Moving Target Defense for Placement of Intrusion Detection Systems, Risk Assessment and Countermeasure Selection

Sailik Sengupta[1] and Avinash Patil[1] and Mrunal Bodhe[1]

***Abstract*— Cloud based software systems often have a complicated networking structure which makes the security analysis of such systems challenging problem. In this project, we seek to address this problem in three parts– (1) identification of a placement strategy for Intrusion Detection Systems in an enterprise network while respecting the cloud service provider's limited budget and the customer's privacy demand, (2) risk assessment using the attack graph threat model leveraging the Common Vulnerability Scoring System metrics and (3) determining the vulnerabilities one should fix in order to obtain the maximum return on investment. In the end, we provide a website that takes as input an attack graph file (in xml format) and provides the results of these three modules, highlighting experimental results of our algorithms.**

Keywords. **Risk Assessment, Intrusion Detection Systems, Moving Target Defense, Attack Graph, Countermeasure Selection**

Project Demo. `https://goo.gl/TTcb4o`

Project Code. Uploaded to GitLab. We can give access to the private repository on github upon request. As some of this code might be used for an ongoing research work, we have not released this on a public repository.

I. INTRODUCTION

(a) In order to detect live attacks on a cyber-systems, administrators often deploy a variety of Intrusion Detection Systems (IDS) [1]. These systems investigate traffic on the wire ([2], [3]) or monitor access to files on a machine ([4], [5]) to flag anomalies that can result in to potential attacks. Cloud service providers (or defenders), who host multiple services on their service platform, often encounter non-trivial challenges when it comes to deploying IDS.

The biggest challenges include (1) investigating all traffic over the network leads to scalability issue and unwillingness from third party users hosted on the cloud platform due to privacy reasons, and (2) IDS often needs to run processes on the machines that could be otherwise, provided as computing resources, thus incurring costs for the cloud service provider.

Given a defender's limited budget, we look at the problem of placing IDS systems on the cloud network. It is trivial to see that if we place IDS systems for monitoring certain attacks on specific systems, the attacker (with reconnaissance on its side) will eventually be able to figure out our placement strategy. At this point, it can always select attacks that circumvents our IDS system, thus passing though our cloud network undetected. To address this, we propose the concept of Moving Target Defense for placement of IDS systems on cloud networks such that the attacker is always kept guessing as to weather their attack will remain undetected or not.

To that extent, we formulate this scenario as a two-player Stackelberg Game where the defender and attacker are the players. Defender's each action represent a specific placement of IDS and the attacker's action represent a particular CVE (Common Vulnerability and Exposures) it can exploit. The utility values of this game are obtained by leveraging the metrics designed by security experts in the Common Vulnerability Scoring System (CVSS). We obtain the Stackelberg Equilibrium for this game, which results in the optimal mixed strategy for the defender that it can use to switch among the various IDS placements.

(b) Attack Graphs provide valuable insights to security experts who evaluate the security of cyber-systems. We will perform attack (AND-OR) graph analysis that considers all paths that lead to the goal state/node and how the different interactions on each path affect the probability with the each node/state can be compromised.

(c) In this project, we use the term *countermeasure* to denote a code or configuration deployment that fixes a certain vulnerability in the system. In realistic settings, a software firm does not have unlimited resources to address all the security bugs at one go. Hence, a key question is *which security bug should be fixed first?*

(d) For intelligent countermeasure node selection, we remove one vulnerability at a time and recalculate cumulative probability to reach goal node. Notice that fixing a vulnerability modifies certain attack paths and can also disconnect the graph, pruning away large subgraphs. On this new graph, we can again apply the procedure mentioned in step (b) to obtain the probabilities of reaching the goal node, we call p_g. The vulnerability that, upon removal, results in the highest decrease in the value of p_g is the one the application owner should fix (or deploy a countermeasure against). In essence, we define the return of investment as the decrease in the probability with which the goal state is compromised.

