Zheng J, Namin AS. A survey on the moving target defense strategies: An architectural perspective. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(1): 207–233 Jan. 2019. DOI 10.1007/s11390-019-1906-z

A Survey on the Moving Target Defense Strategies: An Architectural Perspective

Jianjun Zheng and Akbar Siami Namin

Department of Computer Science, Texas Tech University, Lubbock, Texas 79409, U.S.A.

E-mail: {jianjun.zheng, akbar.namin}@ttu.edu

Received July 15, 2018; revised October 14, 2018.

Abstract As the complexity and the scale of networks continue to grow, the management of the network operations and security defense has become a challenging task for network administrators, and many network devices may not be updated timely, leaving the network vulnerable to potential attacks. Moreover, the static nature of our existing network infrastructure allows attackers to have enough time to study the static configurations of the network and to launch well-crafted attacks at their convenience while defenders have to work around the clock to defend the network. This asymmetry, in terms of time and money invested, has given attackers greater advantage than defenders and has made the security defense even more challenging. It calls for new and innovative ideas to fix the problem. Moving Target Defense (MTD) is one of the innovative ideas which implements diverse and dynamic configurations of network systems with the goal of puzzling the exact attack surfaces available to attackers. As a result, the system status with the MTD strategy is unpredictable to attackers, hard to exploit, and is more resilient to various forms of attacks. There are existing survey papers on various MTD techniques, but to the best of our knowledge, insufficient focus was given on the architectural perspective of MTD strategies or some new technologies such as Internet of Things (IoT). This paper presents a comprehensive survey on MTD and implementation strategies from the perspective of the architecture of the complete network system, covering the motivation for MTD, the explanation of main MTD concepts, ongoing research efforts of MTD and its implementation at each level of the network system, and the future research opportunities offered by new technologies such as Software-Defined Networking (SDN) and Internet of Things (IoT).

Keywords moving target defense, network security, Software-Defined Networking (SDN)

1 Introduction

1.1 Easy Targets: Static Nature of Networks

Computer system administrators have been struggling with harmful attackers since the beginning of the digital era. In the past, when not many computer systems were interconnected, attackers had very limited capabilities to launch harmful attacks. The situation, however, has quickly turned into a complex one, as more and more computers are interconnected for the purpose of sharing data and services. Nowadays, even without having physical access to vulnerable targets, attackers can utilize emailing services, craft malicious network packets, or employ some other network services to launch attacks remotely and anonymously. The security status of one computer may have significant impact on the security status of all connected computers and thus on the entire network hosting these computers.

In a highly connected network, attackers can utilize any exploited computer system as a platform for launching further attacks against other connected computers and eventually the entire network. Moreover, by utilizing powerful publicly available scripting tools, these malicious entities are capable of automating their attack scenarios at their convenient time and location and launch them with little cost. To defend against malicious attackers, computer administrators or security administrators must closely monitor their systems around the clock, analyze network traffic carefully, patch up any discovered vulnerabilities, and keep all computers and software systems up to date. As the network size grows, the workload and the cost of main-

Survey

This project was supported in part by the National Science Foundation of USA under Grant Nos. 1516636 and 1564293.

tenance and more importantly security management grow rapidly. Therefore, security administrators must employ new tools and techniques to protect the underlying operational infrastructure. For instance, the use of network scanning and fingerprinting tools (e.g., Nmap[1]) is very popular nowadays. Ironically, professional attackers also utilize these types of tools to detect vulnerabilities in the network with the intention of exploiting the discovered security loopholes in the system.

Besides scanning tools, other network defense services are also implemented, such as firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS) with the aim of monitoring network traffic and blocking unauthorized access to valuable data and assets on networks.

Any vulnerable component in a system is associated with a great risk that might be exploited by attackers. The collection of such vulnerabilities is referred to as the attack surface usually examined by malicious entities over the network[1]. In computer networking, attack surface may refer to any system resources exposed to attackers, such as software residing on the hosts, communication ports between hosts[2] or active virtual machine (VM) instances[3]. In order to strengthen the system and enhance the security of the underlying infrastructure, network administrators must clearly tag the attack surface and then employ security techniques to minimize or eliminate it. For instance, to ensure timely the installation of OS security patches, the security administrators need to shut down unused services, close unused open ports, delete obsolete user accounts, and remove unnecessary software. It is a typical and effective approach to defend against various attacks.

However, even with these sophisticated security defense tools and other resources invested, cyber attackers still are able to discover vulnerabilities in network systems. According to the Symantec Internet Security Report published in April 2016[2], on average a new zero-day vulnerability was found each week in 2015, more than double compared with 2014. The report also showed more than 75% of all legitimate websites had unpatched vulnerabilities and 15% of legitimate websites had critical vulnerabilities. Some possible reasons for this situation are as follows. Firstly, it is a heavy and expensive workload for administrators to maintain and manage the efficiency and performance of the large enterprise networks. As a result, security checkups and updates are often delayed. Secondly, not all administrators have enough ongoing training on the latest security measures that can be utilized and deployed to enhance the network security. Thirdly, the existing architectures for typical networks are usually designed to ensure the reliability of processes for mission critical operations, but are less focused on the security aspect. Therefore, the configurations and settings of these networks usually remain unchanged for a considerably long time. While the static nature of the existing architectures for networks is beneficial to ensuring the high performance of business operations, it could leave networks a clean target for attacks and give attackers big advantages over security administrators considering the time and resources invested: it takes trivial efforts for attackers to discover and exploit such static networks because they can do it anytime and anywhere without being concerned about changes in the configuration of the targeted infrastructure. In order to change this situation and better defend networks, new defense strategies are indeed needed.

1.2 Moving Target Defense: A Game Changer

Moving Target Defense (MTD) is one of the game changing strategies proposed in the National Cyber Leap Year Summit in 2009[3]. MTD is based on the idea of security through diversification and its main focus is to dynamically and randomly change the configurations of a target, i.e., a host computer or a whole network. Such random changes in the configuration of systems increase uncertainty, complexity and unpredictability, making it computationally harder for attackers to exploit such a constantly changing environment.

This paper presents a comprehensive literature survey on Moving Target Defense (MTD) to date with a leveled approach based on how MTD is researched, evaluated, and implemented.

This survey paper is organized as follows. Section 2 presents our survey methodology and contributions. Section 3 provides the background information of MTD. Section 4 covers the cyber attacks that can be addressed by MTD strategies. Section 5 presents the classifications of MTD strategies. Section 6 provides an overview of the existing MTD analysis work. Section 7 sketches future research directions for MTD, and Section 8 concludes this survey paper.

[1]https://nmap.org, Nov. 2018.

[2]https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf, Nov. 2018.

[3]https://www.nitrd.gov/nitrdgroups/index.php?title=National_Cyber_Leap_Year_Summit_2009, Nov. 2018.

2 Survey Methodology

The official website of Department of Homeland Security defines the MTD approach as follows[4]:

"*Moving Target Defense (MTD) is the concept of controlling change across multiple system dimensions in order to increase uncertainty and apparent complexity for attackers, reduce their window of opportunity, and increase the costs of their probing and attack efforts. MTD enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency.*"

However, among all research articles related to moving target defense techniques we surveyed, none of the reviewed papers have provided a formal definition of moving target defense, including the three previous survey papers[4−6], but the characteristics of MTD are described in almost every paper.

By compiling the information from different sources such as [4], the National Cyber Leap Year Summit in 2009, and NITRD[5][6], we provide a formal definition of Moving Target Defense in this survey paper as follows.

A target refers to an entity or asset, such as an application, a computer, or a system that is exploitable by adversaries with malicious intentions. Moving Target Defense in this survey paper refers to the techniques that can change a target's properties or configurations randomly and regularly or the techniques that can increase a target's uncertainty and unpredictability, with the goal of enhancing the security defense of the target while maintaining the essential functionality of the target intact.

With this definition in mind, the authors initially collected around 100 published papers with relevant keywords and subjects, in addition to many online articles. Each paper was carefully reviewed and summarized. Then we finalized around 80 papers in this survey based on the publication date, relevance to MTD, and the publication venues. Most of the chosen papers were from the ACM and IEEE digital libraries ranging from the late 90s, when the basic idea of MTD was first introduced, to the time of the writing of this survey paper. The selected papers then were grouped into three categories with respect to the implementation approaches of the MTD techniques addressed in each paper: 1) the OS level approach, 2) the software/application level approach, and 3) the network level approach.

- The OS level approach includes research studies that aim to implement MTD at the operating system and the machine code level.
- The software/application level approach covers research efforts that aim to integrate MTD into software or application natively or through third-party tools before the software or application is released.
- The network level approach focuses on research findings that aim at introducing MTD into specific hosts on a network and even the entire network.

These three categories are not mutually exclusive. For instance, TALENT[7,8] uses a portable checkpoint compiler to compile the application for different architectures. This compiler technique can be categorized as the software/application level approach, but since multiple MTD techniques are integrated and orchestrated in TALENT to protect mission critical applications on a network, we can also consider TALENT as a network level approach of MTD. The authors believe the paper selection strategy resulted in a representative collection of papers reflecting the state-of-the-art of this line of research.

2.1 Architectural Perspective of MTD

Okhravi *et al.*[4] published a technical report for the Department of Defense on various moving target defense techniques. It provides a technical and qualitative summary of different moving target techniques, threat modeling, and their strengths along with their weaknesses. The work is very informative and useful, but it is less focused on quantitative and analytical aspects of the MTD line of research.

Published by Cai *et al.*[5], a collection of papers in moving target defense field is reviewed. The papers are categorized in three main topic areas: 1) MTD theory, 2) MTD strategy, and 3) MTD evaluation. The review lays a good foundation of MTDs but missing some necessary information. More precisely, some detailed information on the history of MTD, which explains the recent shift of MTD research from application-oriented to network-oriented, is missing from the research paper. It is also less focused on discussing the weaknesses of the

[4]https://www.dhs.gov/science-and-technology/csd-mtd, Nov. 2018.

[5]https://www.nitrd.gov/cybersecurity/documents/NITRDHST2010.pdf, Nov. 2018.

[6]https://www.nitrd.gov/cybersecurity/documents/NITRD_IEEE_SSP_2011.pdf, Nov. 2018.

surveyed MTD strategies. Finally, Software-Defined Networking (SDN) has become an important and integral part of MTDs, but it is only briefly mentioned in the survey paper published by Cai *et al.*[5]

A recently published survey by Lei *et al.*[6] reviews MTD research work from the perspectives of the theory and design of MTD, the key techniques in MTD, and the application of MTD. This review provides in-depth information of MTD and future research opportunities, but the application of MTD in their paper only focuses on the network level and is less emphasized on the coverage of the application of MTD on the lower level, such as ASLR and software diversification. Our survey paper, in addition to providing in-depth analysis and reviews on the low-level approaches to MTD, also discusses the strength and weakness of the surveyed MTD techniques and covers some new research work in the Internet of Things and other small embedded systems.

This survey paper provides an alternative view and comprehensive survey on MTD research that covers a wide range of aspects of MTD such as the MTD history, implementation, strength and weakness comparison, and an overview of existing MTD analysis. In comparison to the survey papers published by Cai *et al.*[5] and by Lei *et al.*[6], this work is more focused on architectural aspects and classifications of MTD strategies. This survey divides MTD strategies into three broad categories based on the architectural view, where a given MTD strategy aims to protect a system: 1) the OS level, 2) the software/application level, and 3) the network level. This type of structure of the survey not only describes the practical and implementation insights of the MTD research, but also shows a clear focus shift in the MTD research: from the low level (the OS level and the application level) to the high level (the network level). The architectural view of MTD techniques and the research focus shift revealed by our survey are major differences between ours and the previous survey papers[4–6]. Table 1 shows a road map of the survey.

