International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 9, Number 2, pages 188–200, July 2019

# Analyzing the Effect of Moving Target Defense for a Web System

Wai Kyi Kyi Oo
Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishi-ku, Fukuoka 819-0395, JAPAN

Hiroshi Koide
Graduate School of Information Science and Electrical Engineering,
Research Institute for Information Technology, Kyushu University
744 Motooka, Nishi-ku, Fukuoka 819-0395, JAPAN

Kouichi Sakurai
Graduate School of Information Science and Electrical Engineering, Kyushu University
744 Motooka, Nishi-ku, Fukuoka 819-0395, JAPAN

Received: February 15, 2019
Revised: May 6, 2019
Accepted: May 10, 2019
Communicated by Shuichi Ichikawa

**Abstract**

Moving target defense (MTD) is a feasible idea for reducing the ratio of successful attacks by altering or diversifying the attributes or parameters of a protected system. As a result of applying MTD techniques to a system, an attacker would have more difficulties in launching attacks. Although several MTD techniques have been proposed for different types of attack, estimating the effectiveness of combining these MTDs remains a challenge. With the aim of setting up a method for evaluating MTDs, we first propose a model composed of two MTD diversification techniques to compare an attack success ratio between theoretical and experimental probability. To validate the proposed model, we conducted an experiment involving an actual attack and then analyzed how our MTD model can adequately estimate a binary-code injection attack. Results show that the rate of attack success is 100% when MTD diversification is not implemented, while the rate is reduced depending on how many variants can be diversified in a target system. Our method is an important first step toward establishing a method for evaluating MTDs, as well as predicting an MTD's defensive abilities.

*Keywords:* moving target defense, cyber defense, attack success ratio, binary-code injection, diversification, web system

## 1 Introduction

With the huge rise in the number of networked systems in the cyberspace, users are increasingly at risk and vulnerable to various types of cyber threats; thus, effective defense against these cyberattacks is of paramount importance to researchers and security experts. As a web system is a primary interface for an information system that provides web services as well as important information to

users, it is particularly attractive to adversaries. Before an attack is launched, attackers already have an edge because they have sufficient time to perform penetration tests to investigate a victim system repeatedly until they reach their goals [1]. If they can identify a single flaw or vulnerability in the victim system, they can compromise the system by deploying malicious codes through this vulnerability. Even a common vulnerability can be exploited in different systems with multiple attacks. Moreover, attackers have the advantage of cost; a small-scale attack can easily be expanded to a large scale at minimal cost, because the homogeneity in existing network configurations or attributes is deterministic and static [1]. All of these situations can be exploited for creating, exploiting, and spreading cyberattacks easily. While the defensive side rests in a passive position, the offensive side is at an advantage in terms of time, information, and cost. Although several traditional defense approaches and mechanisms are available, and have been explored for remedying this situation, they hardly match the attackers' advantage in terms of time, information, and cost. To reverse this asymmetric situation between attackers and defenders, the MTD concept was proposed as a "game-changing" theme in cybersecurity at the IEEE Security & Privacy Conference in 2010 [2]. Although there is no standard definition of the MTD technology, the main idea is to create, evaluate, and deploy mechanisms, and these techniques should be diversified, or constantly altered and adjusted over time to make it harder and costlier to launch an attack. The MTD technology can also limit the exposure of vulnerabilities, reduce the chances of an attack, and increase the protected system's resilience [2].

The implementation of the MTD technology on computing systems is comparatively new, and is only recently receiving scholarly attention in the cyber security field. Although different MTD mechanisms have been proposed over a decade, sufficient evaluation and analyses of these techniques are yet to be conducted. In order to fill this gap, the contributions of this study are as follows:

- We propose a theoretical moving target defense method that combines two diversification techniques with the aim of not only establishing an evaluation method for MTDs when applied to a web system but also measuring the effectiveness of the proposed MTD system.
- We then conduct an experiment involving actual attacks to quantitatively analyze the idea of an MTD defense system that can reduce the probability of an attack success rate.

Although the ideal implementation of our research is to prepare an actual system, for instance, a Linux system that has a different system call numbering and executable and linkable format (ELF) magic numbering with the aim of disrupting binary-code injection attacks, in this case, we conducted these experiments on a pseudo-experimental environment. In this pseudo-experimental environment, the assumption is that our web system imitates the shuffling system call number and the variation of different ELF magic numbers. Our model can be viewed as a first step toward studying the essence of an MTD approach and its applicability to a real-world information system. Because our study is a preliminary work, and solely focuses on the attacker's perspective, we will undertake a realistic evaluation of the impact of our model, and conduct a more extensive study in future. Based on the experimental results, our method shows that the MTD defense system decreases the window of attack opportunities, while increasing the effort expended on finding and successfully executing an attack. Moreover, even if an attacker launches a successful attack, it may take longer to exploit the system because of the MTD techniques deployed.

