Double Helix and RAVEN: A System for Cyber Fault Tolerance and Recovery

Michele Co, Jack W. Davidson, Jason D. Hiser, John C. Knight, Anh Nguyen-Tuong, Westley Weimer
Department of Computer Science
University of Virginia, Charlottesville, VA 22904
{mc2zk,jwd,hiser,jck,an7s,wrw6y}@virginia.edu

Jonathan Burket, Gregory L. Frazier, Tiffany M. Frazier
Apogee Research
4075 Wilson, Blvd, Arlington, VA 22203
{jonathan.burket,glfrazier,tiff}@apogee-research.com

Bruno Dutertre, Ian Mason, Natarajan Shankar
SRI International, Computer Science Laboratory
333 Ravenswood Avenue, Menlo Park, CA 94025
{bruno,iam,shankar}@sri.com

Stephanie Forrest
Department of Computer Science
University of New Mexico, Albuquerque, NM 87131
forrest@cs.unm.edu

ABSTRACT

Cyber security research has produced numerous artificial diversity techniques such as address space layout randomization, heap randomization, instruction-set randomization, and instruction location randomization. To be most effective, these techniques must be high entropy and secure from information leakage which, in practice, is often difficult to achieve. Indeed, it has been demonstrated that well-funded, determined adversaries can often circumvent these defenses. To allow use of low-entropy diversity, prevent information leakage, and provide provable security against attacks, previous research proposed using low-entropy but carefully structured artificial diversity to create variants of an application and then run these constructed variants within a fault-tolerant environment that runs each variant in parallel and cross check results to detect and mitigate faults. If the variants are carefully constructed, it is possible to prove that certain classes of attack are not possible. This paper presents an overview and status of a cyber fault tolerant system that uses a low overhead multi-variant execution environment and precise static binary analysis and efficient rewriting technology to produce structured variants which allow automated verification techniques to prove security properties of the system. Preliminary results are presented which demonstrate that the system is capable of detecting unknown faults and mitigating attacks.

CCS Concepts

•**Security and privacy** → **Operating systems security; Software and application security;** *Intrusion detection systems; Vulnerability management; Software security engineering;* •**Software and its engineering** → *Compilers;*

CISRC '16, April 05-07, 2016, Oak Ridge, TN, USA
© 2016 ACM. ISBN 978-1-4503-3752-6/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2897795.2897805

1. INTRODUCTION

One of the most widely deployed approaches to cyber security is artificial diversity [1, 4, 17, 18]. Unfortunately, currently deployed diversity approaches suffer from a number of deficiencies and limitations. First, these approaches are probabilistic. Depending on the granularity and associated entropy of the diversity approach, a persistent attacker can compromise the system [9]. If the compromised system is within a trusted enclave, then the attacker might be able to perform other attacks to compromise additional machines. Moreover, current diversity techniques rely on keeping secrets such as randomization keys and code or data location. Derandomizing attacks, probing or side-channel information leakage attacks have time and again broken through diversity defenses. With enough time and determination, a skilled adversary can bootstrap even a small piece of leaked or inferred knowledge into a working exploit [2, 13, 14].

To address these concerns and others, in 2006 some of us proposed an architectural framework, called N-variant systems, for the systematic use of artificial diversity to provide high assurance detection of large classes of attacks [5]. The framework executes a set of automatically diversified variants on the same inputs, and monitors their behavior to detect divergences. See Figure 1.

The diversity is structured so that the exploitation sets of the variants are disjoint. The advantage of this approach is that it requires an attacker to compromise all variants simultaneously with the same input. Because the variants are structured to have disjoint exploitation sets, conduct of large classes of important attacks is impossible. As a simple but powerful example, consider two variants whose text address spaces are disjoint (we call this structured diversity non-overlapping code or NOC). Any attack that depends on a code location (for example an ROP-attack [12]) cannot succeed in both NOC variants simultaneously.

This paper presents the status of a collaboration between Apogee Research and the University of Virginia to build

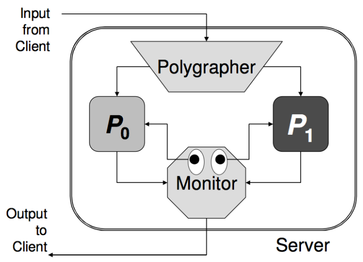

Figure 1: N-Variant Framework (from original USENIX paper [5])

a production-quality cyber-fault-tolerant system based on the N-variant concept as part of DARPA's Cyber Fault-tolerant Attack Recovery (CFAR) program. The program's goal is to apply the power of structured diversity as a defense to protect vulnerable COTS software, achieve low-overhead N-variant execution by exploiting the power of current and future multi-core architectures, and provide the ability to withstand sustained and persistent attacks by supporting the ability to hot-swap new variants to replace potentially compromised variants.

