# 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 possible exploits on a particular application and how each of these exploits would affect the defender. 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 $\Theta$) and $\tau$ attackers (denoted by $\Psi = \{\Psi_1, \Psi_2, \ldots, \Psi_\tau\}$). 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 $\mathcal{P}$ across these attacker types that determines which attacker attacks at a given time step but this distribution may not always be known to the defender. The defender has a set $C = \{c_1, c_2, \ldots c_n\}$ of $n$ configurations that it can deploy. Each configuration has a set of vulnerabilities $\mathcal{V}_c = \{e_1, e_2, \ldots, e_{|\mathcal{V}_c|}\}$ that can be exploited. We define the set of all vulnerabilities by $N = \bigcup_{c \in C} \mathcal{V}_c$. 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 of type $f(t)$ (where $f$ is a function that maps a round to the attacker type 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 may not be known a priori but remains constant throughout. 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 [? ]. It is important to note that we do not require the rewards to be constant throughout; they can be stochastic or even adversarial in nature.

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 utility, 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 that the defender receives, i.e.

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

## 3 REGRET LOWER BOUNDS

When we only have one attacker type and the rewards at each round are adversarial, it is easy to see that this problem is very similar to the multi-armed bandit problem. This similarity allows us to use algorithms like FPL+GR[cite] or Exp3 [cite] to obtain strategies with good regret guarantees. More specifically, the algorithm minimizes the difference between the total welfare 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 loss function. This gives

*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

us the following 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] argue, this notion of regret does not have any meaning when there are switching costs. They 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] 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. So for online learning for moving target defense, we introduce a new notion called *deception regret*. This combines the best of both regret notions. The baseline it compares the welfare with is the performance obtained when the defender plays the best pure strategy in hindsight assuming that the attacker has been *deceived* by the earlier switches of the defender. The deception regret of an algorithm is:

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

$$- \left(\sum_{t=1}^{T} r_t(\Psi_{f(t)}, a_t(v_1, v_2, \ldots, v_{t-1}), v_t) - s_t(v_{t-1}, v_t)\right) \tag{2}$$

Unfortunately, even when the rewards are constant, when there is only one attacker and this attacker is capable of exploiting some vulnerability in all the configurations, the deception regret is $\Omega(T)$.

PROPOSITION 3.1. *When there is only one attacker, then the deception regret given by Equation 2 is lower bounded by* $-\min_{e \in \mathcal{V}} \max_{v \in C} r(\psi_1, e, v) \frac{T}{|C|}$

PROOF. □

What Proposition 3.1 implies is that it will be impossible to come up with an algorithm which achieves sub-linear regret. Note that any strategy achieves $O(T)$ regret since the maximum possible regret is $2T$. Therefore our main focus is on creating scalable heuristics that have high welfare when used on real life data with realistic attacker strategies.

## 4 CASE 1

In this section, we consider a situation where we can observe the vulnerability being exploited as well as the attacker type trying to exploit the vulnerability. While this is similar to the model presented in [? ], our model does not observe the utility that the attacker receives. Note that even if we do not know the probability distribution across attacker types, it would be fairly easy to estimate it since we know which attacker type attacks at every round. To come up with a solution for this case, we use an extension of the multi armed bandits for our setting. This algorithm is described in Algorithm 3. - This part is unclear. Is the MAB extension used to model the problem or is it solution to the problem ? Either way we should present the MAB and the extension for it and then present the algorithm. Is this new algorithm or is it an existing algorithm that we are presenting as solution here ?

**Algorithm 1:** GR-Reward

```
$K_r(a_t, v_t, \Psi_{f(t)}) = M$
for $k = 1, 2, \ldots, M$ do
    Follow the FPL step in the previous algorithm to
      produce $\tilde{v}$ as a simulation of $v_t$
    if $a_t \in \mathcal{V}_{\tilde{v}}$ then
        $K_r(a_t, v_t, \Psi_{f(t)}) = \min(K_r(a_t, v_t, \Psi_{f(t)}), k)$
end
$K_r(a_t, v_t, \Psi_{f(t)}) = \frac{K_r(a_t, v_t, \Psi_{f(t)})}{\mathcal{P}_{\Psi_{f(t)}}}$
```

**Algorithm 2:** GR-Switch

```
$K_s(v_{t-1}, v_t) = M$
for $k = 1, 2, \ldots, M$ do
    Follow the FPL step in the previous algorithm to
      produce $\tilde{v}$ as a simulation of $v_t$
    if $\tilde{v} = v_t$ then
        $K_s(v_{t-1}, v_t) = \min(K_s(v_{t-1}, v_t), k)$
end
```

The FPL-MTD algorithm retains the essence of the FPL-UE algorithm [? ] but it has a few modifications to adapt it to our setting.

