Advanced Adaptive Application (A3) Environment- Initial Experience[1]

Partha Pal, Rick Schantz, Aaron Paulos
Raytheon BBN Technologies
Cambridge, MA
{ppal,schantz,apaulos}@bbn.com

John Regehr, Mike Hibler
University of Utah
Salt Lake City, Utah
{jregehr, mike}@flux.utah.edu

ABSTRACT

In this paper, we describe the prevention-focused and adaptive middleware mechanisms implemented as part of the Advanced Adaptive Applications (A3) Environment that we are developing as a near-application and application-focused cyber-defense technology under the DARPA **C**lean-slate design of **R**esilient, **A**daptive, **S**ecure **H**osts (CRASH) program.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication, Invasive software, Unauthorized access;*

D.2.0 [**Software Engineering**]: General—*Protection mechanisms*

General Terms

Security

Keywords

Execution Environment, Middleware, Preventive Adaptation, Innate Immunity, Survivable Applications.

1. INTRODUCTION AND MOTIVATION

The current way of running applications on host platforms often impedes cyber-defense. Multiple applications share the physical host and the OS. Isolation techniques like SELinux [1] exist, but because of implicit sharing of various host resources, the security policies frequently are not tight enough and as a result, a compromise in one of the applications often leads to disruption or corruption in the operation of other collocated applications. Stronger isolation technologies such as separation kernels [2] although available, are primarily used to enforce separation between multiple levels of security, and not among applications within individual security domains. In addition, application's interactions with the environment through the network, storage system or the user interface (UI) also take place in shared spaces. This makes it difficult to tightly monitor application behavior and enforce application specific controls, and resulted in various rings of *perimeter* security—at the network boundary or at the host boundary—that monitor and control the *aggregate* of multiple protectorate constituents. Furthermore, many of the existing perimeter security techniques such as firewalls, OS or process level security policies, anti-virus and intrusion detection and prevention systems are signature-oriented making them ineffective against novel attacks.

Middleware 2011 Industry Track, December 12th, 2011, Lisbon, Portugal.
Copyright 2011 ACM 978-1-4503-1074-1/11/12...$10.00

As recent reports [3] indicate, adversaries are still succeeding in getting through the perimeter defenses. In most cases, it is the applications that run on the hosts and the data these applications manage that are the target of these attacks. We argue that no matter how secured the perimeter or the OS is, applications with complex logic, structure and interactions will still have flaws. And such flaws will be discovered and exploited by the adversary who will often gain access and privilege in the network and host environment via social engineering and compromising collocated enclaves, hosts and applications with weaker security. Security measures *near or at the* application that go beyond detection and prevention, and aim to tolerate the impact caused by unknown and unforeseen attacks are therefore urgently needed.

In this paper we introduce the Advanced Adaptive Applications (A3) Environment, an innovative middleware designed for defending individual applications against novel attacks. The A3 environment is a *middleware* because it mediates the protected application's execution and interaction with the physical host resources such as the disk, network and UI devices. *Adaptation* is a major underlying theme of A3's defenses, which in the context of survivability, ranges from graceful degradation to recovery, and to changing the system so that successful past attacks do not succeed anymore. A3 carries forward our prior successes in adaptive defense and survivability research [4,5,6,7] which assumes that no defense is absolute, attacks will happen and often succeed; and argues that although adaptation is key to survival, successful defense must include prevention focused defenses observation mechanisms designed to pick up undesirable conditions as well. This paper will primarily focus on the foundational aspects of A3, and will offer a deep dive into the prevention focused defensive capability. Prevention-focused defense is one of the three main defensive capabilities of A3, the other two being recovery techniques based on advanced state

[1] This work is being supported by the Unites States Air Force and DARPA under Contract No. FA8750-10-C-0242. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

management and replay with modification to improve the application's defense based on past successes and failures.

The main contributions of this paper are: 1) a special breed of execution-containing security-focused adaptive middleware that mediates the protected application's interactions with the environment, b) a framework to structure and impose prevention focused adaptive control on an application's interaction with the environment, c) a foundation for novel recovery and replay based improvement, and d) initial results establishing the feasibility of effective and efficient implementation of innovative, near-application and application-centric defenses.

A3 technology is being developed as part of the DARPA CRASH program, which pursues innovative R&D into the design of new computer systems that are highly resistant to cyber-attack, can adapt after a successful attack to continue rendering useful services, learn from previous attacks how to guard against and cope with future attacks, and can repair themselves after attacks have succeeded. Complementing the application-level and application-focused approach taken by A3, a number of other efforts in the CRASH program are developing techniques for security enhanced processor architectures (e.g., tagged instruction and execution), OS based security techniques (e.g., information flow control), and programming language and compiler technologies (e.g., randomizing compiler to produce variants with different vulnerability profiles, security focused invariants and assertions that can be embedded in the application during development and enforced at runtime).

