Moving Target Defense under Uncertainty for Web Applications

Paper #XXX

ABSTRACT

Moving target defense (MTD) has emerged as a technique that can be used in various applications to reduce the threat of attackers by taking away their ability to perform reconnaissance. However, a majority of the existing research in the field assumes unrealistic access to information about attacker motivations when developing MTD strategies. Many of the existing approaches also assume complete knowledge regarding the vulnerabilities of a particular application and how each of these vulnerabilities can be exploited by an attacker. In this work, we aim to create algorithms that generate effective moving target defense strategies that do not rely on prior knowledge. Our work assumes the only information we get is via interaction with the attacker in a repeated game setting. Since the amount of information that can be obtained through interactions may vary from application to application, we provide different algorithms that account for the different levels of information to identify optimal switching strategies. We then evaluate our algorithms using Common Vulnerabilities and Exploits mined from the National Vulnerability Database to show that our algorithms significantly outperform the state of the art.

KEYWORDS

Moving Target Defense; Online Learning; Information Uncertainty

1 INTRODUCTION

Add Later

Lets add something quick here so we get some structure

1.1 Our Contribution

Add Later.

1.2 Related Work

Add Later.

2 MODEL AND PRELIMINARIES

We consider a repeated game played with two players - a *defender* (denoted by Θ) and τ *attackers* (denoted by $\Psi = \{\Psi_1, \Psi_2, \dots, \Psi_\tau\}$). At each round (or timestep) of this game, one of these attackers tries to exploit the application set up by the defender. We assume there exists a probability distribution $\mathcal{P}$ over the set attackers which decides which attacker attacks at a given round. The defender has a set $C = \{c_1, c_2, \dots c_n\}$ of n *configurations* that it can deploy. Each configuration has a set of *vulnerabilities* $\mathcal{V}_c = \{e_1, e_2, \dots, e_{|\mathcal{V}_c|}\}$ that can be exploited. Since each vulnerability has an exploit associated with it, we use thease two terms interchangably. We define the set of all vulnerabilities by $\mathcal{V} = \bigcup_{c \in C} \mathcal{V}_c$. This vulnerability set for each configuration may not be known before hand but we assume that no configuration is perfect i.e. every configuration has some vulnerability.

At the t'th round, the defender chooses a configuration to deploy (denoted by $v_t \in C$) and the attacker $f(t)$ (where f is a function that maps a round to the attacker at that round) chooses a vulnerability to exploit (denoted by a_t); the attacker can also choose not to exploit any vulnerability during a turn. Furthermore, we assume that there is a cost incurred when switching between configurations which is known a priori and remains constant throughout. This is not an unrealistic assumption to make since the switching cost can be estimated during the testing phase before the deployment of the application. The cost incurred by switching from configuration c to configuration c' in the t'th round is denoted by $s(c, c') \in (0, 1]$. The game is played for T rounds.

For each vulnerability $e \in N$, for each configuration $c \in C$, for each attacker type $\psi \in \Psi$, we define a reward to the defender at the t'th round denoted by $r_t(\psi, e, c)$. The reward $r_t(\psi, e, c) \in [-1, 0)$, if an attacker successfully exploits a vulnerability that the defender's configuration has i.e. $e \in \mathcal{V}_c$ and is 0 otherwise. Note that the different attacker types will result in the defender obtaining different utilities because of the varying attacker capabilities e.g., some attackers may not have the expertise to carry out certain attacks [1]. For a specific attacker, we assume these rewards are constant throughout.

We make no assumptions about the attacker strategy since the attacker may not be fully rational. We also do not make assumptions about the attacker rewards, since these values will be hard to observe and even harder to find out beforehand. We however assume that the attacker has access to information from the previous rounds and is capable of reconnaissance.

Our goal in this paper is to maximise the total utility the defender receives, i.e.

$$\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t, v_t) - s_t(v_{t-1}, v_t)$$

2.1 Regret

When we have infinitely many attacker types who are adversarially chosen, it is easy to see that this problem is very similar to the adversarial multi-armed bandit problem. This similarity suggest that algorithms like FPL+GR[cite] or Exp3 [cite] will output strategies with good external regret guarantees. More specifically, the algorithm minimizes the difference between the total welfare it obtains and the welfare of the best fixed strategy in hindsight. However, since the algorithms themselves do not consider switching cost, to use these algorithms, we must include the switching cost in the

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 2020, Auckland, New Zealand

https://doi.org/doi

loss function. This gives us the following external regret guarantee:

$$\max_{v \in C} \left(\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t, v) - s_t(v_{t-1}, v) \right) - \left(\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t, v_t) - s_t(v_{t-1}, v_t) \right) \leq O(\sqrt{T})$$

