Action Recommendation for Cyber Resilience

Sutanay Choudhury, Luke Rodriguez,
Darren Curtis, 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

This paper presents an unifying graph-based model for representing the infrastructure, behavior and missions of an enterprise. We describe how the model can be used to achieve resiliency against a wide class of failures and attacks. We introduce an algorithm for recommending resilience establishing actions based on dynamic updates to the models. Without loss of generality, we show the effectiveness of the algorithm for preserving latency based quality of service (QoS). Our models and the recommendation algorithms are implemented in a software framework that we seek to release as an open source framework for simulating resilient cyber systems.

Categories and Subject Descriptors

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

Keywords

cyber security, cyber resilience, recommendation engine

1. INTRODUCTION

With cyber defenders waging an asymmetric war against cyber attackers and, for all practical purposes appearing to be loosing, researchers around the world are increasingly looking into designing *resilient* cyber infrastructures that can *survive* not only stochastic failures but also targeted attacks. Informally, *resilience* can be 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 [5]. 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.Cyber resilience borrows from the paradigm of fault-tolerance in information systems such as, the Internet or distributed computing platforms where failures of components are quite common. However, there are significant differences. Cyber resilience goes much beyond cyber fault-tolerance to require robustness against targeted attacks. It is easier to model and reason about fault-tolerance when failures (such as a that of a router on Internet or node in a compute cluster), system objectives (such as routing a packet or finishing a computation) and how the former affect the latter are well defined. A cyber enterprise, on the other hand, is an elaborate web of applications, software, storage and networking hardware with complex dependencies among them some of which are often be loosely defined. Each organization builds its own such web based on its mission or organizational objectives (We refer to this as the *enterprise-web* $\mathcal{E}$). The resultant webs are drastically different for every organization although some high-level similarities exist between organizations with similar objectives, such as universities. 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 model or a common framework for performing what-if analysis is a necessary first step towards quantification of organizational resilience. Such a model would allow one to identify resilience bottlenecks and/or recommend actions to make the system resilient.

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:

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? We outline a number of common scenarios in Table 1 that companies such as VISR needs to address on a day to day basis. How can a unified model be used to address a large class of scenarios and determine real-life actions such as turning off systems, disabling users or selectively blocking communication across machines?

1.2 Our approach

Unfortunately, there has been very limited work done in this area and we take the first step to fill the gap. Towards this end, we use a graph theoretic approach to develop an unifying model for representing the infrastructure, behavior and missions of an enterprise

Table 1: Description of impediments that can be addressed by the Enterprise-Web model

Failure/Attack	Description	Impacts	Resilient System Actions
Denial of Service	Disrupts services via overloading	G_{app}	Modify G_{app} . Introduce an intermediary layer between the attacker and the target service, or selectively block communication between server and a subset of clients.
Software/hardware failures	Web server DB2 fails as application runs out of memory or a power glitch disables R2.	G_{phy}, G_{app}	Harden important nodes in G_{phy}, G_{app}. Add redundancy into the systems to push out the "inflection point" shown in Figure 2 .
Point-of-Sale/Pass-the-hash attack	HR1 is compromised. Flat network facilities attacker move to other systems.	G_{access}	Segment G_{access}. The network should be as segmented as possible. The trade-off between a hub and spoke structure (flat network) vs segmented will be additional cost and complexity for extra security.
Web-Application Attack/Phishing	Intern clicks on a phishing email by mistake. Malware gets installed on his computer and starts communicating with command and control.	G_{app}, G_{access}	Spot command and control traffic, data exfiltration in G_{app}. . Segment G_{access}.

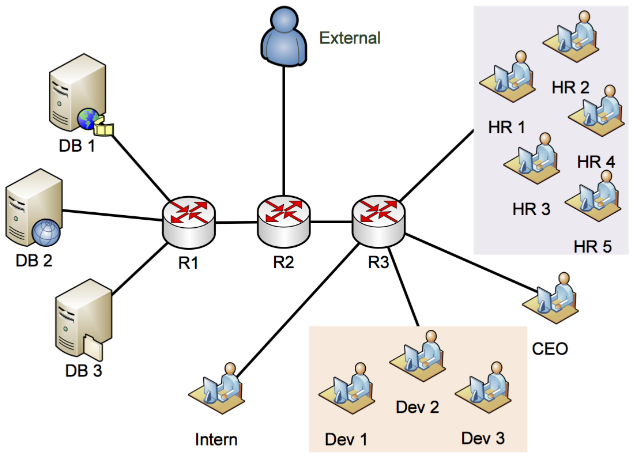

Figure 1: Our motivating example: A fictitious small e-commerce company named VISR.

