A Security Evaluation of a Novel Resilient Web Serving Architecture: Lessons Learned through Industry/Academia Collaboration

Yih Huang and Anup K. Ghosh
Center for Secure Information Systems
George Mason University
4400 University Drive, MS 5B5
Fairfax, VA 22030
huangyih@cs.gmu.edu and aghosh1@gmu.edu

Tom Bracewell and Brian Mastropietro
Integrated Defense Systems
Raytheon Company
2461 S. Clark St Ste. 1000
Arlington, VA 22202
{bracewell, Brian_J_Mastropietro}@raytheon.com

Abstract

We have previously developed a virtualization-based web serving architecture and a prototype to enhance web service resilience under cyber attack. The proposed system utilizes replicated virtual servers managed by a closed-loop feedback controller without humans in the loop. We have replicated the prototype at the Raytheon Company, which conducted a thorough penetration test and security examination. In this paper, we present the Resilient Web Service (RWS) and describe its security evaluation by Raytheon of a prototype implementation. We then present new research directions that address previous weaknesses and discuss the ongoing efforts of designing the next generation RWS architecture.

1. Introduction

Software flaws and vulnerabilities used in network services, including web services, are inevitable. When such flaws are triggered or exploited, corresponding services are compromised or even disrupted entirely. To address the issue, GMU has previously developed and published a novel Resilient Web Service (RWS) [1] framework, shown in Fig. 1. (The same framework was termed Trustworthy Controller in [1].)

RWS uses virtualization with feedback control to achieve fault tolerance, intrusion analysis, security and automatic recovery. It is transparent to the applications it supports, and accesses to application source code are not needed in order to integrate into a RWS-based system.

As shown in Fig. 1, a RWS system runs a pool of virtualized server (VS) replicas on one or more hosts. The server constituents in the VS pool are dynamically brought online or taken offline by the RWS. At any given time, some virtual servers are online serving requests, some are taken offline for reversion to clean state, while others are finishing present requests but not accepting new requests. Each client request is directed by a load balancer, or client dispatcher, in the RWS to one of the virtual servers.

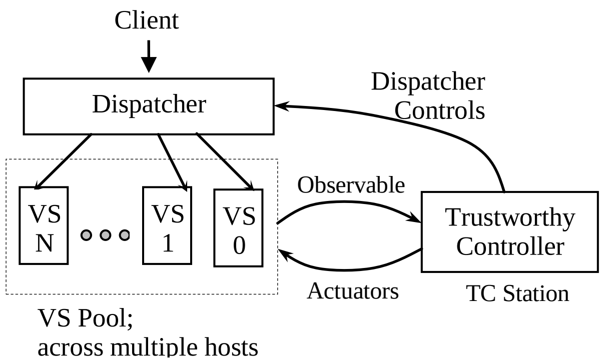

Figure 1. Resilient Web Service (RWS) Architecture

The RWS uses a closed-loop feedback control to monitor and manage the virtual server pool through the Trustworthy Controller (TC) in Fig. 1. Intrusion and anomaly sensors in the network, server boxes, and in each VS report observable status and events, such as server CPU/memory usage, system call behavior and attack alarms. Based on preponderance of evidence in these data, the RWS TC will invoke actuators to address perceived threats and service deterioration. Controls include restarting services in a VS, killing suspicious processes, and reverting virtual servers back to a clean state.

Software rejuvenation is scheduled on a periodic basis to restore virtual servers to their pristine state. While one VS is being reverted, others are still online providing services. By periodically restoring each VS to a pristine state, RWS limits the online exposure of all servers, ensuring that even undetected, successful attacks (false negatives) will be thwarted from being persistent. RWS imposes an upper limit on the VS exposure time; when a VS reaches its exposure time limit, it is reverted even in the absence of negative reports. The accumulation of

This work was partially under AFOSR grant # FA9550-07-1-0527 and DARPA contract # N66001-06-C-2050.

anomaly or intrusion reports from a VS will prompt RWS to revert the VS sooner.

RWS uses virtualization as the foundation for system integrity. In particular, a virtualization feature, called snapshot, is used to periodically revert VSs to pristine state. Unlike system reboots, which remove only memory corruption, VS reverts remove both memory and file system corruptions, including rootkits, backdoors, and Trojan horse programs.

