CyberX:
A Biologically-inspired Platform for Cyber Trust Management

Mohamed Azab
The Bradley Department of Electrical
and Computer Engineering, Virginia Tech
Email: mazab@vt.edu

Mohamed Eltoweissy[1]
Egypt-Japan University of Science and Technology
Email: Mohamed.eltoweissy@ejust.edu.eg

Abstract—**: Today numerous infrastructure assets remain alarmingly susceptible to advanced, targeted cyber attacks. There is a need to construct trustworthy high-quality protection and defense solutions capable of securing such valuable assets. In our work, we realize such cyber trust through trace-resistant moving-target defense, resilience against failures and attacks, and autonomous trustworthy cooperative defense. Achieving these goals would require software development, management, and operation platforms that support elasticity, diversity, awareness, cooperation, and intelligence. In this paper, we propose CyberX, a situation-aware trustworthy management platform. CyberX utilizes our Cell Oriented Architecture (COA) capability that separates the main design concerns: data, logic and physical resources to employ runtime diversity via hot shuffling of similar-function different-quality-objective code variants. CyberX employs such diversity to modify the application implicit behavior at runtime utilizing autonomous execution elasticity and adaptability. Further, CyberX enhances application resilience against failures and attacks via multi-mode recovery and real-time, context- and situation-aware adjustment of shuffling and recovery policies. Using analysis and simulation, results show that CyberX-managed COA-based software systems can efficiently adapt to maintain the desired performance and resilience objectives even in hazardous, unstable environments at a reasonable overhead.**

Keywords software management platform, system diversity, moving target defense, biologically-inspired software architecture, resilience, hot shuffling.

I. INTRODUCTION

Today, cyber systems form the backbone of national critical infrastructures, which means that cyber security incidents on such systems could have significant disruptive impact on the reliability and safety of many of the systems that we rely on to maintain our everyday life. Both researchers and practitioners have been paying considerable attention to the cyber security problems for more than two decades. However, the problems are far from being comprehensively resolved. Cyber trust was defined by the National Science Foundation with the goal to develop new insights and fundamental scientific principles that would lead to software and hardware technologies on which people can justifiably rely [12].

To achieve the cyber trust vision and simultaneously improve the nation's cyber security posture, cyber trust should support a portfolio of defense techniques that when synergistically composed into a comprehensive solution would enable adequate and trustworthy defense provisioning. In our work cyber trust encompasses trace-resistant moving-target defense, resilience against failures and attacks, and autonomous trustworthy allied-defense. We surmise that enabling cyber trust would require software development, management, and operation to be based on five main pillars: elasticity, diversity, awareness, cooperation, and intelligence.

Currently software products depend mostly on static or partially dynamic architectures where data, logic, and/or physical resources are primarily tightly coupled. Multiple attempts have been presented in the literature to partially decouple these design concerns [1, 2, 3]. However, up to our knowledge our Cell Oriented architecture (COA) is the only architecture that comprehensively supports intrinsic separation of design concerns needed for runtime re-programmability, intrinsic autonomic online composability, and dynamic software adaptation and elasticity.

In this paper, we propose CyberX, a situation-aware trustworthy management platform that utilizes the COA features to realize the aforementioned pillars. COA is a biologically-inspired architecture with active components termed Cells that support development, deployment, execution, maintenance and evolution of software. Cells separate logic, state and physical resource management. Cells are realized in the form of intelligent capsules that encapsulates executable applications defined as code variants. Cells are dynamically composable into organisms that are bound to functional roles at runtime. CyberX manages such construction to enable online re-programmability, hot code-swapping, local/global situation awareness, and automated recovery.

CyberX enables applications to dynamically adapt to runtime changes in their execution environment via runtime diversification of multiple similar-function, quality-objective – different code variants. Reliability, performance, robustness, survivability, compatibility, scalability, and mobility are examples of such attributes.

CyberX utilize the COA feature of enabling the application to exchange real-time status and recommendation messages with the host Cell for administrative purposes to enhance the Cell local application awareness and to enable application driven adaptation. CyberX use these messages to guide the Cell runtime quality-attribute manipulation towards accurate and prompt adaptation. Further, CyberX collects, analyze and trustworthy-share these messages and status reports constructing a real-time sharable global view of the Cell network.

CyberX enhances the system resilience by multiple recovery modes to cover different application requirements and host configurations. CyberX offers a prompt and accurate fine-grained recovery for resourceful hosts executing critical applications, and a more resource efficient course-grained

[1] The author is also affiliated with the Bradley Department of ECE at Virginia Tech and the ECE Department at University of Arizona

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250512

recovery for less critical applications. CyberX uses the COA loosely coupled features to allow applications to seamlessly change their current active recovery modes based on context, environment, or application-objective change.

The main contributions of this paper are as follows:

- A biologically inspired architecture with the following capabilities:
  - Intrinsic separation of design concerns (data, logic, and physical resources); and
  - Employing a mission-oriented application design and inline code distribution to enable adaptability, and online dynamic re-tasking;
- Elastic system design and platform-managed control enabling the following:
  - Runtime diversity employment for hot manipulation of quality attributes to effect trace-resistance and moving target defense;
  - Multimodal, autonomous situation-aware recovery system for enhanced system resilience; and
  - Dynamic and autonomous change of shuffling and recovery policies according to run-time changes in the execution environment.

The balance of this paper is organized as follows. Section 2 describes the Cell Oriented Architecture. Section 3 describes CyberX management platform design. Section 4 illustrates the CyberX-managed multimode failure recovery. Section 5 presents an execution scenario for a CyberX-managed application. Section 6 discusses the evaluation of the proposed system. Section 7 presents a brief literature survey. Finally the paper concludes in section 8 which also outlines future work.

II. THE CELL ORIENTED ARCHITECTURE

The COA is an employment of a mission-oriented application design and inline code distribution to enable adaptability, dynamic re-tasking, and re-programmability. The Cell is the basic building block in COA. The COA Cell is inspired from the biological Cell in its independent, generic, composable construction.COA Cell is an abstraction of a mission-oriented autonomous active resource. Generic Cells termed stem-Cells, are seamlessly created by the host-side middleware or the COA Cell DNA (CCDNA). Further, they participate in emerging tasks through a process called specialization. The CCDNA is a middleware program that allows a physical workstation to host Cells and facilitates Cell physical resource allocation and management.