and the dependencies among them (section 2). We also point out how the model is built from real world data sources (netflow, active directory, event logs etc.). We discuss how a diverse class of impediments can be handled by operations on the model (Table 1). Next, we present the resilient action recommendation as an optimization problem (section 3, equation 4) and introduce an algorithm (Algorithm 1). Without loss of generality, we show how the algorithm can achieve resiliency against attacks that causes degradation of the QoS of web services. We present our simulation approach in detail in section 4, followed by experimental results in section 5.

1.3 Contributions

We make the following major contributions in this work.

1. We introduce a multi-network model to capture the behavior of the enterprise. We show the model is expressive to address diverse impediments and act upon to achieve resiliency.

2. We introduce an algorithm for recommending resilience actions based on dynamic updates to the model. We establish a set of constraints that need to be satisfied for resiliency.

3. We demonstrate the validity of the algorithm through simulations and experiments.

2. MODELING THE ENTERPRISE-WEB

We begin with a formal definition of a *mission*.

DEFINITION 1. MISSION A *mission* $\mathcal{M} = (\mathcal{A},\mathcal{U},\mathcal{R})$ is a 3-tuple set of *applications* $\mathcal{A}$ driven by a set of *users* $\mathcal{U}$ on a set of *resources* $\mathcal{R}$.

Next we describe the enterprise-web model as a set of graphs, $\mathcal{E} = (G_{phy}, G_{app}, G_{access}, G_{host})$.

DEFINITION 1 We define $G_{phy} = (V_{mac}, E_{link})$ as the simple graph simulating the physical network.

This network is likely to belong to one of the following topologies: a) star network where a central node is connected to the rest of the nodes in the graph, b) a ring with trees connected, or c) a hierarchical composition of the above. We build this graph using traceroute-derived data and schemas obtained from system administrators.

DEFINITION 2 We define $G_{app} = (V_{app}, V_{user}, E_{flow})$ as a directed multigraph representing the application level behavior.

This models all requests made by either a user or an application to an application node in the graph, of which there may be multiple. We build this graph from network traffic (netflow) data.

DEFINITION 3 We define $G_{access} = (V_{user}, V_{mac}, E_{perms})$ as a directed bipartite graph where each edge in E_{perms} indicates that a user in V_{user} has access to a host in V_{mac}.

These permissions allow us to keep track of which machines could be compromised by a single malicious user, or how an attacker can laterally move through the network. We build this graph from inferring information from event logs, audit logs or active directory data.

DEFINITION 4 We define $G_{host} = (V_{app}, V_{mac}, E_{host})$ as a directed bipartite graph connecting applications in V_{app} to the machines in V_{mac} that they are hosted on.

Connecting them in this way allows us to properly determine which services an attack on a particular node might degrade, or vice-versa. We build this graph from event logs.

Using these different views of the entire enterprise-web allows us to focus on which types of assets and connections are critical to detecting and mitigating different kinds of attacks. For example, addressing a denial of service attack (Table 1, row 1) will require using G_{app} only. We exclusively focus on this use case in the next section and remainder of the paper.

3. RECOMMENDATION ENGINE

OBJECTIVE The recommendation engine 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 a subset of four graphs that make up $\mathcal{E}$. 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 $\mathcal{E}(t+1) = \mathcal{A}(\mathcal{E}(t+\delta_t))$.

In this paper we demonstrate a proof-of-concept recommendation engine that only uses one metric and looks specifically at the quality of service of the network applications by examining G_{app}.

The action recommendation engine works by observing the network flow between users and servers and inferring the server load (i.e., requests to be processed) through a latency-request function which is associated with the server capability. The output is a score indicating the recommendation of blocking the network flow from a subset of users. Borrowing from the concepts of stress and strain in mechanical problems, let x_j denote the number of current requests at server j and let x_j^s denote the number of requests at the steady state for server j. Figure 2 shows a critical request value (i.e., the knee point) above which the latency has a surge increase for server j is denoted by x_j^*. Below the critical request value x_j^*, the latency grows linearly with the increase of request.

