DEMO: Action Recommendation for Cyber Resilience

Luke Rodriguez, Darren Curtis
Sutanay Choudhury, Kiri Oler,
Peter Nordquist
Pacific Northwest National Laboratory
firstname.lastname@pnnl.gov

Pin-Yu Chen
University of Michigan Ann Arbor
pinyu@umich.edu

Indrajit Ray
Colorado State University
indrajit@cs.colostate.edu

ABSTRACT

We demonstrate an unifying graph-based model for representing the infrastructure, behavior and missions of an enterprise. We introduce an algorithm for recommending resilience establishing actions based on dynamic updates to the models and show its effectiveness both through software simulation as well as live demonstration inside a cloud testbed. Our demonstrate will illustrate the effectiveness of the algorithm for preserving latency based quality of service (QoS).

Categories and Subject Descriptors

H.1.m [**Information Systems**]: Models and Principles

Keywords

cyber security, cyber resilience, recommendation engine

1. INTRODUCTION

Resilience is defined as the ability of an organization to continue to function, even though it is in a degraded manner, in the face of *impediments* that affect the proper operation of some of its components. Impediments can be randomly occurring failures of software services or hardware systems in an enterprise, or it may be unavailability of services or systems as the consequence of a cyber attack. A cyber enterprise is an elaborate web of applications, software, storage and networking hardware with complex dependencies among them. Although their building blocks may have been designed to be robust against failures, it is not easy to answer if the enterprise-web as a whole is resilient. Therefore, developing a unifying framework for performing what-if analysis is a necessary first step towards quantification of organizational resilience.

1.1 Motivation

We use a fictional small e-commerce company named VISR as our running example (Figure 1). VISR has a CEO, an intern, and a team of developers and HR professionals. Its physical network is divided into two subnets. R1-R3 are routers connecting these subnets. All the users are mapped to the subnet on right, while its in-house services are hosted on the left subnet. VISR has three missions:

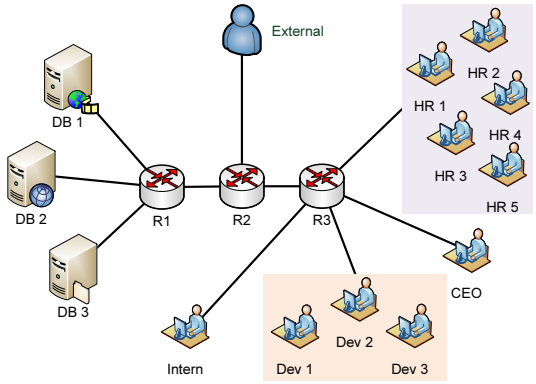

Figure 1: Illustration of the small e-commerce company used as the motivating use case.

1. A sales mission, as online sales is its primary source of revenue. This necessitates guaranteeing the availability of DB2 and the integrity of information inside it.
2. CEO's strategic mission. CEO's workstation has intellectual property information whose confidentiality and integrity needs to be guaranteed.
3. Product development mission which requires availability of DB1 and DB3 to the Dev group, HR group and the intern.

With the missions being defined, we raise the big question: What does mean for VISR to be resilient? In a longer version of this paper [1] we outline a number of common scenarios that companies such as VISR needs to address on a day to day basis. In our demonstration, we use our unified model to address these scenarios and determine what real-life actions such as turning off systems, disabling users, or selectively blocking communication across machines might be effective.

1.2 Approach

In order to fully explore the problem, we present a demonstration on two different environments. First, we present a software simulator that represents exactly the structure in Figure 1 and in which all machines and traffic are generated deterministically from a random seed. This framework allows us to test our scenarios and methods with a great degree of freedom in order to explore how our model reacts to various inputs. Doing so provides a great theoretical foundation, but more is required to prove that this truly

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
ACM 978-1-4503-3832-5/15/10.
DOI: http://dx.doi.org/10.1145/2810103.2810104 .

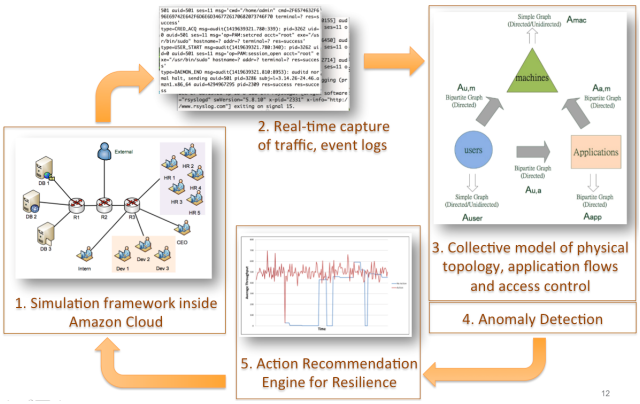

Figure 2: Demonstration pipeline executing inside Amazon Web Services.

can integrate into real-world practice. For this reason, we also introduce an Amazon Web Services (AWS) implementation whose configuration is shown in Figure 2. In this framework we can see how our theory interacts with practice and test the ability of our model to recommend real actions in real-time. Using AWS allows us a great deal of flexibility with how we configure the network, but still is grounded in reality enough to help us notice and correct any inaccuracies in our purely theoretical model.