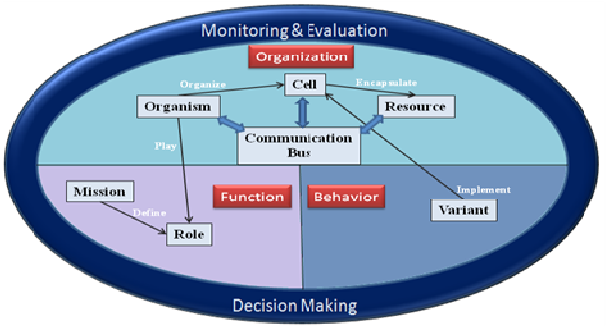

Figure 1. Components of our COA

We envision applications built over COA as a group of cooperating roles representing mission objectives. The term organism is used to represent a role player that performs a dedicated mission. An organism might be composed of a single or multiple Cells based on its objectives. Figure.1 illustrates the different components of the COA. We will briefly illustrate the main design aspects of the COA Cell. More details about COA can be found in [13].

A. *The Cell*

Conceptually, the Cell is the smallest active resource in a distributed computing platform. Cells are intelligent, and independent, autonomous, single-application capsules "sandbox" that acquires, on the fly, application specific functionality in the form of an executable code variant "The specialization process". Cells act as a simple virtualization environment isolating the executable **Logic** from the underlying **Physical** resources. Figure. 2 illustrates an abstract view of a COA Cell at runtime. The Cell is dynamically composable into larger structures "organisms" representing complex multi-tasking applications.

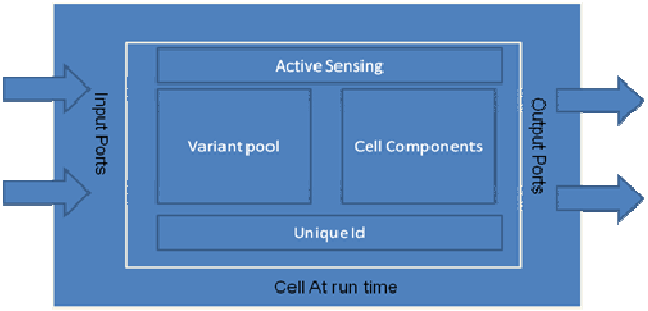

Figure 2. COA Cell at runtime

A single workstation can host one or more Cells, providing a flexible way to share the physical resources among multiple applications. Figure.3 illustrates the main components of the COA Cell briefly described as follows.

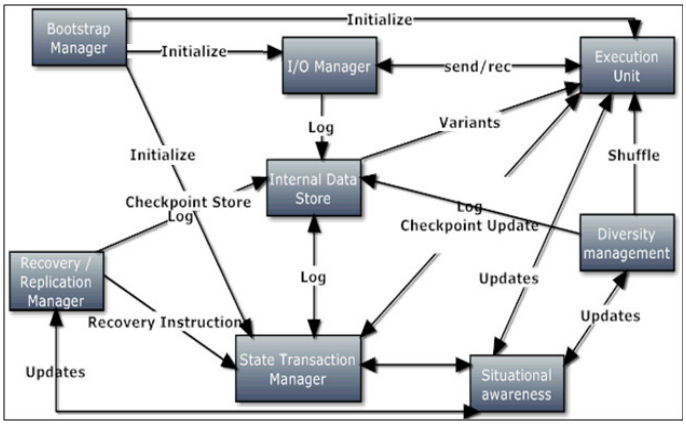

Figure 3. The Cell

Cells are instantiated at bootstrapping when the bootstrap manager initializes the Cell components and ports with the appropriate parameters based on the bootstrap context. The communications unit (I/O manager) handles local and remote

I/O communication setup, I/O logging, and IP/Port/Virtual naming resolution.

The specialization process occurs when the execution unit receives an executable COA-ready code variant that represents the application specific functionality that the Cell should acquire.

A **COA-ready variant** is a program that enables check-pointing and frequent reporting through a predetermined channel using predetermined syntax. We isolate the **Data** from the **Logic** and **physical** resources by forcing committing all sensitive data to remote data storage before each checkpoint. The data is committed using a dedicated data channel provided by the infrastructure. The program must ask for, and start execution from an infrastructure provided starting point. This point is zero for fresh Cells. Finally, programmers have to provide at least two similar-function different-quality-objective variants to enable CyberX quality attribute manipulation.

The execution unit starts by launching the selected variant with the appropriate parameters "Ex., the Cell Id". The execution unit is also responsible for the termination and replacement of the executing variants based on incoming shuffling commands. All issues regarding diversity employment-methodology, shuffling policy, "shuffling frequency, commanding, and variant selection" are the responsibility of the diversity-management unit.

The State Transaction Manager (STM) is responsible for monitoring the variant execution progress. It is the only unit with direct access to the executing application through a dedicated communication channel. STM reports checkpoint change and other incoming application requests and status reports to the appropriate units "ex, holding shuffling frequency change, objective change requests, etc".

The recovery manager is responsible for adjusting the recovery settings, and recovery mode change. Additionally, recovery manager cooperates with the execution unit to restore and synchronize checkpoints in case of failure-recovery. Recovery manager is also responsible for sending the Cell beacon messages to the tracking servers. These messages include the last checkpoint reported by STM, and other reports regarding Cell state reported by the situational awareness unit; and any other administrative messages needs to be delivered to the Global Management Servers (GMS). The details about CyberX multimodal failure recovery processes are illustrated in section IV.

