electronics

MDPI

Article

Intelligent On-Off Web Defacement Attacks and Random Monitoring-Based Detection Algorithms

Youngho Cho

Department of Computer Engineering, Graduate School of National Defense Management, Korea National Defense University, Nonsan 33021, Korea; youngho@kndu.ac.kr

Received: 26 October 2019; Accepted: 11 November 2019; Published: 13 November 2019

Abstract: Recent cyberattacks armed with various ICT (information and communication technology) techniques are becoming advanced, sophisticated and intelligent. In security research field and practice, it is a common and reasonable assumption that attackers are intelligent enough to discover security vulnerabilities of security defense mechanisms and thus avoid the defense systems' detection and prevention activities. Web defacement attacks refer to a series of attacks that illegally modify web pages for malicious purposes, and are one of the serious ongoing cyber threats that occur globally. Detection methods against such attacks can be classified into either server-based approaches or client-based approaches, and there are pros and cons for each approach. From our extensive survey on existing client-based defense methods, we found a critical security vulnerability which can be exploited by intelligent attackers. In this paper, we report the security vulnerability in existing client-based detection methods with a fixed monitoring cycle and present novel intelligent on-off web defacement attacks exploiting such vulnerability. Next, we propose to use a random monitoring strategy as a promising countermeasure against such attacks, and design two random monitoring defense algorithms: (1) Uniform Random Monitoring Algorithm (URMA), and (2) Attack Damage-Based Random Monitoring Algorithm (ADRMA). In addition, we present extensive experiment results to validate our idea and show the detection performance of our random monitoring algorithms. According to our experiment results, our random monitoring detection algorithms can quickly detect various intelligent web defacement on-off attacks (AM1, AM2, and AM3), and thus do not allow huge attack damage in terms of the number of defaced slots when compared with an existing fixed periodic monitoring algorithm (FPMA).

Keywords: web defacement attack; on-off strategy; random monitoring algorithm; web security

1. Introduction

Web defacement attacks refer to a series of attacks that illegally modifies web pages in unauthorized manners for malicious purposes. According to recent statistics provided by ZONE-H [1], 500,000 websites over the world were defaced only in 2018, and around 100,000 defaced-websites were reported during the first quarter of 2019. Detail reports on major web defacement incidents can be found in [2].

Typical types of web defacement attacks vary from changing main images of websites to launching drive-by-download attacks that stealthily inject a malicious link into a web page through which malwares are automatically downloaded to web users' devices which accessed the defaced web pages [3,4]. Recently, the latter type is more often reported because the attacker can construct a large-scale botnet that consists of compromised personal computers, laptops, smartphones, appliances, and Internet of Things (IoT) devices. With the botnet, attackers can easily achieve their intended goals, such as launching distributed denial-of-service (DDoS) attacks to certain websites.

In general, web defacement attacks are performed as follows (see Figure 1). First, the attacker (A) maliciously modifies one or more web pages (or source codes) stored in the web server (WS) by

exploiting security vulnerabilities of the WS. For example, A injects a malicious link (downloader) to malwares stored in malicious server (M) which cooperates with A. Such malicious link is injected in a way that system administrators or normal users cannot easily identify its existence within the defaced web page. Next, when a web user (U) accesses the WS, U is automatically connected to the external malicious server M through the injected malicious link and then malwares are downloaded to U from M; these processes proceed such that U does not know that they are happening, and the number of U (victims) can be hundreds, thousands, or even more depending on the popularity of web services provided by the WS. Once the U is infected with downloaded malwares, U becomes a bot which is under the control of A (or a bot master). After that, A starts launching its actual intended secondary attacks such as extracting critical information from U or DDoS attacks by using the botnet that consists of many Us (bots) [4].

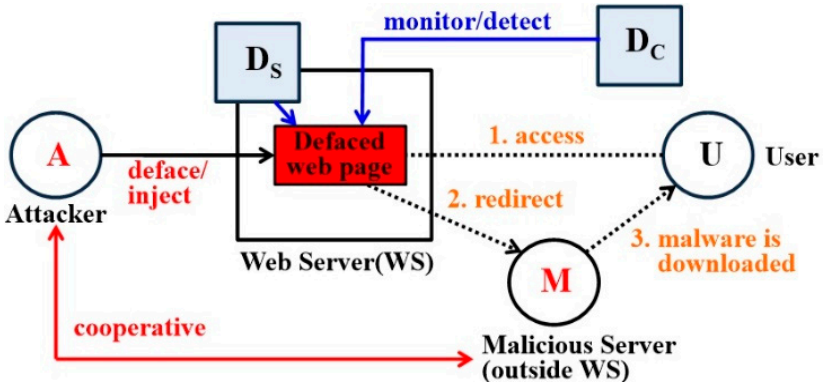


Figure 1. An illustration of web defacement attacks and existing detection systems; WS: web server; A: attacker; Ds: server-based detection system; Dc: client-based detection system; M: malicious server.

As shown in Figure 1, existing detection approaches against web defacement attacks can be classified into either server-based detection approach or client-based detection approach [5–21].

In the server-based detection approach [5,6], the detection system (D_S) is installed in the WS, and it regularly monitors web pages in the WS and checks if they are modified in unauthorized ways. Once the D_S detects modified web pages by attackers, the D_S raises an alarm and reports it to a server administrator or CERT (Computer Emergency Response Team) for further investigation and timely response. To check unauthorized modification of web pages in the WS, various file integrity monitoring methods [22–25] can be used. However, when the attacker successfully defaced web pages of the WS, the WS cannot be trusted because the attacker may have some or full control over the WS in that it is common for attackers (hackers) to try to obtain the root privilege of the WS, and then install backdoors in the WS after hacking applications of the WS. For example, the attacker can obtain operating system root privilege by launching various privilege escalation attacks [26,27]. Once the attacker gains root privilege of the WS, the attacker can disable security software such as the file integrity checker or local monitoring tool. Consequently, there is no guarantee that server-based detection methods work properly when web defacement attacks are successfully launched.

On the other hand, in the client-based detection approach [7–21], the client-based detection system (D_C) is located outside the server WS and monitors web pages in the WS remotely. D_C behaves as a common web user U; D_C periodically accesses the WS and collects web pages from the WS. After downloading web pages from the WS, D_C checks if they are defaced by using various detection techniques. Since D_C is located outside the WS, its detection process can be more trustful compared with the server-based approach. In addition, the client-based approach has some advantage over the server-based approach such that it can detect web defacement attacks performed in man-in-the-middle position between the web server WS and the client U.

Meanwhile, most existing researches on client-based detection mainly focused on either proposing new detection methods or improving attack detection accuracy [7–21]. Interestingly, to the best of

our knowledge based on our extensive survey, there are no studies that explain how frequently their monitoring and detection processes should be performed. Most existing client-based detection methods simply monitor web pages with a fixed periodic monitoring cycle (interval) or they do not even mention about the monitoring cycle. However, such fixed monitoring cycles can be seriously vulnerable to intelligent attackers because detection systems with a fixed monitoring cycle can be completely avoided by intelligent attackers. In this paper, we first justify why fixed periodic monitoring should not be used by introducing *intelligent on-off web defacement attacks* that can completely avoid client-based web defacement detection systems with a fixed monitoring cycle. Since we introduced intelligent on-off web defacement attacks newly in this paper, we cannot provide real industry incident cases and reports. The primary goal of this study is to let security researchers and engineers understand this potential cyber threat to the Internet, and thus let them motivate more research and develop effective security mechanisms to defend against such attacks in advance.

Our contributions in this paper are as the following:

- We introduce a new intelligent on-off web defacement attack model that can completely avoid existing client-based detection methods using fixed periodic monitoring.
- We propose to use a random monitoring defense strategy against intelligent on-off web defacement attacks as a promising countermeasure, and conduct a simple probabilistic analysis that shows how random monitoring defense strategy can be effective in detecting such attacks.
- We devise two random monitoring algorithms, the Uniform Random Monitoring Algorithm (URMA) and the Attack Damage-based Random Monitoring Algorithm (ADRMA), against intelligent on-off web defacement attacks and provide extensive experiment results that show their detection performance by comparing with a fixed periodic monitoring algorithm (FPMA).

The rest of this paper is organized as follows. In Section 2, we review existing client-based detection methods against web defacement attacks. In Section 3, we introduce a new intelligent on-off web defacement attack. In Section 4, we justify that the random monitoring strategy can effectively defend against the intelligent on-off attack strategy, and propose two random monitoring detection algorithms (URMA and ADRMA). In Section 5, we conduct extensive experiments to show the performance of our proposed algorithms by comparing an existing fixed periodic monitoring algorithm (FPMA). Finally, we conclude with future research directions in Section 6.