The rest of this paper is structured as follows. Section 2 provides an overview of existing MTD mechanisms and methods. Section 3 discusses the proposed evaluation model, and the two MTD diversification methods studied, in relation to the proposed model. Section 4 presents the experimental environment for the case study. Section 5 is about the experimental result, and Section 6 provides the analysis of MTDs based on experimental results. Finally, Section 7 includes conclusion and recommendation for future work.

## 2 Moving Target Defenses

Moving target defense was pioneered by private industry, military research, and academic study; it is actively researched in the field of security innovation today. Studies on MTD technology

are broadly classified into four categories: theory-based, strategy-based, diversification-based, and evaluation-based studies [2, 3, 4, 5]. Table 1 is the overview of these four studies and the relevant existing methods on each study. The theory-based study investigates the key components that can determine the effectiveness of MTD implementation in a conceptual way. The strategy-based study produces advantageous strategies that meet the expected safety goals, after examining the condition of an existing system and potential security threats. These effective strategies can be generated by modeling and designing various security methods. The diversification-based research is a feasible solution that is currently being implemented in real computing systems. Under the diversification-based study, existing research can be divided into three methods, depending on the layers of a system. The application or software attributes of the system are diversified to produce different variants. In the platform or host-level method, multiple operating systems or host machines are configured to increase attackers' difficulty in performing the penetration test or reconnaissance. In the MTD methods based on the network-level method, IP addresses, and service port number, in some cases, two or more network configurations are transformed or diversified to achieve different behaviors. In order to measure the effect of these several MTD techniques, prior research work have evaluated them based on four methods: experiments based on an attacker and defender model, simulation-based method, mathematical evaluation method, and mixed analysis method.

Although MTD techniques are new in the security research field, a few techniques have already been available and applied in the real world. The address space layout randomization (ASLR) technique [6] is the most useful and successful MTD technique that has already been implemented in modern operating systems. ASLR is a dynamic runtime environment method and a memory-location technique that can thwart code-injection attacks by randomizing the memory layout of an application program code. Another notable and conventional technique for thwarting code-injection attacks is the instruction set randomization (ISR) technique [7]. The idea is to hide the behavior of instruction set of the target system by randomly altering the instructions used by the system. As a result, an injected code will not be correctly exploited because of the different behaviors. A technique for measuring the effectiveness of ISR by analyzing two attack variations has been proposed [8]. Most existing studies have focused on the theoretical analysis or basic simulations to examine the effectiveness of the MTD approach; furthermore, all of them focused on a specific MTD method. Some prior works propose producing frameworks in order to evaluate the impacts of a combination of MTDs. Connell et al. proposed a quantification framework by combining multiple existing MTDs in order to analyze the extent to which these MTDs could reduce the likelihood of successful attacks [9]. Another framework [10] was also studied to evaluate the security effects on a network system to determine the MTDs that were most effective in a realistic network system. However, the authors mainly focused on extracting data for the effectiveness of MTDs, and did not attempt to estimate an attacker's effort. Huang et al. conducted a similar research work to predict the best choice of MTD approach for optimal protection [11]. Evans et al. presented a model based on dynamic diversity defense, and studied the effectiveness of MTD defenses in specific scenarios, against specific attacks. They examined the effectiveness of three types of MTDs, ASLR, ISR, and data randomization, for different types of attacks [12].

Table 1: Overview of MTD studies and methods

| **MTD Studies** | **Methods** |
|---|---|
| Theory-based study | Conceptual method |
| Strategy-based study | Modeling and design method |
| Diversification-based study | Application or software-level method<br>Platform or host-level method<br>Network-level Method |
| Evaluation-based study | Attacker-defender experiment method<br>Simulation-based experiment method<br>Mathematical-based experiment method<br>Mixed analysis method |

# 3 Mathematical Model

In this section, we discuss our proposed security evaluation method for MTDs as they perform two diversification techniques. We implement our method with two actors: an attacker and a defender. The defender is a web server that provides a web service, whereas the target of the attacker is to deploy a malicious code in a web server through a specific vulnerability in its web application successfully. In this situation, our web server is designed to disrupt this malicious code from being executed normally in the system by using two MTD diversification techniques.