Figure 2 shows the decision making loop in which the recommendation engine operates. This is an automated process that takes two sets of data as input: a steady-state description of the system along with a current snapshot. The process then calculates metrics of the system health such as latency distribution, frequency of authentication requests by processing the data relevant to the model description. If degradation is detected, the steady-state and current snapshots are compared in order to make a recommendation of an action to take that is tailored to restoring that particular metric of system health.

Given the enterprise at time t, $\mathcal{E}(t)$, the occurrence of an impediment Δ, drives the system to an intermediate state $\mathcal{E}(t+\delta_t)$. If $R_k(\mathcal{E}(t)) > R_k(\mathcal{E}(t+\delta_t))$, where R_k are metrics derived from latency or authentication statistics, then a resilient system will need to *act*. An action $\mathcal{A}$ is a resiliency preserving action if it transforms the system state such that $R_k(\mathcal{E}(t+1)) > R_k(\mathcal{E}(t+\delta_t))$, where $\mathcal{E}(t+1) = \mathcal{A}(\mathcal{E}(t+\delta_t))$. We apply this engine to recommend actions on both the simulated framwork of VISR and the AWS implementation.

2. SOFTWARE SIMULATOR

Experimental evaluation of resilience demands the ability to do the following: A) simulate steady state behavior, B) simulate impediments, and C) observe the system responding to the impediments. We did not find any existing data source that contains all three phases of resilient behavior. Using Denial of Service attacks (DoS) an example, there are many publicly available network traffic data sources capturing a DoS attack. However, we could not find any open dataset that captures the period of attack and the target system's subsequent recovery. Observing this dynamism is critical to quantitative studies of resilience and provides the motivation to develop and demonstrate a new simulator.

Our simulator supports a host of random graph generation tools, such as those available in NetworkX in Python. These graphs could be generated to represent the different components of our system, and then stitched together to create our model. However, while the kinds of graphs available to be randomly generated can be representative of many kinds of complex systems, we found that they did not accurately reflect the structure of our cyber networks. Therefore, we chose to use a pre-determined network configuration to build the model and execute our testing. We developed a modeling language that allows us to describe the behavior of our test company in a configuration file. Next, our code processes the configuration file and generates a series of snapshots of data conforming to the model. The data output is similar to network traffic flow (also referred as netflow) datasets captured in real environment.

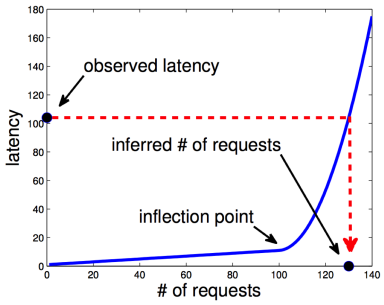

Figure 3: Latency-request function shown in Eqn. 1

We make a few assumptions about the nature of netflow data. First, we assume that all requests are roughly the same size, and thus assign them byte-sizes from a normal distribution with a mean of 50 and standard deviation of 5. More complicated, however, is the question of how to model flow duration or latency. Here we use the latency-request function shown in (1) for simulation.

$$f_j(x_j|x_j^*) = \begin{cases} a_j x_j + d_j, & \text{if } x_j < x_j^*; \\ c_j \cdot (x_j - x_j^*)^{b_j} + a_j x_j + d_j, & \text{if } x_j \geq x_j^*, \end{cases} \quad (1)$$

For server j, the independent variable x_j in the function is the number of requests currently being processed by the system receiving the requests. Figure 3 shows a plot of this function. To simplify, we assume that all systems share the same parameters, i.e., $(a_j, b_j, c_j, d_j, x_j^*) = (10^{-4}, 4, 10^{-1}, 10^{-1}, 100)$ for all servers.

Of particular note is the value of x_j^*, which determines the point of inflection at which the quality of service of the system beings to degrade quickly, as the latency of requests increases steeply. We add an element of uncertainty by using this equation to determine the mean of a normal distribution with standard deviation equal to one tenth of the mean, and draw a duration from this distribution. This is done by determining how many requests are currently being processed by a machine at the point at which a new flow is to be generated to it, and then calculating (1) for this value of x_j. To model the distribution of netflow requests through time, we first determine which users will be using which applications. Once this has been determined, 400 requests are put in a queue to be generated across the 200 seconds of model time with exact times chosen uniformly at random. The model then steps through these requests sequentially according to the randomly picked start time, with its duration assigned as described in the previous paragraph. This provides the base model on top of which attacks and restorations can be implemented.