2. Related Works

In this section, we briefly introduce previous studies on client-based monitoring and detection methods against web defacement attacks. With the recent advancements and popularity of machine learning (ML) techniques, many studies using various ML techniques have been conducted in this area as follows.

Borgolte et al. [10] proposed Meerkat which is a web defacement detection system using various techniques used in the computer vision field. In the training stage, Meerkat extracts high-level features from screenshots of monitored web pages by using image processing techniques and ML techniques together, and then generates a set of features of monitored web pages; Meerkat works based on a deep neural network in this stage. In the monitoring stage, Meerkat uses generated features of monitored web pages to examine whether current monitored web pages are defaced.

Medvet et al. [12] used a genetic programming technique to learn monitored web pages without any prior knowledge (learning phase), and to monitor the corresponding web pages at pre-determined intervals (monitoring phase). In addition, Bartoli et al. [16,17] proposed Goldrake which is a framework that uses sensors and alarms to automatically check remote web resources' integrity.

Kim et al. [9] proposed an n-gram based detection method that uses N-Gram-based Index Distance (NGID) to validate dynamic web pages. In [19], they proposed a defense mechanism for detecting web pages in a remote site and two threshold adjustment methods to lower false alarm rate.

Hoang and Nguyen [18] proposed a hybrid defacement detection model that is designed based on the combination of the ML-based detection and the signature-based detection.

Davanzo et al. [4] assessed the performance of several anomaly detection approaches designed based on ML techniques in terms of false positive ratio and false negative ratio. They conducted extensive experiments by using around 300 dynamic web pages for three months.

In addition to ML-based detection methods, various client-based detection methods have been proposed as follows.

Kim et al. [7] proposed a website falsification detection method in which web crawlers regularly collect web pages from a website, extract codes and images from the collected web pages, and determine whether web pages are defaced by analyzing the extracted codes and images in terms of similarity.

Masango et al. [13] proposed a WDIMT (Web Defacement and Intrusion Monitoring Tool) that operates like a web vulnerability scanner. When web defacement is detected, WDIMT can automatically recover the defaced web page by using its original web file stored before it is defaced. Similarly, Kals et al. [14] proposed Secubat, which is designed based on penetration testing techniques.

Park and Cho [11] proposed CREMS (Client-based Real-time wEb defacement Monitoring and detection System) that periodically examines web pages to see if they are defaced. Specifically, CREMS compares each web page's source codes every second and measures similarity after comparison. If the measured similarity value is below a certain threshold, CREMS raises an alarm and reports it to system administrators for further investigation. In addition, by using its source code matching algorithm, CREMS can locate the exact place where malicious codes are injected within a defaced web page.

According to our extensive survey, most previous works can be classified into either proposing new web defacement detection methods or improving detection accuracy of existing detection approaches. Interestingly, we observed that no studies clearly described how their monitoring cycles are set or should be set. For example, some detection methods [8,11,16,20,21] monitor web pages with a periodic or fixed monitoring cycle without clear explanations, and some works [7,9,10,12,15,18,19] did not even explain in detail about their monitoring method and cycle.

Meanwhile, recent advances of information and communication technology (ICT) techniques including AI (artificial intelligence) and ML (machine learning) techniques make cyberattacks more intelligent and sophisticated. Consequently, we should not ignore the possibility that attackers can avoid or nullify existing security systems by exploiting vulnerabilities that can be discovered by analyzing their operational patterns and behaviors [4].

For this reason, in this paper we introduce intelligent on-off web defacement attacks in order to show how existing client-based web defacement detection systems using fixed monitoring cycle can be vulnerable and then discuss our defense strategy and methods against such attacks.

3. Intelligent On-Off Web Defacement Attack

In this section, we explain why existing client-based detection approaches with fixed monitoring cycle can be easily, completely nullified by on-off attack strategy, and then we introduce a new intelligent web defacement attack model based on the on-off attack strategy.

3.1. The Security Weakness of Client-Based Detection Methods with a Fixed Monitoring Cycle

First, we describe a general description of client-based defense approaches with fixed monitoring cycle. Figure 2 shows an example of a web defacement detection system with a fixed monitoring cycle $\boldsymbol{c}$ = 10 seconds, which means that the detection system monitors and examines a web page every 10 seconds. We assume that the monitoring cycle $\boldsymbol{c}$ necessarily exists since no detection systems can monitor continuously due to their limited computing resources, the complexity of monitoring algorithms, etc. In this example, we assume that the unit is second for simplicity, but depending on detection systems, the unit of monitoring cycle can be second, milli-second, or even smaller. In addition, if we consider each monitoring cycle as a monitoring round (MR), then one MR consists of 10 time slots. As described in Figure 2, if the first monitoring activity is done at the first monitoring slot ($\boldsymbol{ms_1}$),

every $(10t + 1)$-th time slot will be examined by the detection system where $t = 1, 2, \ldots, \infty$. A simple fixed periodic monitoring algorithm (**FPMA**) is described in Algorithm 1.

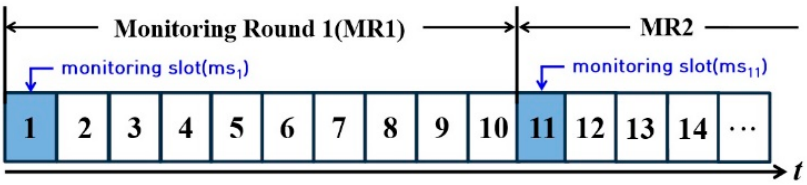


Figure 2. An example of a client-based web defacement detection system with a fixed monitoring cycle c (c = 10 seconds).

Algorithm 1: Fixed Periodic Monitoring Algorithm (FPMA).

Input:
Number of slots: n
Current time: $t_{current}$
Start time of current monitoring round (MR): $t_{MRstart}$
Fixed monitoring slot: ms_{fixed}
Output:
Detection result: ***detection_result***

```
1:  begin
2:      while (t_MRstart ≤ t_current ≤ (t_MRstart + n − 1)) :
3:      if t_current == (t_MRstart + ms_fixed − 1):
4:          // monitor( ) checks if web pages are defaced
5:        detection_result ← monitor()
6:      else:
7:          // monitor( ) is not performed
8:          continue
9:  end
```

Next, we describe how an intelligent web defacement attacker with an on-off attack strategy can avoid and nullify the above monitoring mechanism. We have the following assumptions (AS1-AS4):

- AS1: Attacker can discover some security vulnerabilities of its target web server such as **WS**
- AS2: Defender (client-based web defacement monitoring system located outside **WS**) monitors web pages stored in **WS** periodically (every 10 seconds)
- AS3: Attacker can modify web pages in **WS** by AS1
- AS4: Attacker can figure out monitoring cycle c and previous monitoring slots at the time t

Based on the above assumptions (AS1-AS4), the attacker can also figure out the next monitoring slots (blue-colored slots) at t. Figure 3 shows every possible monitoring slots (red-colored slots) at which the attacker can safely launch web defacement attacks to the victim server. Thus, except for the monitoring slots, the attacker can deface web pages at the red-colored time slots (non-monitoring slots).

When we define *Attack Success Rate* $(ASR)(\%) = \frac{Num.\ of\ defaced\ time\ slots}{Num.\ of\ all\ time\ slots} \times 100$, ASR is 90% in this example; in other words, the attacker is able to deface its target web pages for 90% of time even without being detected by the monitoring system. Note that a defaced (time) slot means that the web deface attack is successfully launched at that time slot, and we use the term for the rest of this paper. Moreover, instead of defacing web pages for all time slots, the attacker can selectively choose some part of time slots for defacement in an on-off manner. In this case, ASR will decrease according to the amount of chosen attacking slots, but it will become more difficult to detect such attacks. In this paper, we name this type of attack as *intelligent on-off web defacement attack* and describe the attack model with a generalized algorithm (Algorithm 2) in Section 3.2.

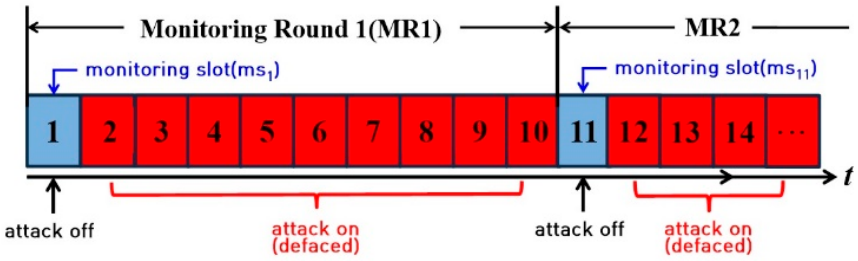


Figure 3. A description of an intelligent on-off web defacement attack; this intelligent attacker defaces web pages only for the red-colored time slots safely by avoiding the blue-colored monitoring slots.