Our prototype also utilizes the off-the-shelf CylantSecure software for intrusion detection. Cylant is an in-kernel behavior-based anomaly detection engine that builds a model of normal system behavior. When a VS deviates from the trained model, Cylant signals the anomaly.

Because every VS in RWS returns to its pristine state periodically, it is stateless by definition. One might intuitively deduce that the limit excludes RWS from stateful services. Fortunately, the industry is generally moving away from stateful servers, lest crashes of individual servers affect user satisfaction. In particular, the REST framework for web application development [2] ensures that all web servers are stateless while still being able to provide stateful services. For this reason, a web application running on a RWS system must be REST conformant, a.k.a. RESTful. The REST framework enjoys wide support, including major server operating systems, such as Windows, Solaris, and Linux. RESTful libraries are available for popular web application languages, such as Java, PHP, Python, Ruby and Perl. It is also supported by major cloud computing services, including Google App Engine, Microsoft Azure and Oracle Sun Cloud.

In this paper, we present the results of a security evaluation of our prototype conducted by the Raytheon Company. Due to its strong connection to military, the company enforces extremely high standards for server security and is experienced with all types of cyber threats. Its evaluation of our prototype provides invaluable information in future research. The paper presents the weaknesses in our RWS prototype discovered by Raytheon. In addition to the vulnerabilities in implementation, the Raytheon team also identified architectural weaknesses in the RWS framework. It is our belief that the lessons presented here are also useful to other research in similar areas. We present remedies to these problems uncovered in the security evaluation and present the second generation design, called RWS-MT, to addresses existing weaknesses.

RWS-MT (Resilient Web Service – Moving Target) will introduce diversity and randomness to several software components of the system to present a moving target to cyber adversaries, including diversifying VS operating systems, virtualization technologies, and web server and application software. Moreover, we will create combinations of these components to exponentially increase diversity. We acknowledge that introducing large numbers of different technologies increases management complexity and will introduce automatic VS management tools and update capability to help ease the problem of managing diversity.

It must be emphasized that the RWS evaluated here does not provide web application security. If a PHP code allows SQL injection attacks, it is considered as an application bug. RWS does not prevent the consequential backend SQL server data corruption. While these problems are mitigated to some extent through diversification in RWS-MT, this paper emphasizes the integrity of frontend, virtualized web servers.

The rest of the paper is organized as follows. We will first review related work in Section II. The results by Raytheon penetration tests are presented in Section III. We will present our plan for the next generation RWS in Section IV. Preliminary assessment of the next generation architecture is presented in Section V. Conclusions and future work are given in Section VI.

2. Related Work

Computer virtualization was first introduced by IBM in 1972 [3]. It has recently experienced a powerful revival. Representative present virtualization technologies include VMware products [4] and Xen [5].

A significant body of work in program instrumentation exists to protect servers, including web servers, against attacks, to recover after attacks, and also to make servers fault tolerant [6, 7, 8, 9]. While these techniques can tolerate the effect of a fault, they can no longer guarantee the session semantics, and the program may no longer operate correctly. For example, with Failure-Oblivious Computing [6], an invalid memory reference that could have crashed the program instead produces a "manufactured" result that at least allows the program to continue execution. However, there is no guarantee that the program will still be in a correct or consistent state.

An issue common to the above techniques is that they require access to source code and are ideally used by the developer of the code, rather than by the acquirer of the server software. While there is prominent open-source web software, such as Apache, many others are not. In contrast, the RWS architecture does not require access to source code. In addition, we are able to provide continuous service with guarantees of integrity.

Various types of randomization/diversification techniques have also been investigated to thwart attacks. Representative examples include instruction set randomization [13], address space layout permutation [14], redundant data diversity [15], design and configuration diversity [16], and randomness in operating system interface [17]. Rather than focusing on a particular aspect of the system for randomization, the RWS-MT seeks to find compatible combinations of off-the-shelf, mature technologies to create diversity and randomness.