2. DESIGN AND IMPLEMENTATION

The key idea underpinning the A3 environment is to isolate individual applications into dedicated containers such that (i) application-specific defensive adaptations do not interfere with the operation of other applications and (ii) all interactions of the protected applications can be subject to mandatory mediation. We argue that if the application executable is *pure* (i.e., it may contain vulnerability, but is not corrupted with attack code) at inception, the only way it can be compromised is through its interactions with the environment i.e., via disk storage, network and user interfaces. Unfortunately, in a modern general purpose computing platform, the interface between an *application* and the *environment* has gotten out of control with touch points at many known and unknown surfaces. Isolation of an application in a container enables us to organize the applications interactions into storage, network and UI *channels*, which in turn enables us to put up *crumple zones* that subject these channels to mandatory mediation and act as buffers that absorb the initial blow of attacks (and potentially *crumple*), preventing the attack from reaching the protected application.

A3 prevention-focused defenses are concentrated in the Crumple Zones (CZs). The CZs essentially impose a *space-time dilation* upon the application's interaction: the interactions are intercepted, and can be watched, analyzed, processed and transformed in the defense's timeframe, changing the equation for the attacker—the attacker no longer has the advantage of hiding in a general purpose host running various applications and services. Instead, the A3 container is dedicated to the single application and the attacker has to play by the rules of the protected application.

A3 recovery-focused defenses leverage the isolation of the protected applications into dedicated containers and stands on the hypothesis that not all parts of the application's state is equally important. In particular, some state information is absolutely crucial and needs to be retained and the rest can be discarded, or recreated from other saved information. This differential treatment of state information needed for recovering a crashed or compromised application is at the core of A3's Advanced State Management (ASM) that enables different flavors micro-reboots with different timeliness and consistency profiles on top of standard reboot and rollback based recovery options.

For adaptive immunity, i.e., the ability to improve the defense over time, A3 relies on *Replay with Modification* (RwM) — a capability that enables us to roll back the protected application to a past state, modify the protected application (e.g., a new variant) or its security configuration (e.g., the inspection and transformation based rules in the crumple zones), and perform replay-based experiments. If the recorded events contained an attack, i.e., triggered a vulnerability that compromised the crumple zone or the protected application, the experiments are used to determine an alternate configuration of the protected application that does not suffer the same outcome.

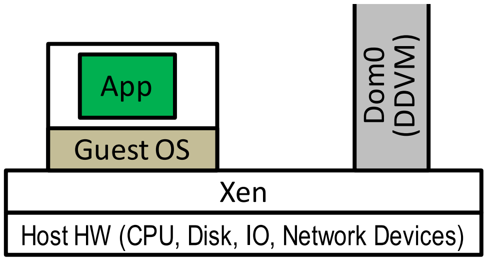

Figure 1: Guest VM as A3 Container

In the first year of the four year project, we have prototyped the basic containerization mechanism, and the storage and network crumple zones. We are currently working on ASM and RwM. The Crumple Zones and container-isolated applications along with the ASM and RwM capabilities collectively form the envisioned A3 environment—where the mediated channels and crumple zones prevent the protected application from falling victim of attacks, ASM enables faster and diversified recovery when the application does succumb to failure and compromise, and RwM facilitates changes in the configuration that prevent the protected application from succumbing to the same attacks.

2.1 Basic Containerized Isolation

A3 uses *virtualization,* specifically the Xen hypervisor as the basic containerization mechanism. An A3 container running a protected application will be a Dom U (guest) VM (see Figure 1), with its network, UI and storage channels logically connected to the VM that runs the device drivers that manage the hardware devices. In Xen, this could be the Dom0 VM, or one or more specialized DomUs running the device drivers as advocated by security enhanced operating systems like Qubes [8] and L4 [9]. In our current prototype Dom0 is the designated Device Driver VM (DDVM). In the future, we plan to put the device drivers managing physical devices in their individual DDVMs separate from Dom0.

Using a container VM to encapsulate the protected application implies that the application has the impression of having an entire machine to itself including its own (virtual) disk. But disk storage is also used frequently by applications to share information—for instance, a file created by Word can subsequently be used by an email application when the user wants to send the file to someone by email. Or, in order to read a file received by email, another application such as Word needs to be launched. If Word and email client applications run in their own dedicated container VMs, the file created by Word will remain in the container running word, and the file received by the email client will remain in the container running the email client application. Note that under A3, each application runs in its own dedicated container and therefore, there needs to be a way to share files between the two containers. More specifically, this points to the need for synchronizing the virtual disks of different container VMs. This is a unique issue for the storage channel because of the semantic difference between virtualizing a storage device like a disk and a network interface or an IO device: whereas a virtual network interface or IO device primarily acts as a multiplexer, the virtual disk also acts a (longer term) buffer. The A3 storage channel therefore manages the mapping and synchronization of physical disk content that are shared across multiple applications into the respective virtual disks by using a commit mechanism which enforces the following policy: after N (configurable) number of updates, the storage channel commits the virtual disk and remaps and remounts the virtual disks (of other VMs) that share the same file systems of the physical disk. The value of N dictates the synchronization delay.