3.2. Attack model: Intelligent on-off web defacement attacks

When the intelligent on-off web defacement attacker successfully defaced a certain web page WP, let $WP_{original}$ be the original web page of WP and $WP_{defaced}$ be the defaced web page of WP. To avoid being captured by a client-based web defacement detection with a fixed monitoring cycle, the intelligent on-off web defacement attacker acts as follows (see Algorithm 2).

- Attacker stores $WP_{original}$ before defacing it;
- To avoid a monitoring slot, the attacker calculates (or estimates) the next monitoring slot ms_{next} by a detection system based on current time $t_{current}$, monitoring cycle c, and previous monitoring slot $ms_{previous}$;
- When $t_{current}$ is not ms_{next}, attacker defaces WP;
- When $t_{current}$ is ms_{next}, attacker does not deface WP; if the web page is already defaced, attacker replaces $WP_{defaced}$ with $WP_{original}$ to avoid being captured by defender.

Algorithm 2: Intelligent On-Off Web Defacement Attack

Input:
Current time: $t_{current}$
Previous monitoring time (slot): $ms_{previous}$
Monitoring cycle (fixed): c
Original web page: $WP_{original}$
Defaced web page: $WP_{defaced}$
Output:
State of web page: WP_{state}

```
1:  begin
2:    while (true):
3:      if (t_current − ms_previous) != c:
4:        WP_state ← WP_defaced        # attack mode is on
5:      else:
6:        WP_state ←WP_original        # attack mode is off
7:  end
```

4. Random Monitoring-Based Defense Strategy and Two Detection Algorithms

In this section, we claim that a random monitoring strategy can effectively defend against intelligent on-off web defacement attacks by conducting a simple probabilistic analysis, and design two web defacement attack detection algorithms based on the random monitoring strategy.

4.1. Defense Strategy: Random Monitoring

In many computer and network security problems, it is often assumed that attackers are in superior positions than defenders. For example, defenders have limited resources but need to care for many

defense spots (weak points) of their assets while attackers are able to successfully launch attacks if they can exploit at least one vulnerability of defenders' assets. For this reason, many computer and network security problems are formulated as unfair games between the attacker and the defender [28–30].

One of the effective defense strategies is to assign defenders' limited small defense resources to large defense spots in random ways, so that attackers cannot figure out which spots will be monitored [29,30]. For example, in [30], a defender uses a random patrol strategy to capture attackers in many defense spots because the defender cannot patrol all patrol spots at the same time, and a fixed periodic patrol method can be easily avoided by attackers. As another effective defense method, moving target defense (MTD) has been actively studied to defend against attackers targeting our assets, such as network devices and data, by moving the assets (or changing the locations of the assets) randomly and frequently and to thus make it very difficult for attackers to accurately target assets when they want [31–34]. In this paper, we will use the former random defense strategy to detect the intelligent on-off web defacement attacks because our research focus is to detect attackers rather than avoiding attackers; we note that studying the latter MTD in this research problem is out of the scope of this paper, but MTD techniques can be very effective for protecting our assets from attackers.

We now justify why the random monitoring strategy can effectively defend against the intelligent on-off web defacement attacks by using a simple probabilistic analysis. Variables and notations used in the analysis are as follows:

- $\boldsymbol{n}$**:** the size of monitoring round (MR) or the size of the monitoring cycle; thus, $\boldsymbol{n}$ is the number of slots that consist of one MR. Each slot in MR can be identified by an index such as $s_1, s_2, \ldots, s_i, \ldots, s_n$. As we explained in Section 3.1, $\boldsymbol{n}$ can vary depending on the performance of defense systems. Given $\boldsymbol{n}$, detection system can monitor only once at s_j where $j \in [1, \boldsymbol{n}]$.
- $\boldsymbol{S_{DS}}$: A finite set of all possible slots from which the defender chooses one slot during one MR; Thus, given $\boldsymbol{n}$, $\boldsymbol{S_{DS}} = \{s_1, s_2, \ldots, s_n\}$ and the cardinality of $\boldsymbol{S_{DS}}$ ($|\boldsymbol{S_{DS}}|$) = $\boldsymbol{n}$.
- $\boldsymbol{S_{AS}}$: A finite set of all possible combinations of slots from which the attacker chooses one or more slots for launching defacement attacks during one MR. Thus, given $\boldsymbol{n}$, $\boldsymbol{S_{AS}} = \{s_1, s_2, \ldots, s_n, (s_1, s_2), (s_2, s_3), \ldots, (s_{n-1}, s_n), \ldots, (s_1, s_2, \ldots, s_n)\}$ and $|\boldsymbol{S_{AS}}| = 2^{\mathrm{n}} - 1$; $\boldsymbol{S_{AS}}$ = the power set of $\boldsymbol{S_{DS}}$ - null set $\emptyset$.
- Random variable $\boldsymbol{X}$: slots that the attacker chooses
- Random variable $\boldsymbol{Y}$: one slot that the defender chooses
- Let $\boldsymbol{P}[\boldsymbol{X} = \boldsymbol{s_i}^+]$ be the probability that $\boldsymbol{X}$ contains $\boldsymbol{s_i}$.

By the definition, two random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent each other. Since the total number of elements of $\boldsymbol{S_{AS}}$ is $2^n - 1$, the probability $\boldsymbol{p}$ that the attacker will be detected during one MR can be obtained by:

$$\boldsymbol{p} = \sum\nolimits_{i \in S_{AS}} \boldsymbol{P}[\boldsymbol{Y} = \boldsymbol{i}]\boldsymbol{P}\left[\boldsymbol{X} = \boldsymbol{i}^+\right] = \frac{\left(2^{n-1} - 1\right)}{\left(2^n - 1\right)}. \quad (1)$$

By using Equation (1), the probability $\boldsymbol{p(r)}$ that the attacker will be detected during $\boldsymbol{r}$ consecutive MRs can be obtained by:

$$\boldsymbol{p}(\boldsymbol{r}) = 1 - (1 - \boldsymbol{p})^{\boldsymbol{r}} = 1 - \left(1 - \frac{\left(2^{n-1} - 1\right)}{\left(2^n - 1\right)}\right)^{\boldsymbol{r}} = 1 - \left(\frac{2^{n-1}}{2^n - 1}\right)^{\boldsymbol{r}} \quad (2)$$

When $\boldsymbol{n} = 10$, $\boldsymbol{p} \simeq 0.4995$ according to Equation (1). According to Equation (2), $\boldsymbol{p}(\boldsymbol{r})$ becomes higher than 0.87 when $\boldsymbol{r} \geq 3$. As shown in Figure 4, as $\boldsymbol{r}$ grows, $\boldsymbol{p}(\boldsymbol{r})$ grows quickly and eventually converges to 1. Meanwhile, even if the intelligent on-off web defacement attacker knows that the defender is monitoring only one slot during one MR, it is very unlikely for the attacker to avoid being detected for a long time (many MRs) when the attacker keeps defacing web pages in on-off manner. Consequently, the random monitoring strategy can effectively defend against intelligent on-off web defacement attacks.

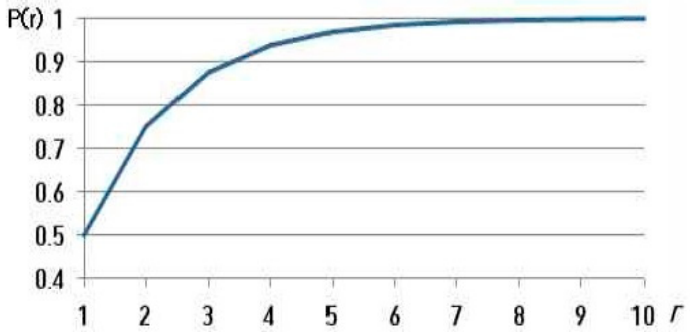


Figure 4. *P(r)*, the probability that the attacker will be captured by random monitoring method when the number of MR = *r* and the size of MR (*n*) = 10.

4.2. Design of Two Detection Abased on Random Monitoring Strategy

Based on the random monitoring strategy, we design two detection algorithms against intelligent on-off web defacement attacks: 1) Uniform Random Monitoring Algorithm (**URMA**) and 2) Attack Damage-based Random Monitoring Algorithm (**ADRMA**). In this study, our goal is not to design the best random monitoring algorithm in terms of detection performance, but to show you various ways of designing random monitoring algorithms. For the detection performance of our algorithms, we will explain in Section 5.

4.2.1. Uniform Random Monitoring Algorithm (URMA)

The uniform random monitoring algorithm (URMA) chooses one slot per one MR in a uniform manner and checks if web pages are attacked in the chosen slot.

As described in Algorithm 3 below, when each MR starts, URMA first selects one slot from $\boldsymbol{n}$ slots according to the uniform distribution; the probability that each slot is selected is $1/n$. Next, if current time $\boldsymbol{t}$ is equal to the chosen slot, monitoring operation is performed to see whether web pages are defaced. For remaining slots, monitoring operation is not performed by assumptions we mentioned in Section 3.