The most similar intrusion tolerance solution to RWS is SCIT [10]; they both use virtualization, replicated VS images, reversions, and rotations. There are, however, critical distinctions. First, SCIT uses a simple rotation algorithm to revert VSs at a fixed time interval. In contrast, RWS installs anomaly detection engines in every VS to enable event-driven reversion within a fixed reversion cycle. This allows clean VS to stay online for longer fixed cycle times, resulting in lower overhead – most of the performance degradation is taken during reversions. In addition, fixed reversion cycles define windows of exposure. The longer the cycle time the longer the exposure and the lower the overhead of the system. For shorter windows of exposure, the SCIT system will experience higher overhead and potential user disruption from frequent recycling. RWS does not force reversions when they are not needed, but will provide an upper bound on persistence through a fixed reversion time that can be set according to the throughput needs of the environment. Second, SCIT uses an open source version of Terracotta [11] to share information among servers. Terracotta is a client-server memory clustering service based on the concept of Network Attached Memory (NAM). It must run on a server inside the intranet and will retain states, including corrupted/poisoned ones. SCIT therefore cannot be described as completely stateless and can continue to propagate poisoned states. RWS, in contrast, is stateless because its requirement on web applications to be RESTful. Non-persistent state of the web service is ensured on each reversion. Moreover, our next generation RWS framework will introduce diversification/randomness to VS images.

Related research on intrusion detection systems (IDSs) are outside the scope of this paper, but we emphasize that IDS solutions will either miss attacks (false negatives) or produce too many false alarms (incorrect detections) that tend to "numb" system administrators to real events. By removing humans from the loop and enforcing non-persistence, we address false positives and false negatives with our approach.

3. Tests by Raytheon

3.1. Attacks through conventional vulnerabilities

The GMU team replicated the RWS prototype in a Raytheon lab for the Raytheon team to conduct penetration tests. The penetration tests succeeded in attacks are listed below. All of these exploits required that an attacker first gets a shell on VS through a flaw in Apache. The Cylant detection software was not disabled during the attacks.

1. An attacker can deface a VS. Even though VS reverts will later restore the original content, some users may see the defaced page.
2. An attacker can send any file between the client and a VS. This is a far more serious issue. Once sensitive data are obtained by an adversary, later reverts do not help. Periodic reverts do limit the amount of data to be leaked, such as source codes and large database contents.
3. An attacker can make sensitive data public by combining the above attacks. This is achieved by making sensitive files, such as /etc/passwd, part of the defaced web page.

The Raytheon team advised GMU to remove tools that allow hackers to easily poke around a VS's internals or the whole environment. Those tools include, but not limited to, uname, id, netcat and ping. Additionally, all attacks above relied on a single Apache bug. This demonstrated the value of diversity in web serving software. We will further explore the issue later.

Secondly, we configured the web server such that the software components run with root privilege. This privilege level was unnecessary and was a remnant of our system administration. The principle of least privilege dictates that the software be run with user-level privileges, which would have thwarted these attacks. Moreover, Linux security modules, such as SELinux, can be deployed to further harden all parts of the system.

3.2. RWS-specific weakness

Raytheon also identified weaknesses that are RWS specific in nature, listed below. We will address these issues in the next section.

1. Intrusion detection software within a VS is subject to attacks. In particular the daemons that send Cylant reports can be killed.
2. An attacker can send a properly formatted Cylant profile with forged information to TC. Because the time-to-expire for a VM is calculated using the Cylant distance, a negative Cylant distance can be repeatedly be sent to the TC, which will keep the VS from reaching a state to retire. Thus an infiltrated machine can be kept from reverting while other machines report normal behavior.
3. RSA keys and BASH history files can be stolen. Stealing the RSA keys would allow an attacker to use the keys in attacks on other boxes, while stealing the BASH history is used in attacks to study administrator usage of a machine.
4. Fixed PIDs. In the RWS prototype, web services in a VS were pre-launched before taking the snapshot of an VS. The Raytheon team pointed out that the practice creates identical Apache processes PIDs in all VSs. Even with VS reversions, once a hacker learns the fixed Apache PIDs, the hacker can quickly kill those processes immediately after getting into other VSs.

4. New Directions of Next Generation RWS

In this section we discuss the solutions to the problems observed by the Raytheon team to create the next generation RWS framework, *RWS-MT* (Moving Target). The discussion below does not constitute a complete design yet. Its purpose is to present new directions we intend to explore in the future.

4.1. Hardening RWS-MT infrastructure

One important recommendation from Raytheon is that RWS has to protect not only the system but also itself. Of course, we will in the future apply the best security practice on all RWS-MT components, with properly limited privileges. The following issues are RWS specific.

In additional to in-guest intrusion detection systems (IDS), RWS-MT could deploy hypervisor-level IDS like VMScope or VMware's VMSafe. In-guest IDS could also be further secured using technologies such as SIM [12].