(e) Finally, as opposed to a REST API, we develop a website that take as input an attack-graph file (in `xml` format) and outputs (1) probabilities of reaching the goal, (2) countermeasure selection that results in the highest return of investment and (3) mixed strategy based on which IDS should be placed (see Fig. 1).

Arizona State University {`mbodhe, apatil20, ssengu15`}`@asu.edu`

II. System Model and Problem Description

In this section, we will describe the system models for the three components we worked on. Since these are meta level components and can be used as a third party API given any attack graph, we have not integrated them with the `thoughtlab` environment. In each of the subsections, we elucidate the assumptions we make in defining our problem and talk abour the solutions we propose after this section.

A. Modeling IDS Placement

We consider two agents– the defender $\mathcal{D}$, who is trying to deploy IDS and the attacker $\mathcal{A}$, who is trying to remain undetected while attacking the system. Furthermore, we denote the an *attack* on our system as a two-tuple $\langle ip, cve \rangle$ which mentions the Common Vulnerability and Exposures (CVEs) used to exploit a certain machine. Given a system's attack graph, we extract a set A of all possible attacks.

The attacker's action set corresponds to A. If the defender has enough resources necessary to place a detection mechanism for detecting all the attacks $a \in A$, we have the best case scenario. Unfortunately, as mentioned before, for cloud networks these use up a lot of the valuable computing resource which could be used instead to earn more revenue. Thus, we assume that the defender mentions a constraint variable k to denote the number of attacks it can monitor at any given point of time. If $|A| = n$, then the defender can choose to deploy any of the $\binom{n}{k}$ placements, where k among the n attacks can be detected by deploying an IDS. These placements are the defender's action set.

As an example, consider your system has two machines $\{ip_1, ip_2\}$ and there are two vulnerabilities $\{cve_1, cve_2\}$ in ip_1 and one vulnerability $\{cve_3\}$ in ip_2. Thus, the attacker's action set is $\{\langle ip_1, cve_1\rangle, \langle ip_1, cve_2\rangle, \langle ip_2, cve_3\rangle\}$. Let us further assume that the defender can deploy two IDS ($k = 2$), which will cover any two of the three attacks. In this case, it can choose one placement strategy from the set

$$\left\| \begin{array}{l} \{(\langle ip_1, cve_1\rangle, \langle ip_1, cve_2\rangle), (\langle ip_1, cve_1\rangle, \langle ip_2, cve_3\rangle), \\ (\langle ip_1, cve_2\rangle, \langle ip_2, cve_3\rangle)\} \end{array} \right.$$

Notice that selecting one placement strategy out of the three placement strategies and sticking to is not secure in the long run since the attacker $\mathcal{A}$, with reconnaissance on its side, will eventually figure out the attacks against which $\mathcal{D}$ has placed detection mechanisms and will then attack to evade detection. Thus, we propose the defender to use the concept of Moving Target Defense [6] in selecting the IDS placement strategy, i.e., $\mathcal{D}$ selects one placement strategy out of the three mentioned above at time period t and switches to a different one at time period $t + 1$.

As shown in [7], simple randomization strategies like uniform random selection is shown to be insecure when used to switch among the different placement strategies. Thus, we now describe a game theoretic modeling that might help us in generating better strategies, with provable security guarantees. Similar to the model the authors describe in [7], in our case $\mathcal{D}$ has to deploy a cyber-system with IDS first, after which the attacker plays. Thus, we model our scenario as a Stackelberg Game.

Before we define the rewards for each player in our game, note that the strategy set for our defender has $\binom{n}{k}$ strategies. As the size of the enterprise system increases, the number of strategies explode combinatorially. The existing methods [7], [8] devised for finding the Stackelberg equilibrium of a game with arbitrary reward metrics becomes computationally expensive, failing to scale beyond a certain limit. To simplify the problem we assume a reward structure that can be factored into smaller components, which we define now, and later talk about solvers we adapt for our case.