## 3.1 Basic Principle and Assumption

MTD is considered a proactive diversity defense for reducing an attacker's success rate. In general, attackers can exploit different attack types depending on vulnerabilities that he or she can identify following reconnaissance. As we focus on the exploitation phase of the cyber kill chain [13], in our research, we assume that attackers have already identified the vulnerability through which they can invoke a malicious code with the aim of executing it in the web server. We first focus on evaluating two MTD techniques that can disrupt the attacker's exploitation at the low-level of a system. Hence, we do not consider the attacks that attempt to exploit at the higher application-level of the system. Moreover, we do not consider data storage-level because we target the examination of the binary-code injection attack. We assume that the attacker, through guesswork, generates multiple variants of a program to exploit the binary code being executed successfully.

In our method, we assume that two diversifications of system call numbering and ELF magic numbering are implemented on the web server, the defender. For the threat model, we assume that there is a web application that has a vulnerability being exposed to the public. We regard this vulnerability as exploitable, and there is no file restriction on the web application. Thus, the attacker tries to take advantage of this vulnerability by deploying the executable binary file as attack requests to the web server. The web server will probably accept these requests, and execute and download them as files. As we concentrate only on the attacker who attempts to compromise the system by exploiting the malicious binary code, we do not consider the legitimate users in our method.

## 3.2 Proposed Model

As the main goal of our research is to explore the impact of MTD techniques in web systems, our evaluation framework is based on four features: a web server, a web service, a specific vulnerability of the web application, and two MTD techniques. We define $K - MTD$ as a kind of web system configured with two MTD techniques to provide a web service $W_S$, and the vulnerability of the web service is defined as $S_v$. For the configuration of the two MTD techniques, we define the system call numbers diversification as $N_{MTD}$, and the executable and linkable file (ELF) magic number diversification as $M_{MTD}$. The time taken to accomplish a successful attack, from the commencement of exploitation to the conclusion, is defined as $t_s$. Each successful attack is defined as $S_a$.

To determine the probability of a successful attack, we define $\mathrm{P}(S_a)$ for each MTD technique. The rate of attack success varies for each MTD technique. We assume that the probability of a successful attack is ($\mathrm{P}(S_a)$=1), when the static (normal) system is deployed in a web system, because the attacker can execute an attack easily without expending too much effort. If we assume the attacker knows the vulnerability, $S_v$, of the web service, $W_S$, he or she may then exploit this vulnerability to compromise the $K - MTD$ server. When considering code-injection attacks, we intend to analyze the extent to which the attack success ratio can be estimated by deploying two diversification techniques. The proposed method for estimating the attack success ratio of each MTD diversification technique is as follows:

- System Call Number Diversification, $N_{MTD}$: If an $N$-kind system call is used in a shell code, the attack success ratio for one injection will be $\mathrm{P}(S_a) = \frac{1}{N!}$.
- ELF Magic Number Diversification, $M_{MTD}$: If an $M$-kind magic number is used in an executable binary file, the attack success ratio of one injection will be $\mathrm{P}(S_a) = \frac{1}{M!}$.

- The attack success ratio for the combination of two MTD diversification techniques will be: $\mathrm{P}(S_a) = \frac{1}{N!} \times \frac{1}{M!}$.
- To determine the mean attack successful time for the combination of two MTDs, if one attack needs to be successfully exploited at $t_s$, the mean attack success time will be: $\frac{1}{2} \times N!M! \times t_s$.

We also assume that each MTD diversification technique has distinct characteristics of attack time and number of targets. If we have several $K - MTD$ systems deployed with different configurations of the two diversification techniques, it is expected that the attack successful time needed to compromise the $K - MTD$ servers will take longer than when a single system is under attack.

## 3.3 Parameter for Expected Result

To evaluate the attack success rate based on our MTD model, we presume to take the numbers of requests sent to the server and time taken to accomplish these requests as the expected parameters that we want to achieve. The expected parameters we hope to achieve from our proposed model following the experiments are listed in Table 2.

Table 2: The expected parameters to be achieved following experiments

| **Parameters** | **Symbols** |
|---|---|
| Number of attack requests per method | $K_r$ |
| Time taken for a request | $t_s$ |

## 3.4 Code-Injection Attack

Code-injection attacks occur when an attacker injects arbitrary or malicious codes into a vulnerable application on the host's operating system in order to gain unauthorized access to the target system, extract sensitive information, or destroy the target system's hardware. In this case, the arbitrary code is invoked directly through vulnerability in the web application that allows users to upload any files. Many code-injection attacks can be found in web security; a shell injection or binary code injection is a form of attack in which a small code is used as the payload in the exploitation of a program's vulnerability. First, the malicious code is scripted as a program file, and compiled by a compiler; then, it is linked with other library wrapper functions by a linker to generate the output file. After generating a binary output file, the script file for injecting the binary code is built. This code can be implemented in various programming languages such as PHP, Ruby, or even Python. Then, an attacker will channel this code to the running service of a computing system as several requests. After the task of uploading this code to the service of the system is accomplished, many types of unwanted actions such as editing, deleting, or downloading any additional files on the system can be performed. Before the victim system takes action to disrupt this attack, the system is probably compromised; thus, the stage is set for a large-scale attack.