The communications between all RWS-MT components should never be sent in plaintext. In this way, a hacker cannot study the formats of messages viewed over the network. Also, sanity checks should be in place to detect, for instance, negative values of intrusion alarms.

Automatic authentication. One primary design goal of RWS is to tolerate attacks and maintain some level of service availability *without* humans in the loop. For this reason, the TC in Fig. 1 uses *automatic* authentication for all communications with the VSs. This is done by storing TC's SSH public keys in all VSs. If an attacker obtains the key, he or she can connect to VS without authentication. This problem can be resolved by removing the public key file in the VS immediately after the TC authenticates itself and connects to the VS. The same applies to the bash history file and any sensitive information not required in online services. Only after the removal of all such information will the VS serve online requests. We point out the removed files will be restored after the next revert, allowing for automatic authentication in the next cycle.

Recall that all successful attacks in Section III rely on a flaw in Apache to create a BASH shell. Attack kits are available to exploit well-known vulnerabilities such as the one we left in an un-patched version of Apache. In RWS-MT, it is mandatory that update checking be part of the "life cycle" of VSs. If a software package needs upgrade or patching, then affected VSs must be taken off-line for upgrade. As part of our design goal for RWS-MT, patching will be automated without humans in the loop.

4.2. Diversification

Another important aspect of RWS-MT is diversification. This is motivated by the weaknesses in the present prototype caused by monoculture web services and also the desire to present a moving target and increase uncertainty to adversaries that attempt to attack otherwise fairly static systems. Next, we identify the core components to diversify.

Operating systems. Window, Solaris, BSD, and Linux servers have different strengths and weaknesses. The same observation applies to the many variants of BSD and Linux. There is a large array of operating systems one can theoretically build into RWS-MT systems. In practice, however, the task of managing and securing too many different operating systems can be formidable. Presently, we plan to use Windows, Solaris and Linux systems because of their very different characteristics and widely available expertise. While we believe none of these is impenetrable, by presenting different OSs as part of the VSs at a given time, we diversify the attack surface while increasing uncertainty for the attacker on which vulnerabilities will be presented for a particular connection.

Virtualization technologies. Just like any software, virtualization software has vulnerabilities that can be exploited. If there are multiple host machines, they do not have to use the same virtualization technologies to run VSs. One host server could run Xen while another could use VMware. This increases the uncertainty for hackers if they try to exploit the virtualization layer.

Moreover, there are two types of virtualization technologies that are based on very different concepts: hypervisor-based and OS-level virtualization. The former includes Xen, VirtualBox [18] and VMware. Examples of the latter include Solaris Zones and OpenVZ. In some cases you can mix multiple virtualization technologies on one host. For example, a Xen based host can run Linux VSs in Xen, VMware, and OpenVZ at the same time. Moreover, some VSs may run on multiple layers of virtualization. For instance a Linux VS can be a Solaris zone where the Solaris system itself is also a VS running on VMware virtualization. By presenting both diverse as well as layered virtualization, the attack surface area is both further constrained as well as uncertain for attackers.

Web serving software. Besides Apache and IIS, there are several other choices of web server software. Again, each of them has different vulnerabilities and runs on different (sometimes multiple) platforms. Recall that, in all successful attacks reported in Section III.A, an attacker first gets a shell on VS through a flaw in Apache. Before this bug has been fixed, diversification of web servers in RWS will result in different instances of different web servers being presented to client requests.

Web application implementations. This kind of diversity can be achieved by the N-version programming approach in identical or different languages. With different vulnerabilities in different implementations, hackers need to guess the right technique to exploit. Presently, we are in the process of evaluating the effectives and difficulty of this technique.

Randomizations. Another powerful way to increase the level of difficulty for successful cyber attack is to increase unpredictability in the attack surface area for cyber adversaries. Starting with the fixed PID problems discussed in Section 3.2, VS snapshot should not have been taken after the Apache is started. Launching Apache should happen after the VS is reverted but before it goes online, producing unpredictable Apache PIDs. By the same token, there should be randomness in the lifespan (cycle time) of VSs and in the mix of the types of different of virtualizations, operating systems and web serving software. Previously developed techniques for randomizing process memory layouts and/or instruction set opcode can also increase uncertainty for the adversary – thus creating a moving target.