We will denote an attack (which is a $\langle ip, cve\rangle$) as a and use A to denote the set of all attacks against our system. For each attack, the defender $\mathcal{D}$ can either choose to *cover* it by placing a IDS to detect it or keep it *uncovered*; and the total number of attacks the defender can cover is k. We now define four utility values for each attack, for being covered or not, and for the two players as follows.

$$\langle U^{\mathcal{D}}_{c,a}, U^{\mathcal{D}}_{u,a}, U^{\mathcal{A}}_{c,a}, U^{\mathcal{A}}_{u,a}\rangle$$

where $U^{\mathcal{D}}_{c,a}$ and $U^{\mathcal{D}}_{u,a}$ denotes the utility to the defender for covering and not covering an attack a respectively. Similarly, $U^{\mathcal{A}}_{c,a}$ and $U^{\mathcal{A}}_{u,a}$ represent the utility to the attacker when they attack and the attack a is detected (since a was covered) or not, respectively. The values for these symbols are obtained by leveraging the knowledge of security experts encoded in the Common Vulnerabilities Scoring System (CVSS) [9].

The CVSS metric provides two qualitative scores for each CVE–the Impact Score (IS) and the Exploitability Scores (ES). We can combine both of these to calculate a third score, known as the Base Score (BS). For each attack a in our model, we will use these scores subscripted with a ,i.e. IS_a, ES_a and BS_a. We will use CVSS-v2 metric for our reward definition.

$$\begin{aligned} U^{\mathcal{D}}_{c,a} &= +\frac{1}{c_a} \\ U^{\mathcal{D}}_{u,a} &= -1 * IS_a \\ U^{\mathcal{A}}_{c,a} &= -1 * ES_a \\ U^{\mathcal{A}}_{u,a} &= +1 * BS_a \end{aligned}$$

where $c_a (> 0)$ is the cost of deploying a contingency plan for containing the effect of attack a, if at all. We now provide some rational for modeling the rewards in this particular manner. The value of $U^{\mathcal{D}}_{c,a}$ is positive since it detected an attack, but is inversely proportional to the cost. Since the value of the base score is $\in [0, 10]$, one might want to set *stronger lower bounds* on the score c_a (like $c_a \geq 0.1$). Otherwise, the defender's reward can shoot up well beyond 10. When the defender does not place an IDS for detecting an attack, i.e., an attack is not covered, it incurs negative utility ($U^{\mathcal{D}}_{u,a}$) proportional to IS.

For the attacker $\mathcal{A}$, if it uses an attack a which the defender can detect (has it covered), denoted as $U^{\mathcal{A}}_{c,a}$, then it obtains a negative utility proportional to the time and cost it had to

Fig. 1. The defender, who has limited mobile robot resources, deploys these for surveying a large number of areas using a randomized strategy. The attacker is kept guessing as to where they should attack so that they are not detected.

invest in doing it, which is (somewhat) measured by ES. Also, since $\mathcal{A}$ gains nothing by doing this attack, no gains are added to it. Lastly, when the attacker uses an attack for which the defender has not placed an IDS, we give a positive utility that (conceptually) adds the IS and subtracts the cost (ES). Since BS already does this in a way so as the impact and cost is scaled properly, we use it directly.

B. Risk Assessment Modeling

To obtain the risk or probability value with which a node in our system will be compromised, we use an attack graph model. Each edge coming out of a node in the graphs, represents an attack step. Each such attack step has a value $\in [0, 1]$ associated with it, which represents the probability of successfully exploiting the vulnerability present in that node. We calculate this value using the Base Score (BS) obtained from CVSS scores mentioned above.

C. Modeling Countermeasure Selection

As stated above, in the context of this project, we define the countermeasure to denote a code (or config) fix that the $\mathcal{D}$ can do to remove a particular attack a from the system. The question then is, given developer effort to stop one attack, which $a \in A$ should we fix, which translates to which countermeasure should we deploy.