Algorithm 3: Uniform Random Monitoring Algorithm (URMA)

Input:
Number of slots: $\boldsymbol{n}$
Current time: $\boldsymbol{t_{current}}$
Start time of current MR: $\boldsymbol{t_{MRstart}}$
Output:
Detection result: ***detection_result***

```
1:  begin
2:  if t_current == t_MRstart − 1:
3:      ms ← choose one slot for detection slot
4:      according to uniform (1, n)
5:      while (t_MRstart ≤ t_current ≤ (t_MRstart + n − 1)):
6:          if t_current == (t_MRstart + ms − 1):
7:              // monitor( ) checks if web pages are defaced
8:              detection_result ← monitor( )
9:          else:
10:             // monitor( ) is not performed
11:             continue
12: end
```

4.2.2. Attack Damage-Based Random Monitoring Algorithm (ADRMA)

When defenders' resources are limited, they need to wisely use their resources to defend against attackers. One of such ways is that defenders use their defense resources such that the amount of attack damages introduced by attackers can be minimized. Based on this rationale, unlike URMA using uniform randomness against attackers, we design a different random monitoring algorithm ADRMA that considers a factor of attack damage introduced by attacks.

For simplicity, let us consider a case where the size of MR is three (that is, $n = 3$). Then,

- S_{DS} = {1, 2, 3}
- S_{AS} = all subsets of S_{DS} – null set ∅ = {1, 2, 3, (1, 2), (1, 3), (2, 3), (1, 2, 3)}.

In this example, the total number of attack combinations that can be selected by the web defacement on-off attacker is 7 (= $2^3 - 1$) except ∅; we do not consider ∅ because it means that the attacker does not launch attacks. When the defender chooses defense slot d and the attacker chooses attack slots $S_{AS}(i)$ from S_{AS} where i is an index of attack combination, let $D_{Attack=S_{AS(i)}}(d)$ be the amount of attack damage introduced by the attacker. Then, given d and $S_{AS}(i)$, $D_{Attack=S_{AS(i)}}(d)$ is obtained by summing up the amount of attack damage introduced by each attack slot of $S_{AS}(i)$ as:

$$D_{Attack=S_{AS(i)}}(d) = \sum_{i \in S_{AS}} D_{Attack=i}(d) \tag{3}$$

For example, assuming that the amount of attack damage for a single slot is 1, if the attacker launches web defacement attacks at slot 1 and slot 3 and the defender monitors slot 3, $D_{Attack=S_{AS}(5)}(3) = D_{Attack=1}(3) + D_{Attack=3}(3) = 1 + 0 = 1$ (see the blue-colored column in Table 1). That is, the amount of attack damage ($D_{Attack=1}(3)$) is 1 since the attack slot 1 is not monitored and thus attack at slot 1 is valid, and the amount of attack damage ($D_{Attack=3}(3)$) is 0 since the attack slot 3 is monitored and thus attack at slot 3 is invalid. On the other hand, when slot 1 is chosen for defense slot, $D_{Attack=S_{AS}(5)}(1) =$ 0 because the attacker will be captured at slot 1 which is monitored by the defender (defense slot = 1).

Table 1 shows the attack damages calculated for each combination of d and $S_{AS}(i)$ by this manner. We can see that $D_{Attack}(1) = 4$, $D_{Attack}(2) = 6$ and $D_{Attack}(3) = 8$, and as d grows, $D_{Attack}(d)$ also grows. Meanwhile, a rational defender should not always choose the first slot for monitoring since the attacker may not choose the first slot always.

Table 1. Attack damages given a defense slot and attack slots (when the size of MR = 3).

d	$S_{AS}(i)$							$D_{Attack}(d)$	
	$S_{AS}(1)$	$S_{AS}(2)$	$S_{AS}(3)$	$S_{AS}(4)$	$S_{AS}(5)$	$S_{AS}(6)$	$S_{AS}(7)$		
	1	**2**	**3**	**(1,2)**	**(1,3)**	**(2,3)**	**(1,2,3)**	∑	**ratio**
1	0	1	1	0	0	2	0	4	2
2	1	0	1	1	2	0	1	6	3
3	1	1	0	2	**1**	1	2	8	4

When n is given, the attacker can consider choosing one from at most $2^n - 1$ combinations of attack slots. Given n and d, $D_{Attack}(d)$ can be easily, efficiently calculated by the below Equation (4), which operates in **O**(1) in terms of algorithmic time complexity. Derivation of Equation (4) is shown in Appendix A.

$$D_{Attack}(d) = \begin{cases} (n-1)2^{n-2} & for\ d = 1, \\ (n+d-2)2^{n-2} & for\ 2 \le d < n, \\ (n-1)2^{n-1} & for\ d = n. \end{cases} \tag{4}$$

● Design of Attack Damage-based Random Monitoring Algorithm (ADRMA)

Now we design a random monitoring algorithm by using $D_{Attack}(d)$. As shown in Table 1, $D_{Attack}(1) = 4$, $D_{Attack}(2) = 6$ and $D_{Attack}(3) = 8$. The ratio of $D_{Attack}(1)$, $D_{Attack}(2)$, $D_{Attack}(3)$ is 2

: 3 : 4. The higher $D_A(d)$, the larger damages the defender will be likely to get. **ADRMA** chooses a defense slot according to the inverse ratio of $D_{Attack}(d)$. In this approach, a slot with lower $D_{Attack}(d)$ will be more likely chosen as a defense slot than a slot with higher $D_{Attack}(d)$.

As shown in Algorithm 4, ADRMA works in two steps. In Step 1, given d and n, ADRMA calculates attack damage $D_{Attack}(d)$ for each slot according to the Equation (4). In Step 2, each time MR starts, ADRMA chooses one from n slots randomly according to the inverse ratio of $D_{Attack}(d)$. After that, ADRMA checks if web pages are defaced at the chosen defense slot and does not check for the remaining slots.

Algorithm 4: Attack Damage-Based Random Monitoring Algorithm (ADRMA)

Input:
Number of slots: n
Current time: $t_{current}$
Start time of current MR: $t_{MRstart}$
Output:
Attack damage D_A
Defense slot ds
Detection result: $detection\_result$

begin
  // Step 1: Calculate D_{Attack} to choose a defense slot
  for each d **in** [1, n]**:**
    if $d == 1$**:**
      $D_{Attack}(d) = (n-1)2^{n-2}$
    if $2 \leq d < n$**:**
      $D_{Attack}(d) = (n+d-2)2^{n-2}$
    if $d == n$**:**
      $D_{Attack}(d) = (n-1)2^{n-1}$
  // Step 2: Choose a defense slot and Monitor
$ds \leftarrow$ choose one slot randomly by using $D_{Attack}(d)$
    such that a slot with lower $D_{Attack}(d)$ will be
    more likely chosen as a defense slot than a slot
    with higher $D_{Attack}(d)$.
  while $(t_{MRstart} \leq t_{current} \leq t_{MRstart} + n - 1)$**:**
  if $t_{current} == (t_{MRstart} + ds - 1)$**:**
      // monitor( ) checks if web pages are defaced
    $detection\_result \leftarrow$ monitor( )
    else:
      // monitor( ) is not performed
      continue
end

5. Experiment Results

5.1. Experimental Objectives and Methods

5.1.1. Purpose of Experiments

The main purpose of conducting experiments here is to show how effectively our two random monitoring algorithms (URMA and ADRMA) work against various intelligent on-off web defacement attacks.

For this purpose, with Python 3 programming language, we implemented three intelligent on-off web defacement attack models (AM1, AM2, and AM3) based on Algorithm 2 as we described

below in detail. In addition, we implemented our two random monitoring algorithms URMA and ADRMA according to Algorithm 3 and Algorithm 4, respectively. Moreover, to compare with our random monitoring algorithms, we implemented a simple fixed periodic monitoring algorithm (FPMA) according to Algorithm 1, and for simplicity FPMA always monitors the first slot of each monitoring round (MR).