4.3. Challenges

We acknowledge that introducing a large array of technologies and their combinations greatly increases management complexity. Too much complexity to manage may outweigh the benefits of the uncertainty it introduces for cyber adversaries. The challenge is to find the right balance that maximizes diversity and intrusion tolerance with *manageable* efforts to create and maintain such a system. For instance, creating N-versions of web applications may prove more costly than using off-the-shelf components that are already diverse.

5. Preliminary assessment

Our in-progress RWS-MT prototype plans to use four operating systems (Windows, Solaris, RedHat, Ubuntu), two web serving software platforms (IIS and Apache), and three virtualization technologies (VMware, Solaris Zone and OpenVZ). The feasibility of implementation diversity in web applications is under investigation.

5.1. Compatibility issues in diversified components

It would be naïve to believe that O_n operating systems, V_n virtualization technologies, W_n web serving software, I_n implementations produce $N = O_n \times V_n \times W_n \times I_n$ functional VSs. In reality, working combinations N′ will be smaller than N. For instance, IIS runs only on the Windows platforms and does not have the multiplicative effects on overall diversity. In this section, we explore compatibility issue with emphasis on operating systems and virtualization.

Table 1 gives the compatible combinations of VSs running different operating systems and on top of different virtualization technologies. In the table, phrases like "On VMware in a Zone" or "On VMware in OpenVZ" indicate the zone or the OpenVZ residing in Solaris/Linux VMware virtual machines, effectively implying two layers of virtualization.

Table 1. Compatible Combinations of Various Server Platforms and Virtualization Technologies

Windows VS	Solaris VS	RedHat/Ubuntu VS
VMware VS on Windows, and Linux hosts	VMware VS on Windows, and Linux hosts in Solaris host Zone, On VMware in Zone	VMware VS, on Windows, and Linux hosts Zone in Solaris host, OpenVZ in Linux host On VMware and in Zone, On VMware and in OpenVZ

As we can see, there are 2+4+6*2=18 compatible combinations. The above number assumes that Windows servers run IIS and other platforms use Apache. Since Apache also runs on Windows servers, we have two more varieties on the Windows VSs, leading to N′=20 different VSs. Further assuming we replicate the service in two implementations, we have N′=40 different types of VSs, by diversifying four factors in the system: 1) operation systems, 2) virtualization, 3) web serving software, and 4) implementations.

5.2. VS Diversity and attack success probabilities

Assuming that for an attack to succeed, it requires vulnerabilities in K of the above 4 diversified factors existing simultaneously in any one of the N′ different VSs. We show in Table 2 the probabilities for such an attack to succeed. Precisely, we show the value of 1/C(K,N′). In the table, we also *speculate* the probabilities if N′=100. This can be achieved by introducing new platforms such as the many variants of Linux and BSD, as well as new virtualization that is compatible with all above operating systems. One such example is VirtualBox [18].

As one can see in the table, if an attack against a given type of VS requires vulnerabilities in two or more diversified factors, it is very difficult to succeed. The success chance for K=2 is less than 1% even with the most conservative case of N′=20. Attacks that need only one diversified factor to be vulnerable have small but non-negligible chances to succeed. This is the reason why the importance of system hardening as follows from the Raytheon evaluation is important for K=1. In Raytheon's evaluation, the attack against Apache (K=1) was successful in creating a bash shell. A shell (script) may or may not do real harm to the system. Real damage described in Section 3 would be difficult had the recommended OS hardening measures been implemented in our first prototype, effectively increasing K to 2.

Table 2. Attack Success Probabilities by Exploiting K Diversified Factors (All in a Column VSs are Different).

K	20 VSs	40 VSs	100 VSs
1	5.0000000%	2.5000000%	1.0000000%
2	0.5263158%	0.1282051%	0.0202020%
3	0.0877193%	0.0101215%	0.0006184%
4	N/A	0.0010942%	0.00002550%

It must be emphasized that the above figures are rudimentary. On one hand, it ignores the cases where one attack method may compromise more than one type of VSs. For instance, an attack exploiting bugs in a particular version of Apache will succeed, at least in the first step, in all VSs using the same version of Apache regardless the underlying OS and virtualization method. Moreover, an attacker typically tries more than one attack kits/paths. On the other hand, enforcing diversity in presented VS images mitigates this concern. Similarly, the presented figures ignore the beneficial impact of anomaly detection, intrusion detection systems, and our feedback-control system.