3 Moving Target Defense: Background

Since the introduction of Moving Target Defense at the National Cyber Leap Year Summit in 2009, many researchers in many research articles have quickly adopted the term. The fundamental concepts of MTD can be traced in many other areas with a long history. For example, the "shell game" is a good illustration of the moving target concepts. In this game, an operator places a target such as a pea under one of the three identical face-down shells and shuffles them quickly for many times. When stopped, the player, who can correctly identify which shell contains the pea, wins. Since the target (the pea) is randomly moving, it is difficult for the player to identify it. This dynamic defense strat-

Table 1. Roadmap of the Survey

Topic	Classification	Subsections & References
Introduction (Section 1)	N/A	[1–3]
Methodology & Contribution (Section 2)	N/A	[4–8]
Background of MTD (Section 3)	N/A	[9–19]
Cyber Attacks that MTD Can Defend Against (Section 4)	N/A	4.1 Reconnaissance Attack 4.2 Code Injection Attacks 4.2.1 Buffer Overflow 4.2.2 SQL Injection[20] 4.2.3 Cross-Site Scripting (XSS)[21] 4.3 DDoS Attack[22,23] 4.4 Computer Worm Attack[18]
Key Strategies in MTD (Section 5)	OS level approaches	[24–31]
	Software level approaches	5.2.1 Software Diversification[32–43] 5.2.2 Software Diversification Through Middleware[20,33,36,44–46]
	Network level approaches	5.3.1 IP Address Randomization[47–55] 5.3.2 Virtualization-Based MTD[7,8,19,56,57] 5.3.3 Decoy-Based MTD[52,58] 5.3.4 Software-Defined Networking Based MTD[59–65] 5.3.5 Lightweight MTD[66–68]
Overview of Existing MTD Analysis Studies (Section 6)	N/A	[2, 8–10, 25–30, 40, 49, 50, 69–80]

egy was gradually adapted into the computer security as computer administrators start to recognize the fact that perfect security is unattainable and start to shift their focus on building alternative defensible systems rather than perfectly securing systems. Randomness and diversity are two essential components in the dynamic defense strategy and many defense research studies involve these two components.

In 2001, the PaX team introduced a technique called Address Space Layout Randomization (ASLR)[7] for the Linux kernel to defend memory corruption attacks by randomizing the memory address locations of key data areas of a process. The implementation of ASLR was effective and other major operating systems, such as Windows, OS X and Solaris, later implemented ASLR on their own platforms.

In 2003, the Instruction Set Randomization (ISR) technique was proposed independently[9,10] with different designs and implementation and presented in the 10th ACM Conference on Computer and Communication Security. Unlike ASLR, ISR works at the machine code level and aims to prevent code injection attacks by randomizing the instruction set before sending it to the processor for execution. The vulnerability of application/software is a major security threat not only to the computer where the application is installed but also to the network to which the computer is connected. In 2004, Just and Cornwell[11] and Stamp[12] pointed out that monoculture enabled attackers to quickly attack application vulnerabilities in a large scale and they proposed to introduce diversity in software to mitigate vulnerabilities. Many research studies confirm that diversity in software is effective[13−17].

As networking has become an essential part of every company's infrastructure, network security has attracted researchers' attention[18,19]. In 2009, the National Cyber Leap Year Summit was held, and Moving Target Defense idea was introduced by researchers as a new defense strategy, and many research studies have emerged ever since. For instance, Springer published two books on Moving Target Defense and ACM started the first Moving Target Defense Workshop in 2014.

MTDs are not meant to replace the current security defense strategies, such as the reduction of attack surfaces, but to add another layer of security defense to assist security administrators in employing a dynamic approach for protecting the operational infrastructure. The ultimate goal of MTD is to employ various techniques and approaches to dynamically and randomly change the configuration of the system in an unpredictable manner. This will help in increasing the uncertainty and unpredictability of the system behavior and accordingly results in a constantly changing attack surface leaving malicious entities puzzled with the targeted system. MTD not only can confuse attackers and increase their burden to launch attacks, but also can thwart some common cyber attacks pertinent to this type of dynamic security strategy.

4 Cyber Attacks Targeted with MTD Strategies

The randomness, diversity, unpredictability, and uncertainty introduced into a system by MTD techniques can effectively defend against a certain range of cyber attacks. This section gives an overview of four types of popular cyber attacks and the MTD strategies against them.

4.1 Reconnaissance Attacks

Rather than being a true attack, reconnaissance is technically the initial phase of almost all attacks. During the reconnaissance phase, attackers utilize various techniques and automated tools to probe and scan characteristics of a certain target with the aim of gathering important information about the target such as OS types, running services and protocols, and open ports, for potential vulnerability exploitation and attacks. A system with an enabled MTD strategy is effective against a reconnaissance attack because the random changes in the system orchestrated by the MTD strategy will invalidate the information that the attackers previously obtained.

4.2 Code Injection Attacks

A code injection attack happens when attackers take advantage of vulnerabilities in a computer program or the underlying system to inject and execute a piece of malicious code. A carefully crafted code injection attack can even grant the attacker with the administrative privilege to control the whole system. There are different techniques that attackers use to achieve to inject the malicious code, and some well-known techniques are listed below.

[7]https://pax.grsecurity.net/docs/aslr.txt, Nov. 2018.

4.2.1 Buffer Overflow

Buffer overflow is a vulnerability in computer programs or the underlying languages that allows data to be written into the buffer in the memory without checking the boundary of the buffer. Attackers can use carefully-crafted data to overrun the buffer boundary to inject the malicious code past the end of the buffer, resulting in a buffer overflow attack. Address Space Layout Randomization (ASLR) is an effective MTD technique against buffer overflow and some other code injection attacks that rely on the address location of the memory layout. Instruction Set Randomization (ISR)[9,10] is another MTD technique that has been claimed to be effective against any code injection attack including buffer overflow attacks. ASLR and ISR will be discussed more in Subsection 5.1.

4.2.2 SQL Injection

The SQL injection is a type of code injection targeting vulnerable data-driven applications. When an application fails to filter or sanitize the user inputs before sending it to the database engine for execution, the attacker can exploit this vulnerability to inject malicious SQL queries into the user inputs to gain the access to the protected data and even play the role of the administrator of the database server.

SQLrand[20] is introduced as an MTD technique to prevent the SQL injection attack at the language level and will be discussed more in Subsection 5.2.2.

4.2.3 Cross-Site Scripting (XSS)

A cross-site scripting attack is another type of code (client-side script) injection, in which malicious scripts are injected into vulnerable websites viewed by other users. When a user clicks a script-injected URL or visits a web page infected by a malicious script, the malicious script will be executed on the user's computer without the knowledge of the user. To protect users from XSS attacks, all major web browsers include whitelist-based protection feature. On the other hand, Portner *et al.*[21] believed that utilization and maintenance of such whitelisting was inflexible and therefore they proposed a new approach to defend against XSS attacks based on the principles of moving target defense. This approach will be discussed more in Subsection 5.2.2.

4.3 DDoS Attack

A Distributed Denial of Service (DDoS) attack usually targets certain hosts on a typical network via two steps to launch: 1) attackers compromise and take control of a myriad of computers on different networks; 2) attackers organize the compromised computers in such a way that they simultaneously send large volume of traffic messages to flood the target host and eventually force it to shut down or deny all service requests from legitimate users. As proposed by several researchers[22,23], MTD is an effective technique for defending and mitigating DDoS attacks. More descriptions will be presented and discussed in Subsection 5.3.1.

4.4 Computer Worm Attack

A computer worm is a stand-alone computer program created with malicious purposes. These malicious programs can cause various damages such as consuming excessive network bandwidth, stealing information from infected computers, and taking control of the infected computers for launching some other attacks. Randomly changing hosts' IP addresses based on the MTD principle can be an effective way to prevent the spread of the computer worms, as stated in [18] which will be discussed more in Section 5.3.1.

5 Key Strategies in Moving Target Defense

This survey divides MTD strategies into three broad categories with respect to the architectural perspective: 1) the operating system level, 2) the software level, and 3) the network level.

5.1 Operating System Level Approaches

Some programming languages, e.g., C and C++, offer low-level memory manipulation but do not provide built-in libraries and packages specifically for protection against buffer boundaries violations. Hence, programs written in these languages are often vulnerable to buffer overflow attacks if no explicit boundary checking is implemented in the given program. Moreover, once programs are loaded into the main memory, an attacker can easily target and inject malicious code into those memory locations.

In the past, the prevention of buffer overflow attacks mainly relied on the implementation of buffer bounds checking by programmers. Due to human errors, however, buffer overflow problems continued to exist in many released software applications. As a matter of fact, this type of vulnerability had been a commonly exploited vulnerability until the introduction of

Address Space Layout Randomization (ASLR) in major operating systems.

Address Space Layout Randomization (ASLR) is a moving target defense strategy that prevents the exploitation of various memory corruption vulnerabilities, such as buffer overflow attacks, by randomizing the memory address locations of key data areas of a process. As a result, it will be extremely difficult, if not impossible, for attackers to predict the memory address layout of the target's loaded program for launching buffer bounds attack through code injection. ASLR was first implemented and released for the Linux PaX project in July 2001 and was later implemented in other major operating systems such as Windows, Mac OS, and Solaris. The effectiveness of ASLR depends largely on the performance of the randomization algorithm, which is usually determined by the number of memory addresses available for randomization procedure. It has been proven that ASLR is less effective when implemented in computers whose architecture is 32-bit[24].

The number of possible settings of a randomized memory layout can be computed by the following equation:

$$N_c = 2^n,$$

where n is the number of address bits available for randomization. For 32-bit computers, only 16 of 32 address bits can be utilized for randomization, yielding up to $2^{16} = 65\,536$ different settings for the randomized memory layout. This is a relatively small number and a simple brute force attack with 65 536 linear probes can find out about all combinations within a short time. For 64-bit architecture, however, there are at least 40 address bits available for randomization which yields at least $2^{40} = 1\,099\,511\,627\,776$ combinations. Even though it is possible to figure out all the combinations with a brute force attack, the magnitude of the attack will be large, and the attack will be quickly detected and appropriate mitigation strategy can be adopted prior to the attack being successful.

Besides being vulnerable to brute force attacks on a 32-bit system, ASLR can also be bypassed by using memory disclosure technique[8][9] on vulnerable applications on a system[25−27]. To prevent memory disclosure issues, some schemes have been proposed by randomizing the data and code segments of each application[28,29]. However, Snow *et al.*[30] suggested that these schemes may not be as promising as expected and call for research on developing more comprehensive defense techniques.

It is worth noting that to further protect the operating system from code injection attacks, a technique called non-executable memory protection was proposed and adopted by major OS vendors[10][11]. The non-executable memory protection in an operating system marks all writable regions of memory as non-executable and it can prevent any injected code from being executed. All current major operating systems have implemented ASLR and non-executable memory protection mechanism to provide protection against code injection attacks.

In addition to buffer overflow attacks that target the memory layouts, some other types of attacks inject code to the low-level machine code, i.e., the instruction set. This type of code injection usually does not necessarily cause programs or the underlying operating system to crash. Therefore, these types of attacks are harder to detect. To defend against this type of attacks, a technique called "Instruction Set Randomization (ISR)" was proposed independently by University of New Mexico research group and Columbia University research group, which uses a key-based XOR operation to randomize the instruction set before it is passed to the processor[9,10]. The processor will then use the same key to reconstruct the instruction set for execution. Without having knowledge about the key, any code injected after the randomization will not be executed by the processor because the processor will fail to reconstruct the instruction set. Since the ISR technique works at the low level, it is transparent to applications, languages, and compilers. The implementation of ISR in practice, however, is limited mainly due to two major drawbacks as noted by the Columbia group[10]: 1) the performance overhead is high, and it needs hardware support to be practical; 2) ISR requires applications to be statically linked to work efficiently. Later, Hu *et al.*[31] proposed a new implementation of ISR by utilizing the Software Dynamic Translation (SDT) and Advanced Encryption Standard (AES). SDT provides a virtual execution environment that is responsible for loading, encrypting applications, decrypting ap-

[8] https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf, Nov. 2018.

