2021 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
CYBERSECURITY OF GROUND SYSTEMS TECHNICAL SESSION
AUGUST 10-12, 2021 - NOVI, MICHIGAN

Advanced Cyber Testing With Virtualization

William Wysocki[1], Greg Price[1], Steve Friedman[1], Adrianne Conage[1]

[1]Raytheon CODEX, Annapolis Junction, MD

ABSTRACT

The growing sophistication and emergence of widespread cyber threats today has driven the DOD to place Cyber Resiliency requirements on new and legacy defense systems. The DOD has recently garnered a massive defensive DevSecOps effort aimed at defining structured practices to unify software (Dev), Security (Sec), and operations (Ops) under the umbrella of more OpSec-driven engineering practices. According to the DOD DevSecOps practicum referenced in this document [1], "Practicing DevSecOps provides demonstrable quality and security improvements over the traditional software lifecycle, enabling application security, secure deployments, and secure operations in close alignment with mission objectives."

Modern systems often contain greater networking capability and are therefore more exposed to cyber-threats. Legacy systems were often conceived prior to the field of cyber warfare maturing, resulting in unpatched potential vulnerabilities that could be exploited through trusting computing relationships. Each type of system presents different Cyber Resiliency requirement challenges. This paper explores the power and flexibility of employing virtualization technology as a tool for cyber-focused testing on both new and legacy defense systems across the DOD.

Citation: William Wysocki, Greg Price, Steve Friedman, Adrianne Conage, "Advanced Cyber Testing With Virtualization", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 10-12, 2021.

1. Introduction

Cyber Resiliency has matured from a passing concern to a requirement driver for multitudes of complex defense systems across the DOD. The sophistication of even moderate cyber threats today requires targeted, multifaceted solutions to address these problems in their entirety. Unfortunately, the development lifecycle, particularly for established legacy systems, is often riddled with unforeseen complexities that make safe modernization slow and resource intensive. It has become common for developers to extend legacy systems by networking them with modern technologies. By doing so, legacy systems have become exposed to modern threats.

Defense systems are commonly created on specialized hardware, making effective cyber testing difficult due to limited availability. System Integration Labs, or SILs, perform the bulk of requirements testing today, verifying correct software behavior of both hardware and software. These SILs are often saturated with tasking during development, performing regular gauntlets of validation testing. To complicate matters, cyber testing uses techniques that may expose hardware to risk of damage and corruption, a risk that many expensive SILs cannot afford.

Risk aversion and limited availability greatly limit the ability to perform effective cyber testing.

Virtualization provides numerous benefits that augment traditional testing infrastructure. Traditional infrastructure is constrained by the number of physical systems available to conduct testing activities. Tests run on real hardware require careful preparation, validation, and sanitation of test pipelines at conclusion ensure integrity and consistency of results many test cases. This cycle of testing can be immensely time consuming, ultimately limiting the scope, throughput, and frequency of tests that can be run within a designated time frame. Virtualization, on the other hand, allows testers and developers to recall known good states from an existing archive, reducing test times, and ensuring the integrity, repeatability and validity of results without the constraints of real hardware.

Virtualization provides data collection and analysis opportunities that are impossible on physical hardware. Snapshots of a hardware state in a virtual environment can be manipulated, deterministically replayed, and analyzed after the system has crashed. Instead of a reviewing a stack traces and memory dumps, a full replay of execution can be saved. This analysis capability is a boon to developers, accelerating error analysis and triage. Virtualization helps standardize these tools and techniques across all virtualized systems, enabling reuse across projects and integration with standard commercial tools.

In the following sections, we will describe how virtualization can greatly aid in improving cyber resiliency across DOD platforms. It breaks the constraints of physical hardware access and eliminates the risk of accidentally damaging or corrupting expensive SILs. It provides greater introspective capabilities than hardware and can be integrated with industry standard tools. Virtualization can be a powerful tool in the effort to reduce cyber risk in today's modern networked environment.

2. Background

2.1. Cyber Resilience

> Cyber Resiliency is defined as the ability to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks or compromises on systems that use or are enabled by cyber resources. Systems with this property are characterized by having security measures "built in" as a foundational part of the architecture and design. Moreover, these systems can withstand cyber-attacks, faults, and failures and can continue to operate even in a degraded or debilitated state, carrying out mission-essential functions, and ensuring that the other aspects of trustworthiness (in particular, safety, and information security) are preserved [2].

Mission-critical systems face an increasing threat from persistent malicious actors with considerable resources and advanced technologies tailored to discover software vulnerabilities. However, according to the GAO [3], "In operational testing, DOD routinely found mission-critical cyber vulnerabilities in systems that were under development, yet program officials GAO met with believed their systems were secure." This has led DOD to place resiliency requirements on new system development and order reviews of legacy systems for cyber resiliency.