6. Conclusion and Future Work

With the help from Raytheon, the prototype RWS intrusion tolerance framework has gone through stringent security evaluation and penetration tests. Together we have learned and presented practical lessons in both prototype implementation and architectural design. Drawing upon these lessons, we have assessed a next generation architecture for RWS based on diversification. We presented a rudimentary framework for analytically evaluating the benefits of introducing diversity against cyber adversaries. This paper described a university and industry collaboration that utilized core strengths of both organizations to improve the efficacy of a design and implementation.

Our next step is to evaluate the most effective way to increase and automate diversification. Management complexity is also a critical factor. From the above calculation of the different types of VSs, we found that implementation diversity is most powerful in increasing different VSs – M implementations increase the diversity by M times. This assumes the implementation is platform neutral. However, it would be very difficult to maintain multiple implementations of an application with exactly the same behavior. This is especially true if the web applications continue to evolve, as they always do. We will explore automatic diversification techniques to aid in creating multiple diverse images of functionally equivalent software systems.

In general, technologies that have multiplicative effects on overall diversity N' are preferred over additive solutions. For instance, VirtualBox is another virtualization technology compatible with all above operating systems. Adopting it doubles the overall diversity N' -- it is multiplicative. As a counter example, consider a new application implementation based on the .NET platform. It is limited to Windows VSs and makes only additive contribution to N'. Our ongoing research is to investigate those multiplicative factors for maximum benefit and minimal management complexity.

7. References

[1] Yih Huang, Anup K. Ghosh, "Automating Intrusion Response via Virtualization for Realizing Uninterruptible Web Services," Eighth IEEE International Symposium on Network Computing and Applications (NCA ‘09), 2009.

[2] Fielding, R. T. and Taylor, R. N. 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technology. 2, 2 (May. 2002), 115-150.

[3] R. J. Creasy. The origin of the VM/370 time-sharing system. IBM J. Research and Development, 25(5):483-490, September 1981.

[4] VMware, http://www.vmware.com.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, “Xen and the Art of Virtualization,” Proceedings of the nineteenth ACM symposium on Operating systems principles, October 2003, NY, USA.

[6] Rinard, M., C. Cadar, D. Dumitran, D. Roy, T. Leu, and J.W. Beebee, “Enhancing server availability and security through failure-oblivious computing,” in Proceedings of the 6th Symposium on OSDI, December 2004.

[7] Sidiroglou, M.E. Locasto, S.W. Boyd and A. Keromytis, “Building a Reactive Immune System for Software Services,” in Proceedings of the USENIX Technical Conference, 2000.

[8] Qin, F., J. Tucek, J. Sundaresan, and Y. Zhou, “Rx: treating bugs as allergies – a safe method to survive software failures,” in Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP), pp. 235-248, 2005.

[9] Sidiroglou, S., O. Laadan, A. Keromytis, “Using Rescue points to Navigate Software Recovery (Short Paper),” in Proceedings of the IEEE Symposium on Security & Privacy, pp. 273-278, May 2007, Oakland, CA.

[10] Yih Huang, David Arsenault, and Arun Sood. “Incorruptible System Self-Cleansing for Intrusion Tolerance”. Performance, Computing, and Communications Conference, IPCCC 2006.

[11] The Terracotta project. http://www.terrracotta.org

[12] Sharif, M., Lee, W., Ciu, W., & Lanzi, A. “Secure In-VM Monitoring Using Hardware Virtualization.” Paper presented at the 16th ACM Conference on Computer and Communications Security, Chicago, IL, 2009.

[13] G. Kc, A. Keromytis, V. Prevelakis. Countering Code Injection Attacks with Instruction Set Randomization. ACM Computer and Communications Security. 2003.

[14] Kil, C., Jun, J., Bookholt, C., Xu, J., and Ning, P. 2006. Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software. In *Proceedings of ACSAC’06, 2006*.

[15] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox, and J. W. Davidson. “Security through redundant data diversity.” In 38th IEEE/IFPF International Conference on Dependable Systems and Networks (DSN’08), Anchorage, Alaska, USA, 2008.

[16] A. Bessani, A. Daidone, I. Gashi, R. Obelheiro, P. Sousa and V. Stankovic. Enhancing Fault/Intrusion Tolerance through Design and Configuration Diversity. 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS 2009).

[17] M. Chew and D. Song. Mitigating Buffer Overflows by Operating System Randomization. Tech Report CMUCS-02-197. December 2002.

[18] VirtualBox. http://www.virtualbox.org/