2013 International Conference on Computational Science

Resilient Dynamic Data Driven Application Systems (rDDDAS)

Glynis Dsouza a, Salim Hariri a, Youssif Al-Nashif a, Gabriel Rodriguez b

a NSF Center for Cloud and Autonomic Computing, The University of Arizona, Tucson, AZ, 85721, USA.
b Computer Architecture Group, University of A Coruña, Campus de Elviña, s/n 15071 A Coruña, Spain.

Abstract

There is a growing interest in Cloud Computing for delivering computing as a utility. Security in Cloud Computing is a challenging research problem because it involves many interdependent tasks including vulnerability scanning, application layer firewalls, configuration management, alert monitoring and analysis, source code analysis, and user identity management. It is widely accepted that we cannot build software and computing systems that are free from vulnerabilities and cannot be penetrated or attacked. Consequently, there is a strong interest in resilience approach because of its potential to address the cybersecurity challenges. Our is based on using the Dynamic Data Driven Application System (DDDAS) and Moving Target Defence (MTD) strategies to develop resilient DDDAS. The Resilient Applications utilize the following capabilities: Software Behaviour Encryption (SBE), Replication, Diversity, Automated Checkpointing and Recovery. Software Behaviour Encryption employs spatiotemporal behaviour encryption and a moving target defence to make active software components change their implementations and their resources randomly and consequently evade attackers. Diversity and random execution is achieved by “hot” shuffling multiple functionally-equivalent, behaviourally-different software versions at runtime (This encryption of the execution environment will make it extremely difficult for an attack to disrupt the normal operations of a cloud application. Also, the dynamic change in the execution environment will hide the software flaws that would otherwise be exploited by a cyberattacker. Checkpointing is used to save the current state of the task to a reliable storage and thus enabling rollback recovery if it is required to tolerate cyberattacks and mitigate their impacts. We use the Compiler for Portable Checkpointing (CPPC), a tool for automatically inserting portable checkpoints into the code.

We also evaluate the performance and overhead of running three applications in our rDDDAS environment. Our experimental results show that the rDDDAS environment can be used to develop resilient cloud applications are resilient against attacks with around 7% in execution time overhead.

© 2013 The Authors. Published by Elsevier B.V.
Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science

Keywords: DDDAS, Moving Target Defense, Software Behavior Encryption, Cloud Computing, Resilience Applications.
1. Introduction

Cloud systems and their ubiquitous penetration in our daily life increase the need for their services to be secure and resilient to cyber attacks and/or malicious faults. It has been noted that much of the vulnerability in computing systems can be attributed to the monoculture [1] or lack of diversity in software systems. It is practically inevitable that software will contain flaws. Current software monoculture enables attacks to spread rapidly and thus exposing the systems to large-scale attacks by well-informed attackers. Inspired by the resilience of diverse biological systems, our approach is based on a diversity-based defense mechanism against software attacks that utilizes spatiotemporal software diversity to enhance software system security, survivability and resilience.

Moving Target Defense has been identified as a game changer approach to build self-defending systems [2]. Some works presented a wide range of Moving target Defense (MTD) techniques to continuously change network configurations or parameters, firewall settings, operating systems, memory addresses, instruction sets, or application execution environments [3][4]. For example, in [3], the IP addresses are dynamically changed while maintaining existing connections. One can also randomize the configuration space [5] where the configuration variables of a system are randomized, while ensuring the availability of end to end services.

In [6], the authors presented a survey of several software fault tolerance techniques. The fault tolerance techniques that are based on diversity include dual-node redundant operating stations with hardware or software result comparison [10], Recovery Block Station [8], Distributed Recovery Block with acceptance Test [9], Voting Triple Modular Redundant Computing Stations [10] and N version programming [11]. Also, the authors in [12] have described several diversity defense techniques used in popular operating systems. These include Address Space Randomization [13], Instruction Set Randomization [14], and Data Randomization [15]. In our approach, we adopt diversity technique to the application execution environment, redundancy in the resources used to run the cloud services and randomly changing the versions and resources used to make it prohibitively expensive for attackers to figure our current execution environment and succeeding in exploiting vulnerabilities and launching attacks.