Developing cyber-resilient systems requires incorporating cyber-testing technologies early in development, not after the system has been fielded. Failing to engage in cyber testing during development, when subject matter expertise is at its peak, significantly increases resource consumption and task difficulty when assessing and mitigating software errors. Delaying cyber testing beyond requirement testing, integration, and fielding compounds this issue further. Legacy systems suffer immensely from insecure development conventions due to being conceived prior to modern technological trends in networking and connectivity; organizations responsible for maintaining these systems often struggle to understand their security posture as a result. Legacy systems can therefore be regarded as an ideal case study for demonstrating the near impossibility of addressing security after-the-fact.

2.2. Cyber Testing

The NIST Cybersecurity framework [4] presents foundational principles behind the NIST Secure Software Development Framework (SSDF) [5], which provides a framework for integrating security requirements into every step of software development. It states that, when software is unavailable and must be produced, "[testing] executable code to identify vulnerabilities and verify compliance with security requirements" shall occur. The SSDF states projects shall "[integrate] dynamic vulnerability testing", "[use] automated fuzz testing", and "[document] the root cause to each discovered issue".

Dynamically testing embedded or system-level software requires running the compiled software on the target hardware, or in a representative example of that hardware (for example, developer kits, SIL, or emulator). Dynamic cyber testing, or "fuzzing", is typically some form of unstructured test without a well-defined end-state or testing period. This form of testing fundamentally differs from standard requirements testing, as it explicitly attempts to cause unintended behavior. It is extraordinarily difficult to foresee emergent properties of complex systems, which is why unstructured testing has been effective at discovering

vulnerabilities even when a system has strict requirements-based testing.

What is not discussed by NIST is the scale and introspection capabilities required to perform effective dynamic testing. Hardware-based execution environments (developer kits, SILs) are not scalable, nor do they provide the introspection faculties required to execute effective dynamic testing campaigns. This means existing test infrastructure is ill-suited to meet new cyber-testing requirements.

Virtualization provides a clear path to resolve these issues.

2.3. Virtualization

Virtualization is the creation of a "virtual" representation of a system or hardware that can execute software as if it were on the real system. For embedded systems, such as many DOD platforms, this involves using an emulator or hypervisor to execute system software on desktop or server environments. Virtualization platforms provide three key features that real hardware does not.

First, it provides greater system availability when real hardware is expensive, precious, or simply unavailable. As legacy systems age, it is common to experience hardware scarcity issues as components cease being manufactured. Systems under development often experience hardware scarcity issues due to cost of prototypes, manufacturing capacity, and the need for SILs. Virtualization makes execution environments available to all developers for testing on commodity hardware, such as desktops.

Second, virtualization technologies provide interposition capabilities that are unavailable on real hardware. This can include interposing upon execution for coverage collection and intercepting communications between internal components. This additional interposition is often required for advanced cyber-testing techniques, such as coverage-driven fuzzing and dataflow analysis.

Finally, virtualized systems are scalable. Scaling embedded system hardware for testing represents a severe maintenance and cost penalty for programs. Effective cyber testing requires running millions of tests per second, making real hardware a poor choice for both cost and practical reasons.

The following sections discuss how virtualization can be leveraged to perform effective cyber testing and achieve greater cyber resilience.

3. Improving Cyber Resiliency

The current state of cyber resiliency for DOD systems has been extensively documented. The GAO report [3] issued in October 2018 extensively covers the current state of cyber readiness in defense systems, and the root causes of the underlying failures.

> Multiple factors contribute to the current state of DOD weapon systems cybersecurity, including: the increasingly computerized and networked nature of DOD weapons, DOD's past failure to prioritize weapon systems cybersecurity, and DOD's nascent understanding of how best to develop more cyber secure weapon systems. Specifically, DOD weapon systems are more software and IT dependent and more networked than ever before. ... Nevertheless, until recently, DOD did not prioritize cybersecurity in weapon systems acquisitions [3].

There is a growing awareness of the need to standardize cybersecurity assessment practices for software products—much in the same way rigorous security standards currently are applied to software deployed on real-time operating systems (RTOS) in the avionics and automotive industries. As a case study, airborne platforms must now adhere to a set of rigorous testing standards defined by the DO-178B/C [6]. A growing number of federal certifications and testing standards have begun emphasizing security testing as a criterion for being awarded prestigious operational compliance levels. One such set of standards includes the Common Criteria [7].

> The Common Criteria enable an objective evaluation to validate that a particular product or system satisfies a defined set of security requirements [8].