The latency-request function f_j for server j takes the form

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

where the latency-request function is characterized by the parameters a_j, b_j, c_j, d_j and x_j^*. An illustration is shown in Fig. 2.

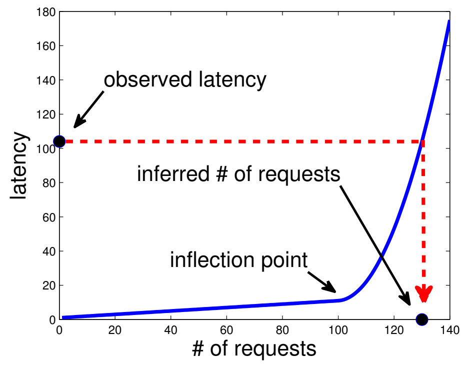

Figure 2: Illustration of the latency-request function. The inflection point is caused by the critical request value. The number of requests in a server is inferred via the observed latency and the latency-request function.

It is easy to check that f_j is a differentiable convex function with respect to x_j for any $a_j \geq 0$, $b_j \geq 1$, $c_j \geq 0$ and $d_j \geq 0$. From (1) the gradient of f_j at x_j is

$$\nabla f_j(x_j|x_j^*) = \begin{cases} a_j, & \text{if } x_j < x_j^*; \\ b_j c_j \cdot (x_j - x_j^*)^{b_j - 1} + a_j, & \text{if } x_j \geq x_j^*. \end{cases} \tag{2}$$

Let n_{user} and n_{server} denote the number of users and servers, respectively. We denote $\mathbf{r} = [r_1, r_2, \ldots, r_{n_{user}}]^T$ as an $n_{user} \times 1$ binary column vector. $r_i = 1$ if the network flow from user i is recommended to be blocked; otherwise $r_i = 0$. Let $\mathbf{A}$ denote the $n_{user} \times n_{server}$ matrix where its entry $[\mathbf{A}]_{ij}$ characterizes the number of requests between user i and server j. Essentially $\mathbf{A}$ is the matrix representation of G_{app}, and since we are only dealing with G_{app} in this paper, we use the notation $\mathbf{A}$ instead of $\mathbf{A}_{app}$ for the sake of brevity. When blocking users according to $\mathbf{r}$, the number of removed requests at server j can be expressed as

$$\sum_{i:r_i=1} \mathbf{A}_{ij} = \sum_{i=1}^{n_{user}} \mathbf{A}_{ij} \mathbf{r}_i = \mathbf{r}^T \mathbf{A} \mathbf{e}_j, \tag{3}$$

where $\mathbf{e}_j$ is a canonical column vector such that all entries of $\mathbf{e}_j$ are zero except its j-th entry being 1. Therefore when users are removed according to $\mathbf{r}$, the latency of server j can be represented as $f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*)$. Let $h_j(\cdot)$ be a nondecreasing differentiable convex function such that $h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right)$ evaluates the cost of latency changes between the subsequent latency $f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*)$ and the steady state latency $f_j(x_j^s)$. Furthermore, we incorporate user priorities into the model such that each entry in the $n_{user} \times 1$ nonnegative column vector $\mathbf{q} = [q_1, q_2, \ldots, q_{n_{user}}]^T$ reflects the cost of removing an user.

By relaxing $\mathbf{r}$ to be real-valued in $[0,1]^{n_{user}}$, we formulate the action recommendation problem as an optimization function

$$\begin{aligned} \text{minimize} \; & \sum_{j=1}^{n_{mac}} h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right) \\ & + \lambda \cdot g(\mathbf{r}) + \beta \cdot \mathbf{q}^T\mathbf{r} \\ \text{subject to } & r_i \in [0,1] \; \forall \; 1 \leq i \leq n_{user}, \end{aligned} \tag{4}$$

where $g(\mathbf{r})$ is the regularization function on $\mathbf{r}$ and $\lambda \geq 0$ is the regularization parameter for $g(\mathbf{r})$. The term $\mathbf{q}^T\mathbf{r}$ specifies the cost of user removal according to $\mathbf{r}$ and $\beta \geq 0$ is its regularization parameter.