## 3.5 System Call Numbers Diversification

A system call plays the main role in requesting a service from a computer program to the kernel of an operating system; in other words, the interaction between the user mode and kernel mode is mediated through the system calls. As shown in Figure 1, when a user or computer program requests a service; for instance, file opening or writing to an operating system's kernel, it can invoke file system calls such as open() or write(). System calls can be generally invoked directly or using corresponding C library wrapper functions that perform the necessary steps to invoke the system call. The C wrapper function and the system call it invokes are similarly named. However, it may sometimes be different in some cases. If a user program needs to access the resources or services from the operating system's kernel, the user program must use at least one or more system calls because it is the only entry point into the system's kernel.

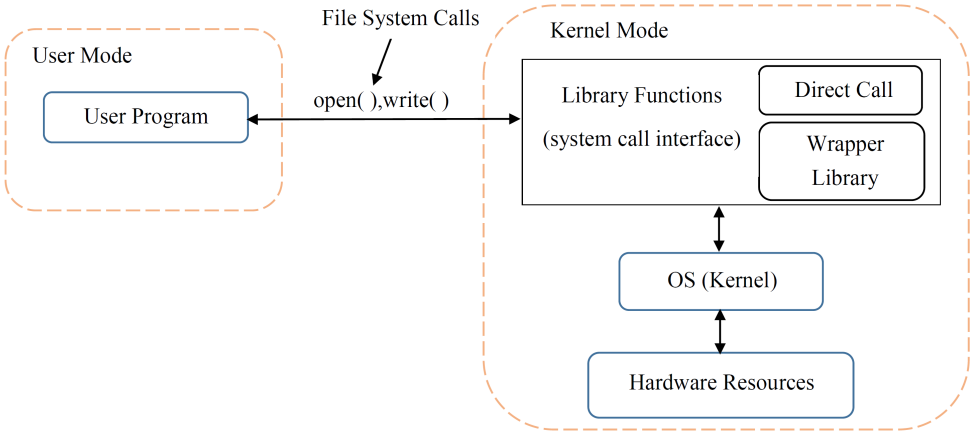


Figure 1: Basic architecture of system call interface

System calls vary according to the system's architecture. Usually, system calls for the 32-bit architecture might differ from that of the 64-bit architecture in Linux and Unix-based operating systems. Each system call has its own index number referred to as system call number. The number of system calls can vary according to the kernel version. System call numbers are listed with their specific index number in a system call table, and can usually be located in the directory "/usr/src/kernel-version/arch/x86/syscalls/syscall_32.tbl or syscall_64.tbl" for 32-bit and 64-bit architectures respectively. In the Unix-based platform, there may be more than 320 system calls used to provide the service between the kernel and user programs. Table 3 shows some examples of system calls, their specific index numbers, and the name of entry points.

Table 3: Examples of system call numbers and their entry vectors

| number | name | entry point |
|---|---|---|
| 0 | read | sys_read |
| 1 | write | sys_write |
| 2 | open | sys_open |
| 3 | close | sys_close |

In this study, we analyzed Linux 64-bit architecture of kernel generic version 4.13, and the system call number defined can be found in the directory "/usr/include/x86_64-linux-gnu/asm/unistd_64.h". As an instance of analyzing a system call and how its system call number is defined, we examined execve( ) system call, application programming interface (API) function call that is used to execute an executable binary or a script file in a 64-bit Linux system. As shown in Listing 1, it has three parameters; the first parameter is either a binary we want to execute, or a script including strings. The second parameter is an array of argument strings related to the program we want to execute, and the third parameter can be any argument we want to pass to the program. The system call number of execve is defined as 59, Listing 2, its entry point is named as *sys_execve*.

In code-injection attacks, attackers commonly use execve system call to execute a shell; for example, "/bin/sh" for the purpose of gaining unauthorized access to resources of a target system. As a result, they can be able to run system commands, and perform unwanted actions with high privilege.

Listing 1: execve() system call funtion in Linux 64 bit architecture

```
#include <unistd.h>

int execve(const char *filename, char *const argv[], char *const envp[]);
```

Listing 2: execve() system call number defined in Linux 64-bit generic kernel

```
#define __NR_execve 59
```

In this study, we aim to evaluate the MTD system call number diversification method that is among the applications of the instruction set architecture randomization technique [7], and we employed our method to render a binary executable injection. Previous research works [14, 15] have proposed this MTD randomization technique. In this research, we attempt predicting the impact of this technique from the attacker's perceptive. If an attacker attempts to exploit the system's vulnerability by directing a malicious code to the system, one or more system calls must be used in the code to gain control of the system. Hence, the system calls numbers and its entry point in the kernel are shuffled in the protected web server. As a result, the user program that tries to invoke the system by using system calls must also be shuffled. If not, it will not be correctly executed.