Recently, there have been notable strides in identifying key components for governing more comprehensive system cybersecurity assessment and validation activity. To this aim, the DOD has begun to formalize practices for cybersecurity testing. The DOD has outlined these practices in the DOD Cybersecurity Test and Evaluation Guidebook (T&E).

> The purpose of this guidebook is to promote data-driven, mission-impact-based analysis and assessment methods for cybersecurity and evaluation (T&E), and to support assessment of cybersecurity, system cyber survivability, and operational resilience within a mission context by encouraging planning for tighter integration with traditional system T&E [9].

The DOD publishes a yearly Director, Operational Test and Evaluation [7] (DOT&E)—a report for Congress providing an on-going register of approved operational test plans as part of the joint oversight program. It is evident from the DOT&E report that the DOD continues to bolster its defensive posture by expanding its list of cybersecurity operational testing initiatives.

Despite these requirements, it remains common to engage in cyber testing after development, as opposed to during development. The question is, what is causing this cyber testing gap?

3.1. The Cyber Testing Gap

As noted previously, a critical gap in DOD cyber testing is that penetration and vulnerability assessments are not performed early enough in the product lifecycle. Many programs wait until a formalized test and evaluation (T&E) phase to perform these important tasks—by which time it is too late for a developer to refocus efforts on cyber mitigation activities.

Cyber testing employs strategies that differ from traditional software development processes but require a similar time investment (from design to deployment) that is often not factored into the total software lifecycle. Cyber testing is a path tailored toward improving cyber resiliency and should therefore employ the same tactics an attacker would use to compromise a system in the wild. Effective cyber testing should be iterative in nature. The DOD Cybersecurity Test and Evaluation Guidebook [9] defines a series of phased iterations for conducting cybersecurity verification analysis of system attack surfaces.

> The goal of cybersecurity T&E is to identify and mitigate exploitable system vulnerabilities impacting [the] operational resilience of military capabilities before system deployment to include safety, survivability, and security. Cybersecurity T&E Phases 1 and 2 are the essential first steps of the T&E planning process that support system design and development. Phase 1 and 2 should be performed in a cyclic fashion and repeated throughout each phase to ensure a thorough understanding of the requirements and any changes within the attack surface [9].

This Cybersecurity T&E Guidebook also highlights the importance of engaging cybersecurity professionals at the inception of a product lifecycle as a means of understanding the scope of threats and mitigation strategies to govern subsequent design decisions. By engaging a cybersecurity specialist early, it is easier to generate a custom system profile for conducting future cybersecurity audits, ultimately yielding more comprehensive and intelligent tests around potentially high-risk and critical features of a product. Lack of cybersecurity expertise, engagement, and planning during the system design phase is another notable gap in many development efforts.

While the need to test may seem obvious, cyber testing carries requirements that may appear counterproductive to optimal system development in a traditional environment. Cyber testing increases use of scarce and expensive laboratory resources, and may represent a destructive risk.

With constrained hardware resources, testing becomes a burden to developers, increasing development time and cost. When dependent on a SIL, trade-offs are made to balance time and effort spent in functional testing versus cyber testing due to hardware scarcity. These trade-offs widen the cyber testing gap. It is simply impractical to achieve scaled testing for embedded systems with real hardware, and this puts cutting-edge cyber testing techniques out of reach.

Destructive testing is a software assessment method that involves purposefully interacting with a system using malformed inputs with the intent of discovering errors or failure states. In some cases, destructive testing carries the very probable risk of permanently damaging a target system or hardware. This risk is often unacceptable in cases where hardware resources are constrained.

To achieve effective cyber testing during system development, we must solve the problems of execution environment availability, enable testing at scale, and make offensive vulnerability research methods accessible to system developers. This is exceptionally difficult for legacy systems.

3.2. Legacy Systems in a Cyber World

There are two classes of systems struggling with cyber security requirements today, fielded (legacy) systems and systems under active development. Legacy systems may date back many decades, before the birth of cyber security as a professional field. Many of these systems were developed without security requirements and have significant flaws ripe for exploitation by a malicious actor. Legacy systems undergoing cyber testing have traditionally done so after deployment.

The question is: how do we test legacy embedded systems whose hardware is no longer manufactured (or is otherwise unavailable) and/or does not scale? As we discuss in section 4, cyber testing requires significant scale and interposition capabilities, neither of which may be possible with legacy hardware. By creating virtual representations of these embedded devices, we circumvent the issue of hardware availability, making the device available at any required scale (desktop, range, cloud, etc.).