5.1.2. Three Intelligent On-Off Attack Models

- **AM1** (most aggressive): In this attack model, we assume that the attacker knows how FPMA operates but does not have any knowledge about our random-monitoring algorithms. In AM3, the attacker is very aggressive such that it tries to deface all slots except the first slot of each MR monitored by FPMA.
- **AM2** (moderately aggressive): In this attack model, we assume that the attacker knows not only FPMA but also the existence of our random-monitoring algorithms. Unlike AM1, the attacker in AM2 does not attack all safe slots. Instead, the attack tries to deface one or more slots randomly until he/she is detected. Specifically, the attacker will decide whether it deface each slot according to attack rate $\boldsymbol{R_A}$. For example, if attack rate $\boldsymbol{R_A}$ = 80%, the attacker will launch defacement attack at each slot with the probability = 0.8. Thus, the higher $\boldsymbol{R_A}$ is, the more aggressively the attacker defaces. In our experiment, we used $\boldsymbol{R_A}$ = 80%, 60% and 40%.
- **AM3** (least aggressive): Like AM2, we assume that the attacker knows not only FPMA but also the existence of our random-monitoring algorithms. Unlike AM2, the attacker in AM3 randomly chooses only one slot for each MR until he/she is detected by our random monitoring algorithms as the following. Assuming that the size of MR = $\boldsymbol{n}$ and slot 1 is the monitoring slot by FPMA, slot $\boldsymbol{i}$ will be more likely chosen by the attacker than slot $\boldsymbol{j}$ where $\boldsymbol{i} \geq \boldsymbol{j}$ and $2 \leq \boldsymbol{i}, \boldsymbol{j} \leq \boldsymbol{n}$. This attack model is designed by considering that the attacker may think that slot 2 just after slot 1 is the most safe slot for launching defacement attacks because slot 2 is the most distant slot to the next monitoring slot (slot 1 + $\boldsymbol{n}$) while slot n is the most dangerous slot at which the attacker may be detected by FPMA.

5.1.3. Experimental Methods and Metrics

In our experiments, one experiment proceeds as follows. First, each monitoring round MR (whose size |MR| = $\boldsymbol{n}$) starts, the attacker randomly chooses one attack combination that consists of one or more slots for launching the defacement attack according to three attack models (AM1, AM2, and AM3), and also the defender chooses one defense slot according to three monitoring algorithms (FPMA, URMA, and ADRMA). After that, each experiment checks whether the launched attack is monitored at the chosen defense slot; that is, we conclude the launched attack is monitored if any slot of the chosen attack combination by the attacker matches the chosen defense slot by the defender. Finally, each experiment terminates either when the attack is monitored by all three monitoring algorithms (FPMA, URMA and ADRMA) or when the number of monitoring rounds reaches 100 rounds; as we will explain later in Section 5.2, the latter condition is necessary since FPMA could not detect any of implemented intelligent attack models while our random monitoring algorithms URMA and ADRMA could detect all attack models successfully within a couple of monitoring rounds. We conducted all our experiments on our laptop (with Intel 7th Gen Core i5 and RAM 4GB) by running simulation programs which we implemented with Python 3.

For experiment result analysis, we use the following experiment metrics ($\boldsymbol{N_{MR}}$, $\boldsymbol{N_{ES}}$, $\boldsymbol{N_{DS}}$, $\boldsymbol{N_{AD}}$, and $\boldsymbol{AADR}$) to compare three monitoring algorithms in the presence of three attack models:

(1) The number of elapsed monitoring rounds until the attacker is detected ($\boldsymbol{N_{MR}}$) and the number of elapsed slots until the attacker is detected ($\boldsymbol{N_{ES}}$): These two metrics explain how quickly a monitoring algorithm can detect the attacker in terms of attack detection speed; recall that one monitoring round consists of n slots.

(2) The number of defaced slots until the attacker is detected (N_{DS}): This metric explains how long the attacker can successfully launch the web defacement attack until he/she is detected by a monitoring algorithm. That is, N_{DS} indicates the amount of damage caused by the attacker and N_{DS} is measured by counting the total number of slots that the attacker has defaced successfully until the attacker is detected by a monitoring algorithm.

(3) The number of successful attack detections for each monitoring round m ($N_{AD}(m)$) and accumulated attack detection rate by monitoring round m ($AADR(m)$): These metrics measure how successfully and quickly a monitoring algorithm can detect the attack as monitoring round m increases. By using $N_{AD}(m)$, we can obtain $AADR(m)$ by

$$AADR(m) = \frac{\sum_{i=1}^{m} N_{AD}(i)}{Total\ num.\ of\ launched\ attacks} \quad (5)$$

Then, by definition, if a monitoring algorithm could successfully detect the attacker by monitoring round k, $\sum_{i=1}^{k} AADR(i) = 1$. We will use this metric to see how attack detection rate changes as the monitoring round m grows.

We conducted 10,000 experiments and measured the average value of the above metrics. For the size of monitoring round MR (|MR| or n), we used 5 and 10. We consider one slot as the base time unit in our experiments (e.g., one slot = one second).

5.2. Experiment Results and Analysis

Table 2 shows results obtained by conducting extensive experiments according to the experimental methods described in Section 5.1. We explain the results and our analysis on them as follows.

Table 2. Experiment results.

Size of Monitoring Round |MR|		|MR| = 5			|MR| = 10		
Attack Models	Metrics	FPMA	Proposed Algorithms		FPMA	Proposed Algorithms	
			URMA	ADRMA		URMA	ADRMA
AM1	Elapsed MR (N_{MR})	Not detected	1	1	Not detected	1	1
	Elapsed Slots (N_{ES})	Not detected	3.5	3.31	Not detected	6.05	5.48
	Defaced Slots (N_{DS})	400	1.5	1.31	900	4.05	3.48
AM2 (R_A = 80)	Elapsed MR (N_{MR})	Not detected	1.25	1.24	Not detected	1.24	1.24
	Elapsed Slots (N_{ES})	Not detected	4.73	4.5	Not detected	8.38	8.38
	Defaced Slots (N_{DS})	323.24	1.8	1.63	727.57	4.76	4.37
AM2 (R_A = 60)	Elapsed MR (N_{MR})	Not detected	1.64	1.63	Not detected	1.63	1.64
	Elapsed Slots (N_{ES})	Not detected	6.66	6.46	Not detected	12.23	12.01
	Defaced Slots (N_{DS})	244.75	2.08	1.98	545.57	5.43	5.3
AM2 (R_A = 40)	Elapsed MR (N_{MR})	Not detected	2.27	2.31	Not detected	2.47	2.46
	Elapsed Slots (N_{ES})	Not detected	9.84	9.85	Not detected	20.72	20.11
	Defaced Slots (N_{DS})	174.29	2.38	2.37	364.36	6.38	6.12
AM2 (R_A = 20)	Elapsed MR (N_{MR})	Not detected	3.31	3.34	Not detected	4.61	4.64
	Elapsed Slots (N_{ES})	Not detected	15.03	15	Not detected	42.07	41.86
	Defaced Slots (N_{DS})	121.34	2.81	2.83	195.16	7.2	7.19
AM3	Elapsed MR (N_{MR})	Not detected	3.91	3.69	Not detected	8.9	8.6
	Elapsed Slots (N_{ES})	Not detected	17.55	16.27	Not detected	84.34	80.91
	Defaced Slots (N_{DS})	100	2.91	2.69	100	7.9	7.6

First, FPMA could not detect all intelligent on-off web defacement attacks (AM1, AM2, and AM3) in our experiments as shown in Table 2. This is because all intelligent attack models in our experiments are designed according to Algorithm 1 such that the attacker knows exactly which slots FPMA will monitor and thus is able to avoid FPMA's monitoring. As shown in Table 2 and Figure 5, the average N_{DS} (defaced slots) varies according to attack models. As we explained in experimental

methods, although FPMA could not detect attacks, we measured the average N_{DS} when 100 monitoring rounds elapsed because without this condition, experiments will not stop and N_{DS} will continue to grow endlessly.

The result shows that AM1 has the highest N_{DS} because it is the most aggressive attack model in our experiments while AM3 has the lowest N_{DS} because it is the least aggressive attack model. For AM2, as attack rate R_A decreases from 80% to 20%, N_{DS} also decreases almost linearly because R_A for each slot decreases. In addition, except AM3 where only one slot is attacked regardless of the size of MR, N_{DS} when |MR| = 10 is much larger than N_{DS} when |MR| = 5 because the number of successful defaced slots per one MR is 4 when |MR| = 5 and 9 when |MR| = 10, respectively. Consequently, we can see that as the size of MR grows, the attack damage will also grow.

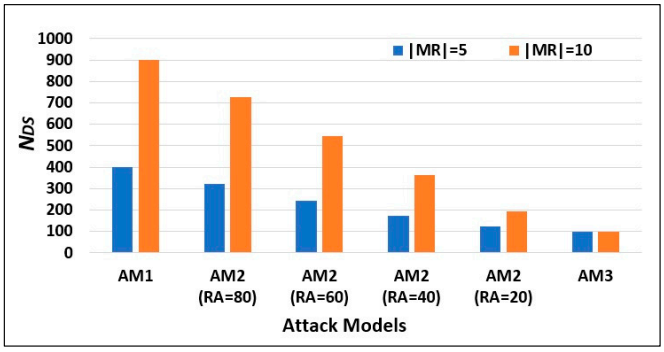


Figure 5. The number of defaced slots until the attacker is detected (N_{DS}) by FPMA in the presence of various attack models (AM1, AM2(R_A = 80, 60, 40, and 20%), and AM3); FPMA cannot detect all type of attacks.