Here we assume $h_j(x)$ takes the form

$$h_j(x) = \begin{cases} 0, & \text{if } x < 0; \\ x^{\phi_j}, & \text{if } x \geq 0, \end{cases} \tag{5}$$

for any $\phi_j > 0$. Therefore the latency cost between current state and the steady state $h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right)$ is zero if the subsequent latency $f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*)$ is less than the steady state latency $f_j(x_j^s|x_j^*)$, and $h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right)$ increases with $f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*)$ when $f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) \geq f_j(x_j^s|x_j^*)$. The regularization function $g(\mathbf{r})$ is set to be the ℓ_1 norm $\|\mathbf{r}\|_1 = \sum_{i=1}^{n_{user}} |r_i|$, which is a non-smooth convex surrogate function that promotes sparsity in $\mathbf{r}$. That is, the solution suggesting few user removals will be preferred for the optimization problem. Let $\nabla h_j(x)$ denote the gradient of h_j at x and let

$$H(\mathbf{r}) = \sum_{j=1}^{n_{mac}} h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right). \tag{6}$$

Then the gradient of H at $\mathbf{r}$ can be expressed as

$$\nabla H(\mathbf{r}) = -\sum_{j=1}^{n_{mac}} \nabla h_j\left(f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right) \cdot \nabla f_j(x_j - \mathbf{r}^T\mathbf{A}\mathbf{e}_j)\mathbf{A}\mathbf{e}_j, \quad (7)$$

where

$$\nabla h_j(x) = \begin{cases} 0, & \text{if } x < 0; \\ \phi_j \cdot x^{\phi_j - 1}, & \text{if } x \geq 0. \end{cases} \quad (8)$$

It can be shown that $H(\mathbf{r})$ is a differentiable convex function with respect to $\mathbf{r}$.

The optimization problem can be solved via proximal algorithms such as the fast Iterative shrinkage thresholding algorithm (FISTA) [1]. The action recommendation algorithm is summarized as follows. In practice the current request x_j and steady state request x_j^s are inferred via the observed latency average and the specified latency-request function f_j as described in Fig. 2. The following functions are used in the action recommendation algorithm:

$$F(\mathbf{y}) = \sum_{j=1}^{n_{mac}} h_j\left(f_j(x_j - \mathbf{y}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right) + \lambda\|\mathbf{y}\|_1 + \beta\mathbf{q}^T\mathbf{y}; \quad (9)$$

$$Q_L(\mathbf{y},\mathbf{z}) = \sum_{j=1}^{n_{mac}} h_j\left(f_j(x_j - \mathbf{z}^T\mathbf{A}\mathbf{e}_j|x_j^*) - f_j(x_j^s|x_j^*)\right) + \sum_{j=1}^{n_{mac}} (\mathbf{y}-\mathbf{z})^T(\nabla H(\mathbf{r}) + \beta\mathbf{q}) + \frac{L}{2}\|\mathbf{y}-\mathbf{z}\|^2 + \lambda\|\mathbf{y}\|_1 + \beta\mathbf{q}^T\mathbf{z}. \quad (10)$$

$\mathrm{T}_\alpha : \mathbb{R}^n \to \mathbb{R}^n$ is the shrinkage operator defined by

$$[\mathrm{T}_\alpha(\mathbf{x})]_i = (|x_i| - \alpha)_+ \operatorname{sgn}(x_i), \quad (11)$$

where $[\mathrm{T}_\alpha(\mathbf{x})]_i$ is the i-th element of $\mathrm{T}_\alpha(\mathbf{x})$, $(x)_+ = \max\{0, x\}$, and sgn(x) is the sign function of x.