This solution alone, however, is not a silver bullet. Emulations are expensive to produce, and the cost must be weighed against the expected lifetime of the underlying system. This is especially true of legacy systems, whose schematics and source code may be inaccessible. If the system is scheduled for replacement soon, it may be more cost effective to increase monitoring rather than spend a year creating a virtualization. We discuss this challenge in section 6.1 (Reducing Cost of Virtualization).

3.3. Virtualization and the Future of Secure System Development

Embedded system developers often lack sufficient execution environments to execute the requirement, compliance, and security testing needed to truly assess a

platform's security posture. These developers rely heavily on non-representative environments, such as cross-compiling software for desktop environments. While cross-compilation may satisfy functional testing requirements, it may miss bugs related to the production architecture and hardware.

In some cases, developers may have access to hardware developer kits (HDK) or SILs. When HDKs are available, testing may occur at the desktop scale, but not at an acceptable scale for security testing. SILs compound this limitation, forcing many developers to share access to common hardware for requirements testing. However, SIL test environments will never be completely displaced by virtualization. Siddapureddy [10] explains why many testing requirements demand extremely accurate simulation and structural requirements to determine the survivability of a system in adverse operational conditions. This paper therefore constrains its discussion of SIL test engineering to aspects of software assurance testing and is not suggesting replacement of SILs with virtualization platforms.

Leveraging virtualization in the early phases of system development serves as a major force multiplier, both for velocity of system development and for cyber resiliency requirements. Virtualization makes embedded systems available at the desktop for rapid and scaled test development. This provides developers the chance to build fuzz testing frameworks when key resources, such as system experts, are readily available.

In the following section, we discuss why this type of desktop and scaled availability is critical for test development.

4. Effective Cyber Testing

Threat actors are constantly seeking new attack surfaces and vulnerabilities. Every component that communicates with external systems represents a risk, as it provides a way for external actors to affect internal operation. These actors often leverage fuzzing as a way of exploring attack surfaces and discovering vulnerabilities, in hopes of providing access vectors for their attacks.

Ransomware, for example, requires system access to perform its attack. In recent years, attackers have leveraged ransomware to attack financial institutions, government entities, educational systems, and critical infrastructure. Without a system access vector, these attacks would be difficult to execute. While many attacks use access vectors such as phishing, more sophisticated actors may use undisclosed (zero-day) code execution and privilege escalation vulnerabilities to achieve system access. It is exactly these types of vulnerabilities developers should defensively fuzz for during development.

Defensive fuzzing is a proven method that industry leaders such as Microsoft are using to secure their software. SAGE, a tool that uses fuzzing at scale, has proven results. SAGE found approximately one-third of all the bugs discovered by file fuzzing during the development of Microsoft's Windows 7. Finding these bugs has saved Microsoft millions of dollars, and has saved the world time and energy by avoiding expensive security patches to more than one billion PCs [11]. Fuzzing remains one of the most effective tools in the vulnerability researcher's toolkit for assessing the approximate stability and engineering quality of a piece of software.

4.1. Cyber-Testing-Driven Requirements

Fuzzing is the act of randomly (or pseudo-randomly) exploring the state of a program with generated input (as opposed to known or structured input). Fuzzers require significant scale (millions of tests per second) and feedback (code coverage data) to be effective. American Fuzzy Lop (AFL), for example, uses compile-time coverage faculties for some software, and can use a special emulator mode that is compatible with QEMU. AFL has been used across the industry for dynamic security testing and is considered one of many industry-standard tools.

If we are to apply fuzzing techniques during embedded system development, new requirements must be placed on the overall development process. These requirements are derived from common fuzzing methods and industry standard fuzzing frameworks. The most common problem with applying industry standard tooling is availability of the underlying execution environment.

While AFL can be used with QEMU (an open-source emulator), there is no central team or company responsible for providing developers a QEMU emulator for their platform. To complicate the matter, the open-source community is often wary of working with government-adjacent entities.

In addition, while commodity hardware, such as x86, may have extremely robust onboard hardware-based virtualization and introspection faculties, embedded systems (often ARM-, PPC-, or MIPs-based) simply do not. Lacking these faculties, tooling becomes dependent on whatever emulator is available—and most simply do not implement the introspective capabilities required to meet cyber testing needs.

From this, we derive three major requirements for a virtualization platform truly suitable for cyber-focused testing. First, it must by scalable. Second, it must provide sufficient introspection. Finally, maintenance must be offered as a service, as opposed to being reliant on open-source communities.

4.2. Scalability of Virtualization-Based Testing

We define scaled testing in three ways:

- **Horizontal Scale** – Testing code across many system configurations.
- **Vertical Scale** – Testing code across many inputs or interactions.
- **Distributed Scale** – Testing distributed computations dependent on many components.