D. Softwares used

Python 3.5 (also Python 2.7 for Gurobi), Flask, MongoDB, NVD-CVE ID database, Data files from attack graph modules(XML), NetworkX libraries, Github.

E. Security Model (optional)

We consider a strong threat model for the attacker. In this, we assume they have access to all the available Common Vulnerabilities and Exploits (CVEs) present in our system. For the placement of detection mechanism module, we further assume that the adversary, who (by default) has the power of reconnaissance, will infer the defenders policy as to how detection mechanisms are placed in the Cloud Network and acts rationally using this knowledge.

III. SOLUTION METHODS

In this section, we discuss solution methods to fulfill the primary objectives of our project, mitigate the added risks we had foreseen in our initial project proposal and showcase the website we develop to showcase our results (see Fig. 1).

A. Obtaining the Equilibrium Strategy for IDS Placement

Let T denote the set of k *tokens* the defender can allocate to *cover* k of the n attacks that it wants to detect. Now, let the variables p_a represent the probability with which an attack a is covered by one of the k tokens and $p_{a,t}$ represent the

probability with which a particular attack a is covered by a particular token $t \in T$. Having defined the probabilities p_a, we can define defender's expected utility for deploying an IDS to detect a particular area a as $U^{\mathcal{D}}_{c,a} * p_a + U^{\mathcal{D}}_{c,a} * (1 - p_a)$. Similarly, we can define the attacker's expected utility for using a particular attack a as $U^{\mathcal{A}}_{c,a} * p_a + U^{\mathcal{A}}_{c,a} * (1 - p_a)$. We will now modify the optimization problem defined in [10] so that we maximize the defender's and attacker's utility given that an attacker attacks the area a^*.

$$
\begin{aligned}
\max \quad & U^{\mathcal{D}}_{c,a^*} p_{a^*} + U^{\mathcal{D}}_{c,a^*}(1 - p_{a^*}) && (1)\\
s.t. \quad & p_a \in (0,1) \ \ \forall\, a \in A \\
& p_{t,a} \in (0,1) \ \ \forall\, a \in A, t \in T \\
& \sum_{a \in A} p_{t,a} = 1 \ \ \forall\, t \in T \\
& \sum_{t \in T} p_{t,a} = p_a \ \ \forall\, a \in A \\
& U^{\mathcal{A}}_{c,a} p_a + U^{\mathcal{A}}_{c,a}(1 - p_a) \leq U^{\mathcal{A}}_{c,a^*} p_{a^*} + U^{\mathcal{A}}_{c,a^*}(1 - p_{a^*})
\end{aligned}
$$

Before we dive into what the constraints mean, notice that this is a Linear Program (LP) and thus can be solved in polynomial time. The first two sets of constraints ensure that the optimization variables p_a and $p_{t,a}$ are valid probabilities. The third set of constraints ensures that every token is fully utilized in covering the different attacks in A. The equality of this constraint is possible in our case since (1) all our tokes are homogeneous, i.e. any token $t \in T$ can be used to cover any attack $a \in A$ and (2) the number of tokens k $(= |T|)$ is less than the number of attacks n $(= |A|)$. Thus, we prune away solutions that do not fully utilize all the resources. The fourth set of constraints ensure that the probabilities of allocating various tokens to cover an attack a add up to the probability that a is covered. The final set of constraints ensure that the attacker selecting a^* maximizes their utility. Lastly, note that given the values of $p_{t,a}$ one can easily obtain p_a using the fourth set of constraints.

To obtain the (globally) optimal strategy for the defender, we can iterate over all the possible choices areas the attacker can attack and pick the values for which utility of the defender is maximized. This is illustrated in Algorithm 1.