Second, unlike FPMA, all our proposed algorithms (URMA and ADRMA) can successfully detect all type of attacks (AM1, AM2, and AM3) in our experiments. For attack detection speed, as shown in Figure 6, as the attack rate for each lot grows (AM3 → AM2(R_A = 20%) → AM2(R_A = 40%) → AM2(R_A = 60%) → AM2(R_A = 80%) → AM1), N_{MR} also decreases. This means the attack detection speed of both our monitoring algorithms also increases. This result is clear because as the number of attack slots grows according to the attack rate, the possibility that the attacker will be detected also increases. Meanwhile, regardless of |MR| (= 5 or 10), N_{MR} is the same when AM1 is used because the AM1-based attacker defaces all slots except slot 1 and all our algorithms capture the attacker at slot 2. On the other hand, as the attack rate for each slot decreases (AM2(R_A = 80%) → AM2(R_A = 60%) → AM2(R_A = 40%) → AM2(R_A = 20%) → AM3), the difference of N_{MR} when |MR| = 5 and |MR| = 10 becomes lager as described in Figure 6. When AM3 is used, the attacker launches web defacement attacks randomly at only one slot per one MR, the detection speed is relatively slow, but the attack damage is not very high; we will explain the reason below in detail. Figure 7 shows the number of elapsed slots until the attacker is detected (N_{ES}) when |MR| = 10, which is similar with the results of N_{MR}.

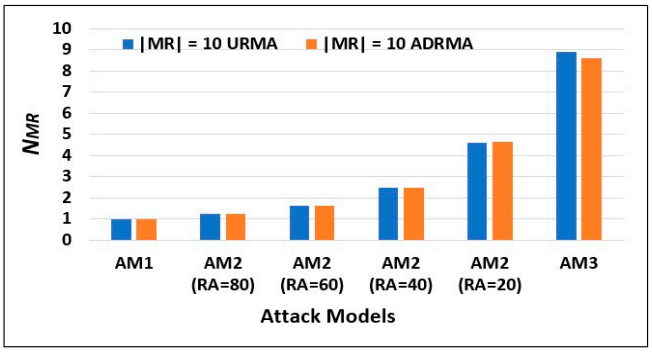


Figure 6. The number of elapsed monitoring rounds until the attacker is detected (N_{MR}) when |MR| = 10.

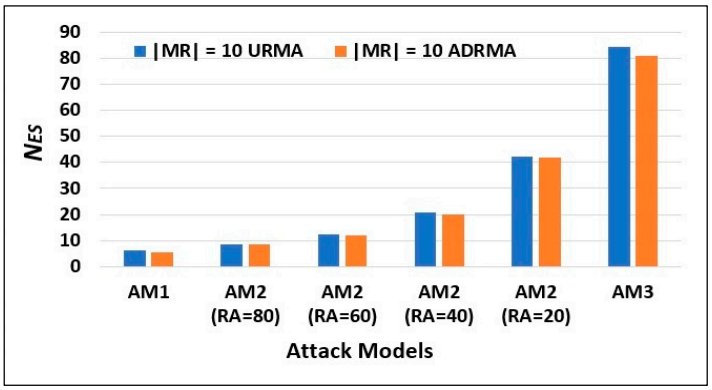


Figure 7. The number of elapsed slots until the attacker is detected (N_{ES}) when |MR| = 10.

Third, in addition to N_{MR} and N_{ES}, $N_{AD}(m)$ and $AADR(m)$ show the attack detection speed of monitoring algorithms according to monitoring round m, and as shown in Figures 8 and 9, our monitoring algorithms could detect most of attacks in early stage of monitoring rounds, especially in the presence of aggressive attacks models (AM1 or AM2(R_A = 80%)). In particular, Figure 9 shows how the accumulated attack detection rate $AADR(m)$ of our two monitoring algorithms changes when AM2 is used. We can see that as m grows, $AADR(m)$ converges to 1 quickly although the growth rate of $AADR(m)$ can vary according to attack models. Intuitively, as the attack rate increases, the growth rate of $AADR(m)$ also increases.

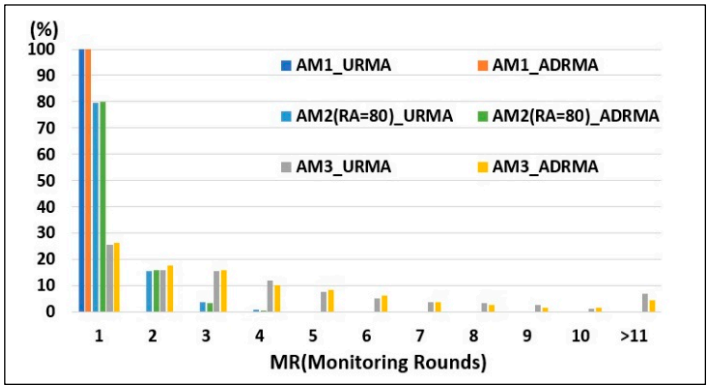


Figure 8. The number of successful attack detections for each monitoring round m (N_{AD}[m]) by URMA and ADRMA when |MR| = 10.

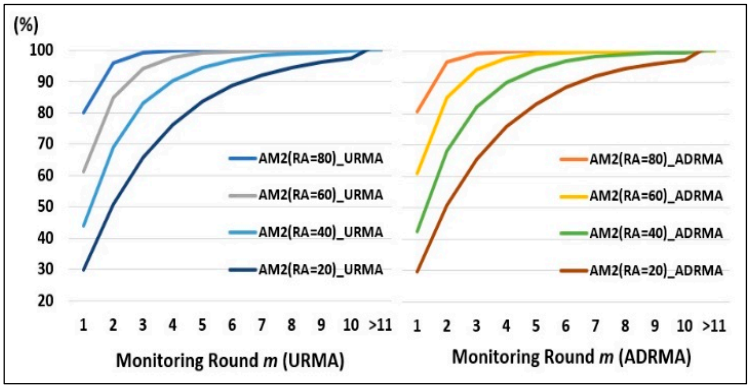


Figure 9. Accumulated attack detection rate by monitoring round m (AADR(m)) in the presence of AM2-based attacks.

Fourth, as shown in Table 2, N_{DS} (the number of defaced slots until the attacker is detected) shows that all attack models could not make huge damage (many defaced slots) in the presence of our random monitoring algorithms, especially compared with the case where FPMA cannot detect attacks at all and thus the attack damage (N_{DS}) continue to grow endlessly. As shown in the below Figure 10, regardless of |MR|, all attack models could not deface more than eight slots in our experiments. Among all attack models, AM3 could make the largest attack damage, but even AM3-based attacker could deface only 7.9 slots at most and then captured by our monitoring algorithm (URMA). This is because random monitoring defense strategy works effectively against intelligent on-off web defacement attack models by quickly capturing such attacks.

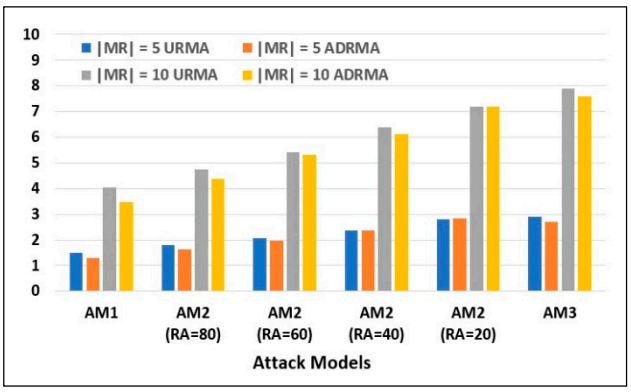


Figure 10. The number of defaced slots until the attacker is detected (N_{DS}) by URMA and ADRMA.

Fifth, ADRMA allows slightly smaller attack damage (the number of defaced slots) than URMA in most attack models used in our experiments (see Table 2 and Figure 10). This is because ADRMA was originally designed to randomly choose a defense slot such that the amount of attack damage can be reduced as we discussed in Section 4. For example, as you can see in Table 2, for |MR| = 10, when compared with URMA, ADRMA could reduce N_{DS} by 8.19% and 14.07% when AM2(R_A = 80) and AM1 was used, respectively. This means that the detection performance of random monitoring algorithms can vary according to their design characteristics. Nevertheless, we can see that both our random monitoring algorithms could effectively defend against all attack models in our experiments by allowing very small deface slots.