However, as one can see and as [cite Arora] argue, this notion of regret does not have any meaning when there are non-zero switching costs. [cite Arora] consider an attacker strategy which depends on all the previous configurations of the defender and define an alternate notion of regret known as policy regret. This regret takes into consideration how the attacker would have behaved if you had played a pure strategy instead and then compares the performance of the bandit algorithm to the performance of the best pure strategy in hindsight. More specifically, let a_t be the attacker strategy at time t which takes as input, the actions $v_1, v_2, \ldots, v_{t-1}$. Then, the policy regret of an algorithm would evaluate to

$$\max_{v \in C} \left(\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t(v, v, \ldots, v), v) \right) - \left(\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t(v_1, v_2, \ldots, v_{t-1}), v_t) - s_t(v_{t-1}, v_t) \right)$$

[cite Arora] show that the lower bound for policy regret is $\Omega(T)$ in the presence of an adaptive adversary.

However, in a moving target defense context, the baseline used in policy regret does not make much sense either since it is a stationary baseline. This allows the attacker to perform reconnaissance and learn the best possible strategy to inflict damage on the defender. So this baseline can have an arbitrarily bad performance. This means that even though, in realistic settings any pure strategy will perform poorly, in theory no algorithm can guarantee doing better than this. From this, we conclude that even though the bandit approach may result in good welfare, the regret of these algorithms can be arbitrarily bad or can have no meaning.

3 HIGH INFORMATION

In this section, we consider a scenario where the defender fully knows the vulnerabilities of every configuration that they deploy, the number of attackers and the reward that the defender would get when an attacker attacks a vulnerability. At every time step, we observe what vulnerability was attacked, which attacker attacked it and the reward obtained. The only two things the defender does not know is the strategy of the attacker and the probability distribution across attacker types. Since we do not know the probability distribution among the attackers, we assume the attackers are chosen adversarially. This setting corresponds to an older application where enough research has been done to determine the vulnerabilities of each configuration that is being deployed and past data about attacks has helped determine the different kinds of attackers who are trying to exploit vulnerabilities in this software and the damage they can do.

When the rewards $r(\psi, e, c)$ are constant for each attacker, vulnerability and configuration tuple, then we can model this as a repeated game against a *master* attacker who chooses both an attacker and a vulnerability. **Need to figure this out**.

4 LOW INFORMATION

In this section, we only assume that we can observe rewards at the end of each time step. We assume no knowledge about the set of vulnerabilities or the attacker which attacked at a given time step. This setting corresponds to a new application which uses much newer software whose vulnerabilities are unknown and the different types of attackers trying to exploit the software are unknown as well.

If all the attacks and the rewards are chosen adversarially, this resembles an adaptive multi-armed bandit problem with switching costs. Even when the switching costs are 0, the lower bound on policy regret can be shown to be $\Omega(T)$. However, we propose an algorithm that does well for most realistic attacker types. This algorithm is an extension of the Exp3 algorithm [cite Auer 2002] and the main difference is that we add some padding to reduce the number of times the algorithm switches so as to minimize the number of switches. The detailed algorithm is presented in Algorithm 1.

Algorithm 1: Padded Exp3

```
$t = 1$
$w_c(t) = 1 \forall c \in C$
for $t = 1, 2, \ldots$ do
    $\eta \sim U[0, 1]$
    if $\eta < \eta'$ then
        $c_t = c_{t-1}$
    end
    else
        $p_c(t) = (1 - \gamma) \frac{w_c(t)}{\sum_{c' \in C} w_c(t)} + \frac{\gamma}{|C|}$
        Choose $c_t$ according to the probabilities $\{p_c(t)\}_{c \in C}$
    end
    Receive reward, $r_t$ and switching cost $s_t$
    $w_c(t + 1) = w_c(t) exp\{\frac{\gamma r_t}{p_c(t)|C|}\}$;
end
```

This algorithm retains the essence of the Exp3 algorithm in the sense that it computes a probability distribution over all the configurations using some weights and chooses an action using this probability distribution. But the key difference is that this step is only done very few times to reduce the number of switches that the algorithm makes. In other words, at time t, the probability of choosing c_{t-1} is padded to ensure that the number of switches done by the algorithm is low.

5 EXPERIMENTS

Add Later

5.1 Identifying Critical Vulnerabilities

6 CONCLUSION

Add Later

REFERENCES

[1] 2017. A game theoretic approach to strategy generation for moving target defense in web applications. In *16th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2017*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 178–186.