Algorithm 1 Action Recommendation Algorithm

```
Input: latency-request function $f_j$ with parameters $\{a_j, b_j, c_j, d_j, x_j^*\}_{j=1}^{n_{server}}$, latency cost function $\{h_j\}_{j=1}^{n_{server}}$, inferred current request $x_j$, inferred steady state request $\{x_j^s\}_{j=1}^{n_{server}}$, user-server request matrix $\mathbf{A}$, regularization parameters $\lambda, \beta$, stopping criterion $\epsilon$
Output: score vector of action recommendation $\mathbf{r}$
Initialization: $\mathbf{r}_0 \sim$ unif$[0,1]^{n_{user}}$. Take some $L_0 > 0$ and $\eta > 1$. Set $\mathbf{y}_1 = \mathbf{r}_0, t_1 = 0, k = 1$.
Step $k$ :
1. Find the smallest nonnegative integer $i_k$ such that with
   $\bar{L} = \eta^{i_k} L_{k-1} =: \frac{1}{s_k}$ and $\widetilde{\mathbf{y}}_k = \mathbf{y}_k - s_k\nabla H(\mathbf{y}_k)$,
   $F(\mathrm{T}_{\lambda\cdot s_k}(\widetilde{\mathbf{y}}_k)) \leq Q_{\bar{L}}(\mathrm{T}_{\lambda\cdot s_k}(\widetilde{\mathbf{y}}_k), \mathbf{y}_k)$
2. $\mathbf{r}_k = \mathrm{T}_{\lambda\cdot s_k}(\widetilde{\mathbf{y}}_k)$
3. $\mathbf{r}_k = \left(\frac{\mathbf{r}_k}{\max_i[\mathbf{r}_k]_i}\right)_+$
4. $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
5. $\mathbf{y}_{k+1} = \mathbf{r}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{r}_k - \mathbf{r}_{k-1})$
6. $k = k + 1$
Repeat Step $k$ until $\|\mathbf{r}_k - \mathbf{r}_{k-1}\|_2 \leq \epsilon$
```

The operation $\mathbf{r}_k = \left(\frac{\mathbf{r}_k}{\max_i[\mathbf{r}_k]_i}\right)_+$ is a projection onto the feasible convex set $\mathbf{r}_k \in [0,1]^{n_{user}}$. To mitigate the effect of random initialization vector $\mathbf{r}_0$ which is drawn uniformly in $[0,1]^{n_{user}}$, one can perform the action recommendation multiple times and take the averaged results as the final score vector.

4. SIMULATION FRAMEWORK

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 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. The physical topology, users, and running applications remain unchanged through the entirety of our simulation, and we systematically randomize the output from the simulation.

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 in (1) for simulation. 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. 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 pro-

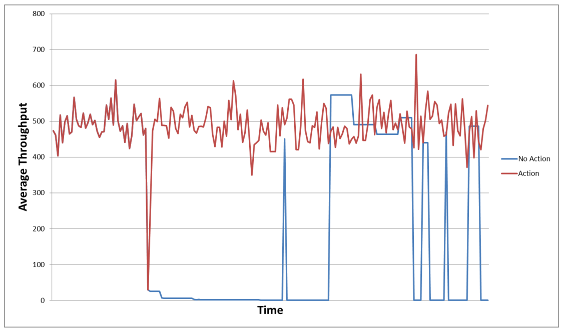

Figure 3: Plot for $x_j^* = 100$ for all j, with and without restorative action.

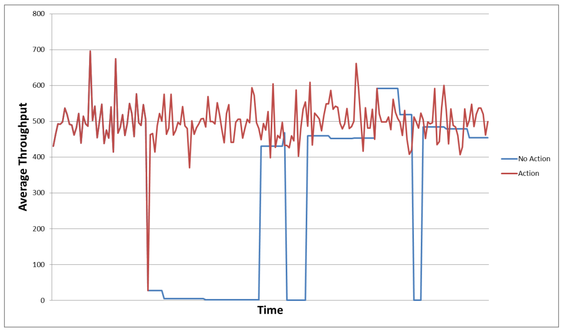

Figure 4: Plot for $x_j^* = 50$ for all j, with and without restorative action.

vides the base model on top of which attacks and restorations can be implemented.

5. EXPERIMENT AND RESULTS

5.1 Impact on recommendation on resilience

For a first pass at attack and system response simulation, we insert a denial-of-service (DOS) type attack by an external user (External in Figure 1) beginning at time 50 on the web server (DB2). At this point, the external user makes 1000 requests to the website host per second over the course of 5 seconds. This has the effect of disrupting all traffics to the web-server machine, which also reduces the quality of service of the email application hosted on DB2. The number of requests processed by the system is inferred by the average median durations and the latency-request function over all completed requests between users to a server. For the recommendation engine we set the regularization parameters to be $\lambda = 10$ and $\beta = 1$. Uniform user priority is used such that the vector $\mathbf{q}$ is a vector of all ones.