2.2 Mediated Channels and Crumple Zones

Even though the UI, Network and Storage channels logically connect the A3 Container and the DDVM running the device drivers managing physical devices, the actual path is through the hypervisor. Therefore, extending the hypervisor is one possibility to implement the channel mediation and the crumple zones. However, since the mediation policies are highly application-specific, this approach would require building/configuring a custom hypervisor for each protected application. Furthermore crumple zones are expected to fail (i.e., *crumple* under attack) under attack, which in the case of hypervisor-extension approach, will threaten the integrity and liveness of the hypervisor. To avoid these issues we designed the bulk of the crumple zones functionality outside the hypervisor—specifically, CZs are implemented as interposed VMs—the mediation policy and controls are either implemented within the VM containing the protected application or in individual VMs that are part of the A3 conglomerate representing and acting as the protected application.

Interposing a crumple zone VM relies on Xen's basic inter-VM communication technique of using a circular buffer in a shared memory page. The circular buffer connects the device driver in the guest VM (known as the front-end driver) with the device driver in the DDVM that is responsible for managing the physical device (known as the back-end driver). For storage and UI channels, only one circular buffer is used, whereas for network channels a pair of buffers is used. The sharing of the memory pages is implemented by "grant tables" and strictly controlled *share* or *transfer* primitives. In this grant table based paradigm, sharing or transfer of data cannot be done without one side first making a hypervisor call. Our VM-based implementation of containers rely on the trustworthiness of the virtualization mechanism, i.e., the hypervisor responsible creating VMs, assigning and managing VM identities is treated as part of the trusted computing base (TCB), the grant tables and shared memory circular buffer between the front-end device driver at the guest VM and the back-end device driver (that manages the actual hardware devices) at the DDVM provides a fairly strong non-bypassable way to mediate channel interaction outside of the hypervisor. A CZ VM can be inserted in front of the Dom0 DDVM such that an A3 container VM trying to use the real hardware devices (via the back-end drivers in the DDVM) must go through the CZ VM. Because the intercepted traffic is now available to a VM, we are not limited to looking at the contents of the circular memory buffers—our mediation policies can inspect, interpret and process the intercepted information at various levels of the system stack.

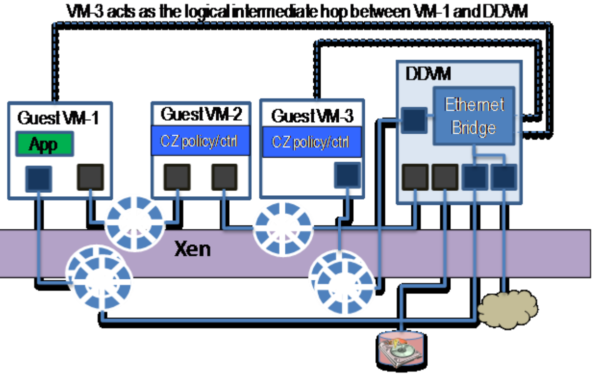

Figure 2: Insertion of Storage and Network CZ

Figure 2 shows the A3 conglomerate for protecting an illustrative application. Guest VM-1 is the container running the protected application (APPVM henceforth), Guest VM-2 is the storage CZVM and Guest VM-3 is the network CZVM. The storage CZVM and the corresponding split-pair device drivers (shown in grey) at DDVM form the storage channel, and the network CZVM and the corresponding device drivers (shown in blue) in the DDVM form the network channel. Note that although each of the CZVMs introduces an additional circular buffer indirection in the original paths connecting the APPVM and the DDVM, there is a slight difference. The storage CZVM connects with the APPVM presenting a backend driver and also with the DDVM presenting a front end driver. Whereas the network CZVM behaves much like the APPVM in the sense that it only has a front end driver, and connects only with the DDVM. The network channel is designed in this way to take advantage of the Xen's standard networking infrastructure (the Ethernet Bridge and supporting mechanisms at the DDVM): in a sense, the network CZVM acts like an intermediate hop between the APPVM and the DDVM.