### 3.6 Executable and Linkable Format (ELF) Magic Number Diversification

In a computer system, ELF is the format of the executable files, binaries, and libraries in Linux. The executable files include the binary data that stores the specific machine language. When we invoke an executable file, the operating system must know how to load it into the memory properly, and how to solve dynamic library dependencies, and where to jump into the loaded executable to start executing it.

The basic format of ELF file is presented in Figure 2. An ELF header describes the format of a file at the beginning, followed by program header table in which an executable or shared object file describes a segment or other information the system needs to prepare the program for execution. Each section represents a part of a file; an executable code is always placed in a .text(code)section, and .rodata contains read-only data. In data segment, all data variables initialized by the user are placed in a .data section. A section header table presents extra information about the sections of a file.

Each ELF has a header containing general information about the binary, and it is fixed at a position in a file, followed by the file data. The ELF header starts with some magic numbers, and has four bytes that contain information about the file. The ELF magic numbers are used to indicate ELF files, and are the very first few bytes of a file. We can use a common tool "readelf"to determine whether the file format is executable or not. The tool, "readelf", is used to display the information about ELF object files, and it has several options of what particular information we want to analyze.

As we focus on the part containing the magic number, we can use the command, "readelf -h /bin/ls | grep Magic" to display the contents of magic bytes. The output of the magic bytes, Listing 3, shows 16 bytes identify "/bin/ls" as an ELF executable file. The first four bytes is the actual magic number, and identifies the file format of executable. If there is an attempt to execute a file in the system, the operating system declines the task when the specific magic number of that file is not matched. Based on this insight, we achieve a variant of the MTD diversification that randomizes a magic number, depending on how many variants of the magic bytes can be generated.

Listing 3: Analysis of ELF magic bytes /bin/ls with readelf tool

```
$ readelf -h /bin/ls | grep Magic
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
```

## 4 Case Study

To validate our proposed method to obtain the attack success ratio by analyzing the expected attack time from the actual exploitation attacks, we carried out actual exploitation attacks. This section

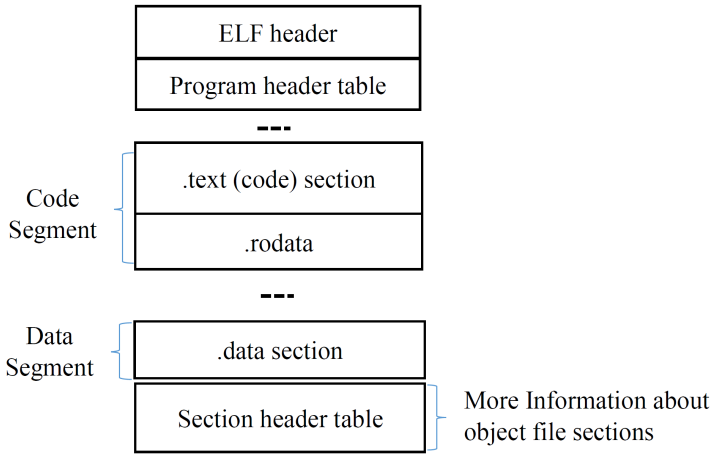


Figure 2: Basic architecture of ELF file format [16]

presents the setting of the experiment for both the attacker-side and defender-side, as well as the attack strategy.

## 4.1 Experiment Setup

Our experimental platform is built on a Window desktop machine with 64 GB RAM and 4.2 GHz 18 core processors. A virtualization technology, Oracle VirtualBox version 5.2.8, is used to create an attacker machine and a web server machine. For network configuration between two machines, we utilized the bridged network, which supports virtual machine (VM) to VM, as well as host to VM. The IP addresses assigned to both the attacker and the web server machines are 10.0.1.37 and 10.0.2.10, respectively. We use Linux distribution, Ubuntu operating system for both machines because of its reputation for efficiency and fast performance.

The attacker VM machine was deployed with GNU compiler collection (gcc) 5.4.0, the standard compiler for most Unix-like operating systems, to generate an executable binary file, and Python requests library 2.21.0, to simplify requests via HyperText Transfer (HTTP) Protocol. A simple script implemented by the C programming language is created first, and compiled by gcc to produce the binary file.