The situational awareness unit, is responsible for providing the needed situational and context awareness information to the other Cell units in order to support their runtime decisions. It monitors the internal and the external surroundings and generates guideline reports for all Cell units. Additionally, situational awareness unit informs the GMS with awareness reports through messages attached to the Cell frequent beacon messages. GMS use these beacons to generate more meaningful status reports. These reports holds information, directions, and commands that CyberX wants to deliver to a certain area in the network. For example, if one of the Cells reported a malicious event that might affect other neighbor-Cells, GMS might command other Cells to change their current variant to a more secure variant.

In the COA Cell, decision-making tasks are totally distributed among Cell units. The global operation of the Cell is handled by the real time cooperation and autonomous synchronization between all these units.

III. THE CYBERX MANAGEMENT PLATFORM

CyberX is a situation-aware trustworthy management platform that utilizes the COA features to enable a wide set of features and capabilities. Online re-programmability, hot code-swapping, local/global situation awareness, and automated recovery are examples of such capabilities that participate in the realization of Cyber-trust. In the next subsections we describe CyberX architecture, the main components participating in its construction and the functionality of each component. Further, we will discuss the communication aspects and security issues with and within CyberX.

A. CyberX platform architecture

CyberX is composed of a set of central powerful nodes we will address them as servers. These servers cooperate autonomously to manage the whole network of Cells. This platform is responsible for the organism creation "composition and deployment of Cells", management, the host side API(s) "CCDNA", real-time monitoring and evaluation of the executing Cells, and recovery management. Further, it provides the necessary management tools for system administrators to manage, analyze, and evaluate the working Cells /organisms.

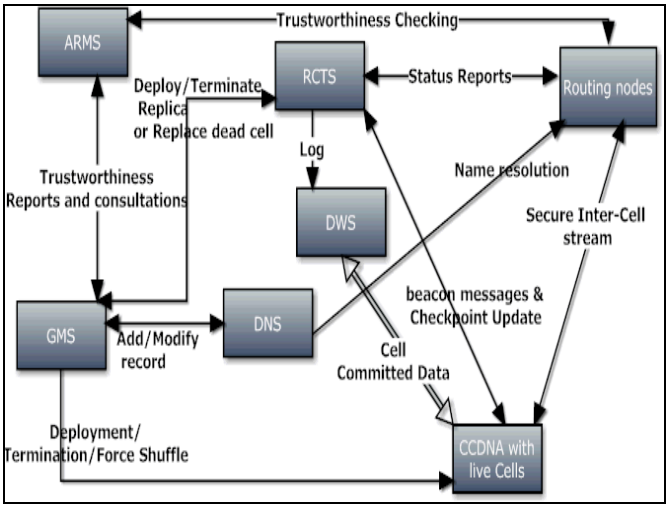

Figure 4. The management platform architecture

Auditing and Reputation Management Servers (ARMS); its main task is to monitor outgoing or incoming Cell administrative messages for the lifetime of the Cell. This information is used to assist evaluating the trustworthiness of the Cell. These servers cooperate with the recovery tracking servers and routing nodes to frequently evaluate the Cell behavior for any malicious activities. These servers will hold comprehensive reports about each Cell for the lifetime of the Cell.

Recovery and Checkpoint Tracking Servers (RCTS); its main task is to monitor, and store checkpoints changes for all running Cells. Checkpoint updates are always enclosed as a part of the Cell frequent beacon message updates. This server is also responsible for reporting failure events by comparing the duration between consecutive beacon messages to a certain threshold matching the reporting frequency settings of each Cell. Failure events are validated by comparing the recently noticed reporting-delay for a particular Cell to the average reporting-delay within its neighbors and other Cells hosted in

the same host. A Cell failure notice is reported to the global management servers with the last known failure recovery settings, checkpoint, and variant settings to start deploying replacement Cells.

Global Management Servers (GMS); its main task is to manage the underlying COA infrastructure. It is responsible of Cell deployment, coordinating between servers, facilitating and providing a platform for administrative control. It is the only server authorized of issuing Cell termination signals. It can also force Cell migration or change the current active recovery policy when needed. It is responsible of assigning the infrastructure global policy, routing protocol, auditing granularity, registering/revoking new hosts, and keeping/adjusting the host-platform configuration file.

The Data-Warehouse Servers (DWS), it is one of the main components of the infrastructure that participate in the separation between the Data, Logic, and Physical-resources. DWS are distributed through the Cell network, they are responsible for holding and maintaining all the data being processed, and any other sensitive data that the management units want to store. All running Cells are not permitted to store sensitive data on their local memory. All sensitive data has to be remotely stored in a specific DWS through the dedicated data channel. DWS synchronize their data independently.

Distributed Naming Servers (DNS), is responsible for resolving the real host IP/Port mapping to the virtual Cell Id and organism names. The working Cells use this mapping at runtime to direct incoming and outgoing communications. DNS is major player in the COA's separation of concerns that enables virtually seamless, Cell relocation, and workload transition in case of failure recovery. In case of Cell movement, the DNS will be instructed by the GMS to maintain communication redirection.

B. CyberX trustworthy platform-communication

This section discusses CyberX management of the secrecy, authenticity, and anonymity of the inter-Cell communications. We present a suitable key management scheme for various connection types in the system. Further, we will illustrate our mechanism to detect maliciously behaving and problematic Cells. Additionally, we present our secure authentication mechanism securing the inter-Cell communications against identity theft attacks.

In order to maintain the **secrecy of the sensitive information** stored locally or externally, or being exchanged over communication lines; CyberX uses an asymmetric key encryption scheme to encrypt this data. At the deployment time, the GMS assigns a pair of keys to each cell, a public key and a private key. The public key will be used to encrypt all incoming messages to the Cell. The private key will be used to decrypt these incoming messages. The Cell can use the private key to encrypt the sensitive data within the Cell itself, if the situation necessitates that. For example, it can encrypt sensitive data stored in the local hard drive if the drive is being shared by other Cells hosted in the same host. Figure 5, illustrates the architecture of CyberX local security mechanism.

CyberX maintains the Cell to Server, and Cell to Replica **data authenticity** using a set of encryption/decryption keys. At the deployment time GMS attaches the Cell inputs, configuration parameters, the Cell public and private keys, and a pool of public keys for other entities that the Cell might communicate with to the Cell deployment package. The public keys pool includes keys for CyberX servers and routers that the Cell might need to be in direct contact with. Additionally, if the Cell had any replicas at the deployment time, the public keys for those replicas are also included.