[9] https://www.blackhat.com/presentations/bh-usa-08/Sotirov_Dowd/bh08-sotirov-dowd.pdf, Nov. 2018.

[10] https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457155(v=technet.10, Nov. 2018.

[11] http://pax.grsecurity.net/docs/noexec.txt, Nov. 2018.

plication instructions, and verifying the decrypted instructions in preparation for execution. AES is used to replace XOR operations in the previous ISR research work to increase the security. Their experiment showed that the performance results and the increased security make their implementation a viable approach for protecting mission critical server applications, even without special hardware support.

The Columbia University research group[10] stated that the principle of ISR could be applied to protect programming languages such as Perl and SQL[20,27].

5.2 Software/Application Level Approaches

Both ASLR and ISR can significantly prevent low-level code injection attacks on computer systems. On the other hand, attackers have found other ways to execute their malicious code on the target system. The computer worms Conficker[32] and Stuxnet[12] are two instances that demonstrate how attackers have taken advantages of vulnerabilities at the application level, where malicious programs are attached to certain running processes. This type of code "injection" is difficult to detect because the code can stay dormant until a certain process executes or an event occurs. As a remedy, [33] proposes to adopt the ISR technique across all software layers and requires all programs be randomized during the program installation process using various keys. A special run-time environment is however required to translate the randomized programs into valid executable ones before execution. As a result, any program that fails the translation process will not be executed.

The static code analysis tools, either open source such as IntelliJ or Eclipse or proprietary such as Fortify or AppScan[13]–[16], are widely used to detect and fix program bugs and potential vulnerabilities during the early stages of the application development. These tools can be launched to parse a given source code at any time without executing the application. While these tools can detect many potential vulnerabilities such as buffer overflow and SQL injection, they cannot detect semantic errors in the logic of the program. On the other hand, the dynamic program analysis tools execute an application at runtime by using specialized tools and thus are capable of detecting semantic problems such as logic errors, authentication problems, and authorization issues. These dynamic analysis tools, however, require developing effective test cases, which may indicate the high cost associated with this approach, especially when the program under test is large and complex. Ironically, malicious entities also utilize similar static and dynamic analysis tools to discover potential vulnerabilities in an application, assuming having access to the source code of the application, i.e., open source software.

Software producer companies usually employ different secure programming techniques such as obfuscation, wrapper, and encryption to protect their source code from being reverse engineered. However, professional hackers always find a way to circumvent these techniques and reverse engineer the source code. Once vulnerabilities are discovered, they will be used to launch attacks. This situation is attributed to software "monoculture"[11,12], in which the binary code for widely used software is identical and runs on a large scale of computers, where the vulnerability found in one copy of the software can be used to attack all the other copies of the software. To address the monoculture issue and its security consequence, researchers[13−17] proposed to use software "diversification" techniques for defense purposes.

5.2.1 Software Diversification

Software diversification was first introduced as a means for increasing the reliability and fault-tolerance of software through N-version programming approaches[34]. Through N-version programming, the software team generates N semantically equivalent programs from the same initial specifications. The approach greatly reduces the probability of identical software faults occurring in two or more versions of the program. In an analogous way, in security context, software diversity works in such a way that multiple, semantically equivalent versions of the software are generated. As a result, if attackers breach one of the versions, the knowledge gained from the damaged version would not be applicable for the other versions of the

[12] https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, Nov. 2018.

[13] https://www.jetbrains.com/idea, Nov. 2018.

[14] https://www.eclipse.org, Nov. 2018.

[15] https://software.microfocus.com/en-us/products/static-code-analysis-sast/overview, Nov. 2018.

[16] https://www.ibm.com/security/application-security/appscan, Nov. 2018.

software. Therefore, this approach reduces the success rate of the attack.

There exist some other techniques for diversifying software applications. For example, adding a nonfunctional piece of code, called no-op or NOP, randomly into the compiled code may disrupt code injection attacks. The technique diversifies the memory layout of a running application using techniques such as randomly padding stack frames, randomizing the locations global variables, and newly allocated stack frames[13].

Modern compilers are very intelligent and utilize several optimization techniques to improve performance. The optimization techniques employed are a great source of variations for a given program. Super optimization is a compiler technique introduced by Massalin[35] to minimize the size of compiled code using a brute-force search. This type of optimization finds the shortest instruction sequences for a desired function implementation. Jacob *et al.*[17] extended the research of super optimizer to create a super diversifier toolkit to customize programs at the machine-code level. By using a secret key, user-defined parameters, and empirical data, the super diversifier guides the brute-force search procedure to generate an individualized copy of the compiled code. The research results indicate that the user diversification is an effective way to defend against signature-based attacks, in which malware or viruses look for particular byte patterns in the executable parts of the applications[17]. A direct benefit of code individualization is the obfuscation of a given binary code, which is typically used to prevent reverse engineering. Reverse engineering in computer world refers to the process of extracting design information from a software product. It is beneficial and useful for many occasions, but attackers may use this technique to gain the insight knowledge of the targeted software with the hope of finding vulnerabilities. With program individualization, each copy of the program is unique, and the launched reverse-engineering would only work on the local copy of the program that the attacker owns. Thus, the knowledge the malicious entities gain would not be beneficial to attack other copies of the program. However, recent research shows that a diversified program can still be compromised by taking advantage of memory corruptions[27] or stack vulnerability[25].

Based on the ideas of super diversifier, Jackson *et al.*[36] proposed another compiler-based diversification technique. Instead of substituting and pruning the existing instructions, the proposed technique randomly inserts non-alignment NOP instructions to create diversity. A major advantage of this approach is its ability to generate a large number of software variants, resulting in high unpredictability, which can greatly increase the cost for malicious entities to launch attacks against a system. Instead of diversifying the low-level code, Cabutto *et al.*[37] presented a high-level technique called "code mobility". In code mobility, some parts of the binary code blocks are removed from the software before deployment and stored on a remote and trusted server, and when the software is executed on the client computer, the missing binary code blocks will be fetched from the trusted server and injected into the running process' memory for execution. Code mobility aims to protect software from reverse engineering, but Cabutto *et al.*[37] claimed that it can be combined with any software diversification technique to improve software protection.

Franz[38] argued that it is time to apply compiler-generated diversity mechanism to a massive scale and proposed a design idea for massive-scale software diversity (MSSD) and delivery system based on the architecture of online App Store. The core component in the design is a software diversification engine, which compiles the software in a Just-In-Time (JIT) mode to generate a unique version of the software each time when it is downloaded by a client, resulting in a large number of variants of the software. This scale of diversification will make vulnerability exploitation difficult and will increase the cost to the attacker dramatically. Jackson *et al.*[39] extended Franz's research[38] and introduced another design idea called Multi-Variant Execution Environment (MVEE) that can detect the exploitation of vulnerabilities in a program at the runtime. To protect a program, MVEE stores all variants of the program and their normal behavior as a baseline, and then any input to the program will be fed to all variants and be monitored. A malicious input will cause a divergence and MVEE will detect it and alert the user to take actions. MVEE can bring extra security to organizations, but at a cost of computational overhead and performance loss. Therefore, Jackson *et al.*[39] recommended MVEE be implemented in organizations that can trade off performance for extra security.

Challenges in Software Diversification. Despite all the key advantages of various software diversification techniques in security defense, there are three major challenges that need to be addressed prior to the prevalently adapting of software diversification.

1) *Cost of Generating Code Variants.* In order to be effective, a software diversification technique must

generate a sufficient number of variants to ensure high unpredictability, which can be measured by entropy $H(D)$[40]. For a given moving target mechanism, D stands for the dynamic portion of the attack surface and $H(D)$ is the entropy of D defined by:

$$H(X) = -\sum_i p(x_i) \log_b(p(x_i)),$$

where $p(x_i)$ is the probability of the i-th instance of the moving target. If $H(D) \gg 0$, then the technique meets the unpredictability requirement. However, it is computationally expensive to generate all the variants, even for an even small program. In practice, larger programs are more complex and would require a tremendous number of variants. How to efficiently generate the sufficient number of variants with low cost is a daunting challenge. The design of MSSD[38] can generate variants in a massive scale to meet the unpredictability requirement of MTD defense, but the cost can be high. Franz[38] recommended MSSD be implemented in a cloud computing environment for performance and affordability, and argues that the cost associated with the cloud computing can be absorbed either by the software vendor or by the user who downloads the software.

2) *Delivery of Software Variants.* Another challenge in performing software diversification at machine-code level is the distribution of the large number of variants to clients efficiently. For instance, it is a major challenge to deliver $2^{1\,000}$ versions of the software to clients. The physical shipment of the large number of software variants is impractical if not impossible. Moreover, digital downloading is less helpful because it would take up huge amounts of space to store all the software variants on the server.

MSSD[38] can address the challenge in the delivery of software variants because each variant is generated on demand whenever a client requests the software. The software is downloaded to the client without the needs for storing the variant on the server. However, as mentioned above, the cost in MSSD can be high when a large number of variants are generated and downloaded; therefore, more research needs to be conducted on the costs versus variants scale and delivery.

3) *Software Updates and Patches.* Software updates and patches are usually applied for fixing discovered vulnerabilities or bugs introduced through new features in the software application or fixing existing or hidden security problems. The common practice among software companies, such as Microsoft, Google and Apple, is to use a technique, called delta update, to achieve fast delivery and installation of the software updates and patches. A delta update file contains only the code that has changed since the last update of the software, avoiding the update of the entire application. Users need only download and install a small patch file to update the existing software. The delta approach to updating applications turns out to be challenging as each copy of the software running on the client's computer is diversified and unique. This is due to the fact that the delta updating procedure is not aware of the exact location in the code to start the software update process. A straightforward solution to this challenge could be that each client would be required to download a new diversified version of the updated software. However, this is apparently not efficient even for delivering a small software update or patch. Another solution is to use a random key to identify each version that is generated, but it will require a complicated tracking process besides the diversification and delivery process. This will open many future research questions on performance, cost, and security of the complex system.

4) *Integrity and Verification of Software Variants.* When an application is distributed over the Internet, the software company must ensure clients that the downloaded software can be trusted, i.e., it contains no flaws due to transmission failure, or is not being tampered by malicious entities. This process is called Software Integrity Verification[41,42]. For example, a software company may apply a hash function, such as MD5 or SHA1, to generate a unique digital signature from its software and distribute it along with the software over the Internet. Then the client can use the same hash function to generate a signature from the downloaded software and compare it with the distributed signature. If the two signatures match, then the downloaded software can be trusted; otherwise, the integrity of the downloaded software cannot be verified or trusted, and the client should not use the software. When software is diversified, however, each variant is unique and will result in a unique signature for integrity verification. For software with a large number of variants, the integrity verification process could become cumbersome. The efficiency and scalability of the integrity verification of diversified software has not been fully investigated.

5) *Validation of Software Variants.* Software testing is an important phase in the software lifecycle in which software applications are tested as thoroughly as possible by using various rigorous methods, such as functional testing and structural testing, before the software products are released[43]. This important task could be

very challenging when testing diversified software, due to the randomness and unpredictability introduced by software diversification on functional or structural level.

5.2.2 *Software Diversification Through Middleware*

Software diversification can be achieved by using a middleware, which is a special program or an execution environment, installed on the host computer, to provide the diversity for other applications running on the same host. The idea of delegating the diversification process to middleware applications is useful especially when some vulnerabilities are discovered by malicious entities, but the software developers have released no security patches. The middleware program can diversify the vulnerable software and prevent it from being exploited by the attackers.