The defender, the web server VM machine, was configured with Apache server 2.4.18, a well-known web server software for supporting web service. As we chose to use the Python web framework, we implemented the Python application server software, web-server gateway interface (WSGI), mod-wsgi's version 4.3.0, along with the Python Flask web framework, 1.0.2. We then configured a simple web service that was assumed to have the vulnerability of being able to which can accept users' POST requests. In this study, the vulnerability we simulated is unrestricted file upload, a form of web application vulnerabilities that is susceptible to simple attacks; however, simple attacks are considered dangerous because they further weaken the application. From this vulnerability, the attacker can send POST requests by directing malicious script to the server, and executing the code remotely.

## 4.2 Attack Strategy

In our experiment, it is assumed that the attacker knows that our web application has a vulnerability that allows users to upload any file via HTTP. As the first step is to exploit this vulnerability by directing the binary executable file to the system, we have to create a binary file that produces the configuration of normal ELF magic number. Then, the attacker sends a request that has a normal ELF magic byte through the web server's vulnerability, and measures the time taken to get a response from the server. After that, we conducted the experiment by launching several binary

files that have varying ELF magic number diversification as attack requests via HTTP protocol to the web server. The attacker needs to guess all possible variants of the ELF magic number to succeed in the attack, as the MTD diversification method is implemented on the web server. These procedures are presented in Figure 3.

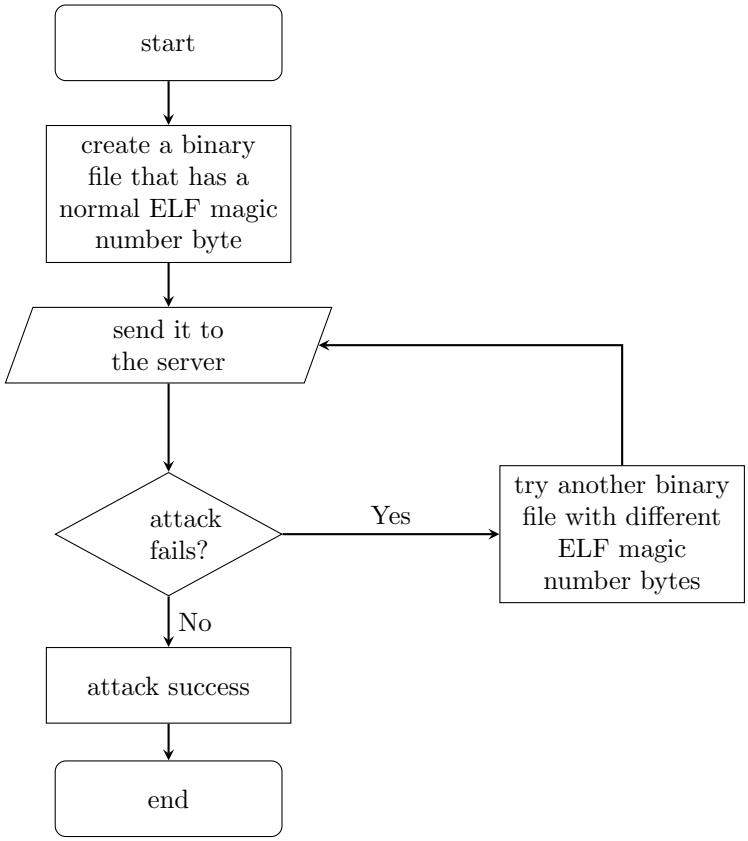


Figure 3: Attack strategy

# 5 Experimental Result

In this section, we discuss the results of the experiment.

## 5.1 Results of ELF Magic Number Diversification Experiment

As shown in Figure 4, from the experimental validation of the proposed model, the number of ELF magic number variants generated was $K_r$, and the time taken to accomplish a successful attack was $t_s$; thus, we achieved the expected parameters and their values. Based on these values, we evaluated the attack success ratio P($S_a$), shown in Table 4. For the ordinary system, the attack success rate is 100%, as no diversification method is used in the system. However, in our MTD system, the attack success rate is less than 1%, as only one attack can be launched (Table 5).

Table 4: Result of ELF diversification method

| **Number of variants diversified($K_r$)** | **Time taken($t_s$)** | **Attack success ratio P($S_a$)** |
|---|---|---|
| 3 | 0.00724 | 0.33 |
| 27 | 3.4236 | 0.037 |
| 140608 | 942.07 | 0.000007 |