2. Resilient DDDAS Environment

Our approach aims at leveraging a closed loop DDDAS-based architecture to develop resilient applications that will prohibitively increase the burden on the adversary to penetrate and exploit cloud vulnerabilities. Our architecture is shown in Figure 1 and has the following components: Replication, Diversity and Automatic Checkpointing, Software Behavior Encryption (SBE) and the Decision Support System (DSS). In the following section, we briefly highlight the main tasks performed by each of the units shown in Figure 1.
2.1. Replication

The concept of design diversity is commonly used in software fault tolerance techniques [16] in order to continue to operate successfully in spite of software design faults. In our rDDDAS approach, we combine the N-version programming [11] and online anomaly behavior analysis techniques. Hardware redundancy is applied by having the task run on different physical nodes in the cloud infrastructure. Each of these physical nodes is designated as a replica.

2.2. Software Behavior Encryption

SBE adopts a Moving Target Defense (MTD) strategy by using spatiotemporal behavior encryption to make active software components change their implementation versions and resources continuously and consequently evade attackers. Diversity is applied by “hot” shuffling multiple functionally-equivalent, behaviorally-different software versions (code implementation) at runtime (e.g., each software task can have multiple versions, where each version can be a different algorithm implemented in different programming language running on different computing systems). This approach will make it extremely difficult for an attack to disrupt the normal operations of a cloud application or service. Also, by incorporating the DDDAS paradigm [24], the feedback we receive from top or bottom layers will be used to adopt the resilience level by dynamically changing shuffling policies, increasing or decreasing the shuffling rate, and scope of executing tasks’ versions and their execution environments. A major advantage of this approach is that the dynamic change in the execution environment will hide the software flaws that would otherwise be exploited by a cyberattacker.

2.3. Decision Support System (DSS)

The primary task of the DSS is to support dynamic decisions among the various components based on the DDDAS paradigm such that the cloud resources and services are dynamically configured to effectively exploit the current state of the cloud system and meet the application security requirements that might change at runtime.

Figure 2 shows how the DSS component applies an acceptance test to task A with three replicas between the output of phase i and the input of phase i+1. The acceptance test will determine which output of the three replicas will be used as the input of the next phase for all the replicas. The Automatic Checkpointing unit continuously collects checkpoint information from the versions on each replica.

To reduce the recovery time to tolerate software attacks or compromised tasks, a switching element data structure (see Figure 2) is used to connect the output of the task operating normally (this could be taken from any of the replicas) to the next stage, and so on. If no attack is detected, the switching elements will be setup to select the first results that are determined to be operating normally as shown in Figure 2. However, if after the acceptance test has determined that a replica, say replica1 is determined to be compromised, the switching module will be setup by DSS, to the case “Replica compromised” and consequently forward the output from
the other replica module for Task A to the next phase. It is clear from this example that the design diversity will enable cyber resources and applications to continue to operate normally in spite of attacks on the tasks and their execution environments.

2.4. Diversity and Automatic Checkpointing

This component is responsible for generating the functionally equivalent-behaviorally different versions required by the SBE. We use the Compiler for Portable Checkpointing for continuously capturing the current state of the system.

3. Software Behavior Encryption

The Software Behavior Encryption (SBE) algorithm encrypts the execution environment by dynamically changing the sequence of execution of task variants by shuffling the task variant running after each execution phase. The dynamic software behavior change makes it more difficult for an attacker to generate a profile with the possible flaws of the executing variant (task versions). The decisions regarding when to shuffle the current variant, the shuffling frequency, and the variant selection for the next shuffle are guided by a continuous feedback from the autonomic and program manager.