As discussed in Subsections 5.1, ALSR, non-executable memory protection mechanism, and ISR are efficient for defending against code injection attacks. However, attackers have already developed a new exploitation called "Return-Oriented Programming" (ROP) to bypass those protections[44]. In ROP-based exploitations, attackers do not inject any code into the system memory, instead they carefully exploit specific vulnerable programs. A successful exploit will enable them to link together a number of machine instruction sequences in the memory, called "gadgets", for execution. Using the gadgets, attackers perform arbitrary operations on a target computer even when the computer has implemented the non-executable memory protection.

Pappas *et al.*[45] proposed a technique called "in-place code randomization", which can be used directly on executable versions of third-party software applications to defend against the ROP exploitation. In-place code randomization is a code transformation technique and uses a statistical model to determine which piece of code can be safely extracted from the compiled binaries, and the transformation is the key to defeat ROP exploitation because the success of ROP exploitation relies on the successful execution of all linked gadgets. Any modified gadget can break the link causing the exploitation to fail.

The experimental results[45] show that in-place code randomization is a practical and effective way to thwart ROP attacks directly on the third-party software, e.g., Adobe Flash Player. The major limitation of this technique, however, is that it cannot be used on software with self-checksum feature because any code injection can cause a checksum to fail.

Inspired by the ISR technique (see Subsection 5.1), Boyd and Keromytis[20] implemented ISR on SQL to defend against SQL injection attacks. The implementation is called SQLrand (SQL randomization). It requires a middleware on the web server to randomize standard operators such as keywords and commands in SQL before accepting user input. On the database end, it requires a proxy to de-randomize the query returned from the web server, and then passes the query to database engine for execution. Any user input containing malicious code will not be derandomized and thus will be dismissed from the execution.

In a book chapter, Portokalidis and Keromytis[33] proposed that the ISR technique can be adopted globally across all layers of the software stack. The basic principle is to randomize keywords, operators and function calls in the language before accepting user input, and then de-randomize the instructions before execution. Two languages are used to demonstrate the process: Perl and SQL, and their results show that ISR is versatile and the implementation is successful with low overhead.

Another approach to software diversification via middleware is to use middleware software components to monitor the target program and offer defense services when needed. Although the target program might not be able to be diversified natively for different reasons, the middleware can be easily diversified in different ways. The combination of the target program and the diversified middleware creates a diversified entity with a better defense strategy. The aforementioned MVEE[36] is an example of using middleware to monitor and protect the target program.

Inspired by a common biological phenomenon, called the symbiotic mutualistic relationship, in which two organisms live together to provide mutual defense against predators, Cui and Stolfo[46] proposed a host-based defense mechanism called "Symbiotic Embedded Machines" (SEM). SEM is a code structure injected into the target program (host) to provide monitoring and defense for the host program. Unlike anti-virus or anti-spyware programs, SEMs are not installed on the target computer, instead each SEM is infused into the host software and yet each is self-contained, self-fortified, and is executed along-side the host software. The code of SEM can be randomized before injecting into the host software and the combination of SEM and the host software creates a unique executable, which makes it much harder for attackers to exploit the system with the knowledge gained from the executable.

The principle of SEM can be used to develop applications, e.g., host-based IDS and rootkit detection. More research and data are needed to evaluate its feasibility and performance.

Challenges in Software Diversification via Middleware. The use of middleware software in this type of diversification techniques requires a special execution platform and it might impose significant performance overhead on the target program and even the entire system. This could be very challenging to implement the middleware at a large scale (i.e., enterprise level) or in an environment where the program performance is mission-critical and thus cannot be sacrificed.

5.3 Network Level Approaches

The network stability and performance are two utmost concerns for enterprises. The network stability refers to the requirement that the network must be able to continue to function even when some hosts on the network become nonoperational for any reason. For network performance, the network must be able to transmit data efficiently and correctly between hosts. These requirements have led to the current conventional network architecture, which is distributed (for stability) and static (for performance). For example, the Internet Protocol (IP)[17], a core protocol in the current Internet architecture, is designed for efficient data transmission between hosts on network that requires an IP address to be assigned to each host. The IP addresses of the source and the destination hosts are critical to the data transmission between hosts. Unfortunately, attackers can utilize this protocol to exploit the target hosts.

Once a network is configured and is operational, the configurations of the network and even hosts usually remain unchanged until further changes need to be implemented. The static configuration of a network and its hosts is important to stabilize the network in a distributed environment. However, it makes the implementation of MTD-based techniques very challenging.

There are many research studies that aim at employing MTD to protect specific hosts on a network, or even the entire network system. This subsection will cover these techniques in four subcategories.

• *IP Address Randomization.* The IP address randomization includes MTD strategies via the randomization of IP addresses, port numbers, and network protocols.

• *Virtualization-Based MTD.* The virtualization-based MTD includes research that utilizes the virtualization technology to create an MTD defense system.

• *Decoy-Based MTD.* The decoy-based MTD section, as the name suggests, includes researches that introduce decoys into a network system as an MTD defense strategy.

• *Software-Defined Networking Based MTD.* The Software-Defined Networking based MTD subsection includes researches that take advantage of SDN technology to provide MTD defense.

• *Lightweight MTD.* This subsection includes the research studies that focus on the use of MTDs on small embedded systems (i.e., wireless sensor networks) and Internet of Things devices.

These subcategories are not mutually exclusive, and some research studies might be a part of multiple subcategories based on the technologies they use. In these cases, we classify the research based on the overall goal of the work and the technology used. For instance, Zhuang *et al.*[47] combined virtualization and IP address randomization in their research. However, we include the research in the IP address randomization subsection because the paper's main focus is to investigate the MTD performance vs the adaption interval and the adaption includes the IP address randomization.

5.3.1 *IP Address Randomization*

The IP address randomization, also called IP address mutation, is a basic MTD technique that has been discussed in several lines of research[18,47–53]. A typical network attack usually starts off with a reconnaissance phase during which the attacker uses various scanning and probing tools (e.g., Nmap) combined with carefully-crafted packets to exploit the target host. The reconnaissance attack is usually an automated and scripted exploitation on a list of predefined IP addresses. Therefore, an intuitive way to countermeasure the attack is to change the host's IP address frequently in order to invalidate the information previously exposed to the attacker. This is the deriving idea behind the IP address randomization technique.

The IP address randomization technique randomly picks a new IP address from a pool of available IP addresses and assigns it to the target host. If the IP address change happens between the reconnaissance phase and the actual attack phase, then it thwarts the attack. Otherwise, the attack will be successful. Fig.1 demonstrates the IP address randomization defense model[53].

[17]https://tools.ietf.org/html/rfc791, Nov. 2018.

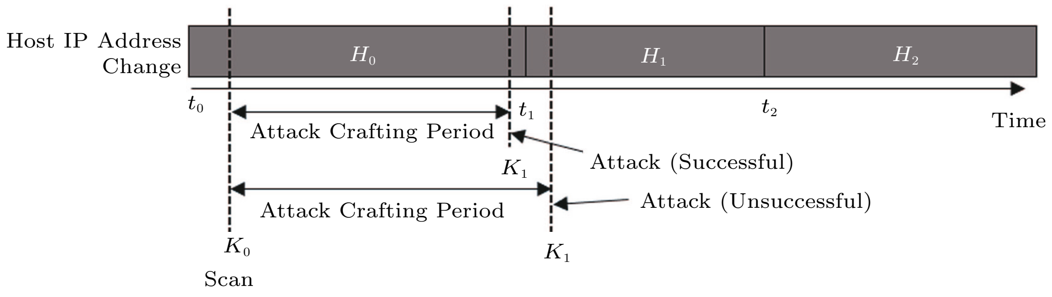


Fig.1. IP address randomization defense model.

In this model, the period (t_i, t_j) is called the IP address retention period, during which the host holds the same IP address until it changes. The attack crafting period (K_m, K_n) refers to the period between the completion of a malicious scan and the launch of an attack. Then the outcome of an attack can be expressed as in (1):

$$O_{(i,j)} = \begin{cases} 1 \quad \text{(successful)}, & \text{if } t_i \leqslant K_m < K_n \leqslant t_j, \\ 0 \quad \text{(unsuccessful)}, & \text{otherwise.} \end{cases} \tag{1}$$

The IP address randomization technique is effective for defending against reconnaissance or scanning attacks. Antonatos *et al.*[18] used IP address randomization technique in their research. They called their technique the "Network Space Address Randomization" (NSAR) to defend against the hit-list worm attack. The hit-list worms are specialized computer worms that are spread to target computers based on predetermined list of IP addresses of vulnerable targets. The technique uses a customized version of Internet Systems Consortium (ISC) open source DHCP server[18], which expires the IP allocations of all clients and then reassigns each client a new IP address. The network address change can invalidate partial or even complete list of IP addresses in the hit-list worm, and hence can slow down the spread of the worm. The preliminary simulation of this research on 1 000 000 vulnerable clients shows that NASR can increase the time of infecting 90% of clients from five minutes without NASR to between 24 and 32 minutes, and can effectively contain the worm infection rate under 15% of the vulnerable clients.

An important feature in the current network architecture is the efficient communication among hosts on the network. To ensure a reliable network, network protocols require hosts to exchange important information such as connectivity status, running services, open ports, and OS types. The information enables the network administrator to objectively monitor the network activities, locate and patch vulnerable hosts, and manage some other aspects of the network. Ironically, attackers also take advantages of this feature to launch "fingerprinting" attack against a network. A fingerprinting attack is an exploitation in which an attacker sends various well-crafted requests to a target host and obtains important information of the network with respect to the responses received from the target.

Al-Shaer[48] proposed an MTD-based architecture, called Mutable Networks (MUTE), to defend against network reconnaissance, i.e., scanning and fingerprinting. The MUTE architecture can periodically create alternative random configurations (called mutation consisting of host's location and other identity information) and apply them to the network without disrupting the normal network operations and services. The periodic random mutation in the network invalidates and deceives the adversary's fingerprinting results. Two techniques are proposed in the MUTE architecture's implementation.

1) *Random Host Hopping.* This technique is the same with the IP address randomization technique as described earlier.

2) *Random Finger Printing.* Host responses are intercepted and randomly modified to contain false fingerprinting information to deceive adversaries.

The randomization of host responses can be achieved through two mechanisms. One is to intercept and modify the session control messages (TCP 3-way handshake) to include false information. The other is to utilize the network firewall to generate positive responses for all denied packets to deceive scanners. Al-Shaer[48] shed some lights on some research challenges on the MUTE architecture, such as fast and unpredictable creation of mutation, deployability, and scalability.

[18] http://www.isc.org/downloads/dhcp, Nov. 2018.

An MTD technique based on IP address randomization requires a large number of unallocated IP addresses to ensure the newly assigned IP address is truly random and unpredictable. This means that any IPv4 (Internet Protocol version 4) network cannot meet this requirement to implement effective IP address randomization MTD defense. The problem is rooted in the fact that the current worldwide use has already exhausted the available IPv4 addresses. Because of this limitation, Network Address Translation (NAT) is commonly used to expand the IP address space. Some research also uses the combination of IP address, port number and running service[47] or uses virtual IP addresses[50] to improve the randomization process. However, the more feasible and efficient solution is to switch to the new Internet Protocol version 6 (IPv6) which can produce over 3.4×10^{38} possible IPv6 addresses.