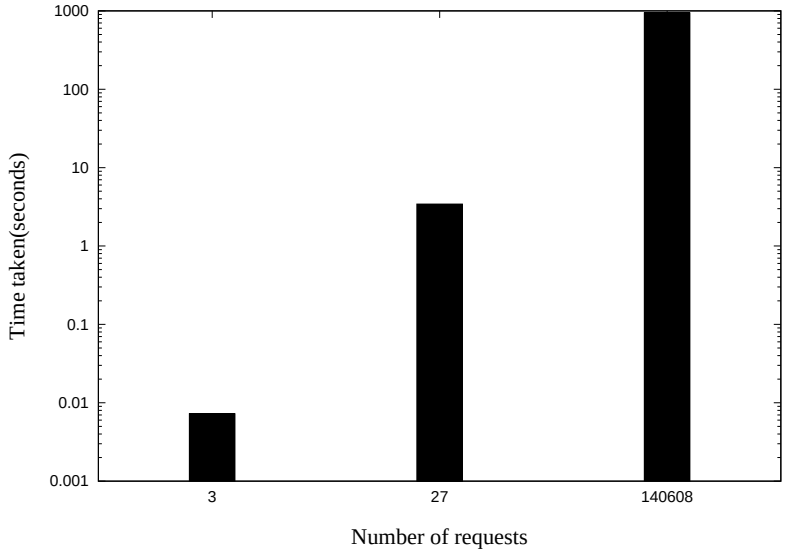


Figure 4: Number of requests and time taken to complete the requests

Table 5: Success rate

| System | Success rate |
|---|---|
| Normal System (with no impact of diversification) | 100% |
| MTD system (with impact of ELF magic number diversification) | less than 1% |

### 5.2 Analyzing System Calls of Attack Program

Though our experimental evaluation of system call number diversification is not included in this study, we analyze the attack binary program by observing the number of system calls invoked. To debug the number of system calls used in the attacker's binary file, we use the "strace" debugging tool to trace a system and the system call of a program. It intercepts and records the system calls that are invoked by a process and the signals that are received by the process. Listing 4 is the command used to monitoring and tracing the execution of the attack on the binary executable program. The output shows how many system calls are used in the attack binary program; 12 system calls are used in exploiting the binary program, though, we prepared the simple binary program. These system calls, their call numbers, and total number of calls are listed in Table 6.

Listing 4: Analysis of system calls in attack program with "strace"

```
for f in /home/user1/flaskapp/test.out ; do
        strace -c $f 2>&1 > /dev/null |
        grep -v ':' | cut --complement -c 42-50 |
        sed '1s/^% /%_/;2d' | head -n -2 ;
done | sed -n '1p;/^%/d;p' | datamash -HW -sg5 sum 4  |
       xargs printf "%-12s\t%14s\n"
```

## 6 Analysis

Based on the experimental results in terms of time $t_s$ and the attack success ratio $P(S_a)$, we can conclude that if the attacker injects the binary code into the static (normal) behavior of the system,

Table 6: List of system calls used in attack program

| System calls | Call number | Total number of calls |
|---|---|---|
| access | 21 | 3 |
| arch_prctl | 158 | 17 |
| brk | 12 | 3 |
| close | 3 | 2 |
| execve | 59 | 1 |
| fstat | 5 | 3 |
| ioctl | 16 | 1 |
| mmap | 9 | 7 |
| mprotect | 10 | 4 |
| munmap | 11 | 1 |
| open | 2 | 2 |
| read | 0 | 1 |
| write | 1 | 1 |

expectedly, he or she can execute an attack with higher successful attack probability in a shorter time. However, when confronted with the MTD system, the attacker needs to expend effort to generate multiple variants in guessing the possible ELF magic numbers for an attack to succeed, but with a zero attack success rate. Thus, the time taken to accomplish the attack is shorter with a higher successful attack ratio in the normal system; conversely, when the time needed to accomplish an attack is longer, the successful attack ratio is lower.

## 6.1 ELF Magic Number Diversification

- For a normal static system that has no transformation impact:
  If the attacker sends a request that has a normal magic number (ELF) to the static server, the time taken to accomplish the attack is 7.023 milliseconds. Therefore, the success rate of attack is 100%.

- For an MTD system that has the ELF transformation impact:
  When the attacker had to guess in the hope of successfully deploying the malicious binary code to the server that incorporates the ELF diversification, 27 attack requests were carried out. Consequently, the time taken to accomplish all these requests was 3.4236s. We also sent out 140608 attack requests to the MTD server, and this took 942.07s.

  Based on the time taken to complete various requests, we can say that as the window of attack opportunities decreases, greater effort is required to find and successfully execute an attack. Moreover, even if an attacker succeeds, it may take longer time to exploit the system because of the MTD techniques deployed on the defender's side.

## 6.2 System Call Number Diversification

For a normal static system that is not shielded by system call number diversification, if the attacker sends an attack program that deployed some system calls, the successful attack rate will expectedly be 100% within a short time. However, when an attacker sends attack requests to the MTD utilizing system call number diversification, the attack success rate is almost zero (Table 7).