In this section we discuss how virtualization can enable each form of testing.

Horizontal Scale

Effective testing requires validating software requirements and behavior on a wide array of hardware and software configurations. For example, a cell phone application may not display identically on two separate phones with different screen sizes. When stakes are low, as is the case with phone applications, cursory testing across a small sample of the many different configurations may suffice.

When stakes are high, as is the case with defense system development, the number of hardware and/or software configurations may be overwhelming to test. Attempts to test all possible runtime configurations is an oppressive requirement and may negatively impact vertical scale testing efforts.

Virtualization makes extensive horizontal testing possible by alleviating hardware availability concerns, and by reducing the time it takes to reconfigure a system under test. For example, reconfiguring a system may simply involve loading a saved state, as opposed to requiring a full hardware reboot.

Vertical Scale

Effective fuzzing campaigns can require execution performance in terms of millions of tests per second. According to Wei Shiyi [12], on evaluating fuzz testing, "the ultimate measure of a fuzzer is the number of distinct bugs that it finds." Some campaigns may require billions of tests in total to identify just a single vulnerability. Hardware-based developer kits and SILs simply cannot support this level of vertical scaling. Figure 1 shows a chart generated from a real fuzzing campaign run against a virtualized flight system interface that plots the number of tests run and crashes discovered.

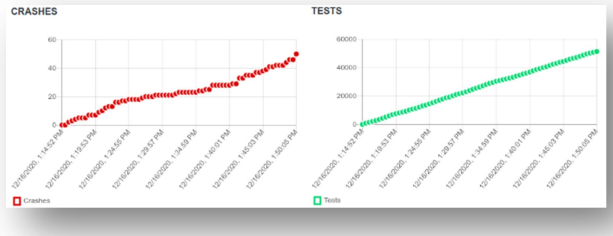

Figure 1. Virtualized flight system fuzz test crash report

Each test may require the system be fully reset to a known-good state. Hardware can often take minutes to fully restart—greatly limiting test throughput. In this case, a one-minute restart cycle would have caused this fuzz campaign to have taken nearly 1000 hours. Instead, executing the tests serially on one piece of hardware, rather than in parallel on many, took only 35 minutes. Acquisition of hardware to achieve sufficient vertical scale is simply infeasible due to cost, availability, and difficulty of maintaining such a hugely complex laboratory environment.

As previously discussed, lacking a representative execution environment at the desktop limits a developer's ability to design and execute scalable unstructured test plans. Virtualization can be deployed at the desktop for test development, and to cloud infrastructure for vertical scaling—limited only by the availability of commodity hardware and server rack space.

Distributed Scale

Modern electronic devices are often complex systems of systems with many discrete compute elements working in parallel to accomplish a goal. Cell phones integrate many commodity components and co-processors to achieve asynchronous computation of data from many external sources (for example, cell tower, Wi-Fi, and touch screen). Similarly, embedded systems often contain an array of co-processors responsible for computing discrete tasks. A fighter jet may have one computer for the cockpit display, another for GPS navigation, and yet more to control weapons systems.

It has become increasingly common for those components to be highly reliant on each other, distributing tasks and executive controls to discrete components of the system of systems. This is good for redundancy and reliability but makes effective dynamic testing outside a SIL extraordinarily difficult. To independently develop these subsystems, developers often produce low-fidelity simulators of missing components—such as scripts that simply replay captured traffic rather than emulate the real behavior.

Low-fidelity simulators lack the ability to replicate true real-world situations, and as a result, effective horizontal scale testing of an individual component becomes infeasible. A virtualized system allows components to be initialized realistically, providing developers an environment to produce meaningful test results.

This topic will be further discussed in section 5 (Testing System of Systems Under Virtualization).

4.3. Interposition and Feedback Fuzzing Under Virtualization

Most secure development frameworks do not specifically address how to design or run effective fuzz tests. Even the NIST SSDF remains vague regarding the definition of "effective dynamic testing". The NIST Information Security guide on Practical Combinatorial Testing [13] comes closest to presenting a comprehensive discussion using the combinatorial testing strategy.

Fuzzing can be considered a class of combinatorial testing that exercises boundary conditions using random inputs. "The key advantage of combinatorial testing derives from the fact that all, or nearly all, software failures appear to involve interactions of only a few [predictable] parameters." [13]

Most secure development frameworks are not designed to be comprehensive one-size-fits-all Swiss army knives. Most frameworks are geared to solving a very narrow and specialized subset of cases within a problem space, or even more disappointingly, simply include a requirement to "*use fuzz testing*". In this section we explore what interposition is required to achieve effective fuzz testing.