The adoption of IPv6, however, introduces new security challenges. For example, instead of relying on a DHCP server to assign IP addresses to hosts, IPv6 protocol allows an IPv6 host to self-configure its unique IPv6 address automatically. This method is called Stateless Address Auto-Configuration (SLAAC). SLAAC can reduce network administrative cost because there is no need to use any DHCP server or NAT (Network Address Translation) service[19]. In SLAAC, the IEEE Extended Unique Identifier (EUI-64)[20] is commonly used to generate the IPv6 address. The address, however, contains the host's MAC address, which can be tracked easily. The exposition of MAC address is especially serious for networks with strict requirements such as military networks and Smart Grids.

Dunlup *et al.*[49] proposed a new strategy called MT6D that utilizes MTD in IPv6-based networks to enhance the security and privacy between trusted hosts. The architecture of any MT6D host consists of an encapsulator, decapsulator, and a Shared Routing Table (SRT). The encapsulator is responsible for transmitting all outbound packets, the decapsulator is responsible for receiving all inbound packets, and SRT stores the addresses of a sender host and its trusted receiver hosts, and the shared symmetric key for each sender/receiver pair. Upon receiving an outbound packet, the encapsulator will obscure the source and destination addresses and other private information in the packet by using the shared symmetric key, and then append it to a new IPv6 header to create a new IPv6 packet which will be transmitted to the receiver host. Upon receiving an inbound packet, the decapsulator will fetch the shared symmetric key and the original addresses in the SRT to reconstruct the data packet, and then deliver it to the host. If a packet is sent to an untrusted receiver, or is received from an untrusted sender, then the encapsulator or the decapsulator will immediately forward the packet to the nearest gateway device for processing. A proof of concept prototype software implementation of the MT6D was provided in [49] and the results showed that the MT6D concept is valid and feasible.

An enhanced implementation of MT6D was later adopted by Groat *et al.*[51] to protect Smart Grid. The existing current electric grid is a network of different power equipment, such as transmission lines, transformers and substations, which delivers electricity from a power plant to the home or business consumers. The first electric grid was built in 1890s and since then the technology has been improved. The increasing complexity and consumption of electricity today have imposed challenges on various aspects of electric grids such as efficiency, reliability and security.

Smart Grid[21] is a new and innovative design of electric grids to address the challenges and to provide reliable and efficient electric energy. By incorporating system controls, computers, and automation technology, Smart Grid enables a two-way interactive communication mechanism between its provider and the consumers. The communication provides its consumers with more visibility and controls over their electric usage. In addition, it enables the electricity provider to efficiently adapt to changes that occurred during the electric demands and to respond quickly and strategically to an electricity disruption.

Groat *et al.*[51] suggested using IPv6 for the Smart Grid communications because by adopting the IP-based communications, Smart Grid can take advantage of the existing and mature Internet technologies and infrastructure. To address the aforementioned security concerns in IPv6, Dunlop *et al.* also proposed to adopt MT6D[49] in Smart Grid to defend against targeted network attack by providing anonymity to consumers' identity. The simulation results[51] show that MT6D is a viable solution for the security defense in the Smart Grid. The major drawback of using MT6D is its overhead associated with latency and data encapsulation and de-capsulation. Moreover, MT6D cannot operate

[19] https://tools.ietf.org/html/rfc4291.html, Nov. 2018.

[20] https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf, Nov. 2018.

[21] https://www.smartgrid.gov/the_smart_grid/index.html, Nov. 2018.

in a network that is managed by a DHCPv6 server. These limitations may prevent the MT6D design from being widely adopted.

Zhuang *et al.*[47] presented a research that combined virtualization and IP address randomization to establish an intelligent moving target defense system. In this system, every mission-critical service, called "role", such as database and Web servers, is executed on a unique virtual machine (VM). Each VM is a dedicated resource component installed on a host, called "resource". The configuration of the network at a moment is called adaption in this design and each adaption consists of attributes such as role, VM ID, host ID and its IP address. A Configuration Manager stores all adaptions to create a resource map and use it to control the packet flow from one resource to another. At random intervals, an Adaption Engine produces a new adaption and passes it to the Configuration Manager to replace the old adaption by matching the role name. If a role was compromised, then the attacker could use it to compromise some other roles and eventually to reach their target role. However, if a new random adaption was generated in time during the attack path, then the attack would be thwarted and the target role would be protected. Therefore, the frequency of the adaption generation can impact how well the defense works. The effectiveness of this approach was confirmed by the simulation results: as the adaption interval decreased (high frequency), the effect of the MTD defense increased[47]. An Analysis Engine was also introduced to analyze vulnerabilities and attack activities and the analysis data can help the Adaption Engine to make more intelligent decision on when to create a new adaption, instead of at a random interval.

Jia *et al.*[22] proposed a DDoS defense mechanism called MOTAG by using IP address randomization. In MOTAG, any protected server is behind a layer of proxy server whose IP addresses are only known to authenticated clients. All traffic between the client and the server is relayed by a proxy node. When a proxy node is under attack, it will be replaced by a new proxy node with a different IP address and its clients will be shuffled among proxies to set up new associations. Their simulation test in MATLAB shows that MOTAG can effectively defend brute-force DDoS attacks.

In MOTAG, the IP address of a proxy server does not change until it is under attack. This reactive nature of MOTAG makes it vulnerable to proxy harvesting attack, as Venkatesan *et al.* later argued[23]. Proxy harvesting attack is a reconnaissance attack in which a malicious insider client sends a large number of requests to the authentication server and collects the IP addresses of proxy servers returned from the authentication server. The collected IP addresses can then be used to launch DDoS attack. Venkatesan *et al.*[23] proposed a proactive defense mechanism against proxy harvesting attack by randomly and regularly reassigning IP addresses to proxy servers to disrupt the reconnaissance efforts. Their simulation results showed that their proposed defense strategy is effective.

An Economic Denial of Sustainability (EDoS) attack targets cloud consumers by exploiting the "pay-as-you-go" charging scheme in the cloud computing. In an EDoS attack, an attacker can send a large number of fraudulent requests to consume the services of the targeted cloud consumer, which eventually exhausts the budget of the cloud consumer and causes the consumer not to be able to host his/her services in the cloud. To defend against EDoS attacks, Wang *et al.*[54] proposed WebTrap, a moving target defense and trap strategy that dynamically changes the resource addresses to hide the actual URL of the resource from the attackers while trapping malicious EDoS attack bots using dynamic trap injection technique. The authors[54] claimed that WebTrap can effectively reduce the cost incurred by EDoS attack and the overhead incurred by WebTrap is negligible.

Challenges in IP Address Randomization. The most challenging part in an IP address-based randomization technique is the requirement of a large IP address space. However, as the advancement of the Internet of Things paradigm pushes the wider adoption of IPv6, the requirement can be satisfied in near future.

Another limitation of IP address-based randomization is the potential "connectivity disruption" when a new IP address is assigned to a host. For instance, during a file transfer session between two hosts, if a host is forced to switch to a new IP address before the file transfer is complete, then the connection will be lost and the service will be disrupted. This is usually addressed in research by using a stand-alone component that monitors the host's connection status. If an active connection is detected, then the IP address randomization will be temporarily suppressed until the connection becomes idle. This technique, however, may decrease the effectiveness of MTD. Although SCIT and MAS maintain the connectivity to the VM online via a controller or dispatcher, neither covers the IP assignment to the VM. MT6D[49], however, is capable of preventing any connection disruption, and it is tested and analyzed by Yeung *et al.*[55] that MT6D can ensure that traffic is not lost during address change period. Besides MT6D, more research is needed to balance the minimal connectivity disruption and the effectiveness of MTD.

5.3.2 Virtualization-Based MTD

Virtualization has been a major driving force behind moving target defense techniques. It enables network administrators to easily and quickly modify the network configurations at low cost such as hosts' operating systems, IP address, port numbers and running services at a lot cost in a virtualized environment, which can be very challenging and costly in a physical network. The Self Cleaning Intrusion Tolerance (SCIT) is a theory proposed by Bangalore and Sood[19] to proactively protect a server by reducing its exposure time on the Internet from several months to less than a minute. To accomplish this task, SCIT hosts an array of virtual machines, each loaded with a copy of pristine and malware-free server. The SCIT controller rotates the server array and determines when to bring a virtual machine (VM) online. In each rotation, a VM is online and accessible for only a short period of time and then it is taken offline and replaced by a new VM. The old VM will be wiped clean and reset back to its pristine state and will be ready to be brought up online by the SCIT controller. The research results show that using SCIT to reduce the exposure time of the VM's can increase the security of servers and thus minimizes any losses caused by an intrusion[19].

Moving Target Surfaces (MAS) is another rotation-based moving target defense technique, proposed by Huang and Ghosh[56] to protect Web services from malicious attacks[56]. The primary difference between MAS and SCIT is that each VM in SCIT is loaded with the same software. Whereas, in MAS each VM is loaded with diversified software mix. The authors[56] used different operating systems and Web servers to make over 1 500 unique combinations (1 554 to be exact) where each combination represents an attack surface. Therefore, each rotation will bring a diversified VM online with different attack surfaces, which increases uncertainty and diversity in the system. Another difference between these two techniques is that SCIT uses a fixed time interval for VM rotation, whereas MAS uses detection engine to enable the clean VM to stay online for longer cycle time, resulting in lower overhead cost than SCIT.

Unlike MAS or SCIT where only specific software or software mix is hosted in each virtual machine, MTD CBITS[57] is a cloud-based approach in which each running component of an IT system is hosted in a virtual machine instance or a cluster of instances, and every component can be replaced with a pristine version of the component. The replacement of virtual machine instances is accomplished through a sequence of adaptions, and their experimental results show that MTD CBITS can efficiently increase the difficulty of attacks against an IT system.

By applying virtualization at the operating system level, Okhravi *et al.*[7,8] proposed a design and implementation of TALENT (Trusted Dynamic Logical Heterogeneity) to protect critical infrastructure applications. The OS level virtualization works at the level of filesystems, memory regions, sockets and kernel objects. It enables TALENT to change the platform on the fly in order to achieve the platform heterogeneity. TALENT also uses a portable checkpoint compiler to compile the application for different architectures and hence is able to migrate a running application across different platforms while preserving the state of the application, as shown in Fig.2. The live-migration of applications across heterogeneous platforms creates a cyber moving target that increases the resilience of the system against attacks.

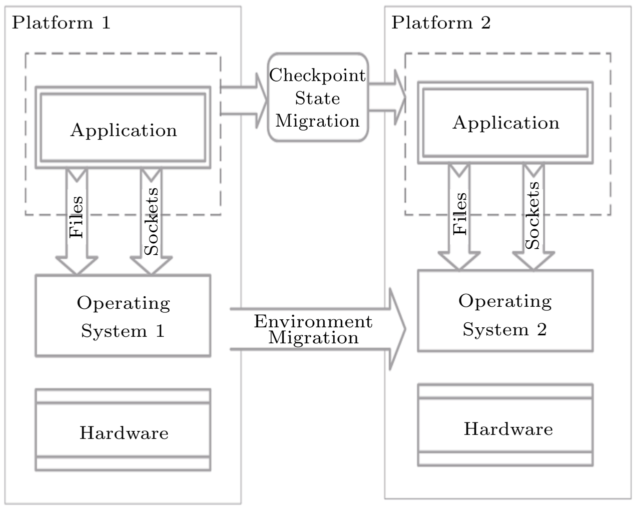


Fig.2. TALENT migration process.

Challenges in Virtualization-Based MTD. The time to reload or restart a virtual machine can affect the performance of the virtualization-based MTD, especially when there are multiple services running on the same VM. More research is needed on the optimization of the virtualization process.

5.3.3 Decoy-Based MTD

Decoy systems, or deception systems, are phony systems set up to trap unauthorized users and track their suspicious activities to better understand the intentions and methodologies of the attack. The concept of deception systems can be traced back to the Stoll's book[58], in which the author described how a phony system can be set up to lure and eventually entrap attackers.