The remainder of this paper is organized as follows. Section 2 briefly describes the design of our CFAR system. The following section gives the current status of the project. Section 4 provides a very brief overview of related work.

2. CFAR DESIGN

The Double Helix and RAVEN CFAR architectures are illustrated in Figure 2. Double Helix processes an application to defend (ATD) and produces a package of variants to be run in the RAVEN multi-variant execution environment. The N-variant bundle includes all the information necessary to run the variants (e.g., program paths, libraries required, etc.), information about the security argument as to which classes of attack are covered, and the policies to apply when an attack is detected (e.g., recovery policies, which new variants to instantiate, etc.). The following sections provide more details about Double Helix and RAVEN.

2.1 Structured Variant Generation

Double Helix processes and diversifies binaries, including libraries, i.e., no source code or debugging information is required and the system can operate on stripped binaries. The ability to operate on binaries provides broad applicability, in particular the ability to process legacy software for which either source code is not available or the translation toolchain is not available because of changing compilers, libraries, or other development assets.

Variants are generated via the process illustrated in Figure 2. Double Helix takes as input an application-to-defend (ATD) that is composed of one or more binary code units (BCU), i.e., an executable file and its dependent shared libraries.

High-precision static binary analysis is performed to gather control-flow and data-flow information that will be used to guide the N-Variant Instantiators. The information gathered during static analysis is stored in the intermediate representation database (IRDB).

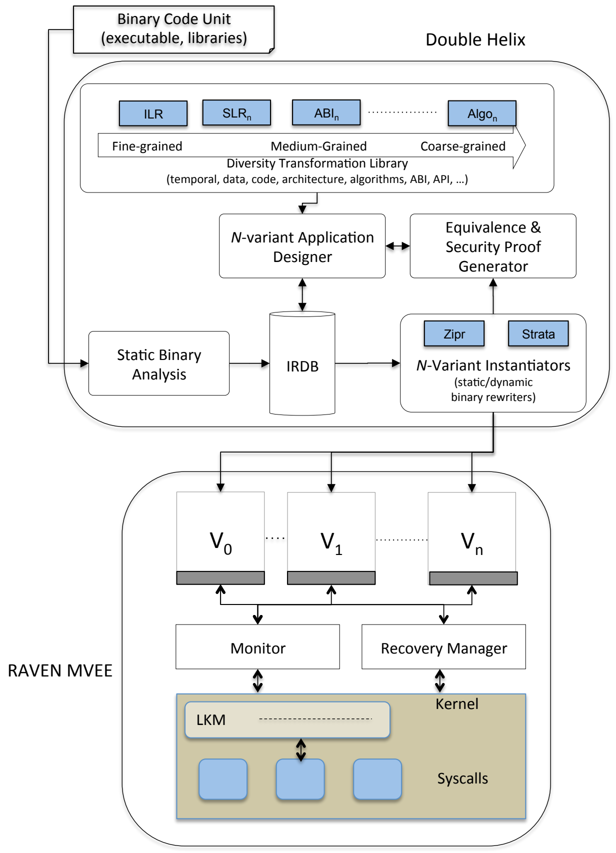
Binary Code Unit (executable, libraries)
Double Helix
ILR
SLR$_n$
ABI$_n$
Algo$_n$
Fine-grained
Medium-Grained
Coarse-grained
Diversity Transformation Library
(temporal, data, code, architecture, algorithms, ABI, API, ...)
N-variant Application Designer
Equivalence & Security Proof Generator
Static Binary Analysis
IRDB
Zipr
Strata
N-Variant Instantiators
(static/dynamic binary rewriters)
V_0
V_1
V_n
Monitor
Recovery Manager
RAVEN MVEE
LKM
Kernel
Syscalls

Figure 2: The architecture of the Double Helix-RAVEN CFAR system.

The IRDB is the information store for the N-variant Application Designer and the N-variant instantiators. It contains information about the instructions that make up the program, their addresses, and their control and data flow. It also contains data about each function (e.g., stack layout, entry and exit points), the global data layout of the program, and other important information such as the targets of the indirect branches.