We maintain a reward estimate $\hat{r}_{e,\psi}$ for each vulnerability $e$ and for each attacker type $\psi$. We also maintain a switching cost estimate $\hat{s}_{c,c'}$ for each pair of configurations.

We then use two steps to decide which configuration (denoted by $v_t$) to deploy. First we estimate the worst case reward of each configuration and denote it by $u_c$

$$u_c = \sum_{\psi \in \Psi} \min_{e \in \mathcal{V}_c} \mathcal{P}_{\psi}(\hat{r}_{e,\psi} - z_{e,\psi})$$

This worst case value is a weighted average of the worst case estimate of all the attacker types where the weights are the probabilities of occurrence of each type.

**Algorithm 3:** FPL-MTD

```
$\hat{r}_{e,\psi} = 0 \quad \forall e \in N, \psi \in \Psi$
$\hat{s}_{c,c'} = 0 \quad \forall c, c' \in C$
for $t = 1, 2, \ldots, T$ do
    Sample $flag \in \{0,1\}$ such that $flag = 0$ with prob $\gamma$
    if $flag == 0$ then
        Let $v_t$ be a randomly sampled configuration
    else
        Draw $z_{e,\psi} \leftarrow exp(\eta)$ independently for $e \in N$ and $\psi \in \Psi$
        Let $u_c = \sum_{\psi \in \Psi} \min_{e \in \mathcal{V}_c} \mathcal{P}_\psi(\hat{r}_{e,\psi} - z_{e,\psi}) \quad \forall c \in C$
        Let $v_t = \max_{c \in C} e^{-g(c)} u_c - \hat{s}_{v_{t-1},c}$
    end
    Adversary of type $\Psi_{f(t)}$ plays $a_t$, you play $v_t$ and get a reward $r_t(a_t, v_t)$ and incur a switching cost $s(v_{t-1}, v_t)$
    Run GR-Reward to estimate $\frac{1}{p(a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)})}$ as $K_r(a_t, v_t \Psi_{f(t)})$
    Run GR-Switch to estimate $\frac{1}{p(v_t/v_{t-1})}$ as $K_s(v_{t-1}, v_t)$
    Update $\hat{r}_{a_t,\Psi_{f(t)}} = \hat{r}_{a_t,\Psi_{f(t)}} + K_r(a_t, v_t, \Psi_{f(t)}) r_t(a_t, \Psi_{f(t)})$
    Update $\hat{s}_{v_{t-1},v_t} = \hat{s}_{v_t,v_{t-1}} = \hat{s}_{v_{t-1},v_t} + K_s(v_{t-1}, v_t) s(v_{t-1}, v_t)$
end
```

Secondly, we use these worst case rewards to compute a strategy that maximises the worst case reward subject to a possible switching cost. We also take into consideration possible reconnaissance and penalise repeating strategies accordingly. This gives us the following assignment

$$v_t = \max_{c \in C} e^{-g(c)} u_c - \hat{s}_{v_{t-1},c}$$

Here $g(c)$ is the dampening factor which penalises repeating configurations

$$g(c) = \begin{cases} \alpha & \text{if } c = v_{t-1} \\ 0 & otherwise \end{cases}$$

The next step is to update the estimates of the algorithm. We update the two estimates as follows

$$\hat{r}_{a_t,\Psi_{f(t)}} = \hat{r}_{a_t,\Psi_{f(t)}} + r_t(a_t, \Psi_{f(t)}) \frac{\mathbb{I}\{a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)}\}}{p(a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)})}$$

$$\hat{s}_{v_{t-1},v_t} = \hat{s}_{v_{t-1},v_t} + s_t(v_{t-1}, v_t) \frac{\mathbb{I}\{v_t/v_{t-1}\}}{p(v_t/v_{t-1})}$$

where $\mathbb{I}$ is the identity function and $p$ refers to the probability. The term $a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)}$ refers to the event where the attacker $\Psi_{f(t)}$ exploits a vulnerability of the chosen configuration and $v_t/v_{t-1}$ refers to the event where configuration $v_t$ is chosen immediately after configuration $v_{t-1}$.