Result: $\hat{p}_{r,a}$ for optimal defender utility
max_val $\leftarrow -\infty$;
$\hat{p}_{r,a} \leftarrow \vec{0}$;
while $a^* \in A$ **do**
  obj_val, $p_a, p_{r,a} \leftarrow$ solve Optimization (1);
  if *obj_val* $>$ *max_val* **then**
    max_val $\leftarrow$ obj_val;
    $\hat{p}_{r,a} \leftarrow p_{r,a}$;
  end
  return $\hat{p}_{r,a}$
end

Algorithm 1: Obtaining globally optimal mixed strategy for the defender.

Although we have a way to obtain the values $p_{t,a}$ (which can be stored in a $k \times n$ 2D matrix P_{kn}), there are no guarantees that we will be able to *implement* this matrix, i.e. we can decompose the $k \times n$ matrix to obtain $\binom{k}{n}$ probability values the corresponds to a mixed strategy for the defender. The Birkhoff Von-Neumann theorem [11] ensures that when certain constraints hold, we can obtain unique probability coefficients for the resource to area allocation vectors, which correspond to the mixed strategy of the defender from the probability matrix [12]. For example, consider the matrix when we have two resources ($k = 2$) three areas ($n = 3$),

$$
P_{23} = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0 & 0.3 & 0.7 \end{bmatrix}
$$

The third and fourth equalities in the optimization problem in 1 ensure that the constraint structure imposed on P_{kn} is a *bihierarchy*, which is sufficient condition for any P_{kn} to be *implementable* [12]. The authors in [10] state a general statement for the Birkhoff Von-Neumann theorem but lack the insight stated in [12]. We use the algorithms stated in [12] to obtain the mixed strategy for the defender. For the matrix P_{23} defined above, the implementable strategies are,

$$
0.1 : \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad 0.2 : \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$
$$
0.2 : \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad 0.5 : \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

In our case, since the tokens are homogeneous, i.e., any token can cover any attack, we can simplify these probabilities to the following mixed strategy for the defender,

$$
0.3 : \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \quad 0.2 : \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad 0.5 : \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
$$

where the first matrix is obtained by adding up the first two decomposed matrices mentioned before. The computed strategy states that the the defender can deploy two IDS at a time–to detect (a_2, a_3) with probability 0.3, (a_1, a_2) with probability 0.2 and (a_1, a_3) with probability 0.5.

B. Risk Assessment

The attack graph information for our system was obtained in `xml` format using MULVAL. We use this information to extract information about (1) leaf nodes that have `vulExists`, i.e., the leaf nodes in the attack graph of our system that contains a exploitable vulnerability, (2) the Host Access Control List (HACL) entries that specify the allowed traffic flow, and (3) AND-OR Rule nodes that define the possible attack paths which we use to build an AND-OR tree using NetworkX libraries. For Leaf nodes with `vulExists` we use CVE-ID to fetch CVSS base score BS from the NVD database and normalize it by ten, which is the upper bound of BS. In contrast, we consider a default value $\alpha (\in [0, 1])$ for exploitation probability of a node for which `vulExists` doesn't exist, which the user needs to provide as input.

Notice that when $\alpha = 1$, it means, in the context of an AND-OR graph, that even though a node does not have a `vulExists` associated with it, it can be exploited with absolute certainty, i.e. probability 1. This is an overly pessimistic view. On the other end, if we assume $\alpha = 0$, then we are making an overly optimistic claim that there

Attack	a_1 =192.168.0.9, CVE-2008-5161	a_2 =192.168.0.7, CVE-2008-5161	a_3 =192.168.0.6, CVE-2016-0128	a_4 =192.168.0.6, CVE-2015-1635	a_5 =192.168.0.6, CVE-2011-0657
$U^{\mathcal{D}}_{c,a}$	0.0	0.0	0.0	0.0	0.0
$U^{\mathcal{D}}_{u,a}$	-2.9	-2.9	-4.9	-10.0	-6.4
$U^{\mathcal{A}}_{c,a}$	-4.9	-4.9	-8.6	-10	-10
$U^{\mathcal{A}}_{u,a}$	2.6	2.6	5.8	10.0	7.5

TABLE I
UTILITY AND COST METRICS FOR EACH ATTACK.

can exist no vulnerabilities in that node, i.e. it cannot be exploited. Thus, a user needs to factor in the knowledge of unknown unknowns (attacks that might exist but have not been classified as CVEs yet) in deciding a good value of α. We find the conditional and cumulative probability of each node (including the goal node that the attacker wants to reach) using a recursive Depth-first search (DFS) taking into consideration the context of AND-OR graphs.