3. AWS IMPLEMENTATION

The implementation of our VISR example on AWS consists of a collection of dynamically started virtual machines (VM), each of

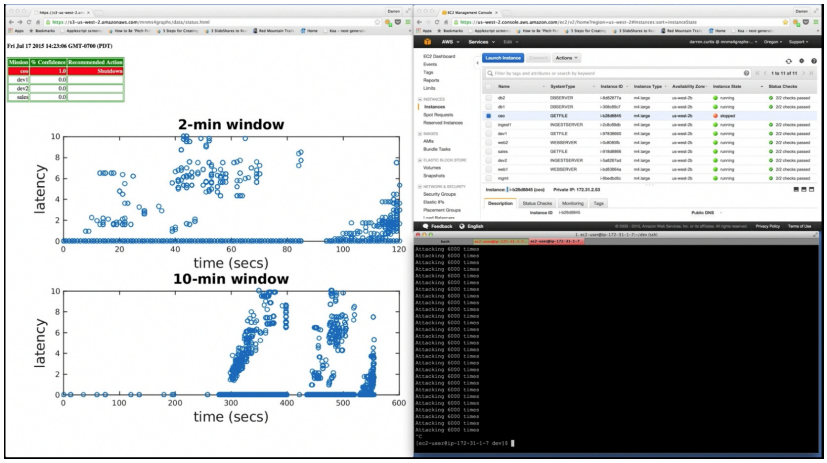

Figure 4: A screenshot of the demonstration setup. The left window shows the impact of attacks launched on the "CEO" machine as well as the recommendations produced for a set of VMs. Red indicates that a particular VM should be turned off. Currently we execute the actions manually fromAWS dashboard (right window).

which have an associated configuration file that describes its behavior (Figure 5). Traffic is simulated via simple traffic generators that mimic HTTP traffic and database server requests, which in turn allows us to simulate various impediments including denial of service, service degradation with overloading, and data deletion on a network random walk. The system state is then captured via netflow, audit logs and sys logs, which are sent to a machine outside of the target setup for processing and for action recommendation.

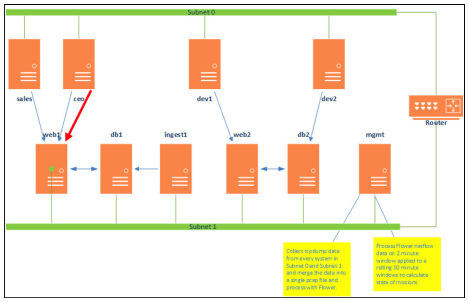

Figure 5: A dynamically configured set of VMs inside Amazon Web Services. Each VM spawns a set of randomized HTTP and database traffic generators whose behavior is specified in the testbed configuration.

The attacks used are composed of two varieties: those directed at disrupting availability and those aimed at impacting data integrity. The attacks on availability use a combination of simple Unix commands and scripted operations to power off or reboot a selected node, overload processor resources, and kill a selected process. The attacks on data integrity work in a similar fashion (i.e. via command line tools and functionalities) to delete all accessible files, delete a specified file, delete a specified directory, change a specified number of characters in a given file, or change a specified number of characters in all accessible files. The attacks all presume that the login information for at least one account has been compromised. Using that information ssh connections are attempted for all known existing nodes and when successful, some combination of attacks are attempted.

The recommendation engine [1] remains unchanged from the simulation framework with one exception; we experimentally learnt

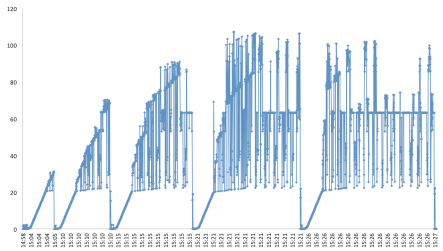

Figure 6: Learning the latency-request function from AWS setup. X and Y-axis represent time and throughput measured in bytes. Each burst represents an attack, and attacks with increasing intensity are launched sequentially.

the function associated with equation (1) from our AWS testbed. We systematically overloaded the system by stepping up the attack volume and measured the latency. Figure 6 shows that throughput (Y-axis) eventually flattens out as we increase the attack intensity. Latency is inversely proportional to the throughput, and each attack provides us with a sample to learn the function.

Figure 4 shows the workflow for the AWS-based demonstration.

4. OPEN SOURCE SOFTWARE

Code used for this demonstration is available as open source at https://github.com/cyber-resilience/cloud-simulation.

5. ACKNOWLEDGMENTS

Presented research is funded by the Asymmetric Resilient Cyber Security initiative at Pacific Northwest National Laboratory, which is operated by Battelle Memorial Institute.

6. REFERENCES

[1] S. Choudhury, P.-Y. Chen, L. Rodriguez, D. Curtis, P. Nordquist, I. Ray, and K. Oler. Action recommendation for cyber resilience. In *Proceedings of the ACM workshop on Automated Decision Making for Active Cyber Defense (SafeConfig 2015)*, 2015.