Interposition

System interposition is the ability to intervene on a given computational task. For example, software-based code coverage systems inject routines at the beginning of each sub-routine in a piece of software. Hardware-based coverage is accomplished via hardware side channels that detect and record program state-transitions.

Many developer kits provide some level of introspection capability (such as basic debugging tools), but rarely provide the flexible runtime introspection tools required to leverage most modern vulnerability analysis techniques.

Virtualization platforms provide a solution to this problem, because introspection can be baked into the virtualized model software. Emulators translate instructions from one ISA to another (for example, PPC to X86). During this translation process, the emulator may inject additional machine code, which fuzzers can leverage to provide feedback in the form of code coverage and data-flow tracking. Despite emulation running slower than real hardware, emulations can be scaled to deliver a non-linear increase in overall test throughput.

Code Coverage

Many safety-critical embedded systems must meet government certification and compliance testing standards. To achieve certification, testing must generate artifacts that prove the system conforms to the standards. One certification requirement dictates that all code must execute and behave according to a well-defined test plan. In other words, the artifacts must show complete code coverage.

Many test efforts entail developing custom branch trace instrumentation to generate these coverage artifacts. Using instrumented builds for testing presents many risks and disadvantages in practical applications, however. Instrumentation causes significant overhead, often contributing to unnecessary program bloat to the extent that hardware-based tests fail to execute or run to completion—ultimately decreasing confidence in the integrity of the results.

In addition, instrumentation may result in code execution bottlenecks, slow-downs, or race conditions that interrupt normal operation not otherwise present on non-instrumented builds. For real-time embedded systems, timing concerns can affect the stability and health of a system. Without reliable normal operation, test results become unreliable.

Virtualization-based execution environments can provide flexible branch tracing systems that can be tailored to achieve the best performance-introspection dynamic possible. Many virtualization platforms, such as QEMU and DejaVM, provide an instruction instrumentation framework that developers can apply in a more flexible and robust manner than hardware-provided faculties. In addition, a virtual platform can control the perception of time by software and can omit the overhead caused by code coverage faculties from its view. This level of control delivers both the introspection and stability required to ensure the integrity of test results.

Feedback Fuzzing

Most industry-standard fuzzers define "feedback" as some form of execution-state information associated with a particular system input. Branch tracing is one of the primary feedback data sources used by fuzzers. This information associates input mutation with program flow deviations. Embedded hardware rarely provides robust branch tracing capabilities, and commodity hardware tracing faculties often come with severe limitations that prevent vertical scalability.

Some virtualization platforms provide coverage systems that can greatly simplify code coverage collection for fuzz testing campaigns. Rather than running instrumented builds for collection, code coverage is collected dynamically using non-instrumented binary artifacts. This eliminates

significant resource expenditure by development and test teams.

Most hardware-based coverage systems (such as Intel PT) are not capable of providing runtime feedback for analysis during execution. The output must be consumed after the test is completed. Some advanced feedback fuzzing techniques require runtime analysis, making hardware faculties unsuitable for use. Emulation-based virtualization platforms, however, can provide flexible tracing feedback based on the fuzzer's requirements.

To be effective at fuzzing, we therefore require a flexible virtualization system that provides a flexible interposition-performance configuration. This allows testers to achieve the greatest feedback generation and vertical scaling possible.

5. Testing System of Systems Under Virtualization

Expanding cyber resiliency for most relevant DOD platforms requires the ability to connect multiple virtual devices together into a system of systems (SoS) configuration. Traditional integration testing occurs in SILs, and is a critical part of the development lifecycle. SIL resources are constrained, comprising primarily physical hardware connected via control software.

For any test plan to leverage virtualization, it must be possible to duplicate SIL configurations within a virtual environment. This may involve modeling some or all portions of the system in software to replicate system functionality. Configuration management is also a challenge in physical SILs. Systems must be restored to known good states and may require multiple configurations to adequately support testing across the different software versions.

Under virtualization, SoS configuration becomes more manageable. Each relevant set of configurations may be saved and recalled without a complete restart of the system. Each different configuration can then be loaded as needed, and testing of each configuration may even occur concurrently. This reduces test plan complexity, system maintenance, and the amount of resources consumed managing, modifying, and synchronizing configurations across various subsystems prior to staging new tests.

When dealing with SoS configurations, shared state awareness can be valuable for debugging issues. With traditional emulators, leveraging introspection features on a single system within a tightly synchronized environment can create undesirable side effects. For example, pausing an embedded device to debug a software issue without proper regard for system watchdog timers can result in the device perceiving an error state and timing out prematurely.