Typically, decoy systems are deployed on a separate network and on a demilitarized zone (DMZ). It contains various known vulnerabilities with appealing but fake data, such as seemingly sensitive decoy documents and passwords to critical system, as well as vulnerable accounts to lure attackers away from the protected network. Not being aware of visiting a decoy system, an attacker may try to exploit the decoy system with the hope of gaining unauthorized access to the system. Meanwhile, each unauthorized activity is being monitored, logged, and analyzed by the system administrator. The analysis not only helps the system administrators to gain insights on the intrusion, but also helps uncover unknown vulnerabilities of the main operational network. Hence, the longer the attacker is trapped in the decoy system without knowing it, the more the data can be collected.

Clark *et al.*[52] conducted a study on the effectiveness of IP address randomization in decoy-based MTD system by deploying a myriad of decoy nodes (hosts) on the same network. They used a virtualization technology to create a collection of virtual nodes representing hosts on the network, in which there was only one decoy node and the rest were real nodes. The only difference between a real node and a decoy node was the response time to request queries made to them (i.e., network scans) and the difference could be observed by the adversary to identify the decoy node. This interaction between the adversary and a virtual node was modeled in the research and the optimal attack strategy for the adversary to identify the decoy node was computed from the model as well as the optimal defense strategy for the defender to randomize the IP addresses. A simulation experiment with 99 real nodes and one decoy node has been reported by the authors[52] to visually characterize the adversary's strategy and the defender's strategy.

Challenges in Decoy-Based MTD. The major challenge in decoy-based MTD systems is the cost associated with the deployment and maintenance of decoy nodes. To provide effective defense, the decoy-based MTD system requires a significantly large number of decoy nodes to be deployed. As pointed out by Clark *et al.*[52], 99 decoy nodes are needed to deploy in order to protect only one real operational node. As a matter of fact, it is recommended to keep the ratio of real and decoy targets on the order of 1:10 000 to significantly slow down and frustrate the attackers. As a result, the expensive cost of deploying and managing decoy nodes might surpass the security benefits of the decoy-based MTD system.

Another challenge in employing decoy-based MTD systems is the legal and ethical issues. For example, if a legitimate user, who is accessing the network, ends up at the decoy host, there could be legal consequences or even a lawsuit. It is especially true when a user is trying to use critical services, such as financial transactions, controller applications, and emergency services. Therefore, it is very critical and yet difficult to differentiate legitimate users and ensure the quality of services to them.

5.3.4 Software-Defined Networking Based MTD

The static nature of traditional network architecture hurdles the effective implementation of MTD techniques. An effective MTD implementation requires a flexible, movable, and programmable environment to operate. Software-Defined Networking (SDN) can provide such an environment. SDN is a new network architecture in which the control logic and decisions are separated from the network devices and are moved to logically centralized controllers. This separation of the control logic and the data flow offers very flexible, programmable and scalable network architecture[59], which is an appropriate platform for implementing MTD strategies. Fig.3 shows a simplified view of the SDN-based network architecture.

The Control Plane is a virtual plane where the SDN controller resides. The SDN controller makes decisions on how the network data packets should be handled. The Data Plane usually refers to the virtual plane that consists of all network devices such as switches that forward data packets per the decisions (i.e., rules) made by the controller. The Application Plane is a collection of applications that can extend the functionalities of the controller. When a data packet arrives at a switch in the Data Plane, the switch will search its database to see if a rule, i.e., a decision from the controller, exists. Once the switch finds the rule, it will handle the data packet by the rule. Otherwise, the switch sends a request to the controller for further assistance. Once a decision is made by the controller, a rule is generated and sent back to the switch to handle the data packet.

The rule itself is also stored on the switch for handling similar upcoming packets. The third-party applications on the Application Plane do not have direct accesses to network devices in the Data Plane, but can communicate with the SDN controller to relay the needed operations. The communications between planes are carried through different APIs built on open standards. The APIs between the application plane and the control plane are called the "northbound" APIs. Whereas, the APIs between the control plane and the data plane is called the "southbound" APIs.

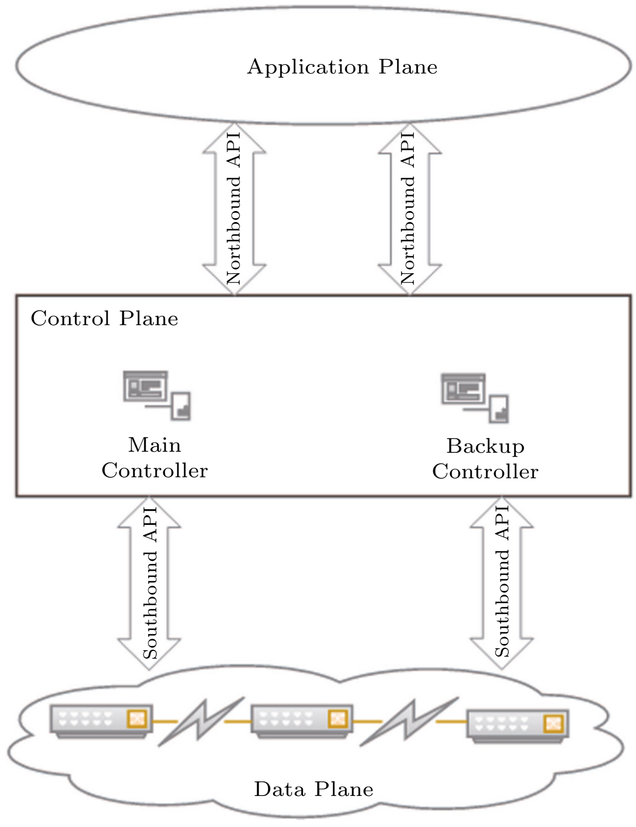


Fig.3. Simplified view of an SDN architecture.

Among all the supported open southbound API's, OpenFlow from Open Networking Foundation (ONF)[22] is the most adopted API that enables the communication between the control plane and the data plane. ONF uses "layer" instead of "plane" to indicate each of the three building blocks in the SDN architecture as shown in Fig.4.

The SDN-based MTD techniques can be implemented in two modes: 1) hybrid-SDN implementation, and 2) pure-SDN implementation. In the hybrid-SDN implementation, the MTD techniques are implemented with the help of both SDN and non-SDN components, such as NAT gateways, routers, and decision-making agent installed on hosts. Whereas, in the pure-SDN mode, the MTD techniques are implemented either as a network application at the application layer, or as a component integrated into the controller layer, as depicted in Fig.4.

A good example of the hybrid-SDN implementation is OpenFlow Random Host Mutation (OF-RHM)[50], which is designed to thwart reconnaissance attacks. With the help of an NAT gateway and an SDN controller, each host's real and unchanged IP address (rIP) is mapped onto a virtual and short-lived IP address (vIP). The vIPs are randomly generated and assigned by the SDN controller and stored in the flow tables of all OpenFlow switches to properly route the network traffic. Since the mutation and assignment of the virtual IP addresses, and also the data route management are all performed in the background without the involvement of the end hosts, OF-RHM can effectively conceal the identities of the protected hosts.

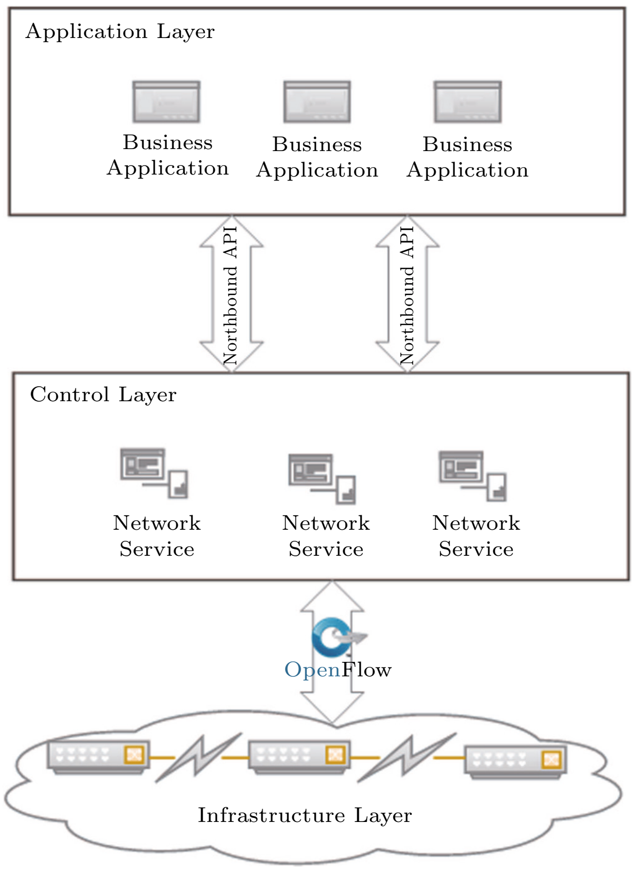


Fig.4. Layered view of OpenFlow-based SDN architecture.

[22]https://www.opennetworking.org/sdn-definition, Nov. 2018.

MacFarland *et al.*[60] proposed a host-based MTD solution by using DNS server, SDN controller and NAT device. When a client application requests to resolve a host name to an IP address, the request is sent to the network's DNS server. Before the DNS server sends back a reply packet with the resolved IP address to the requestor, it elevates the packet to the SDN controller for consideration. The SDN controller, upon receiving the DNS reply packet, randomly generates synthetic IP and MAC addresses, and then creates NAT rules to translate the synthetic IP address into the real IP address and the synthetic MAC address into the real MAC address. The NAT rules are then installed on the NAT device and also on the client's machine to establish a connection between the client and the requested host. This design allows the SDN controller to create a different and moving IP address and MAC address combination for each DNS resolution, preventing attackers from tracking traffic flows, offering both anonymity and unlikability.

Wang *et al.*[61] argued that the host address mutation technique is not effective if an attacker knows the host's domain name. They proposed to use SDN to randomize domain's name as well to protect hosts from reconnaissance attacks. However, the domain name of a company is an important aspect of the company's network when communicating with clients and is purchased through an authorized domain name registrar. The paper does not explain the feasibility of mutating a host's domain name or the potential impact of domain name mutation on the enterprise company.

Packet Header Randomization (PHEAR) is a technique proposed by Skowyra *et al.*[62] to enhance the privacy and security for enterprise networks by using OpenFlow-based SDN. PHEAR requires each host to install a PHEAR proxy that is responsible for rewriting incoming and outgoing packets in order to remove identifiers, and the modified packets will be routed via OpenFlow switches. By hiding or obscuring the packet header information, it not only protects the privacy of the host, but also makes the system more unpredictable to the attackers. A big advantage of PHEAR is that if an SDN infrastructure already exists, PHEAR can be implemented in it without any additional hardware. The experimental results show that PHEAR has low latency and high throughput, which are two important requirements for many network applications.

Kampanakis *et al.*[63] integrated several defense techniques, such as obfuscation of port numbers, service version and OS hiding as well as random host mutation into the SDN controller to provide common network defense against network mapping and reconnaissance attacks. This pure-SDN MTD scheme is implemented in Cisco's One Platform Kit (onePK) and the results show that SDN-based MTD can significantly increase the attacker's overheads.

Cloud computing has been adopted rapidly by companies around the world due to its advantages such as flexibility, cost effectiveness, and mobility. Due to its increasing popularity, cloud networks have attracted attackers. In a typical cloud network, resources (e.g., data and applications) can be located at geographically different locations. More specifically, cloud computing, as a highly distributed architecture, requires innovative technology for network management and security defense. The amazing networking features offered by SDN make it a perfect platform for cloud networks. The logically centralized SDN controller can be used to proactively perform attack analysis and thus apply MTD countermeasures based on the analysis while ensuring the countermeasures not to conflict the security policy[64].