The SBE module will continuously and randomly change execution environment of each application’s task by adjusting the shuffling (changing) cycle, the adversary will not have enough time to figure out the existing vulnerability and the execution environment. Any attack will go through at least three phases: probing, constructing and launching phases. If the environment stays static as it is now, the attacker has plenty of time to identify existing vulnerabilities that can be exploited. However, if the life cycle for any version is much shorter that the time it takes the attacker to launch the attack (see Figure 3) as it will be the case using our SBE algorithm, the attacker will not be able to succeed in exploiting any existing vulnerabilities in cloud services. Hence, the cloud services will be resilient to cyberattacks and continue to operate normally or with acceptable degraded performance.

3.1. Overview

The dynamic software behavior change makes it more difficult for an attacker to generate a profile with the possible flaws of the executing version (task versions). SBE induces diffusion by running the task versions on different physical and logical resources (Linux, Windows, different libraries and file systems, etc.).
decisions regarding when to shuffle the current version, the shuffling frequency, and the version selection for the next shuffle guided by the decision support system component.

Figure 4 shows an example on how SBE can be implemented to encrypt the execution of one task (Task A) with three consecutive phases. During phase 1, we execute version 3 of Task A, version 1 during Phase 2, and version 1 during Phase 3. Each phase ends at the expiration of a timer. At the end of each phase, the current state of the execution is stored as a checkpoint as described in section 2. This checkpoint is passed onto the DSS component to perform the anomaly analysis using acceptance testing techniques.

3.2. Autonomic Management (Self-Management)

The Self Management (SM) architecture is based on our autonomic computing environment (Autonomia) [17]. The SM main functions implemented in two software modules (see Figure 5): Observer and Controller modules. The Observer module monitors and analyzes the current state of the managed cloud resources or services.

The Controller module is delegated to manage the cloud applications and enforce the resilient operational policies. In fact, the Observer and Controller pair provides a unified management interface to support the SM’s self-management services by continuously monitoring and analyzing current cloud system conditions in order to select the appropriate plan to correct or remove anomalous conditions once they are detected and/or predicted.

4. Experimental Evaluation

4.1. rDDDAS Environment

To evaluate our rDDDAS approach, we have implemented a test-bed based on the IBM BladeCenter HS22 Private Cloud [18] at University of Arizona’s Centre for Cloud and Autonomic Computing. The implementation runs on a three node cluster where each is hosting two virtual machines. One of these virtual machines is Windows based, while the other is Linux based.
Figure 6: r-DDDAS Implementation on an IBM Blade system

Figure 7: Experimental Setup

what follows, we describe our experimental environment and the applications used for testing and evaluation of rDDDAS.

4.2. Experimental Results

Figure 7 illustrates the setup used in our experiment. In the beginning, the SBE controller randomly selects a supervisor from a set of three supervisors (one supervisor on each physical node). This supervisor then randomly selects a phase timer for that phase. The master machines on each physical node randomly select the version to be run on each node. Checkpoints are continuously stored on a master machine on each physical node. At the expiration of the phase timer, the last checkpoint from each of the three masters is passed onto the supervisor machine (Steps 12-15 in the algorithm shown in Figure 10). An Acceptance test is run on each of these checkpoints. This test checks the following: a) The solution is within a range, b) The memory utilization of the program is within a normal range, and c) The variable values after subsequent iterations are not too divergent. The latest checkpoint that passes the acceptance test is selected as the output of this phase. For example, if the checkpoints received from the masters have completed iteration 5, 7 and 8, respectively and they all pass the acceptance test, then the checkpoint which has completed the 8th iteration is selected as the output for this stage and the input for the
next phase (Steps 16-21 in Figure 10). At the beginning of the next stage, a new supervisor is selected randomly and the above process is repeated until the final output is received.

4.2.1 MiBench Benchmarks

The MiBench Benchmarks [19] consist of C programs from six categories each targeting a specific area of the embedded market. We used the Basicmath program from the Automotive and Industrial category. We calculated the overhead of our rDDAS architecture for different number of iterations of this benchmark. The results are presented in Figure 8. As seen in the figure, the overhead of our algorithm decreases as program size increases.