In traditional hardware development, you would need to periodically reset a watchdog timer to allow the embedded system to continue normally. A virtualized SoS environment may provide a solution for this problem by controlling all participating virtual machines (VMs) at once. Under such a virtualized environment, pausing any one VM within the system context pauses the entire SoS—retaining a synchronized state.

By virtue of having global command and control of a cluster of synchronized VMs, a virtualized SoS makes it possible to extend other advanced debugging capabilities to all VMs. For example, it becomes possible to extend deterministic replay to a networked system of systems. This enables analysts to quickly explore divergent states or easily follow data as it travels through the system of systems.

In a real-world application, a virtual SIL could be used to more effectively scale testing scenarios for various iterations of firmware across a series of target platforms. In the case of a vehicular system, for example, each primary component of the system capable of processing software instructions and affecting the system state would be modeled according to a set of technical specifications regarding the hardware and how the hardware interacts with the software and any dependent systems. The firmware would then be extracted from the actual system and used as a guide for refining the model and identifying any critical functionality that must be modeled from the original system to ensure a measure of operational fidelity. A test plan for the model would be adapted from the system test plan and requirements tests. The model would then be subjected to a battery of code coverage tests in the virtual environment to exercise all critical functions on the platform. The results of these tests would then be evaluated against a pre-existing set of "known good" results logged from actual systems with repeatable "known good" states. This process would ultimately be used as a validity baseline for proving the behavior of the model itself.

The expectation is that models developed for a virtual SIL are capable of running at instruction-level accuracy and are guaranteed to produce the same expected results as an actual well-behaved system given an identical set of data inputs. As a proof of concept, cybersecurity engineers have successfully developed a virtual SIL for exercising the behaviors of primary flight display (PFD) components as part of a flight system. As a test, engineers were able to validate correctness of the model by composing a series unit tests to exercise functionality and verifying the results against data outputs from an actual PFD system.

6. Future Work

6.1. Reducing Cost of Virtualization

Developing emulators is a difficult task that requires hardware knowledge uncommon among most software development teams. Writing new emulators requires a

dedicated team that is familiar with CPU architecture, drivers, reverse engineering, and communication protocols. Advancements are needed to reduce the upfront cost of emulation and to make emulators more accessible for typical development teams.

It is certainly possible to apply machine-learning techniques to emulation model development. However, the platform on which the models would be build would need to support complex machine learning (ML) algorithms, allowing the ML system to modify system configurations to find configurations that correctly execute the target software.

The size and complexity of the emulated model library also impacts the cost of modeling. As additional systems and system components are developed, those components can be reused across new models. As the corpus of models increases, the cost to produce new models decreases. Additionally, virtual model development is a never-ending job. As new hardware systems and features are developed, support for those features must be added to test software that uses them.

For future defense system development, requirements must be placed on hardware developers and providers to provide virtualized models of their hardware to make security testing accessible. In the meantime, the industry must seek boutique virtualization services to provide these models.

6.2. Consistent But Flexible Test-Driven Development Workflow

A virtualized testing infrastructure should be capable of supporting a diverse range of test scenarios and avoid confining an end user to a specific set of testing requirements. Most traditional test infrastructures are highly specialized, requiring a range of custom equipment and techniques geared to ensuring the hardware performs within optimal operational specifications for the parameters provided. For hardware-centric test environments such as SILs, consideration must be given for power constraints, boot times, and initial operational settings that are key to ensuring the results of test runs can be trusted. Generally, these initial hardware states are dictated as part of the test plan, and therefore must be built into the workflow, often as part of some set of manual operating procedures.

For test-driven development employing virtualization, we want to aim instead for a consistent workflow that allows users to quickly build software for their target VM emulations, mock up tests for these emulations within the test development platform, and validate their software against the models as easily as they would on real hardware. Figure 2 illustrates an example test-driven development workflow starting with model development, progressing through the analysis phase, and finally feeding back into the actual deployment after cyber enhancements have been made.

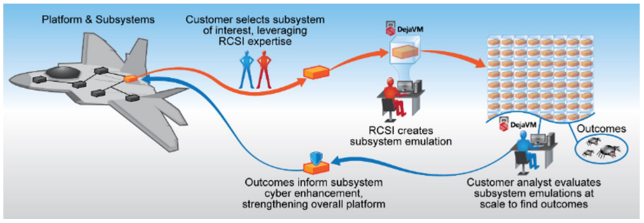

Figure 2. Test-driven development employing virtualization

Virtualized test environments are not necessarily confined to a strict workflow. The challenge then becomes about the economics of testing given an infinite potential. How do we create a solution that is generic enough to be customized and easily integrated into any existing test environment, while still being intuitive for the end-user—we have all dealt with software that traded intrinsic functionality for feature richness. The virtual testing environment is designed to support multiple architectures and platforms configurable, on-demand, from user-defined requirements. The virtual test development workflow should therefore be flexible enough to support these disparate test scenarios, but strive to be intuitive enough for even a novice test writer to get up and running quickly on the platform.