Challenges in SDN-Based MTD. SDN is under continuous and active development by major organizations such as Open Network Foundation and Linux Foundation. It is still considered a new architecture and has not been prevalently adopted around the world. The lack of adoption is a major challenge for implementing MTD techniques in an SDN-based environment. However, we believe this challenge will soon be overcome because major commercial companies, such as Cisco, HP, Google, and Microsoft, have started recognizing SDN as an innovative key technology in the cloud computing.

The lack of enough research in some certain aspects of SDN-based MTD is another important challenge that might limit the adoption of SDN in MTD. Examples of some area topics that require fundamental and emerging research attentions are as follows.

1) *Security and Dependability.* SDN can suffer from single point of failure because if the controller in SDN failed, then the communication between planes may fail and the whole network system may not be operational. The security and the dependability of SDN are ongoing active research topics[65].

2) *Effectiveness of SDN-Based MTD.* Among all the research articles related to SDN-based MTD we surveyed, none of the papers compared the effectiveness of SDN-based MTD approaches with that of the existing non SDN-based MTD approaches. There is a need to

compare the effectiveness and lists the pros and cons of such techniques.

3) *Performance Issues of SDN-Based MTD.* Even though all SDN-based MTD research work provided performance evaluation results, no existing research focus on comparing the performance of SDN-based MTD with that of non SDN-based MTD.

4) *Compatibility of SDN-Based MTD.* Since SDN is not a replacement to the existing network infrastructure, it is crucial to ensure that any SDN technique will be compatible with the current network infrastructure. The lack of compatibility of SDN-based MTD is not only a challenge, but also a future research opportunity.

5.3.5 *Lightweight MTD*

The increasing growth of the use of low-power and low-resource embedded devices such as wireless sensors and Internet of Things devices has introduced new security challenges. The main reason is that many existing MTD strategies cannot be directly applied to these devices due to the limitations of power and computation capacity on the devices.

Casola *et al.*[66] proposed an MTD approach for protecting resource-constrained distributed devices by reconfiguring the devices at the security layer and the physical layer. The reconfiguration at the security layer refers to the selection of a different cryptosystem regularly and the physical layer reconfiguration refers to the full-image replacement of the firmware of the device. The approach is evaluated on a Wireless Sensor Network (WSN) and the results show that it can effectively reduce the attacker's success probability.

Based on MT6D, Zeitz *et al.*[67] introduced an initial design for a Micro-Moving Target IPv6 Defense, μMT6D, to protect the Internet of Things by limiting the exposure time of the device. The initial experiment of the design is introduced later to evaluate the power consumption of the implementation of μMT6D.

Challenges in Lightweight MTD. Wireless sensors, IoT devices, and other embedded systems have limited resources such as power supply, computing capability, and available storage. Any existing MTD strategy that requires heavy and frequent computation might not be feasible for these resource-restrained devices. There are lightweight cryptographic hash algorithms (PHOTON, QUARK, etc.[68]), but research work is still needed to study the feasibility of implementing MTD strategies on these devices with the algorithms and the optimization of the MTD implementations.

6 Overview of Existing MTD Analysis Work

Researchers use various approaches to analyze various forms of MTD strategies in order to gain insights and valuable information and improve the effectiveness and performance of the MTD strategies. Simulation and testbed experimentations[49,69] are two major means for quantitative analysis of the effectiveness and performance of MTD strategies. The commonly used simulation tools include free off-the-shelf OMNet++[53], Mininet[50], NeSSi2[2] and commercial tool onePK[63]. Recently, Virtual Infrastructure for Network Emulation (VINE) has been proposed as an MTD experimentation environment[19] because of its broad capabilities and wide selection of tools, such as network topology generation agents, background traffic and attack generation agents, packet capture agents, and traffic monitoring agents. A case study is included in the research to show that VINE is capable of providing researchers valuable tools to test MTD defense in a number of configurations, and in a number of operation environments.

The analysis approaches in the papers we surveyed can be divided into three major categories.

1) *Strategy-Specific Analysis.* It aims at analyzing a specific MTD strategy by using simulation, experiment, or proof of concept to discover any issues in the target MTD strategy.

2) *Metric-Based Analysis.* In the metric-based analysis, an analysis metric based on MTD requirements is usually utilized to set up a testbed experiment, run the experiment against various MTD techniques, and then use the metric to analyze the result of each MTD strategy tested. Some metric-based analysis may involve a model where is usually a generalized threat model to explain the details of problem addressed[8,69].

3) *Model-Based Analysis.* In the model-based analysis, a model to generalize the features and behaviors of a specific type or general type of MTD strategy is employed to perform the statistical analysis of the model, and then apply simulation or custom experiment to verify the analysis.

6.1 Strategy-Specific Analysis

Since ASLR has been implemented in major operating systems, many studies on the effectiveness of ASLR have been conducted. There are several ways to bypass ASLR[25–27]. To enhance ASLR, some fine-grained ALSR and code randomization strategies have been proposed[9,10,28,29]. The fine-grained ALSR uses techniques such as the permutation of the order of functions and basic blocks and the randomization of the data and code structure to improve ASLR against code injection and memory corruption attacks under the assumption that the disclosure of one single memory address does not allow attackers to deploy attack. However, Snow *et al.*[30] argued that the above assumption can be easily violated. Therefore, the fine-grained ASLR may not be more effective than traditional ASLR. By repeatedly exploiting the memory disclosure, it is possible to map an application's memory layout on-the-fly, search for API functions and gadgets, and use Just-In-Time compilers that the target program is using to finally launch attacks.

Hamlet and Lamb[70] proposed to use dependency graph to outline the impacts on users, defenders and attackers once an MTD strategy is adopted. This impact analysis can be used to measure the possible efficacy and cost of the given MTD and help defenders to choose the best MTD strategy.

6.2 Metric-Based Analysis

Quite often, attackers launch multiple attacks against a target system. Hence, it is more practical to analyze the overall performance of an MTD system under the broad definition described in Section 2. Hobson *et al.*[40] grouped the challenges that relevant MTD techniques have in common into three types.

1) *Coverage.* The exploitable elements of the attack surface must be covered by the moving target defense and no information should be leaked from the static components of the surface.

2) *Unpredictability.* The current or future dynamic changes in MTD should not be predictable by attackers.

3) *Timeliness.* The dynamic changes must be applied between the attacker observations and the subsequent attacker actions.

Three moving target defense techniques are evaluated on how it reacts to each challenge, and a case study of three low-level MTD techniques, ALSR, ISR, and software diversification, is used to demonstrate how the evaluation works[40]. From the analysis point of view, Hobson *et al.*[40] concluded that most of the MTD techniques exhibit weaknesses across all of three criteria. For instance, ALSR can meet the coverage requirement, but cannot guarantee unpredictability and timeliness; whereas, ISR cannot meet the full coverage requirement, but it can provide enough unpredictability and timeliness. Furthermore, software diversification can meet coverage requirement and provide good unpredictability, but timeliness can be difficult to achieve.

Green *et al.*[71] proposed to evaluate network-based MTD techniques with a different set of properties: 1) moving property, 2) access control property, and 3) distinguishability property. The moving property describes how well an MTD technique can alter network information. It has three aspects: unpredictability, vastness, and periodicity. The unpredictability requires that the new destination of any given moving target state should be randomized enough so that no client could guess the new destination. The vastness requires that the destination space of the MTD must be sufficiently large enough to prevent the destination space from being searched exhaustively. Moreover, the periodicity means that the MTD must change with enough regularity to quickly invalidate the information collected previously by the attacker.

The access control property requires that only authorized client can access a moving target system via a mapping system. It has three aspects: uniqueness, availability, and revocability. The uniqueness means the access to the moving target system by a client only belongs to the client and cannot be shared with any other clients. The availability means that a client's access to the moving target system should be guaranteed so that MTD does not cause denial-of-service to the authorized clients. The revocability means that the mapping system should have the ability to terminate or revoke a client's authorization without causing damage to the MTD system. The distinguishability means a system's ability to separate trustworthy clients from the untrustworthy ones.

Green *et al.* further analyzed several MTD techniques including OF-RHM[50] and MT6D[49]. The analysis results show that at least one of these properties was not covered by the evaluated MTD techniques. For example, while both OF-RHM and MT6D cover the moving property, neither of them covers the availability aspect of the access control property.

Okhravi *et al.*[69] proposed a generalized model for dynamic platform techniques which dynamically change various properties of the computing platform, such as instruction set architecture, stack direction, kernel version, OS distribution and machine instance, to make attacks more difficult. The authors[69] first identified four features as a metric to describe the protection by existing dynamic platform techniques, and then used the metric to quantitatively analyze a specific dynamic platform technique developed by them, call

TALENT[8], but the authors believed their analysis can be generalized based on the features of all existing techniques.

Another quantitative approach was proposed by Zaffarano *et al.*[72] by using Cyber Quantification Framework from Siege Technologies along with an evaluation metric with four categories: 1) productivity, 2) success, 3) confidentiality, and 4) integrity. In order to use the metrics to evaluate an MTD system, the authors[72] proposed a mission and attack activity model. The mission activity model represents the legitimate network activities such as sending emails and accessing an FTP server. On the other hand, the attack activity model represents actions or attacks that an attacker would perform. After applying the metrics on the two models, eight individual evaluation metrics will be generated, and the data can be evaluated to describe the effectiveness of the MTD system.

Taylor *et al.*[73] later applied this evaluation framework and metrics to quantitatively evaluate two MTD techniques provided by the Air Force Research Laboratory: ARCSYNE (Active Repositioning in Cyberspace for SYNchronized Evasion) and SDNA (Self-shielding Dynamic Network Architecture). ARCSYNE offers a synchronized IP-hopping to the nodes on a network and protects the network through system-wide encryption and a continuous changing network topology. SDNA is based on cryptographically secure network dynamics on an IPv6 network. Based on the same CQF and metrics as in [72], the authors created different metrics to evaluate the two MTD techniques quantitatively. The metrics show that the attack success is high without MTD, but it significantly decreases when MTD is deployed.

6.3 Model-Based Analysis

Evans *et al.*[74] proposed a model with a defender and an attacker entity for diversity defense strategy, such as ASLR, ISR, and data randomization. In this model, the defender's goal is to provide a service with reliability and performance; whereas the attacker's goal is to exploit the server. The authors[74] conducted an analysis on the effectiveness of a moving target defense through low-level (OS and software levels) diversification technique against five different types of attacks: 1) circumvention attacks, 2) deputy attacks, 3) brute force and entropy reduction attacks, 4) probing attacks, and 5) instrumental attacks. They concluded that the diversification defense is effective against probing attacks, but is either marginally effective or not effective against the other four types of attacks.

Xu *et al.*[75] proposed a three-layered state machine model to evaluate and compare different MTD techniques with the intention of bridging the gap among existing MTD evaluation methods. The authors[75] classified existing MTD methods into low- and high-level methods depending on the application scopes. In the first layer of the model, a state machine is created for each low-level program to identify the required and critical contexts from the program, thus they can be evaluated. In the second layer, each Program State Machine (PSM) is interconnected to form a System State Machine (SSM), thus the interaction between different low-level programs can be modeled. There is also a third layer, which is an evaluation state machine (ESM) and is designed to work as a user interface. For a given attack, changing the state machine in the lower layer could form different MTD combinations. The authors[75] included an example of file decryption and compression to demonstrate the feasibility of this model and claim a more comprehensive evaluation needs to be conducted.

Manadhata[76] proposed to use attack surface shifting as an MTD strategy and used a stochastic game model to define the interaction between a defender and an attacker. The author claimed that the stochastic game model could help the defender to determine an optimal MTD strategy and optimally shift the system's attack surface.