Note that the terms $r_t(a_t, \Psi_{f(t)}) \frac{\mathbb{I}\{a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)}\}}{p(a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)})}$ and $s_t(v_{t-1}, v_t) \frac{\mathbb{I}\{v_t/v_{t-1}\}}{p(v_t/v_{t-1})}$ are unbiased estimators of $r_t(a_t, \Psi_{f(t)})$ and $s_t(v_{t-1}, v_t)$ respectively. This method is used extensively in online learning. Like prior online learning literature (see [? ]), we use geometric resampling to compute an estimate of the inverse of the above probabilities since they cannot be computed efficiently.

The intuition for geometric resampling is as follows: the number of trials required for $v_t$ to be the chosen configuration given $v_{t-1}$ is a geometric distribution with mean $\frac{1}{p(v_t/v_{t-1})}$. So we simulate the configuration process till we choose $v_t$ and the number of trials would be an unbiased estimate of $\frac{1}{p(v_t/v_{t-1})}$. This can be seen in Algorithm 2.

Similar logic can be applied to the reward setting as well. However, the problem with the reward setting as that we cannot simulate multiple attacker types since we do not the rewards but we can advantage of the fact that attacker type selection is an independent occurrence and we know the probabilities of an attacker type occurring i.e.

$$p(a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)}) = p(a_t \in \mathcal{V}_{v_t}) p(\Psi_{f(t)})$$

We know $p(\Psi_{f(t)})$ and we use geometric resampling to estimate $p(a_t \in \mathcal{V}_{v_t})$. This is described in Algorithm 1.

## 5 CASE 2

In this section, we go one step further to assume that we know the number of types and the probability distribution across these types but we cannot observe which type carried out the attack. However, we know that the ranges of utility obtained by each attacker type has low intersection. This is motivated by the setting where attacker types are divided based on skill and the ability to cause harm. We formalise this intuition by defining a metric we call *degree of overlap* and then we propose algorithms for settings with low degree of overlap.

### 5.1 Degree of Overlap

Even though we do not know the utilities before hand, we assume that we know a probability distribution from where defender utilities are drawn from. The distribution for each type need not be the same and we represent this distribution by $\mathcal{D}_{\Psi_t}$ for type $\Psi_t$. From this, the degree of overlap of a security game, $DoO$ is defined by

$$DoO = \frac{\sum_{\Psi_i, \Psi_j \in \Psi} AreaOfIntersection(D_{\Psi_i}, D_{\Psi_j})}{{}^{\tau}C_2}$$

where $\tau$ is the number of types and $AreaOfIntersection$ is the area of intersection of the two probability mass functions when you plot it on a graph. Mathematically, if $f_{D_{\Psi_i}}$ and $f_{D_{\Psi_j}}$ are two continuously distributed probability mass functions of distributions $D_{\Psi_i}$ and $D_{\Psi_j}$ respectively, then

$$AreaOfIntersection(D_{\Psi_i}, D_{\Psi_j}) = \int \min\{f_{D_{\Psi_i}}(x), f_{D_{\Psi_j}}(x)\} \, dx$$

The significance of this parameter is that a game which has a lower degree of overlap should intuitively be classified better by classification algorithm, and therefore should give better performance. A game with degree of overlap 0 should cluster perfectly and give you a performance similar to that of the partial information case (Section 4). This is the intuition we use to take our algorithm one step further to create FPL-MTD-CL (see Algorithm 4). FPL-MTD-CL is just like FPL-MTD (Algorithm 3) but it uses a clustering algorithm to find out the type of the attacker which attacked. We use the exact same policy selection and parameter update rule.