7. Conclusion

Developing systems hardened against modern cyber threats is a difficult problem, and developing secure systems comprising both legacy and modern components has proven impossible to achieve after-the-fact. The authors have presented a case for the benefits of employing virtualization platforms within cyber testing infrastructure to enhance traditional testing environments. Dual-honed efforts are not uncommon as entities both private (financial institutions, education systems, critical infrastructure providers, etc.) and Federal (DOD, Army, NIH, etc.) seek to rapidly improve their cyber security posture. These entities will broaden the scope of the larger, national cyber resiliency campaign.

Our approach focuses on identifying and integrating dynamic testing tools as early as possible in the software development lifecycle. We conclude that by implementing a continuous testing policy and adopting a culture of DevSecOps, cyber-resilient systems arise as a natural byproduct—rendering significant gains in system stability (bugs caught early and often), time factor (faster deployment of patches), and end-user satisfaction (increased reliability and usability).

Software developers should have ready access to testing infrastructure that provides the tools necessary to perform continuous testing across the software development

lifecycle. By requiring development with integrated testing tools occur during prototyping, implementation, and testing phases, developers can generate confidence in their software well before pre-deployment activities commence. We define this form of test integration as "Continual Dynamic Analysis".

A virtualized cyber-testing environment provides a competent middleware solution for cyber testing that can be integrated into most legacy development environments. This approach works well when developers must produce code for highly specialized, scarce, and mission-critical embedded systems such as ASICS. Virtualized test environments enhance the software development process and make it possible to scale testing vertically, horizontally, and for systems of systems. This allows developers to run destructive tests and safely interpose upon system execution in an array of configurations that would prove far less tractable on workstation test benches or System Integration Laboratories.

We allude to the future necessary work to improve the significant upfront costs currently associated with modeling efforts. The costs include gaining a deep understanding of the system being emulated, identifying the system, sub-systems, and components necessary for emulation and debugging the emulation. These steps can require months of dedicated effort and a team of experienced engineers. Despite these challenges, the authors maintain these costs are negligible compared to the billions of dollars lost when critical systems have succumbed to cyber threats.

8. References

[1] "DOD Enterprise DevSecOps Reference Design," Department of Defense Publication, August, 2019.

[2] Ronald S. Ross, Victoria Y. Pillitteri, Richard Graubart, Deborah Bodeau, Rosalie McQuaid, "Developing Cyber Resilient Systems: A Systems Security Engineering Approach," *NIST SP 800-160*, Vol. 2, November, 2019.

[3] United States Government Accountability Office, "Weapon Systems Cybersecurity: DOD Just Beginning to Grapple with Scale of Vulnerabilities," October, 2018.

[4] Matthew P. Barrett, "Framework for Improving Critical Infrastructure Cybersecurity Version 1.1," in *NIST Cybersecurity Framework*, 2018. https://doi.org/10.6028/NIST.CSWP.04162018.

[5] https://csrc.nist.gov/Projects/ssdf.

[6] Parasoft, "DO-178B/C Compliant Software for Airborne Systems," https://www.parasoft.com/solutions/compliance/do-178.

[7] DOT&E (Director, Operational Test and Evaluation) FY 2020 Annual Report. 2020.

[8] https://us-cert.cisa.gov/bsi/articles/best-practices/requirements-engineering/the-common-criteria.

[9] "DOD Cybersecurity Test and Evaluation Guidebook," Department of Defense Publication, February, 2020.

[10] Venu Siddapureddy, Nathan Fountain, David Sanders, Stacy Budzik, "System Integration Laboratory (SIL) is a Key Tool for Establishing and Testing Systems Engineering Discipline," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), Dearborn Michigan, August, 2011.

[11] Patrice Godefroid, Michael Y. Levin, David Molnar. "SAGE: Whitebox Fuzzing for Security Testing," in *Communications of the ACM*, Vol. 55, Number 3, pp 40-44, March 2012. On-line version in ACM Queue 10(1):20, January, 2012.

[12] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, Michael Hicks, "Evaluating Fuzz Testing," in *2018 ACM SIGSAC Conference on Computer and Communications Security* (CCS '18), October 15-19, 2018, Toronto.

[13] Richard D. Kuhn, Raghu Kacker, Yu Lei, "Practical Combinatorial Testing," in *NIST Special Publication*, SP 800-142, October, 2010.