In the event that this is successfully detected by the recommendation engine, the action taken is to block the external user from making any more requests to the web application, and terminating any pending requests made by that user. Since the recommendations are returned in the form of a vector of $< user, score >$ pairs, where $0 \leq score \leq 1$, we must decide when this action should be taken. For this test we assign a weight to each user representing how willing we are to take action and disconnect them. We choose a value of 2 for the external user, 1 for the CEO, and 1.5 for all other users. If the product of this value and the score returned by

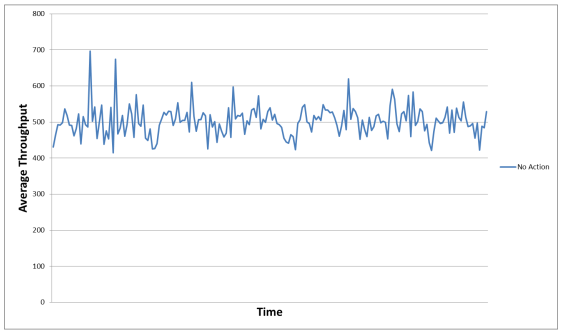

Figure 5: Plot for $x_j^* = 200$ for all j, with no action taken.

the recommendation engine is at least 1, then we take action against that user. For example, if the external user had a score of 0.8, we would take action against them. However, if the CEO had the same score, we would not take action.

For the experiment itself, we generated data from a fixed seed using three different latency functions (Equation (1)), varying only the location of the point of inflection x_j^*. We start with $x_j^* = 100$ for all j. We first run the model through the entirety of the scenario taking no action, and monitor the health of the web server by observing the average throughput over time (measured as $\frac{num_bytes}{duration}$) for all flows that terminate during a given one-second time window (Figure 3). We see that the throughput of the system drops severly when the attack first occurs, and then it appears to alternate between recovery and crashing again. This apparent recovery is an artifact of the fact that we are observing only the flows that terminate during a certain time window, so there may still be many pending requests to the server that are not being taken into account. During the periods of “recovery”, a few flows have been created with shorter duration (and therefore higher throughput), but then there are no flows that terminate in the next windows, providing a throughput of zero. This is very unstable and undesirable behavior. In contrast, Figure 3 shows the throughput of the system when action is taken as recommended by the recommendation engine. We see the same drop in service that we did previously, but our action effectively blocks the degredation from continuing and restores the state of the system to normal.

Figure 4 shows the same two plots for the case where $x_j^* = 50$ for all j. We see that this has very little effect on the overall story observed by this metric. It is possible that if we refined the observation window for our recommendation engine, we would be able to catch the attack earlier in this case. In Figure 5 we see the effect of moving to $x_j^* = 200$ for all j. In this case, we have made our system more robust, and it can handle the DOS attack from the external user without needing to take any action. Thus we never see the degradation of quality of service.

5.2 Effect of user priority on the recommendation

We investigate the effect of user priority vector $\mathbf{q}$ on the recommendation scores by considering two different user priority configurations: one adopting uniform priority (i.e., $q_i = 1$ for all i) and one adopting differentiated priority. For the differentiated priority we set the priority value of the external user to be 5 and set the priority values of other users to be 0. The results are shown in the following table. In the current window only two users (External and Dev3) are communicating with an overloaded server. External has sent more requests than Dev3 and blocking either of the users

Table 2: Recommendation scores with respect to different user priority vector $\mathbf{q}$. The results are averaged over 50 runs.

	uniform user priority	differentiated user priority
user	score	score
External	0.86	0.16
Dev3	0.14	0.84
others	0	0

can restore the system back to the steady state. Observe that in the case of uniform user priority, the score of External is significant due to heavy requests. However, as we impose more user priority on External, the score of Dev3 becomes significant and the score of External is drastically reduced. In both cases, blocking other users will not mitigate the server's load. The results demonstrate the capability of the recommendation engine that the output results can provide accurate recommendations and reflect the effect of differentiated user priority.