Table 7: Estimation of success rate for system call number diversification

| System | Success rate | Analysis |
|---|---|---|
| Normal System | 100% | an attack can be successful within a short time |
| MTD system | zero attack success ratio | 12 system call numbers diversification |

# 7 Conclusion and Future Work

This study presented a model with two MTD diversification techniques to predict the expected time required for an attack to compromise a web system. Toward improving MTD techniques, attack-based experiments are performed to evaluate the security provided by the MTD techniques against binary code injection. In our research, we focus on the idea of altering variants at the low-level of a system to disrupt the execution of binary exploitation. Our approach addresses the unrestricted file uploading vulnerability by incorporating unpredictable diversity with secure architecture in the web server to ward off binary executable file exploitation. This study is a preliminary work; our investigation estimates the attack success ratio with focus on the impact of ELF magic number diversification from the attacker's point of view. We also analyzed the number of system calls used in the simple binary code, and roughly estimated the attack success rate; thus, it is necessary to develop a procedure of the actual attack by using system call number diversification to accurately investigate the effectiveness of MTD. We are convinced that our method can be deployed at different layers of an information system to analyze how MTD diversification techniques can reduce the attack ratio for specific attacks.

Our model can be regarded as the first step toward analyzing the effect of the moving target defense techniques quantitatively. Although we proposed an evaluation method for two MTD techniques, we first carried out the experimental attacks by using ELF magic number diversification in this study. Therefore, we shall further examine the system call number diversification in our future work, in addition to investigating the attack success ratio and the time required to accomplish a successful attack when exploiting shell codes implemented with system calls. The combination of these two methods will also be further studied.

For realistic implementation, we will conduct a more extensive study that will practically evaluate the impact of MTDs on web systems, as well as network systems. To study the effectiveness of MTD techniques, our method can be extended to analyze different MTD techniques for particular attack types based on the parameters we proposed: the number of variants that can be diversified, number of requests, and the time taken to accomplish a successful attack. We believe that our study, by underscoring the necessity of predicting the attacker's approach and making it subject to uncertainties, represents the very first step towards analyzing the impact of MTDs before implementation in a real-world computing system.

# Acknowledgment

This work was supported by Japan Science and Technology Agency(JST), Strategic International Collaborative Research Program (SICORP), Japan.

# References

[1] Gui-lin Cai, Bao-sheng Wang, Wei Hu, and Tian-zuo Wang. Moving target defense: state of the art and characteristics. *Frontiers of Information Technology & Electronic Engineering*, 17(11):1122–1153, 2016.

[2] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media, 2011.

[3] Cheng Lei, Hong-Qi Zhang, Jing-Lei Tan, Yu-Chen Zhang, and Xiao-Hu Liu. Moving target defense techniques: A survey. *Security and Communication Networks*, 2018, 2018.

[4] Jun Xu, Pinyao Guo, Mingyi Zhao, Robert F Erbacher, Minghui Zhu, and Peng Liu. Comparing different moving target defense techniques. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 97–107. ACM, 2014.

[5] Hamed Okhravi, MA Rabe, TJ Mayberry, WG Leonard, TR Hobson, David Bigelow, and WW Streilein. Survey of cyber moving target techniques. Technical report, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 2013.

[6] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307. ACM, 2004.

[7] Gaurav S Kc, Angelos D Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280. ACM, 2003.

[8] Ana Nora Sovarel, David Evans, and Nathanael Paul. Where's the feeb? the effectiveness of instruction set randomization. In *USENIX Security Symposium*, volume 10, 2005.

[9] Warren Connell, Massimiliano Albanese, and Sridhar Venkatesan. A framework for moving target defense quantification. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 124–138. Springer, 2017.

[10] Kara Zaffarano, Joshua Taylor, and Samuel Hamilton. A quantitative framework for moving target defense effectiveness evaluation. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 3–10. ACM, 2015.

[11] Chu Huang, Sencun Zhu, and Yi Yang. An evaluation framework for moving target defense based on analytic hierarchy process. *ICST Trans. Security Safety*, 4:e4, 2018.

[12] David Evans, Anh Nguyen-Tuong, and John Knight. Effectiveness of moving target defenses. In *Moving Target Defense*, pages 29–48. Springer, 2011.

[13] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[14] Zhaohui Liang, Bin Liang, and Lupin Li. A system call randomization based method for countering code-injection attacks. In *International Conference on Networks Security, Wireless Communications and Trusted Computing, NSWCTC*, pages 584–587, 2009.

[15] Monica Chew and Dawn Song. Mitigating buffer overflows by operating system randomization. 2002.

[16] Tool Interface Standards Committee et al. Executable and linkable format (elf). *Specification, Unix System Laboratories*, 1(1):1–20, 2001.