At runtime, Cells can acquire new replicas as a response to a change in the current recovery mechanism. The process starts by a request from this Cell or the RCTS to GMS to deploy new replicas. GMS will reply with the public key and the unique Cell name of the new replica to the requester in an encrypted message using the requester public key.

In order to guarantee the authenticity of all incoming messages, the source id will be enclosed and encrypted with the message. The ARMS will be monitoring inter-Cell behavior with the cooperation of RCTS that keeps track of all Cells activities. Malicious, or problematic Cells, will be terminated, and their terminated Cell id will be blacklisted and announced to all routing Cells.

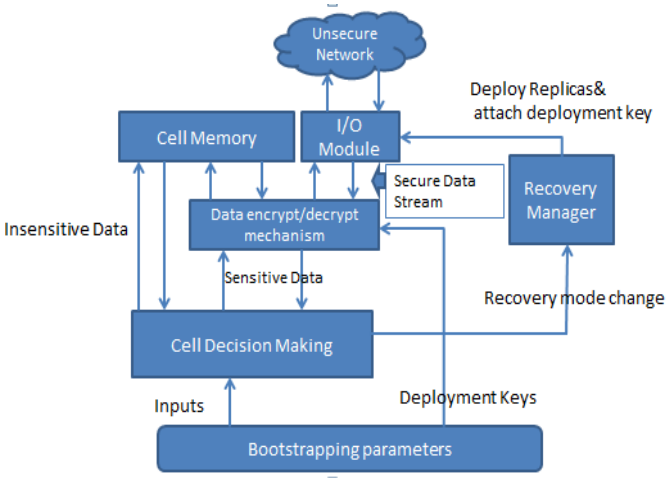

Figure 5. CyberX security framework

In CyberX managed applications**, Inter-Cell communications** can be classified into two main types, administrative related communications, and application related communications. Application related communications are messages being exchanged to serve the application needs and identified by the application designer. The administrative communication messages include, recovery beacon messages between Cells and replicas or RCTS; alerts and events between Cells and ARMS; and messages between Cells and routing nodes.

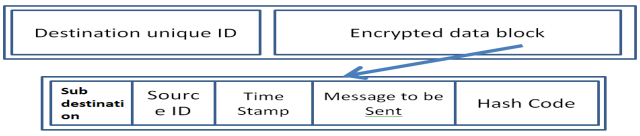

Figure 6. The Inter-Cell message format.

Figure 6 describes an abstract view for **The Inter-Cell message format.** The message is divided into two main parts, the destination id, encrypted data block.

The encrypted data block is divided into five parts encrypted with different keys, sub destination id, the source id, timestamp, message to be sent, and message integrity assurance data "like hash code".

Inter-Cell communications anonymity, Cells are not allowed to directly exchange messages. The reason behind that

is to protect the anonymity of the inter-Cell communications. Cells communicate to intermediate routing nodes to conceal the physical location of the communicating nodes like “replicas, and Cells hosting fractions of the same application”, and to control administrative related communications. CyberX uses intelligent routing cells to anonymize the source and destination of any outgoing message. Doing so can block attackers with access to the network from monitoring outgoing messages searching for a certain transmission pattern “Ex,. Beacon messages between Cells and replicas”. Identifying these patterns can expose the physical location, and the functionality of the destination Cells “replica”.

Figure 7 illustrates a communication scenario between different nodes in the system, cells, replicas, servers. Each node uses the destination public key to encrypt and sign all outgoing messages. We use random router selection for each message (EX,1,2)

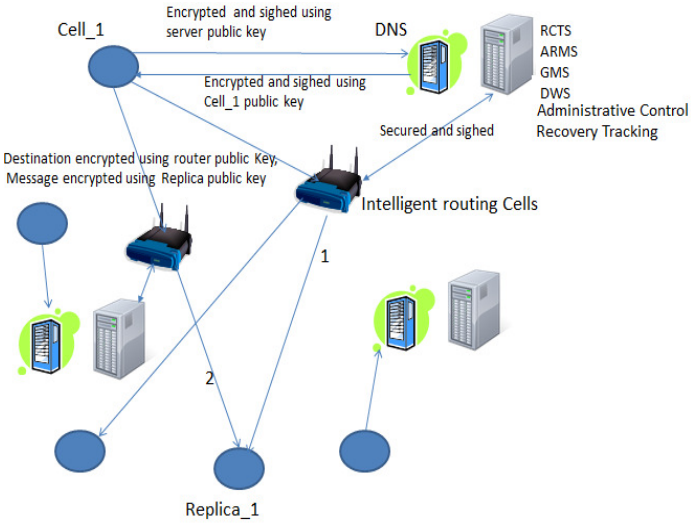

Figure 7. CyberX secure messaging system

Cells are only permitted to directly communicate with routing nodes, and servers. Routing nodes receive the Cell outgoing messages and forward them to their designated destinations in order to hide their physical location. The source Cell will use two different keys to send a message. First, a router public key to encrypt the source ID, and the sub destination ID part of the message. The sub destination ID is the final destination “targeted cell” that the message is indented to be transmitted to. Figure.8 is an example of an incoming message to the router from one of the Cells. Second the final destination key, which will be used to encrypt the message and the integrity check fields.

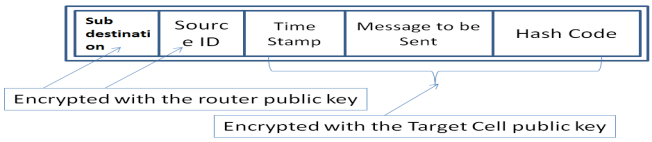

Figure 8. Incoming router message.

The destination ID will be the ID of one of the router close to the Cell. Figure 9 is an outgoing message from the router to one of the Cells. The list of close by routers is preloaded to the Cell at the deployment time, and updated when needed.