2.3 The I/E/T Prevention Framework

To facilitate easy conception, formulation and enforcement of application-specific policy and control that can prevent entire classes of novel attacks we developed a framework to organize the mediation policies that can be enforced in our crumple zones. Apart from thwarting novel attacks for which a-priori known signatures do not exist, the other key design goal of this framework is to provide a structure that can support a generic

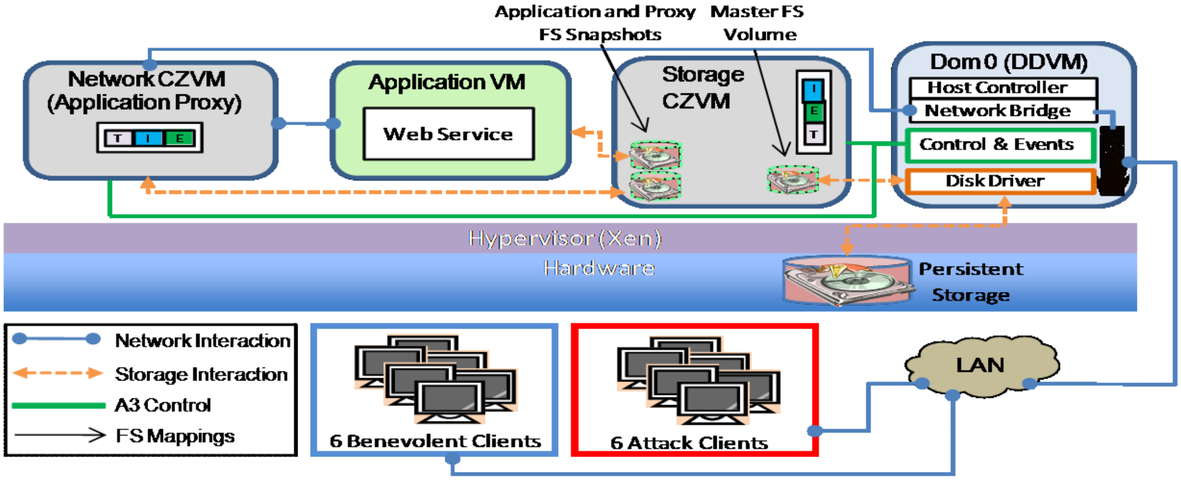

Figure 3: Illustrative Web Server Application in A3

application by accommodating custom specializations. We argue the constraints and consequent reactive adaptations in the following three categories cover a wide range of novel attack avenues.

- Inspect (I): Inspection based policy and control include computing aggregate properties (e.g., rate, size, patterns) of interaction and subjecting them to application-specific operating ranges. This category also includes filtering based on known signatures, and observing the side effects of execution-based policies that are described next.
- Execute (E): This category supports execution or processing of intercepted interaction to determine whether the interaction would cause any known undesired effect on the protected application. If so, such interactions should not be released. On the lighter extreme, E policies may constitute straightforward middleware functions such as marshalling/demarshalling, serializing/deserializing. On the heavier extreme, E policies engage a copy of the protected application executing and responding to the stimuli received on the mediated channels (with appropriate buffering to eliminate spurious side effects) *before* the stimuli are released to the real protected application. As mentioned earlier, I policies may observe the execution of E policies, implementing the cyber security analog of a *try before you buy* or the *7 second profanity delay*.
- Transform (T): Analogous to the way transformation from time domain (amplitude over time) to frequency domain reveals and enables filtering unwanted noise in signal processing, we argue attacks that exploit data format or protocol flaws can be thwarted by data format or protocol conversion. Such transformations need to be semantics preserving and as a corollary application specific. The common input validation techniques currently in use to defend against SQL injection, CSRF and XSS attacks transform parts of the incoming request—which are inbound interactions on the network channel for A3. In addition, a number of modern applications do not have a pre-conceived notion of data format—for example, a search may return anything from text, PDF, word, spreadsheet, audio and video. Service-oriented applications (e.g., web services using WSDL) often are able to negotiate the nature and format of the data exchange at runtime. In these cases, application-specific transformation from one protocol or data format to another is acceptable, and will either eliminate or disrupt the embedded attack code or data. Even with traditional applications where the protocol and data format is not flexible, transformation from one format to another and back to the original will be useful for defense against embedded attacks. This is because in most cases the attacks exploit the vendor-added features or lapses in the specification. If the transformations strictly impose the protocol/data format specifications, malicious elements like embedded scripts can be eliminated or sufficiently disturbed to render then useless. Of course, this scheme may impact the application's operation if the application relies on vendor-added features.

2.4 Current Prototype

Figure 3 shows the A3 test environment where the APPVM runs an illustrative web server application protected by the two CZVMs providing application-specific and prevention-focused adaptive defense capability. As shown, the web server can be invoked (HTTP request) by a browser as well as non-browser clients to manage documents. The server supports multiple users, each authenticates using a token, and can perform a list, upload, download and delete operations on a file store. The server is expected to maintain the file store within a specified directory—and is not expected to write user submitted content to any other directory. The server is also expected to enforce ownership in the sense that a user cannot delete another user's files, but they can openly share. However, the server implementation is inherently buggy, and it is possible for a malicious user to traverse the directory structure, write files in arbitrary places, rename files, execute cross site scripting, upload and execute arbitrary executables etc. This application does not use human interaction, so only the network and storage channels are relevant.