The Diversity Transform Library contains various fine-grained (machine level), medium-grained (programming language/ application level), and coarse-grained (algorithmic level) transformations that can be applied to create variants. Making use of APIs to access and manipulate information stored in the IRDB, these transformations can be applied either using a probabilistic or structured approach. Fine-grained transformations that are planned include instruction location randomization (ILR), block-level instruction location randomization (BILR), inter- and intra-frame stack layout randomization (SLR), heap randomization, probabilistic control-flow integrity, and structured non-overlapping code (NOC). Details of these transformations can be found elsewhere [5, 6, 8]. Examples of medium-grained diversity include transforms such as calling-sequence diversity, basic-block diversity and promoting stack variables to the heap, while coarse-grained diversity includes the use of different algorithms (e.g., different memory allocation libraries, different implementations, etc.) and generation of variants using a popular automated program repair approach called

GenProg [7].

For each variant, the N-variant Application Designer selects, composes, and applies diversity transformations from the various levels of abstraction in the Diversity Transform Library.

Double Helix provides two variant instantiators—a dynamic binary rewriter (Strata) [11] and a static binary rewriter (Zipr). The choice of which to use depends on the application, the diversity transformation(s) that will be applied, and overhead requirements. Static rewriting incurs little or no run-time space and time overhead, while dynamic rewriting does incur modest amounts. However, dynamic rewriting allows the ATD to be transformed at runtime thereby producing a moving target.

The Equivalence and Security Proof Generator provides assurance that any variant derived by the Diversity Transform Library behaves like the original binary on non-malicious input. The equivalence proof generator combines formal proofs and empirical evidence based on testing to provide proof of functional equivalence. The security proof generator compares the behavior of variants after each has been affected by the same attack input and demonstrates that the variants behave in a detectably different fashion, i.e., they diverge.

2.2 Multi-variant Execution Environment

RAVEN is a cyber-fault-tolerant runtime infrastructure that leverages the parallelism of multi-core CPUs to realize an efficient multi-variant execution environment (MVEE). The MVEE executes multiple variants of an ATD in parallel, ensures containment of a corrupted variant until divergence is detected, and performs divergence detection across the variants. RAVEN's MVEE includes a loadable kernel module (LKM), a user-level monitor process, and a recovery manager.

The LKM intercepts system calls to distribute the inputs to the variants, unifies the variants' outputs, synchronizes the variants, and detects divergence of the variants. Input distribution to the variants and output merging allow the n variants to behave like a single process. If divergence is detected, the Monitor is informed so the appropriate actions can be taken.

The Monitor process launches an ATD by requesting instantiation of n variants of the ATD as child processes, informs the kernel via the LKM which processes comprise the ATD, communicates with the LKM regarding divergence detection, and informs the recovery manager if any of the variants crash or if system call divergence between the variants is detected. RAVEN provides a system call cross-checking API to allow N-variant bundle producers to select the manner in which divergence checking is performed. The bundle producer can use the default system call cross-checking mechanism which performs the check on entry to the system call or specify custom state-similarity cross-checking functions.

The Recovery Manager implements the recovery protocol when a variant crashes or divergence is reported. The recovery protocol includes instructing the Monitor to terminate the designated variant and pause the other variants in the set, selecting a new variant, synthesizing state for it, adding it to the variant set, and then re-activating the variant set. The basic functionality of checkpoint and recovery is provided by Checkpoint/Restore in Userspace (CRIU) [16].

3. CURRENT STATUS AND RESULTS

This project has been underway for eight months, and, at this point, Double Helix can analyze and diversify single-threaded ATDs, and run them in RAVEN. As we approach the end of Phase 1, the focus of the project is: (a) to diversify and execute Apache (including the Apache Portable Runtime library, the Apache Portable Runtime Utility Library, and the Perl 5 Compatible Regular Expression Library (libpcre3-dev)) running in its prefork configuration, and (b) to diversify and execute the `thtppd` web server. In Phase 2, we will tackle multi-threaded ATDs. The rest of this section provides details of the implementation status and an unexpected result.

The following diversity transformations are implemented in Double Helix: structured non-overlapping code, inter-frame stack layout randomization, structured stack canaries, and block-level instruction location randomization. These transformations can be composed and applied to a single binary. Currently only Zipr-produced N-variant bundles run in RAVEN. The use of Strata to apply diversity transforms dynamically uncovered an important design issue that is currently being addressed. Recall that the RAVEN Monitor synchronizes system calls among variants and checks them for divergence. Because the diversifications applied by Strata are applied at runtime, the diversifications can be different. By design, Strata in each variant behaves differently (i.e., executes a different sequence of system calls). Consequently, the RAVEN monitor reports divergence. The solution, now being implemented, is to allow Strata, which is considered part of the trusted base, to turn off cross checking of its system calls.