6. RELATED WORK

We are focused on the problem of determining dynamic actions to achieve resiliency with regards to failure of services or hardware, or system compromises. Identifying optimal design strategies that address widely ranging system objectives is a major research theme. Examples of such work include optimal hardening; given a limited budget, such works [3] outline the approach towards identifying software or hardware components that should receive the newest patches or benefit from additional redundancy. Hu et al. [4] explores the well-defined problem of optimal controller placement in software defined networks. Moving target defense [9] and service migration in cloud computing are two strongly related areas. [7] formulate a cloud-based service security model that incorporates cloud-specific features such as virtual machine (VM) migration and compatibility of the migration. The authors propose a probabilistic service deployment strategy that exploits the dynamics and heterogeneity of attack surfaces. Okhravi et al. [6] evaluate dynamic platform techniques as a defense mechanism. Particularly, we draw inspiration from their experiment setups for performing quantitative experiments for resilience use cases. From a graph theoretic point of view, modifying a graph to improve its resilience, or more specifically robustness has been studied by Chan et al. [2]. The authors use connectivity of the network as an objective function for robustness and propose algorithms to modify the graph by adding or deleting edges to control robustness. Ramuhalli et al. [8] describe a mathematical formulation for development and evaluation of autonomous reconstitution algorithms in dynamic cyber environments.

7. DISCUSSION AND FUTURE WORK

In this paper, we took the first steps towards developing a formal methodology to reason about the resiliency of a complex enterprise-web. We presented an unifying graph-based model for representing the infrastructure, behavior and missions of an enterprise and the dependencies among them. The big appeal of this approach is that it allows ingesting multiple data sources such as netflow, event logs etc. into one model, reason about actions in the model space and then transform actions determined in the model space such the deletion of an edge in a graph to a real world action such as blocking communication between a client and a server. We show how metrics computed from the model are utilized to implement an algorithm for recommending resilience establishing actions. We formulate a set of necessary constraints that need to be satisfied for a system to be resilient. We demonstrate the validity of the constraints and effectiveness of the algorithm through simulations and experiments.

At this stage, our model is somewhat simplistic. One of our objectives was to demonstrate the feasibility and advantages of using a graph-based approach for reasoning about system resiliency at which, we believe, we had been successful. Real-world situations involve much more complex dependencies including but not limited to conditional dependencies, cause-consequence dependencies, and temporal dependencies. Our immediate next step is to investigate such dependencies. The current model is also deterministic. One option is to use probabilistic graphical models for model representation and estimate probability values associated with mission events in a dynamic manner. In this context, we plan to develop quantitative metrics for quantifying the damage to mission continuity, cost to attack mission, and cost of implementing security hardening measures to mission quality of service. Lastly, we have not yet studied the effect of organizational policies on system resiliency. It is entirely possible that recommendations suggested by our methodology cannot be executed in practice because of different unique organizational policies and business relationships. Future work involves augmenting the current model to allow modeling of such policy based decision making.

8. ACKNOWLEDGMENTS

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

9. REFERENCES

[1] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] H. Chan, L. Akoglu, and H. Tong. Make it or break it: Manipulating robustness in large networks. SIAM, 2014.

[3] R. Dewri, I. Ray, N. Poolsappasit, and D. Whitley. Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, 11(3):167–188, 2012.

[4] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng. On reliability-optimized controller placement for software-defined networks. *Communications, China*, 2014.

[5] I. Linkov, D. A. Eisenberg, K. Plourde, T. P. Seager, J. Allen, and A. Kott. Resilience metrics for cyber systems. *Environment Systems and Decisions*, 33(4):471–476, 2013.

[6] H. Okhravi, J. Riordan, and K. Carter. Quantitative evaluation of dynamic platform techniques as a defensive mechanism. In *Research in Attacks, Intrusions and Defenses*. Springer, 2014.

[7] W. Peng, F. Li, C.-T. Huang, and X. Zou. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces. In *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014.

[8] P. Ramuhalli, M. Halappanavar, J. Coble, and M. Dixit. Towards a theory of autonomous reconstitution of compromised cyber-systems. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*. IEEE, 2013.

[9] J. Xu, P. Guo, M. Zhao, R. F. Erbacher, M. Zhu, and P. Liu. Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM, 2014.