The storage crumple zone that we used for this application enforces the policy and control described below:

Policy item	Type	Control Action and notes
Path traversal	C ; Inspect	Block requests that try
Write outside	I; Inspect	Block requests that try
Read outside	C; Inspect	Block requests that try
Size limit	A; Inspect	Block requests with large data
Rate limit (blocks read or written)	A; Execute	Abort the write or read that involves too many bytes, thwarting novel attacks that cause infinite read or write loop
Content control	I; Inspect	Block requests that include executables and scripts
Renaming	I; Inspect	Block requests that try

The 2nd column describes the nature of the mediation, i.e., the security attribute (C for Confidentiality, I for Integrity and A for Availability) that would suffer without the policy along with whether the policy is based on Inspection, Execution and Transformation. The 3rd column describes the control actions.

The policy and control imposed by the network crumple zone are similarly described by the following table:

Policy item	Type	Control Action and notes
Fingerprinting	C; Inspect	Drop or "wash" response, thwarts application and transport layer probing
Argument filtering	C, I and A; Inspect	Block requests, stopping application level requests to change directory, exec., malformed requests, arguments
Rate enforcement	A, Inspect and Execute	Block requests, based on source based and aggregate number of packets and requests
Protocol transform	C, A and I; Transform	Normalize requests into a well tested library, thwarting novel attacks that exploit browser and vendor specific extensions
Try before you accept	C, A and I; Execute and Inspect	Block requests, thwarts novel attacks that inject undesired and out of range application behavior

The policies enforced at the crumple zones are usually a mix of generic and application specific policies. The generic policies are applicable to a class of applications (e.g., all web service applications) and can take application specific parameters. Rate and size checking, validation and sanitization of inputs are examples. Application specific policies are can be provided by various stakeholders such as the application developer or the application deployer or the user; and can also be determined by human experts by empirical observation. A specific installation of the web service-based document management application described in this section may have the T policy to transform all word documents into pdfs, and disallow opening of outbound socket connections. The E policy of try-before-accept is a curious mix of generic and specific: the generic aspect is that for any protected application, one needs an application proxy or code that partially emulates the application in one of the crumple zones. At the same time, the proxy, by nature is application specific, for example, in our illustrative application it is essentially a replica of the application running in the network crumple zone. It is also worth noting that E policy is usually complemented by its accompanying I policies that monitor the execution of the E policy (i.e., watches over the proxy). Some of these I policies can be highly generic—such as death the proxy or code performing the execution/processing of channel events, while the others can be application specific such as, watching for the frequency of specific files being down loaded or the frequency of delete operations performed by a individual users.

3. EVALUATION TO DATE

We used a Dell Latitude D820 laptop, an Intel Centrino Duo clocked at 2.33Ghz and 2GB of memory connected to a single router along with a client PC as our experimental set up. The D820 is used to host an illustrative web service application protected by A3 with the APPVM and network and storage VMs as shown in Figure 3, the client laptop is used for sending both benevolent and attack HTTP requests to the protected web-service. We use Xen 3.1.4 with Fedora Core 8 (kernel 2.6.18.8) images for both Dom0 and three DomUs in the prototype A3 environment. Each DomU is configured to have one CPU and 512MB of memory.

For the initial assessment of the overhead associated with A3 crumple zones, we recorded six unique client interactions with a web-service over a 10 minute window and used the recordings to drive our protected web service. Each client represents a different client OS, web-browser and usage-pattern. The usage pattern is roughly categorized as: (i) a batch-bulk download every 2 minutes, (ii) a file download every minute, (iii) *random* heavy burst of eight mixed operations lasting between 10 to 20 seconds, (iv) a upload-download-delete cycle (.2Hz), (v) a constant fast-clicking refresher (.3Hz), and (vi) a heavy uploader every minute. Clients submit and retrieve files ranging from 9KB to 476KB and of multiple mime-types. Taken as a whole, the clients driving the protected web service present a load equivalent of a small-office wiki. The results of the performance assessment experiments are described in Section 3.1

Objective evaluation of how effective a defense mechanism is has always been a challenge, and our experience in evaluating the effectiveness of A3 is no exception. The most credible validation by the community is independent red team experiments, which we expect to undertake later in the project, however, red team evaluation has its own limitations such as the motivation, expertise and resources of the red team and the rules of engagement used in the experiments. Clearly, the answer to questions like *has A3 made the illustrative application completely invincible to attacks* is no. In fact, A3 alone cannot achieve that goal. However, the crumple zones do make attacks on the application that make use or exploit the storage and network channels more difficult to actually affect the application. In the preliminary evaluation we sought to validate that claim by taking the following approach. First, starting with the semantics of the web service application, we developed a number of I/E/T policies that we deployed in the network and storage CZ, and subjected