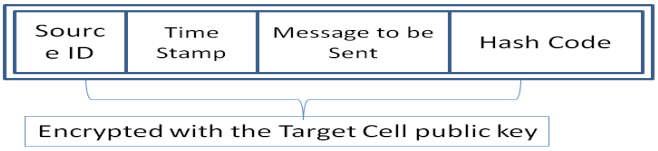

Figure 9. Router outgoing message.

At each routing node, incoming messages will be decrypted using the router private key to extract the source and sub destination information. If the source was blacklisted, the message will be discarded. If the source was not blacklisted, the source ID will be re-encrypted with the destination public key, and attached to the reaming part of the message into a new message to be forwarded to the targeted cell.

We prefer using pre-deployed keys instead of asking for public keys prior communication to block any attempts of a Man in the Middle attack.

We preferred using asymmetric key encryption instead of symmetric key encryption regardless of the added computational cost, because it is hard to use one key for all communicating parities, due to the high cost of key management, and replacement given the large scale of the network of Cells. Further, we used asymmetric key encryption to authenticate the Cell identity and to protect the network against identity theft attacks. The Cell encrypts the source id field using its private key, later it can easily be authenticated by recipients using its unique public key. Additionally, the cost of compromising a Cell is much less with asymmetric encryption as the other Cells will not be affected by revealing the compromised Cell keys. Such keys will be revoked upon detection of exposure. With symmetric key encryption, all the Cells using the compromised keys will be vulnerable to wide set of communication related attacks until the keys gets replaced after detection of exposure.

The presented security mechanism doesn’t protect the Cells against getting compromised. We rely on CyberX full time monitoring of Cells behavior to detect such events. Upon detecting any maliciously behaving Cell, CyberX autonomously block and replace such Cell with a new Cell.

IV. The CyberX Managed Multi-Mode Failure Recovery

CyberX applies diversity techniques to enable autonomous adaptation and performance optimization. Applying diversity might involve multiple execution interruptions. Doing so might lead to multiple coincident failures. Therefore, CyberX is designed to equip COA based applications with an autonomous, dynamic, and situational-aware multi-mode failure recovery mechanism to resolve possible failures. A major outcome of this recovery mechanism is the failure resilience enhancement not only against coincidental failures, but also against malicious induced failures by adversaries.

CyberX dynamically and autonomously changes the Cell recovery-policy to switch between different fault-tolerance granularity levels. Such levels might target reliability, survivability, and resource usage optimization. CyberX offers a fine-grained recovery “Hot-recovery” using replication. The Cell can have one or more replicas on the same physical host. This type of local replication can address only logical failure. For a finer-grained recovery against logical or physical node failure, the Cell might have one or more replicas on different physical hosts. The fine grained recovery comes in two modes, the resource saver, and the fast-recovery modes.

In the resource-saver mode, replicas need to only replicate the STM, I/O unit and local data store units of the Cell. The remaining Cell components stay in hibernation waiting for resurrection when the replica takes over. These replicas will have one variant working all the time and no shuffling or recovery policy change until resurrection. We do that to minimize the resource usage consumed by these replicas. This mode do save the resources but on the account of increasing failure downtime by the time needed to resurrect the Cell.

The fast-recovery mode can achieve virtually no task-transition downtime by using a fully-alive replica Cells. Replicas mimic all the actions of the source Cell except outgoing communications and data change. The execution-transition in this case is a simple network rerouting by a DNS record update. The failure downtime is the time needed to detect failure. The only disadvantage of this mode is the resource duplication needed to keep both Cells alive.

In a resource-constrained environment, CyberX can follow a more coarse-grained recovery "cold-recovery" that might save some of the resources used by replicas while compromising some of the execution states, and increasing the failure downtime.

The default Cell design forces COA Cells to send periodic beacon messages to the RCTS. Such messages contains the last executed checkpoint, some sensitive data, and the currently executing variant. In case of failure, the RCTS notice the delay in beacon message arrival, and investigates the possibility of failure. If failure was detected then the last recovery procedure will be executed as follows:

In case of a failed Cell that follows a **fine-grained recovery mode** then the RCTS will inform MGS to send a resurrection signal to the replica and notify the routers. Additionally, MGS starts to deploy new replicas to replicate the resurrected one. After successful restoration, DNS entry will be adjusted.

If the Cell was following a **coarse-grained recovery mode** then the management will deploy a replacement of the failed Cell while attaching the last checkpoint received by the RCTS to the deployment package. After successful restoration, DNS entry will be adjusted, and the Cell starts execution in recovered-Cell mode. This mode involves negotiating with all Cells in communication to resynchronize any lost execution steps.

The coarse-grained recovery mode is always-on by default enabling the support of multiple concurrent recovery policies. The remote safe store is updated regularly with beacon messages from all working Cells. Each Cell will independently and dynamically set its own message update frequency. Such update frequency could be influenced by the change of the current recovery policy. The update frequency might decrease in fine-grained recovery mode; while they should increase with lower granularity recovery.

CyberX can dynamically change the Cell recovery policy at runtime. The change is guided by the application requirements and host conditions. In stable situations with non-mission critical application, a coarse-grained recovery policy can be used, while in a more hazardous situation, a fine-grained recovery is preferred. The cell utilizes the available information about the current working environment with the application profile to decide the appropriate recovery policy to use. As the surroundings change, the cell changes the current recovery policy to suit these changes.

V. A CyberX managed application

The COA Cell can be built using different techniques based on the desired resource virtualization depth. We implemented the simple and fast version of the Cell to enable quick development of a prototype. We are in the process of realizing a more complex version of the Cell utilizing one of the application virtualization techniques mentioned in [1].

The slow and complex version of the Cell addresses the host resources through a thin hardware virtualization layer. Such indirect addressing increases the execution complexity and the computational cost of the Cell. One of the main advantages of using this technique is enabling uniform variant design where all variants are built to target a uniform virtualized platform regardless of the heterogeneity of the host configuration. Enabling such uniform application design reduces the cost of software production, management, and maintainability; and the effort involved in system upgrades and/or changes. The main disadvantages of such technique are the added workload, and higher risk of failure when compared to the simple version approach.

