remains focused on observing the application interface to the kernel, rather than the structures within the kernel. RESim reliance on internal kernel structures is limited to requiring the address of the pointer to the current process task structure and a handful of offsets within the task structure record, e.g., the offset to the `PID` field. When introducing a new kernel image into a simulation, RESim is first configured to apply heuristics to identify and record these kernel values. Different configurations of the same kernel version may have divergent offsets into the task structure. However, some relationships are invariant, (e.g., the `swapper` process is always PID 0, and its first child will be PID 1), and this allows us to tease out the desired offsets. Identifying the address of the pointer to the current task record can be more challenging. However, since there is always a readily available process-local pointer to the process's own task record, we can resort to brute force memory searches for matching values. These heuristics are effective, and allow us to analyze a range of different kernel configurations of unknown providence.

4.7 Availability

RESim is available on GitHub https://github.com/mfthomps/RESim. It requires Simics Hindsight, versions 4.8, 5 or 6. RESim supports IDA Pro versions 6.8 or 7.

References

[1] DARPA. Cyber grand challenge: Rules version 3. http://archive.darpa.mil/CyberGrandChallenge CompetitorSite/Files/CGC Rules 18 Nov 14 Version 3.pdf, Nov 2016.

[2] Naval Postgraduate School. Resim user's guide. https://github.com/mfthomps/RESim/blob/master/docs/RESim-UsersGuide.pdf, 2022.

[3] Michael F Thompson and Timothy Vidas. cgc monitor: A vetting system for the darpa cyber grand challenge. 2018.

[4] T. Vidas, C. Eagle, J. Wright, B. Caswell, M. Thompson, and H. Sorenson. Designing and executing the world's first all-computer hacking competition: A panel with the development team. *Shmoocon*, Jan 2017.

[5] Wind River. *Simics Hindisght User's Guide, 4.8*. 2015.

[6] Wind River. *Simics Model Builder Users Guide Version 5*. 2019.

[7] Michal Zalewski. American fuzzing lop. https://github.com/google/AFL, 2020.