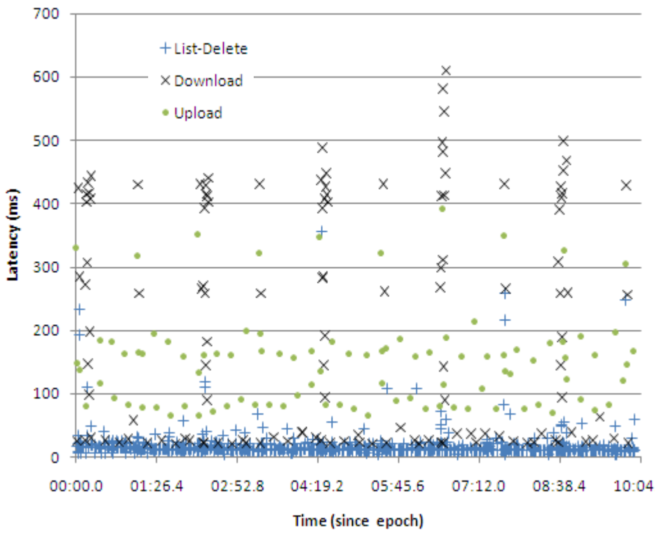

Figure 4: Baseline Latency

both the protected and unprotected application to requests with appropriate privilege that attempt to break the application or cause undesired behavior or effect in the system. Second, we conducted tests where we injected failures into the crumple zones, emulating the effects of novel attacks to observe how the protected application behaves. Section 3.2 elaborates.

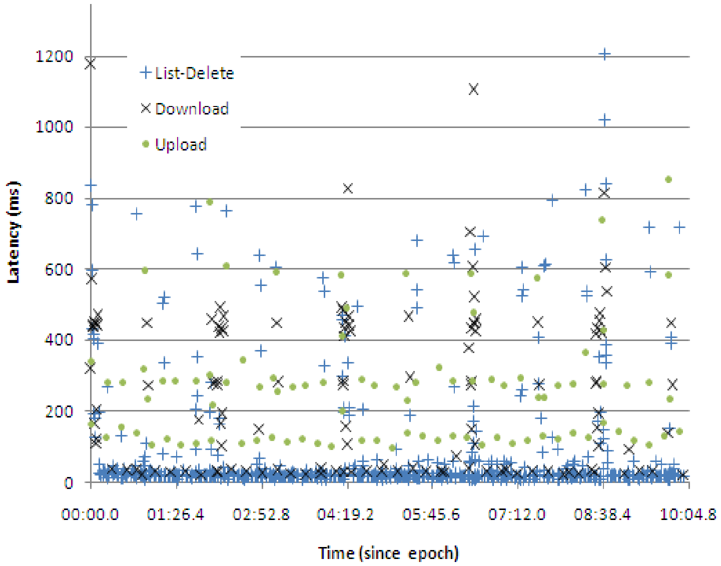

Figure 5: Latency with A3

3.1 Efficiency

Figure 4 and Figure 5 show the client latencies of the unprotected baseline application and the A3 protected application respectively. The protected application uses the default un-optimized A3 policies (all policies and virtual disk committed after every update event). Round-trip latencies are collected from clients replaying the six profiles and are grouped into three categories: list and delete; download and upload. This grouping separates data-heavy ingress and egress flow (*upload/download*) from light-weight requests (*list files*, get *upload form* and *delete a file)*.

The average latency aggregating the different categories is 50.2ms for the baseline, compared to 64.9ms with A3 protection. In other words, the default non-optimized A3 CZs introduce a 29% overhead under this specific load. This overhead may seem low considering that the proxy crumple zone is fully executing and reverse proxying the request to the real application, but we note that the network latency may mask some of the overhead introduced by A3.

We examine Figure 5 to identify which component of A3 contributes most to A3's overhead, and observe an outlier group of list-delete operations and a few spurious download and upload operations. Over both runs, list-delete, download and upload make 80%, 12% and 8% of the total requests respectively. While the average baseline list-delete latencies are 16.4ms (compared with 64.6ms for A3), over 11% of the A3 list-delete outliers have latencies greater than 100ms.