C. Countermeasure selection– Which vulnerability to fix?

As per our modified notion of countermeasure selection, we recalculate the cumulative probability of compromising root node (using the method above) by removing one vulnerable (or by fixing vulnerability associated with attack a) node at a time. The result of recalculation is used to determine δ_a, which is the change in cumulative probability of compromising the root node before and after the removal of that particular vulnerable node. Finally, we output the vulnerable node which, when removed, gives highest $delta$. The return of investment (ROI), in our case, is defined is defined as the decrease in cumulative probability of the goal state when a vulnerability is removed, i.e., $\max_a \delta_a$. Thus the countermeasure selection module chooses the node that results in the highest ROI when removed.

D. Website Development

The website code is developed using Python Flask. The system layout is as shown in the Figure 3. The system uses the attackgraph (in `xml` format (which is the attack graph information extracted from MulVal), α (default value of probability of leaf nodes, except vulExists) and the defenders IDS placement budget (k) as inputs.

The goal state probability module and countermeasure selection module uses the attack graph and α as input and calls the MongoDB database for CVSS scores and NetworkX library for explicit graph generation, that is later traversed. Whereas, the strategic placement of IDS module using the attack graph and k as input and calls scraps NVD webAPI to generate the game rewards. It finally used the Gurobi to efficiently solve the optimization problem.

We get the goal state cumulative probability and the most vulnerable node is selected and a CVE-ID is provided which when fixed will give highest ROI as output of the above modules. A jsonoutput.json file is generated which gives the details about the attack paths, conditional probability. We also get a strategy file (in txt format) as output which gives the defenders optimal strategy matrix.

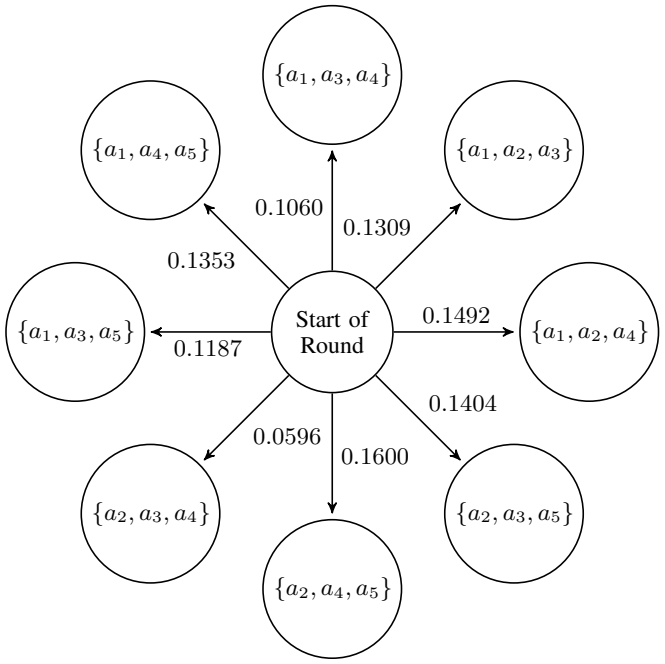

Fig. 2. Optimal mixed strategy of the defender for our scenario. The probability values for picking up one of the eight IDS placements at the start of each round are written on the edges.

IV. EXPERIMENTS

A. Strategic Placement of IDS

We use the network scenario (and the files) provided to come up with the best IDS placement strategy for that instance. Note that our solution methods is still general and can be used for any network scenario.