Last, AM3-based attack, which is the least aggressive attack used in our experiments, could keep launching attacks successfully about 3.69~8.7 times longer than AM1-based attack before being detected by our random monitoring algorithms (see N_{MR} in Table 2 and Figure 6). This is because AM3 is designed to choose only one slot for each monitoring round and thus the possibility that it can be captured is much smaller than other aggressive attack models that choose one or more slots according to their design characteristics. Consequently, AM3-based attacks were able to make larger attack damage by at most 118% than AM1-based attacks in terms of the number of defaced slots N_{DS}. For attackers, the stealthy attack strategy like AM3 can be better to make larger attack damage to defenders than the aggressive attack strategy like AM1 or AM2(R_A = 80). Nevertheless, they will be caught quickly if they continue to launch attacks even in the presence of random monitoring algorithms.

6. Conclusions and Future Works

In this paper, we first reported that existing client-based web defacement detection methods with a fixed monitoring cycle can be vulnerable to intelligent on-off web defacement attacks. Next, we proposed to use random monitoring defense strategy as a promising countermeasure against the intelligent on-off web defacement attacks by providing a probabilistic analysis on how such strategy can be effective in detecting on-off attacks. In addition, we devised two random monitoring algorithms based on the random monitoring strategy and provided extensive experiment results to validate our

approach and to show the detection performance of our random monitoring algorithms. According to our experiment results, our proposed random monitoring algorithms can detect various intelligent web defacement on-off attacks very quickly while their detection performance slightly vary depending on their design characteristics.

Our future research directions are as follows.

First, we will develop a client-based web defacement detection system with our random monitoring algorithms after further advancing and optimizing their detection performance. To this end, we will deploy it in a real network environment and conduct real-time case studies to see how serious intelligent on-off web defacement attacks can be in the real network environment. Based on the analysis results and findings of case studies, we will further improve our random monitoring detection methods to make our system more feasible and efficient. In addition, we will study a hybrid web defacement defense mechanism that combines client-based methods and server-based methods to better defend against various web defacement attacks.

Next, we will further investigate potential security vulnerabilities of current web defacement detection systems that can be exploited for intelligent attackers to avoid them including our random monitoring algorithms. For example, intelligent attackers may try to find the security weaknesses of random function (or randomness extractor) used in random defense systems. If we inappropriately use a random function with a fixed seed value or use a weak random function with known security vulnerabilities carelessly, it can be possible for the attacker to figure out next random monitoring slots with high probability in advance and then simply avoid random monitoring detection systems. Therefore, we should not only use a strong random function but also protect the random function from intelligent attackers.

Last, we would like to extend our research by investigating more broad range of detection and surveillance systems used in various networks. We want to see if our random detection strategy and algorithms can help them better defend against adversaries who want to actively avoid such defense systems. Especially, we are interested in examining various detection and surveillance systems used in VANET (Vehicular Ad-hoc NETwork) and IoT (Internet of Things) environments [35–37].

Funding: This research received no external funding.

Acknowledgments: An earlier version of this paper was presented and selected as one of outstanding papers at the Conference on Information Security and Cryptography-Summer (CISC-S) in June 2017, South Korea. The author would like to thank reviewers for their valuable comments and constructive suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

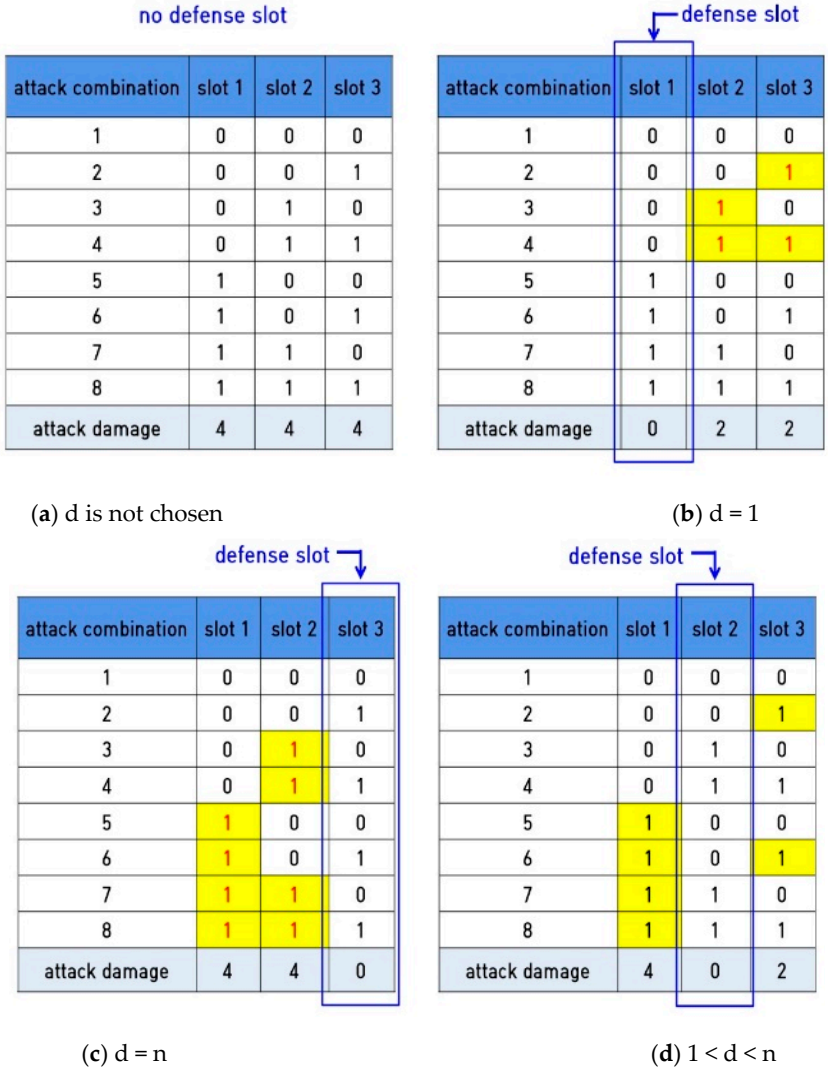
Appendix A

Derivation of $D_{\mathrm{ATTACK}}(d)$ in Equation (4)

Let the size of MR be $\boldsymbol{n}$ and $\boldsymbol{AD(i)}$ be the summation of attack damage that is made by every possible attack combination at a certain slot $\boldsymbol{i}$ (see Figure A1). Recall the definition of $\boldsymbol{D_{Attack}(d)}$ in (3). If there is no defense slot, for all $\boldsymbol{i}$, $\boldsymbol{AD(i)} = 2^{n-1}$ because there are no attack damage for the half of slots that the attacker will not choose (when attack combination = 1, 2, 3, 4). For example, Figure A1a shows an example when $\boldsymbol{n} = 3$ and there is no defense slot, and in this case $\boldsymbol{AD(1) = AD(2) = AD(3)} = 4$ $\left(= 2^{3-1}\right)$.

(1) for $d = 1$

Figure A1b shows an example when $\boldsymbol{n} = 3$ and $\boldsymbol{d} = 1$. When the slot 1 is chosen as the defense slot, $\boldsymbol{AD(1) = 0}$ because either the attacker is captured at the slot 1 (when attack combination = 5, 6, 7, 8) or there is no attack damage (when attack combination = 1, 2, 3, 4). In addition, for each of remaining slots, the amount of attack damage is reduced to 2^{n-2} because there are no attack damages for the half of attack combinations since the attacker will be captured at the slot 1 (when attack combination = 5, 6, 7, 8). Consequently, since the number of the remaining slots except slot 1 is $\boldsymbol{n} - 1$, $\boldsymbol{D_{Attack}(d = 1)} = \sum_{i=1}^{n-1} AD(i) = (n-1)2^{n-2}$.



attack combination	slot 1	slot 2	slot 3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
attack damage	4	4	4

(**a**) d is not chosen

attack combination	slot 1	slot 2	slot 3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
attack damage	0	2	2

(**b**) d = 1

attack combination	slot 1	slot 2	slot 3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
attack damage	4	4	0

(**c**) d = n

attack combination	slot 1	slot 2	slot 3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
attack damage	4	0	2

(**d**) 1 < d < n

Figure A1. An example that shows how D_{Attack} is calculated given n (=3) and d (=0, 1, 2, 3). For example, in (**b**), the yellow-colored slots are valid attack slots when d = 1, and thus D_{Attack} can be calculated as the number of yellow-colored slots by assuming the base unit of attack damage for defacing one slot is 1.

(2) for $d = n$

If the last slot n is chosen as a defense slot, the attack can make attack damage for all previous slots from slot 1 to slot $n - 1$ except the last slot n. Therefore, the total amount of attack damage $D_{Attack}(d = n)$ is $(n-1)2^{n-1}$ which can be calculated by multiplying 2^{n-1} (the amount of damage from each slot) by $n - 1$ (the number of previous slots). Figure A1d shows an example when $n = 3$ and $d = 3$ (the last slot in this case).