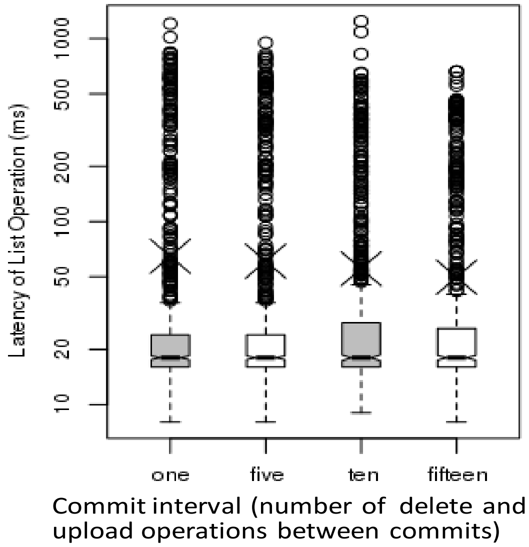

Figure 6: Impact of Commit Interval

In the current prototype, the storage channel commits the virtual disk of the Network CZ host running the application proxy by rsyncing the application proxy's stored-filed directory to persistent storage (the rsync duration for this fixed traffic volume is a consistent 250ms) after a configurable number of operations which modify the application state, here, *upload and delete operations*. At the beginning of the commit sequence, a firewall rule is introduced to block incoming client requests and ultimately wait to reach request-quiescence before starting the rsync. To evaluate A3's sensitivity to commit intervals, we ran four experiments varying the commit interval between one and fifteen as shown in the box-plots in Figure 6. The box-plots show the distribution of client latencies, the mean latency (large X per series) and a trail of outliers for each evaluation.

From the analysis, we notice average latencies decrease (64.6ms, 60.7ms, 55.2ms 49.6ms) as the check pointing interval grows. Likewise, the severity of the outliers decrease as the commit interval grows. We believe that both of these trends, when considered against the relative high volume (i.e., 80% of all requests) of list-delete operations, explain why list-delete

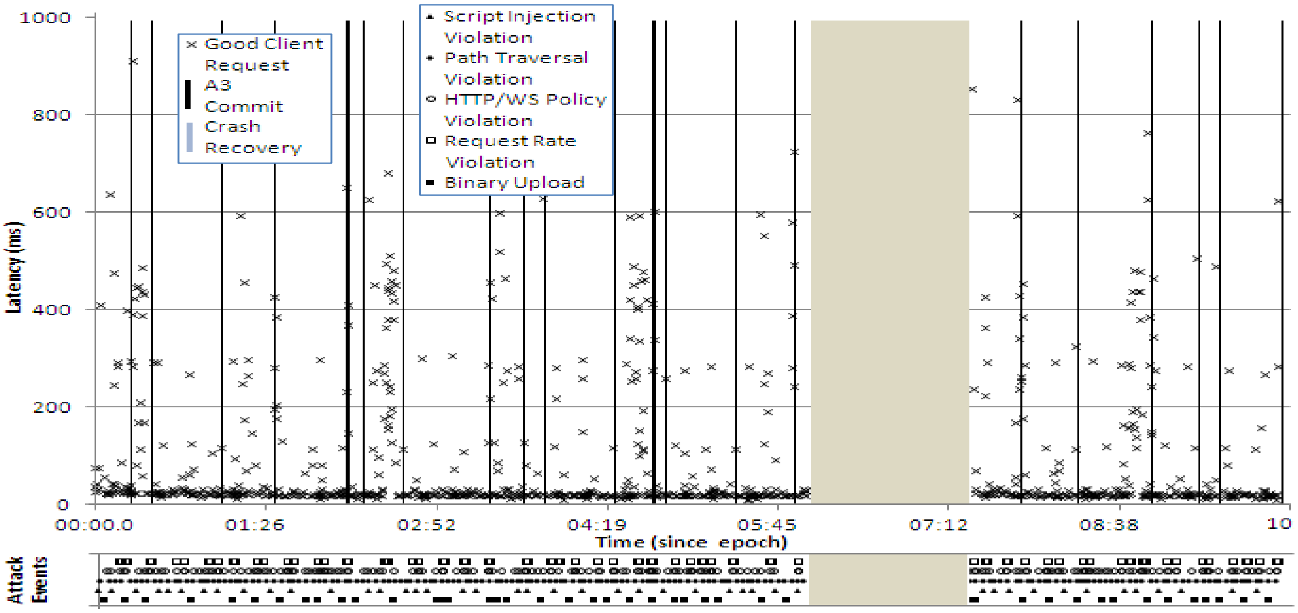

Figure 7: A3-protected Application under Attack

operations fair so poorly for our A3 application. In effect, list-delete operations are more likely to be blocked when a commit occurs, thus resulting in a higher perceived latency.

3.2 Effectiveness

Using a commit interval of 15 modification events and the client mix as described earlier, we introduced six random attacking clients to exercise A3 CZ defenses and record client overheads under attack scenarios. During a ten minute experiment, we used six attack clients attempting a variety of attacks: (i) script injection to compromise future legitimate requests, (ii) path traversal to exfiltrate and infiltrate data, (iii) random fuzzing to scan for buffer overruns and host fingerprinting, (iv) resource exhaustion, (v) upload and execution of binary files and (vi) a fault-injection client that triggers a synthetic crash in the application proxy running the network CZ during the sixth minute. Against our naive web-service without A3's protections the same attacks result in cross-site script attacks, removal of readable host configuration files (e.g., private encryption keys, /etc/iptables), replacement of writable files, fingerprinting via thrown exceptions, resource starvation, and execution of arbitrary shell code and attack binaries. We also verified that the failure injected at the application proxy will also kill the application if injected at the application itself. Figure 7 captures the results.