The simple and fast version technique works only with variants built to match specific deployment platform. This technique gives the executing variants a controlled direct access to the actual host hardware. No hardware virtualization is needed. Such direct access speeds the system response time, and reduces the Cell resource consumption when compared to the complex version technique. Cells instantiate, monitor, and control all the runtime aspects of the variant as descried latter. All communications and data access are only permitted through dedicated units/channels within the Cell. In order to enable emergency Cell-relocation, the variant pool should contain variants matching the destination platform configuration.

As mentioned before a COA-ready program is a program that enables check-pointing with at least two different objective variants enabling quality-attribute manipulation. The checkpoint reporting location has to consider data integrity requirements especially in case of failure. All data has to be committed before checkpoints.

At the deployment time, a new DNS record will be created by the GMS for each Cell. This record indicates the application virtual name to be used for inter-variant communications "if needed by the application designer", the Cell unique id for inter-Cell communication, and the IP of the physical-host hosting the Cell.

The deployment starts when the CCDNA receives the deployment package from the GMS including the Cell globally unique ID(s), the initial checkpoint value, variant pool setup "variant binaries, names; numbers; sets; variant-classification" , the configuration script describing the specs of each variant, the global objective of the application, and any specific specs added by the developer to be considered at time of execution "number of application fractions; fraction-names; ..", the initial shuffling and recovery policy, the needed security level, and the list of security parameters and encryption keys.

The CCDNA instantiates the Cell by constructing the components mentioned in section II passing the provided unique id as a bootstrap parameter. Then the CCDNA starts to interpret the deployment configuration file in order to generate separate configuration files for each Cell unit. Such files will describe modifications to the default task assignment, or special considerations to be taken care off at the time of execution.

The execution starts when the execution unit asks the STM for the starting checkpoint, the STM will get this information as a part of the deployment configuration file. STM will repeatedly provide this information to the execution unit at each shuffling event. The execution unit starts to launch the first variant while passing the appropriate bootstrapping parameters.

The last executed checkpoint value will be held by the STM locally, and remotely at the RCTS that will receive it via the Cell beacon messages.

At runtime, variants will update the STM frequently with the checkpoint advance and any other special needs via a dedicated communication channel.

Quality attribute manipulation: the following is an example for a situation that necessitates manipulating the current targeted quality attribute. An attacker might be able to induce a change in the system surroundings, like a DOS attack that aims to overload the network. CyberX will respond to such change in the normal workload by shuffling the currently executing variant to a more resource efficient variant. CyberX will ask Cells close to the induced event to change their variant to target a different quality attribute (e.g. performance) that suits the induced change in the environment.

At the time of shuffling, the Cell diversity manager gives the shuffling signal to the STM and the execution unit. These units will start the process after the next reported checkpoint and based on the provided shuffling policy.

We have two main realization modes for the shuffling operation the **greedy** and the **light** modes. The system designer can select either one of them based on the available deployment-platform host resources, and the criticality of the application. **The greedy-mode** with seamless handover offers virtually no-downtime but duplicates the resource usage at the time of shuffling, and the **lightweight-mode** offers no-resource increase at the time of shuffling on the account of increasing the transition time by the time needed for variant loading and synchronization. We will briefly describe both.

The greedy-mode "local replication": Upon reception of the shuffling signal, the execution unit starts to load the new variant in freeze "ideal" mode. The new variant will connect to the STM that will locally synchronize the execution checkpoint with it. The communications unit will duplicate all the inputs to the old and the new variant. Upon reception of the ACK Signal from the STM and the communications unit confirming that the synchronization is completed, the execution unit sends pause signal to the old variant, and a resume signal to the new one followed by a termination signal to the old variant.

The lightweight-mode: Upon the reception of the shuffling signal, the execution unit starts by local synchronization with the STM for the checkpoint update. Then it pause the old variant, and informs the STM and communication unit about the execution hold. The communication unit will buffer incoming messages for the duration of the handover. The execution unit will terminate the variant, and starts loading the new variant with the last known checkpoint, and informs the communication unit and the STM about the successful loading to resume execution. The communications unit will send buffered messages to the new variant.

VI. EVALUATION

In this section, we present the results of multiple experiments that were performed using a MATLAB based simulator. These experiments have different objectives regarding evaluating the effect of enabling autonomic adaptation and intrinsic failure recovery on the system performance with respect to failure downtime and resource consumption.

Table I shows the main parameters used in the simulation. The network parameters are mainly static parameters used to setup the experiments, except for the deployment of fresh Cells in the network. The dynamic part depends on a set of distributions mentioned in the column named "Generator ".

The failure or environment-change parameters show the spatiotemporal distribution of failure /environment-change events and the event type that necessities variant change in response to such event. The recovery parameters represent the initial recovery mode for each Cell, and the dynamic recovery change through the experiment lifetime. Deploy-new-Cell parameters represent the rate and location for the deployment of fresh Cells to replace dead or problematic Cells in the network. All experiments had the same period of 6 hours with a sample rate of six minutes giving us 60 samples (time slots) within the network of Cells. The presented parameters in (Run 1) were used to device Figure 10, and 12. We used the parameters in the three runs to evaluate the effect of increasing the failure generation rate illustrated in Figure 11.