**Algorithm 4:** FPL-MTD-CL

```
$\hat{r}_{e,\psi} = 0 \quad \forall e \in N, \psi \in \Psi$
$\hat{s}_{c,c'} = 0 \quad \forall c, c' \in C$
for $t = 1, 2, \ldots, T$ do
    Sample $flag \in \{0, 1\}$ such that $flag = 0$ with prob $\gamma$
    if $flag == 0$ then
        Let $v_t$ be a randomly sampled configuration
    else
        Draw $z_{e,\psi} \leftarrow exp(\eta)$ independently for $e \in N$ and $\psi \in \Psi$
        Let $u_c = \sum_{\psi \in \Psi} \min_{e \in \mathcal{V}_c} \mathcal{P}_\psi(\hat{r}_{e,\psi} - z_{e,\psi}) \quad \forall c \in C$
        Let $v_t = \max_{c \in C} e^{-g(c)} u_c - \hat{s}_{v_{t-1},c}$
    end
    Adversary plays $a_t$, you play $v_t$ and get a reward $r_t(a_t, v_t)$ and incur a switching cost $s(v_{t-1}, v_t)$
    Run GR-Reward to estimate $\frac{1}{p(a_t \in \mathcal{V}_{v_t} \wedge \Psi_{f(t)})}$ as $K_r(a_t, v_t \Psi_{f(t)})$
    Use a clustering algorithm to classify the adversary into type $\Psi_{f(t)}$
    Run GR-Switch to estimate $\frac{1}{p(v_t / v_{t-1})}$ as $K_s(v_{t-1}, v_t)$
    Update $\hat{r}_{a_t,\Psi_{f(t)}} = \hat{r}_{a_t,\Psi_{f(t)}} + K_r(a_t, v_t, \Psi_{f(t)}) r_t(a_t, \Psi_{f(t)})$
    Update $\hat{s}_{v_{t-1},v_t} = \hat{s}_{v_t,v_{t-1}} = \hat{s}_{v_{t-1},v_t} + K_s(v_{t-1}, v_t) s(v_{t-1}, v_t)$
end
```

Note that a low degree of overlap is a very strong assumption and may not be true in a lot of cases. In these cases, we can use Algorithm 3 but assume there is only one attacker type.

## 6 CASE 3

In this section, we consider a scenario where we have minimal information. We assume that we only know about our configurations and we know nothing about their vulnerabilities or the various attacker types that may be looking to attack.

In order to develop an online policy for this case, we use a similar approach except that instead of maintaining a reward estimate for every exploit, we maintain a reward estimate for every configuration. The rest of the algorithm works very similarly.

## 7 EXPERIMENTS

**Add Later**

### 7.1 Identifying Critical Vulnerabilities

## 8 CONCLUSION

**Add Later**

**Algorithm 5:** FPL-MTD-Lite

```
$\hat{r}_c = 0 \quad \forall c \in C$
$\hat{s}_{c,c'} = 0 \quad \forall c, c' \in C$
for $t = 1, 2, \ldots, T$ do
    Sample $flag \in \{0, 1\}$ such that $flag = 0$ with prob $\gamma$
    if $flag == 0$ then
        Let $v_t$ be a randomly sampled configuration
    else
        Draw $z_c \leftarrow exp(\eta)$ independently for $c \in C$
        Let $v_t = \max_{c \in C} e^{-g(c)} \hat{r}_c - \hat{s}_{v_{t-1},c} - z_c$
    end
    Adversary plays $a_t$, you play $v_t$ and get a reward $r_t(v_t)$ and incur a switching cost $s(v_{t-1}, v_t)$
    Run GR-Switch to estimate $\frac{1}{p(v_t / v_{t-1})}$ as $K_s(v_{t-1}, v_t)$;
    Update $\hat{r}_{v_t} = \hat{r}_{v_t} + K_s(v_{t-1}, v_t) r_t(v_t)$
    Update $\hat{s}_{v_{t-1},v_t} = \hat{s}_{v_t,v_{t-1}} = \hat{s}_{v_{t-1},v_t} + K_s(v_{t-1}, v_t) s(v_{t-1}, v_t)$
end
```