The attacks possible on our network denoted as two-tuples alongwith the four different types of utility values is shown in Table I. These were parsed from the `AttackGraph.xml` and the CVSSv2 scores were scraped from the webpage `https://nvd.nist.gov/vuln/detail/<cve_id>`. The cost of deploying a countermeasure for an attack is assumed to be very large ($\forall\ a \in A\ c_a = +\infty$). This means that the defender will not be able to deploy a countermeasure instantly even though it might detect the attack. This makes the values of $U^{\mathcal{D}}_{c,a} = 0\ \forall\ a \in A$.

We consider the defender can put $(k =)3$ IDS to detect three out of the five attacks possible against our system at any instance of time. The optimal switching strategy of $\mathcal{D}$ for this scenario shown in figure 2.

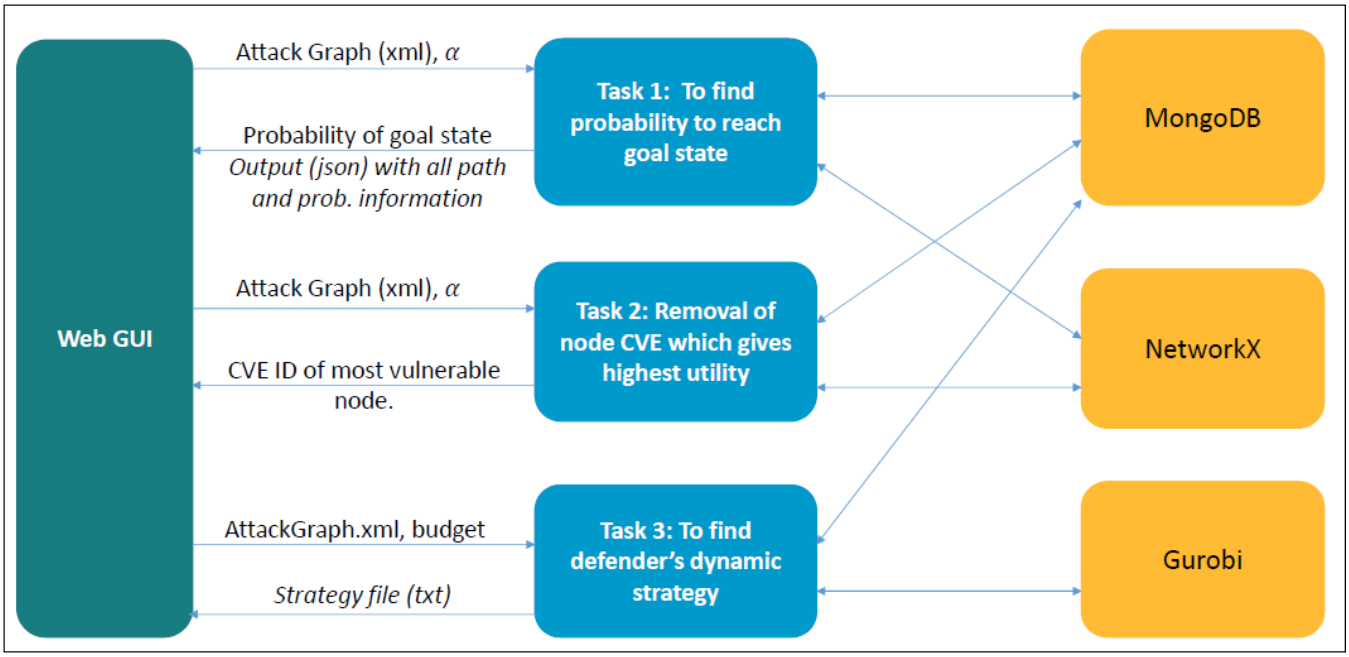

Fig. 3. The Website follows the above dataflow.