The top half of Figure 7 shows a time-series showing the latencies for the six benevolent clients, commit durations, and the crash roll-back recovery duration from the fault-injection. As can be seen in the figure, other than the interval around the 6th minute, the benevolent clients did not suffer any loss of service. In the bottom half of the figure, we group and plot each policy violation in time as reported from the A3 host. Speaking generally, the A3 host blocked binary file uploads and path traversal attempts in the storage crumple-zone, and via the network crumple zone, A3 enforced a strict QoS policy on request rate thus mitigating DoS attempts, filtered <script> tags from submission thus blocking script injection attempts and enforced WS parameter and HTTP header constraints thus disallowing the execution of arbitrary, out-of-bounds fuzzed inputs. Specifically, the attack clients attempted 1,519 attacks and benevolent clients executed 933 requests, which, in effect, doubled the number of clients. In terms of average latency for the benevolent clients, the additional load introduced by the experiment (i.e., request load and A3 I/E/T policies) resulted in a modest 4% of additional overhead. In terms of distribution of performance, we observed a max latency 1% larger than the previous 15-interval experiment which is ultimately negligible.

The final thing to notice in Figure is the grayed-out recovery window. As mentioned earlier, during the sixth minute of the run, we injected a synthetic crash failure emulating a novel attack that would not have been blocked by policy violation in the earlier stages. However, this attack was fully absorbed by the network crumple zone's proxy. Unoptimized, A3 can detect a process crash (observers for more complex failures such as verifying the integrity of the CZ VMs are under development) in the network crumple zone, and complete the subsequent roll-back recovery comprising of tearing down the crumple zone VMs, and restoring them to the last known good checkpoint in under 77 seconds.

4. CONCLUSION AND NEXT STEPS

The initial evaluation of the A3 execution container and constituent prevention-focused mediation and adaptive response indicates that the middleware-based, near-application and application-specific cyber-defense can be effective against novel attacks whose signatures are not known, and such defenses can be mounted effectively.

We are continuing to enhance the CZ policies. A specific case alluded to in the previous section concerns enhancement of I policies. This involves developing observers for more sophisticated compromised behavior and undesirable conditions than process crash. We are using Virtual Machine Introspection (VMI) and application-specific invariants to implement these observers. Work is also underway to support recovery-focused adaptation using ASM and improving the defensive policy and configuration using RwM.

Evaluation of a security solution such as A3 that aims to address the uncertainty and impact of novel attacks is a hard problem. Testing the technology against an application with known vulnerabilities goes part way in demonstrating the effectiveness of the technology, but falls short on evaluating the technology's response to novel attack. We are extending the failure injection approach described in Section 3.2 to emulate the manifestation of novel attacks by using a protected application that is injected with artificial vulnerabilities (i.e., made artificially vulnerable) and exploiting the vulnerabilities in a non-deterministic way. Our future evaluation plan includes testing whether the I/E/T policies of A3 crumple zones can absorb and contain such attack effects, and if not, whether advanced state management and replay with modification can quickly recover and reconstitute a more effective defense.

5. ACKNOWLEDGMENTS

This work is being supported by the Unites States Air Force and DARPA under Contract No. FA8750-10-C-0242.

6. REFERENCES

[1] NSA SELinux website, http://www.nsa.gov/research/selinux/index.shtml

[2] Linux Works LynxSecure Separation Kernel, http://www.lynuxworks.com/virtualization/hypervisor.php

[3] Verizon 2010 Business Data Breaches Investigation Report, http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf

[4] Michael Atighetchi et al. Adaptive Cyberdefense for Survival and Intrusion Tolerance, *IEEE Internet Computing,* vol. 8, no. 6, pp. 25-33, Nov/Dec, 2004

[5] Partha Pal et al. An architecture for adaptive intrusion-tolerant applications, *Software: Practice and Experience*, vol. 36, no 11-12, pp. 1331-1354, Sep/Oct, 2006

[6] Jennifer Chong et al. Survivability Architecture of a Mission Critical System: The DPASA Example. *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, Arizona, pp. 495-504, Dec, 2005

[7] D. Paul Benjamin et al. Using a Cognitive Architecture to Automate Cyberdefense Reasoning. *Proceedings of the ECSIS Symposium on Bio-inspired, Learning, and Intelligent Systems for Security* (BLISS 2008), Edinburgh, Aug, 2008

[8] The Qubes OS web page, http://qubes-os.org/Home.html

[9] Jochen. Liedtke. On micro-kernel construction. *SIGOPS Oper. Syst. Review,* vol. 29, no. 5, pp. 237-250, Dec, 1995