TABLE I. PARAMETERS USED FOR THIS STUDY

Classification	Parameter		Generator	Run1	Run2	Run3
Network	Network size		Static	10*10	10*10	10*10
	# shuffling variants		Static	8	8	8
	Exp_Time		Static	6	6	6
	App_exe_time		normal	50	50	50
	Deploy new Cell	Period	Poisson	20	18	16
		Location	normal	8,3	8,3	8,3
	Resource usage	Cell	Static	5	5	5
		For Replica	Static	3	3	3
		Cell failure		2	2	2
Recovery	Recovery at deploy		normal	8,3	8,3	8,3
	Mode change	Period	Poisson	20	18	16
		Type	normal	8,3	8,3	8,3
Event	Failure or environment event change	Timing (Period)	Poisson	22	20	16
		Location	Normal	11,3	9,4	10,2
		Type	Uniform	10	10	10

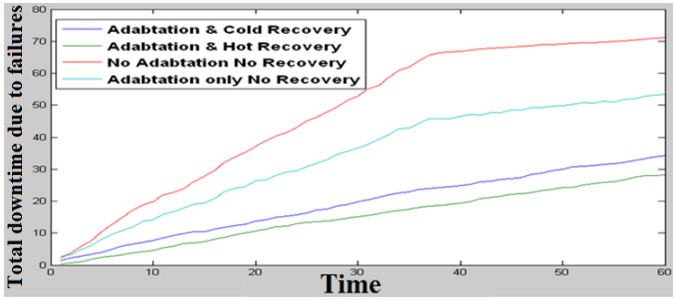

Figure 10. The total downtime in response to failures due to changes for different recovery modes with and without adaptation.

Figure.10 illustrates the effect of failure due to unexpected changes on the total downtime with and without CyberX autonomic adaptation. The figure reflects the different system responses to failures when we activate and deactivate CyberX autonomic adaptation with and without coarse or fine-grained recovery modes. The experiment shows significant improvement in minimizing the failure downtime when the CyberX autonomic adaptation is active as the system adapts autonomously to most of these changes minimizing the number of failures. Additionally, the total downtime significantly decreases when we activate CyberX fine or coarse grained recovery. Both recovery modes will rabidly recover failed Cells minimizing the overall failure downtime.

Figure 11 presents the effect of increasing the failure generation rate by increasing the number of changes over time on the total downtime while utilizing coarse or fine-grained recovery modes. The experiment shows that CyberX fine grained recovery always minimizes the failure downtime when compared to coarse grained recovery. In coarse grained recovery mode, CyberX spend more time instantiating replacement Cells; while in fine grained mode, replicas take over and resume execution first then a new replica is instantiated without holding the execution restoration. Figure 12 illustrates that such fast recovery comes on the expenses of consuming more resources. This figure reflects the total resource usage through the experiment with different recovery and adaptation modes.

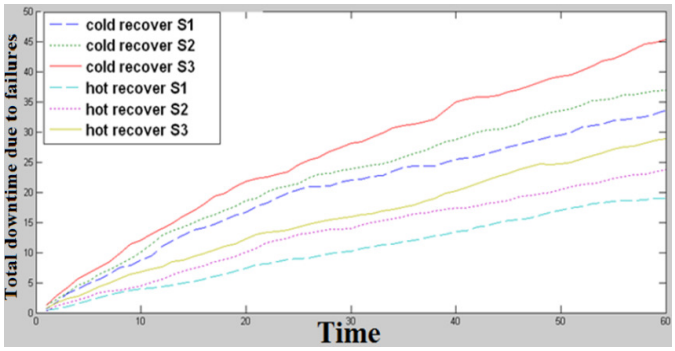

Figure 11. The total downtime in response to increasing failure generation rate for three different experiments and different recovery modes.

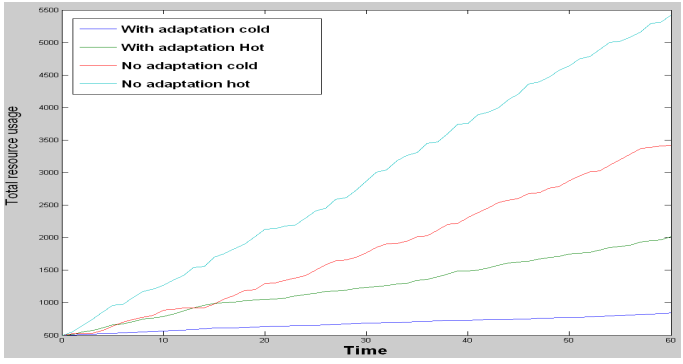

Figure 12. The total resource usage in case of failure with different recovery and adaptation modes

Figure 12 illustrates the effect of using CyberX autonomous adaptation to minimize failures, in saving some of the resource that would have been wasted in the recovery of such failures. Enabling such feature provides some guarantees that the system will always consider using the right resources at the right time while maximizing the system quality-attribute satisfaction-scope. Further, CyberX attempts to recompense resources wasted due to failure recovery by changing the system targeted quality attribute towards optimizing the resource usage after each recovery event. CyberX usually favor using one of the resource efficient variants to resume execution after each recovery event. CyberX do that while maintaining the balance between the different application objectives and targeted quality attribute to the best interest of the application while efficiently maximizing resource utilization.

VII. Related Work

CyberX is designed to manage COA-based systems to enable constructing elastic, dynamic, and adaptable software products with intrinsic support for situation and context aware fault tolerance. Currently software products depend mostly on static or partially dynamic architectures where data, logic, and/or physical resources are primarily tightly coupled. Multiple attempts have been presented in the literature to partially decouple these design concerns. Object (OOA), Agent (AOA), and Service (SOA) oriented architectures are examples [1, 2, 3]. However, up to our knowledge our COA is the only architecture that comprehensively supports intrinsic separation of design concerns needed for runtime re-programmability, intrinsic autonomic online composability, and dynamic software adaptation and elasticity.

Attempts have been presented towards enabling some of these features separately. AOA utilized autonomic building blocks while SOA and OOA used non-autonomic components. Using autonomic building blocks facilitated supporting non-deterministic behavior change in AOA by explicit use of soft computing as presented in [5]. However, supporting online composability is not clear in AOA, while in OAA and SOA it is enabled either by aggregation [6] or by service composition [7].

The COA Cell separates logic from physical resource management by constructing an intelligently-managed elastic thin virtualization layer between the application and the underlying physical resources. Such construction facilitates unifying the execution platform for distributed applications regardless of the configuration of the host platform. Unifying the execution environment waives the load of building platform/OS specific application for each targeted platform. In addition, the maintainability issues are divided between the developer and the technology owner. Software developers are concerned with maintaining the application itself, while the technology owner is responsible for maintaining the execution platform.