(3) for $2 \leq d < n$

As shown in Figure A1c, if the defender chooses one slot j between slot 2 and slot $n - 1$ for defense, we need to consider attack damage before and after the slot j as follows. First, the amount of attack damage for each slot before slot j is 2^{n-1} and the number of slots before slot j is $j - 1$. Next, the amount of attack damage for each slot after slot j is 2^{n-2} which is the half of the former case because attack combination 3, 4, 7 and 8 are excluded additionally and the number of slots after the slot j is $n - j$. Therefore, $D_{Attack}(2 \leq d < n) = (d-1)2^{n-1} + (n-d)2^{n-2} = (n+d-2)2^{n-2}$.

References

1. Zone-H.org. Available online: http://www.zone-h.org/stats/ymd/ (accessed on 15 April 2019).
2. Banff Cyber Technologies. Defacement, B.I.o.W. Available online: https://www.banffcyber.com/knowledge-base/articles/business-implications-web-defacement/ (accessed on 20 January 2019).
3. Bartoli, A.; Davanzo, G.; Medvet, E. The Reaction Time to Web Site Defacements. *Internet Comput.* **2009**, *13*, 52–58. [CrossRef]
4. Davanzo, G.; Medvet, E.; Bartoli, A. Anomaly Detection Technique for a Web Defacement Monitoring Service. *Expert Syst. Appl.* **2011**, *38*, 12521–12530. [CrossRef]
5. Kim, G.H.; Spafford, E.H. Design and Implementation of Tripwire: A File System Integrity Checker. In Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, VA, USA, 19 November 1993; pp. 18–29.
6. Ganger, A.P.; Pennington, A.G.; Strunk, J.D.; Griffin, J.L.; Soules, C.A.N.; Goodson, G.R.; Ganger, G.R. Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior. In Proceedings of the 12th USENIX Security Symposium, Washington, DC, USA, 4–8 Auguest 2003; pp. 1–15.
7. Kim, K.; Choi, S.-S.; Park, H.-S.; Ko, S.-J.; Song, J.-S. Website Falsification Detection System Based on Image and Code Analysis for Enhanced Security Monitoring and Response. *J. Korea Inst. Inf. Secur. Cryptol.* **2014**, *24*, 871–883. [CrossRef]
8. Medvet, E.; Bartoli, A. On the Effects of Learning Set Corruption in Anomaly-Based Detection of Web Defacements. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), Lucerne, Switzerland, 12 July 2007*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 65–78.
9. Kim, W.; Joo, M.; Lee, E.; Lee, D.; Park, E.; Kim, S. N-gram-based dynamic web page defacement validation. In Proceedings of the Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, 23–25 August 2004.
10. Borgolte, K.; Kruegel, C.; Vigna, G. Meerkat: Detecting Website Defacements through Image-based Object Recognition. In Proceedings of the 24th USENIX Conference on Security Symposium, Washington, DC, USA, 12–14 August 2015; pp. 595–610.
11. Park, H.; Cho, Y. CREMS: Client-based Real-time wEb defacement Monitoring and detection System. In Proceedings of the Conference on Information Security and Cryptography-Summer (CSIC-S), Asan, Korea, 3–5 June 2017; pp. 657–658.
12. Medvet, E.; Fillon, C.; Bartoli, A. Detection of Web Defacements by means of Genetic Programming. In Proceedings of the IEEE International Symposium on Information Assurance and Security, Manchester, UK, 29–31 August 2007; pp. 227–234.
13. Masango, M.; Francois, M.; Palesa, A.; Bokang, M. Web Defacement and Intrusion Monitoring Tool: WDIMT. In Proceedings of the International Conference on Cyberworlds, Chester, UK, 20–22 September 2017; pp. 72–79.
14. Kals, S.; Kirda, E.; Kruegel, C.; Jovanovic, N. Secubat: A Web Vulnerability Scanner. In Proceedings of the International Conference on World Wide Web, Edinburgh, Scotland, 23–26 May 2006; pp. 247–256.
15. Kanti, T.; Richariya, V. Implementing a web browser with web defacement detection techniques. *World Comput. Sci. Inf. Technol. J.* **2011**, *1*, 307–310.
16. Bartoli, A.; Davanzo, G.; Medvet, E. A Framework for Large-Scale Detection of Web Site Defacements. *ACM Trans. Internet Technol.* **2010**, *10*, 10–37. [CrossRef]
17. Bartoli, A.; Medvet, E. Automatic Integrity Checks for Remote Web Resources. *IEEE Internet Comput.* **2006**, *10*, 56–62. [CrossRef]
18. Hoang, X.D.; Nguyen, N.T. Detecting Website Defacements Based on Machine Learning Techniques and Attack Signatures. *Computers* **2019**, *8*, 35. [CrossRef]
19. Kim, W.; Lee, J.; Park, E.; Kim, S. Advanced Mechanism for Reducing False Alarm Rate in Web Page Defacement Detection. In Proceedings of the International Workshop on Information Security Applications (WISA), Jeju Island, Korea, 28–30 August 2006.
20. Bergadano, F.; Carretto, F.; Cogno, F.; Ragno, D. Defacement Detection with Passive Adversaries. *Algorithms* **2019**, *12*, 150. [CrossRef]
21. WebOrion Defacement Monitor. Available online: https://www.banffcyber.com/weborion-defacement-monitor/ (accessed on 23 August 2019).

22. Julianto, S.M.; Munir, R. Intrusion detection against unauthorized file modification by integrity checking and recovery with HW/SW platforms using programmable system-on-chip (SoC). In Proceedings of the International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 6–8 March 2018; pp. 174–179.
23. Shi, B.; Li, B.; Cui, L.; Ouyang, L. Vanguard: A Cache-Level Sensitive File Integrity Monitoring System in Virtual Machine Environment. *IEEE Access* **2018**, *6*, 38567–38577. [CrossRef]
24. Smith, C.L. *AIDE-Advanced Intrusion Detection Environment*; Pacific Northwest Nat. Lab.: Richland, WA, USA, 2013.
25. Li, S.; Xiao, L.; Qin, G.; Ruan, L.; Su, S. COW-IMM A Novel Integrity Measurement Method Based on Copy-on-Write for File in Virtual Machine. *IEEE Access* **2018**, *6*, 51776–51790. [CrossRef]
26. Qiang, W.; Yang, J.; Jin, H.; Shi, X. PrivGuard: Protecting Sensitive Kernel Data From Privilege Escalation Attacks. *IEEE Access* **2018**, *6*, 46584–46594. [CrossRef]
27. O'Leary, M. Privilege Escalation in Linux. In *Cyber Operations*; Apress: Berkeley, CA, USA, 2019; pp. 419–453.
28. Moisan, F.; Gonzalez, C. Security under Uncertainty: Adaptive Attackers Are More Challenging to Human Defenders than Random Attackers. *Front. Psychol.* **2017**, *8*, 982. [CrossRef] [PubMed]
29. Nguyen, T.H.; Kar, D.; Brown, M.; Sinha, A.; Jiang, A.X.; Tambe, M. Towards a Science of Security Games. In *Mathematics & Statistics*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 6.
30. També, M. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*; Cambridge University: Cambridge, UK, 2011.
31. Zhang, H.; Zheng, K.; Wang, X.; Lou, S.; Wu, B. Efficient Strategy Selection for Moving Target Defense Under Multiple Attacks. *IEEE Access* **2019**, *7*, 65982–65995. [CrossRef]
32. Connell, W.; Menasce, D.A.; Albanese, M. Performance Modeling of Moving Target Defenses with Reconfiguration Limits. *IEEE Trans. Dependable Secur. Comput.* **2018**, *99*, 1. [CrossRef]
33. Lei, C.; Ma, D.-H.; Zhang, H.-Q. Optimal Strategy Selection for Moving Target Defense Based on Markov Game. *IEEE Access* **2017**, *5*, 156–169. [CrossRef]
34. Sharma, D.P.; Cho, J.-H.; Moore, T.J.; Nelson, F.F.; Lim, H.; Kim, D.S. Random Host and Service Multiplexing for Moving Target Defense in Software-Defined Networks. In Proceedings of the IEEE International Conference on Communications (ICC), Shanghai, China, 26–28 May 2019.
35. Lim, K.; Tuladhar, K.M.; Kim, H. Detecting Location Spoofing using ADAS sensors in VANETs. In Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019.
36. Kim, H.; Ben-Othman, J. A Collision-free Surveillance System using Smart UAVs in Multi Domain IoT. *IEEE Commun. Lett.* **2018**, *22*, 2587–2590. [CrossRef]
37. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J.; Alazab, A. A Novel Ensemble of Hybrid Intrusion Detection System for Detecting Internet of Things Attacks. *Electronics* **2019**, *8*, 1210. [CrossRef]