B. Risk Assessment

In this part, we highlight the probability values of the nodes that have a `vulExists`. These can be easily calculated by normalizing with the base score.

$$\begin{aligned}&\langle 30, CVE-2008-5161, 0.49\rangle,\\ &\langle 34, CVE-2008-5161, 0.49\rangle,\\ &\langle 40, CVE-2016-0128, 0.16\rangle,\\ &\langle 52, CVE-2015-1635, 1.0\rangle,\\ &\langle 64, CVE-2011-0657, 0.5\rangle\end{aligned}$$

Example conditional probability data:

$$\begin{aligned}&\langle 'Property' :' VulExists','Type' :' LEAF',\\ &'Probability' : 0.75,' Path' :' 1, 53, 64'\rangle\\ &\langle 'Property' :' VulExists','Type' :' LEAF',\\ &'Probability' : 0.26,' Path' :' 1, 41, 42, 47, 17, 18, 30'\rangle\\ &\langle 'Property' :' NotaVulExists','Type' :' LEAF',\\ &'Probability' : 1.0,' Path' :' 1, 53, 54, 59, 17, 18, 19, 26, 27'\rangle\end{aligned}$$

C. Which Vulnerability to Fix?

The vulnerability that we should fix for our sample attack graph is `CVE-2015-1635`. We noticed that this vulnerability had $IS = BS = ES = 10$ and even after manual investigation, was unquestionably the vulnerability that maximized the δ, defined earlier.

V. LOGISTICS

A. Project Task Allocation

The high level task allocation is described in Table II.

Mrunal Bonde	33.00%	Development of Rest API for all the modules.
Avinash Patil	33.00%	Attack Graph parsing, analysis and countermeasure selection.
Sailik Sengupta	34.00%	CVE parsing, Game Theoretic utility generation, Optimization solver for mixed policy calculation.

TABLE II
HIGH LEVEL TASK DESCRIPTION AND TIME ALLOCATION.

B. Deliverables

We have delivered the following items:

- Game theoretic modeling for placement of IDS. Proposed an Linear Programming problem that isn't affected by the combinatorial explosion of defender's strategies. We cited research that shows the marginal strategies we obtain can always be implemented to find the optimal defender strategy.
- Given an attack graph, we performed various traversal in it to figure out the cumulative and conditional probabilities of a node being compromised.
- We posed the question of which vulnerability should a defender fix given an attack graph and proposed a brute force solution method, that can help us obtain this.
- We developed a web service that given an attack graph of a system in `xml-` format and the value of α can find the best placement strategy, probability that the goal node is compromised, and which vulnerability should be removed.

VI. RISK MANAGEMENT OF THE PROJECT

Problem. The number of placement strategies mathematically became exponential which made the problem of finding a mixed strategy over this exponential space computationally hard.

Solution. Instead of finding the Stackelberg equilibrium of a game with a general reward structure, we break down the rewards for our players in a careful manner. This makes our problem tractable and helps us to leverage techniques for calculating marginal strategies and adapt one such solver to our specific problem.

ACKNOWLEDGMENT

We would like to thank Dr. Dijiang Huang and Ankur Chowdhary for their relentless help throughout our project. We cherished the though provoking discussions that helped up gain a better understanding of the requirement, insights into techniques, and clarity of the goal for our project.

REFERENCES

[1] Craig H Rowland. Intrusion detection system, June 11 2002. US Patent 6,405,318.

[2] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[3] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[4] Stephen E Smaha. Haystack: An intrusion detection system. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44. IEEE, 1988.

[5] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[6] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[7] Sailik Sengupta, Satya Gautam Vadlamudi, Subbarao Kambhampati, Adam Doupé, Ziming Zhao, Marthony Taguinod, and Gail-Joon Ahn. A game theoretic approach to strategy generation for moving target defense in web applications. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 178–186. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[8] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. AAMAS, 2008.

[9] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6), 2006.

[10] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *AAAI*, 2010.

[11] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[12] Eric Budish, Yeon-Koo Che, Fuhito Kojima, and Paul Milgrom. Designing random allocation mechanisms: Theory and applications. *American Economic Review*, 103(2):585–623, 2013.