Checkpoint and recovery is implemented and the RAVEN system is able to checkpoint an ATD and use the ATD to restart. The ability to hot-swap two variants by checkpointing and transferring state to a new variant is planned for Phase 2.

We are building a PVS model of the x86 instruction set to prove functional equivalence of variants. The model currently consists of 2600 lines and 482 proofs. We are currently able to prove equivalence of simple programs.

In terms of the ability of Double Helix and RAVEN to detect attacks, a third party is developing a corpus of ATDs that include vulnerabilities. They also provide "proofs of vulnerability" in the form of inputs that exploit the vulnerability. Our system is able to detect attacks against vulnerabilities that are covered by our diversity transforms. For example, NOC transform detects exploits that rely on absolute addresses. An important aspect of the project is continuous testing using a range of vulnerabilities.

As an interesting demonstration of the power of N-variant systems, we report on an example in which Double Helix uncovered an unknown bug in an ATD. During testing of an ATD that had been diversified using structured canaries, the system reported divergence on a benign input. Our immediate assumption was that the diversity transform had altered the functionality of the ATD. However, analysis of the divergence revealed a subtle bug in the ATD that, when presented with the provided set of benign inputs, did not demonstrate any failures. Fortunately, the diversity transform acted as an "error amplification" mechanism and quickly exposed the bug which was an off-by-4 error combined with a buffer overread and a buffer overwrite vulnerability.

4. RELATED WORK

The N-variant system concept was first proposed by Cox et al. [5]. Subsequent research proposed additional structured diversity transforms and built similar systems [3, 10, 15].

5. ACKNOWLEDGEMENTS

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

6. REFERENCES

[1] E. G. Barrantes, D. H. Ackley, S. Forrest, and D. Stefanović. Randomized instruction set emulation. *ACM Transactions on Information and System Security*, 8(1):3–40, Feb. 2005.

[2] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, and D. Boneh. Hacking blind. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 227–242, Washington, DC, USA, 2014. IEEE Computer Society.

[3] D. Bruschi, L. Cavallaro, and A. Lanzi. Diversified process replicae for defeating memory error exploits. In *IEEE International Conference on Performance, Computing, and Communications Conference*, IPCCC'07, pages 434–441, April 2007.

[4] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 5–5, Berkeley, CA, USA, 1998. USENIX Association.

[5] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: A secretless framework for security through diversity. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[6] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson. ILR: Where'd my gadgets go? In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 571–585, Washington, DC, USA, 2012. IEEE Computer Society.

[7] C. Le Goues, T. V. Nguyen, S. Forrest, and W. Weimer. GenProg: A generic method for automatic software repair. *IEEE Trans. Softw. Eng.*, 38(1):54–72, Jan. 2012.

[8] B. Rodes, A. Nguyen-Tuong, J. D. Hiser, J. C. Knight, M. Co, and J. W. Davidson. Defense against stack-based attacks using speculative stack layout transformation. In S. Qadeer and S. Tasiran, editors, *Runtime Verification*, volume 7687 of *Lecture Notes in Computer Science*, pages 308–313. Springer Berlin Heidelberg, 2013.

[9] G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi. Surgically returning to randomized libC. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 60–69, Washington, DC, USA, 2009. IEEE Computer Society.

[10] B. Salamat, T. Jackson, A. Gal, and M. Franz. Orchestra: Intrusion detection using parallel execution and monitoring of program variants in user-space. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 33–46, New York, NY, USA, 2009. ACM.

[11] K. Scott and J. Davidson. Strata: A software dynamic translation infrastructure. In *IEEE Workshop on Binary Translation*, September 2001.

[12] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 552–561, New York, NY, USA, 2007. ACM.

[13] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 298–307, New York, NY, USA, 2004. ACM.

[14] A. N. Sovarel, D. Evans, and N. Paul. Where's the FEEB? the effectiveness of instruction set randomization. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.

[15] S. Volckaert, B. De Sutter, T. De Baets, and K. De Bosschere. GHUMVEE: Efficient, effective, and flexible replication. In *Proceedings of the 5th International Conference on Foundations and Practice of Security*, FPS'12, pages 261–277, Berlin, Heidelberg, 2013. Springer-Verlag.

[16] Checkpoint/restore in userspace. http://criu.org.

[17] D. Williams, W. Hu, J. W. Davidson, J. D. Hiser, J. C. Knight, and A. Nguyen-Tuong. Security through diversity: Leveraging virtual machine technology. *IEEE Security & Privacy*, 7(1):26–33, Jan.-Feb. 2009.

[18] J. Xu, Z. Kalbarczyk, and R. Iyer. Transparent runtime randomization for security. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pages 260–269, oct. 2003.