Partially elastic virtualization approaches were presented for loosening the bond between physical and logical resources; where applications are partially compiled at the production phase to be executed over virtual machine host [1, 14]. Such approached can be used to build a uniform execution environment for distributed applications. However, such approaches presented static elasticity and partial separation of design concerns. They did not separate data from logic and physical resources. Such separation is a key enabler for supporting intrinsic fault-tolerance, live-mobilization, and runtime adaptation to frequently changing execution environment.

Our approach provides an intelligent elastic virtualization utilizing mobile software capsules (Cells) that gets specialized at runtime facilitating online re-programmability. This feature when managed by CyberX enables COA Cells to seamlessly move between heterogeneous hosts, while autonomously adapting to any resulted changes. Additionally, CyberX-managed COA Cell can encapsulate different code variants and switch between them at runtime. CyberX utilized this unique feature to enable runtime manipulation of targeted quality attributes. Doing so, facilitates real-time adaptation to execution environment changes optimizing the application performance, resource-utilization, and enhancing its reliability, survivability, and compatibility. Based on our knowledge utilizing any of the available virtualization techniques to enable such features were not possible prior to our work.

Component diversity was investigated in Genesis [8], were the idea of providing both design diversity in the form of multiple variants representing different designs of the same specification as well as data diversity were proposed. Compiler guided code variance approach [9] aimed to present automated massive-scale software diversity by the help of automated variant generation and utilizing multi-core platforms. More advanced diversity employment approaches with the objective of anomaly detection through detecting flow deviation but with fewer constraints were presented in [10, 11]. A major drawback of such solutions is the need for virtualizing every input to the whole set of executing variants at the same logical point to be able to detect the abnormal deviation of the execution flow.

Based on our knowledge utilizing runtime hot shuffling of software variants for quality attribute hot manipulation was not previously investigated. Additionally, failure recovery mechanisms were not investigated as most of these solutions presented static diversity with low probability of failure. None of them investigated the idea of a comprehensive solution that provides elastic, autonomous, resilient, situation-aware platform targeting different quality attributes, while dynamically shuffling its software components to suit changes in the surroundings. Another drawback of these solutions is the massive use of resources to realize diversity using heavy virtualization techniques and multicore or multiprocessor platforms.

VIII. CONCLUSION

In this paper, we presented the CyberX platform designed for cyber trust management through supporting elasticity, diversity, awareness, cooperation, and intelligence. CyberX utilized our COA capability to induce autonomous execution elasticity and adaptability, and to enable adjusting the system shuffling and recovery policies at runtime to meet the continual operational environment changes. Further, CyberX uses its situation-aware, autonomic adaptation and dynamic failure recovery mechanisms to enhance software resilience against failures and attacks. Results showed that CyberX-managed COA-based software systems can efficiently adapt to maintain the desired performance and resilience objectives even in hazardous, unstable environments at a reasonable overhead. Some interesting challenges still to be addressed. include utilizing application-level virtualization to enable seamless Cell migration across heterogeneous platforms, autonomous detection and profiling of environment changes; adjusting shuffling and recovery settings based on context; formalizing an automated variant generation system, and providing alternatives for legacy non-COA-ready software.

REFERENCES

[1] VMware, (2012, Jun), “Application Virtualization made simple,” Available: http://www.vmware.com/products/thinapp/overview.html

[2] G. Lawler, “Distributed architecture for the object oriented methods for interoperability,” Monterey, CA: Naval Postgraduate School, thesis, 2003

[3] C. Hahn, C. Madrigal-Mora, and K. Fischer, “Interoperability through a platform-independent model for agents,” 3rd International Conference on Interoperability for Enterprise Software and Applications, 2007.

[4] C. Seo and B.P. Zeigler, “DEVS namespace for interoperable DEVS/SOA,” Winter Simulation Conference, 2009.

[5] C. Carrascosa, A. Terrasa, J. Fabregat, V. Botti, “Behaviour management in real-time agents,” Fifth Iberoamerican Workshop on Multi-Agent Systems, pp. 1-11, 2004.

[6] A. Tolk, S. Diallo, C. Turnitsa, L. Winters, “Composable M&S Web services for Net-centric Applications,” Journal of Defense Modeling and Simulation 3 (1) 27-44, 2006

[7] P-O. Östberg , E. Elmroth. “GJMF - A Composable Service-Oriented Grid Job Management Framework,” Preprint available at http://www.cs.umu.se/ds, submitted, 2010

[8] J. C. LKnight, J. W. Davidson, D. Evans, A. Nguyen-Tuong, C. Wang, "Genesis: A Framework for Achieving Software Component Diversity," Technical Report AFRL-IF-RS-TR-2007-9, University of Virginia, January 2007

[9] S. Forrest, A. Somayaji, and D. Ackley, “Building diverse computer systems,” 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), pages 67–72, 1997.

[10] T. Jackson, B. Salamat, G. Wagner, Ch. Wimmer, and M.Franz, “On the Effectiveness of Multi-Variant Program Execution for Vulnerability Detection and Prevention,” International Workshop on Security Measurements and Metrics (MetriSec 2010), September 2010.

[11] M. Franz, “E unibus pluram: Massive-Scale Software Diversity as a Defense Mechanism,” New Security Paradigms Workshop 2010 (NSPW 2010), September 2010.

[12] Cyber Trust (2009) Program Solicitation NSF 08-521, Available : http://www.nsf.gov/pubs/2008/nsf08521/nsf08521.htm

[13] M.Azab, R.Hassan and M.Eltoweissy, "ChameleonSoft: A Moving Target Defense System," 7th International Conference on Collaborative Computing, 2011.

[14] R. Spruijt, (2012, Jun). “Application Virtualization Smackdown: Head-to-head analysis of Citrix, Endeavors, InstallFree, Microsoft, Spoon, Symantec and VMware,” Available: http://www.brianmadden.com/blogs/rubenspruijt/archive/2010/09/22/application-virtualization-smackdown-head-to-head-analysis-of-endeavors-citrix-installfree-microsoft-spoon-symantec-and-vmware.aspx