4.2.2 Jacobi based Linear Equation Solver

Linear equations are used to solve a wide range of real world scientific and engineering problems. The Jacobi technique [7] is an iterative technique for solving a set of linear equations under two assumptions:

- The system given by $Ax=B$ has a unique solution
- The co-efficient matrix $A$ has no zeroes on its diagonal.

To solve a set of $n$ equations, we solve the first equation for $x_1$, second equation for $x_2$ as shown in Figure 9.

$$x_1 = \frac{1}{a_{11}} (b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}} (b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}} (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})$$

Figure 8: Overhead for SBE with three phases

Figure 9: Linear Equation Solver using Jacobi’s iteration method

Figure 10: SBE Algorithm used in our experiment
The overhead is given as a function of the number of phases selected to run the application. We calculated the overhead as the additional time taken with our algorithm compared to running the application without SBE. As shown in Figure 12, for programs with higher execution times, the overhead due to SBE reduces significantly. For example, for a program with execution time of 3600 seconds, the overhead percentage for 3 phases is 7%. The number of phases to run each application can be chosen such that it meets the performance and resilient requirements of the application. The DDDAS paradigm will be used to select these parameters at runtime.

In evaluating the resilience of this application, the following attack scenarios are launched against the application execution. Figure 11 illustrates Scenario 1:

1. **DoS Attack (Attack Scenario 1).** We launched a DoS attack on the Windows machine running version V1 during Phase 2 using the mprime library for memory DoS attack. As a result of DoS attack, V1 run was very slow. The DSS detected that the checkpoints received from the other two versions were faster and accurate. Hence the checkpoint from the other machine was selected as the output of this stage and the application continues to operate normally in spite of DoS attack.

2. **Insider Attack (Attack Scenario 2).** As shown in Figure 7 there are three supervisors that directly communicate
with the SBE Controller. Only one is randomly selected as the active supervisor in any given phase. In Scenario 2, in the beginning of the execution, we compromised Supervisor 2 by destroying all the Supervisor services running on it. During the phase when Supervisor 2 was selected, the acceptance test unit on the controller detected that the Supervisor code is not running and it consequently selected another Supervisor.

4.2.3 MapReduce

We have also evaluated our approach using a MapReduce application [21] which uses a parallel data processing model to solve a wide range of large-scale computing problems. The implementation of the Map/Reduce cloud application consists of three physical machines hosting two virtual machines each. Oracle Virtualbox [22] has been used as the virtualization software. The MapReduce word count program [21] is available on each virtual machine.

We evaluated the resilience of the above approach against DoS attack and Insider attacks on one physical machine at a time. The resilient MapReduce application was able to tolerate these attacks and continue to operate normally. The average response time using this algorithm increases by 14% (without attack) and 24% (with attack). For more information on this work, please refer to [23].

5. Conclusions

In this paper, we discussed the need for resilient cloud techniques because we cannot develop cloud applications that cannot be penetrated or attacked. We presented closed loop Resilient Dynamic Data Driven Applications (rDDDAS) architecture and showed how to implement it using four functions: Replication, Software Behaviour Encryption (SBE), Decision Support System and Diversity and Automatic Checkpointing. SBE employs multidimensional software diversity and moving target defense strategies to make it extremely difficult to the attackers to figure out the execution environment and exploits its vulnerabilities. We have evaluated the performance of the rDDDAS approach to provide resilient operations to three cloud applications. We also showed that our approach can tolerate external and insider attacks with low overhead.

Acknowledgements

This work is partially supported by AFOSR DDDAS award number FA95550-12-1-0241, and National Science Foundation research projects NSF IIP-0758579, NCS-0855087 and IIP-1127873.

References

[16] Genesis A Framework for Achieving Component Diversity that was developed for a DARPA program at http://www.cs.virginia.edu/genesis/