Zhu and Başar[77] proposed a game theoretic framework for multi-layer attack surface shifting under the assumption that the attack is launched in multiple stages and accordingly the defense strategy is developed at each layer of the system. At each stage of the attack, the system can respond adaptively based on the feedback information collected and thus can minimize the overall risk.

Zheng and Nanrin[78] proposed to use Markov Decision Process to analyze the impact of various costs on the selection of defense strategy in Moving Target Defense and to help the defender to choose the optimal defense strategy in a certain situation.

Carter *et al.*[79] defined the interaction between a defender and an attacker in the threat model as a typical leader-follower game. By applying game theory and statistical analysis, a dynamic platform technique can be strategically selected. Their results show that the strategic selection significantly outperforms a simple random selection.

Maleki *et al.*[80] presented an MTD analysis framework by using the game theory and a Markov model. The framework can model concrete MTD strategies and provide general theorem about how the probability of a successful adversary defeating an MTD strategy is related to the amount of time/cost spend by the adversary. Two concrete MTD strategies were analyzed by the framework to demonstrate its applicability beyond theoretical analysis.

7 Future Research Directions

From the collection of papers that the authors reviewed, it is apparent that more and more research studies have been carried on the network level. The focus has been shifted from studying a particular MTD technique to a practical MTD system. We list some of the future research directions following the research shifts and trends.

• It is necessary to use other forms of mathematical models for MTD representations and analysis to gain more insights and provide theoretic foundation for potential practical MTD framework.

• A rigorous and fundamental risk and cost analysis and comparison of MTD implementation are required. There are not too many studies related to risk assessments and cost analysis of various forms of MTD strategies.

• Software-Defined Networking offers highly flexible and programmable network architecture. Therefore, it is a candidate for developing and implement MTD system. More research work needs to be carried out in this area, for instance, the effectiveness and performance study on SDN-based MTD strategies, compared with the non SDN-based MTD strategies, the compatibility study on SDN-based MTD in the current network infrastructure, etc.

• As more resource-restrained devices such as wireless sensors and IoT devices are emerging in both home and enterprise networks, new security threats are also introduced and need to be addressed. The existing research work on the lightweight MTD strategies has laid a good foundation in this direction, but more efforts are still needed to deepen and widen the research in this direction.

8 Conclusions

Moving Target Defense has emerged as one of the game changing ideas in computer security defense. A wide range of research work has been done on many aspects of it, from simulation and implementation to evaluation. In this paper, we conducted a comprehensive survey on the current status of MTD research presented in around 80 published papers. By identifying the advantages and challenges of the surveyed MTD techniques, this paper can provide researchers with the guidance for the future MTD research.

As we can observe from Table 2, the network-level implementation of MTD has been gaining more research attention and the research focus has been shifted from studying one specific MTD technique to designing a defense framework with multiple MTD techniques to address various needs of the network security defense. Since the network architecture has been under unprece-

Table 2. Summary

Approach	MTD Implementation	Attacks Addressed	Challenges
OS level approaches	• ASLR • ISR	• Buffer overflow	• Only 64-bit architecture can provide enough randomization • May be bypassed by memory disclosure attack
Software level approaches	• Software diversification • Software diversification through middle	• Buffer overflow • SQL injection attack	• The cost of generating code variants • The delivery of software variants • How to handle updates and patches of variants • Integrity and verification of software variants • Special execution environment and platform • Performance overhead
Network level approaches	• IP address randomization • Virtualization-based MTD • Decoy-based MTD • SDN-based MTD • Lightweight MTD	• Reconnaissance attack • SQL injection • Cross-site scripting (XSS) • DDoS attack • Computer worm attack	• Large amount of IP addresses needed for randomization • Connectivity disruption • Cost associated with decoy deployment and management • Startup time of virtual machine instance • Legal issue of decoy • Lack of adoption of SDN • Lack of research studies on resources-restrained devices

dented attacks, we predict more research to be conducted in MTD at the network layer will emerge.

Software-Defined Networking (SDN) has been gaining attention in both the industry and the research community due to its characteristics such as low management cost, high flexibility and programmability, and dynamic network architecture. By utilizing the advantages of SDN, specialized network applications and services can be easily deployed in an SDN environment with the goal of creating a solid MTD system.

The rapid adoption of resource-constrained embedded devices including IoT devices has introduced new security threats. Existing research efforts show that MTD is a promising candidate for protecting these devices, but more research in this field is needed.

References

[1] Manadhata P K, Wing J M. An attack surface metric. *IEEE Transactions on Software Engineering*, 2011, 37(3): 371-386.

[2] Zhuang R, Zhang S, DeLoach S A, Ou X M, Singhal A. Simulation-based approaches to studying effectiveness of moving-target network defense. In *Proc. National Symposium on Moving Target Research*, June 2012, pp.21-26.

[3] Peng W, Li F, Huang C, Zou X. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces. In *Proc. IEEE International Conference on Communications*, June 2014, pp.804-809.

[4] Okhravi H, Rabe M A, Mayberry T J, Leonard W G, Hobson T R, Bigelow D, Streilein W W. Survey of cyber moving target techniques. Technical Report, Massachusetts Institute of Technology, 2013. http://www.dtic.mil/dtic/tr/fulltext/u2/a591804.pdf, Sept. 2018.

[5] Cai G l, Wang B S, Hu W, Wang T Z. Moving target defense: State of the art and characteristics. *Frontiers of Information Technology & Electronic Engineering*, 2016, 17(11): 1122-1153.

[6] Lei C, Zhang H Q, Tan J L, Zhang Y C, Liu X H. Moving target defense techniques: A survey. *Security and Communication Networks*, 2018, Article No. 3759626.

[7] Okhravi H, Comella A, Robinson E, Yannalfo S, Michaleas P, Haines J. Creating a cyber moving target for critical infrastructure applications. In *Proc. the 5th IFIP WG 11.10 International Conference on Critical Infrastructure Protection*, March 2011, pp.107-123.

[8] Okhravi H, Comella A, Robinson E, Haines J. Creating a cyber moving target for critical infrastructure applications using platform diversity. *International Journal of Critical Infrastructure Protection*, 2012, 5(1): 30-39.

[9] Barrantes E G, Ackley D H, Forrest S, Palmer T S, Stefanovic D, Zovi D D. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proc. the 10th ACM Conference on Computer and Communications Security*, October 2003, pp.281-289.

[10] Kc G S, Keromytis A D, Prevelakis V. Countering code-injection attacks with instruction-set randomization. In *Proc. the 10th ACM Conference on Computer and Communications Security*, October 2003, pp.272-280.

[11] Just J E, Cornwell M. Review and analysis of synthetic diversity for breaking monocultures. In *Proc. the 2004 ACM Workshop on Rapid Malcode*, October 2004, pp.23-32.

[12] Stamp M. Risks of monoculture. *Communications of the ACM*, March 2004, 47(3): 120.

[13] Forrest S, Somayaji A, Ackley D H. Building diverse computer systems. In *Proc. the 6th Workshop on Hot Topics in Operating Systems*, May 1997, pp.67-72.

[14] Cox B, Evans D, Filipi A, Rowanhill J, Hu W, Davidson J, Knight J, Nguyen-Tuong A, Hiser J. N-variant systems: A secretless framework for security through diversity. In *Proc. the 15th Conference on USENIX Security Symposium*, July 2006, Article No. 16.

[15] Gherbi A, Charpentier R. Diversity-based approaches to software systems security. In *Proc. International Conference on Security Technology*, December 2011, pp.228-237.

[16] Neti S, Somayaji A, Locasto M E. Software diversity: Security, entropy and game theory. In *Proc. the 7th USENIX Workshop on Hot Topics in Security*, August 2012, Article No. 5.

[17] Jacob M, Jakubowski M H, Naldurg P, Saw C W, Venkatesan R. The superdiversifier: Peephole individualization for software protection. In *Proc. the 3rd International Workshop on Security*, November 2008, pp.100-120.

[18] Antonatos S, Akritidis P, Markatos E P, Anagnostakis K G. Defending against hitlist worms using network address space randomization. In *Proc. the 2005 ACM Workshop on Rapid Malcode*, November 2005, pp.30-40.

[19] Bangalore A K, Sood A K. Securing web servers using self cleansing intrusion tolerance (SCIT). In *Proc. the 2nd International Conference on Dependability*, June 2009, pp.60-65.

[20] Boyd S W, Keromytis A D. SQLrand: Preventing SQL injection attacks. In *Proc. the 2nd International Conference on Applied Cryptography and Network Security*, June 2004, pp.292-302.

[21] Portner J, Kerr J, Chu B. Moving target defense against cross-site scripting attacks (position paper). In *Proc. the 7th International Symposium on Foundations and Practice of Security*, November 2015, pp.85-91.

[22] Jia Q, Sun K, Stavrou A. MOTAG: Moving target defense against internet denial of service attacks. In *Proc. the 22nd International Conference on Computer Communication and Networks*, July 2013.

[23] Venkatesan S, Albanese M, Amin K, Jajodia S, Wright M. A moving target defense approach to mitigate DDoS attacks against proxy-based architectures. In *Proc. IEEE Conference on Communications and Network Security*, October 2016, pp.198-206.

[24] Shacham H, Page M, Pfaff B, Goh E J, Modadugu N, Boneh D. On the effectiveness of address-space randomization. In *Proc. the 11th ACM Conference on Computer and Communications Security*, October 2004, pp.298-307.

[25] Bittau A, Belay A, Mashtizadeh A, Mazières D, Boneh D. Hacking blind. In *Proc. IEEE Symposium on Security and Privacy*, May 2014, pp.227-242.

[26] Hund R, Willems C, Holz T. Practical timing side channel attacks against kernel space ASLR. In *Proc. IEEE Symposium on Security and Privacy*, May 2013, pp.191-205.

[27] Seibert J, Okhravi H, Söderström E. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *Proc. the 2014 ACM SIGSAC Conference on Computer and Communications Security*, November 2014, pp.54-65.

[28] Pappas V, Polychronakis M, Keromytis A D. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In *Proc. IEEE Symposium on Security and Privacy*, May 2012, pp.601-615.

[29] Wartell R, Mohan V, Hamlen K W, Lin Z Q. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In *Proc. the 2012 ACM Conference on Computer and Communications Security*, October 2012, pp.157-168.

[30] Snow K Z, Monrose F, Davi L, Dmitrienko A, Liebchen C, Sadeghi A R. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization. In *Proc. the 2013 IEEE Symposium on Security and Privacy*, May 2013, pp.574-588.

[31] Hu W, Hiser J, Williams D, Filipi A, Davidson J W, Evans D, Knight J C, Nguyen-Tuong A, Rowanhill J. Secure and practical defense against code-injection attacks using software dynamic translation. In *Proc. the 2nd International Conference on Virtual Execution Environments*, June 2006, pp.2-12.

[32] Porras P. Inside risks: Reflections on Conficker. *Communications of the ACM*, 2009, 52(10): 23-24.

[33] Portokalidis G, Keromytis A D. Global ISR: Toward a comprehensive defense against unauthorized code execution. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.49-76.

[34] Chen L M, Avizienis A. *N*-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. the 25th International Symposium on Fault-Tolerant Computing*, June 1995, pp.113-119.

[35] Massalin H. Superoptimizer: A look at the smallest program. In *Proc. the 2nd International Conference on Architectual Support for Programming Languages and Operating Systems*, October 1987, pp.122-126.

[36] Jackson T, Salamat B, Homescu A, Manivannan K, Wagner G, Gal A, Brunthaler S, Wimmer C, Franz M. Compiler-generated software diversity. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.77-98.

[37] Cabutto A, Falcarin P, Abrath B, Coppens B, De Sutter B. Software protection with code mobility. In *Proc. the 2nd ACM Workshop on Moving Target Defense*, October 2015, pp.95-103.

[38] Franz M. E unibus pluram: Massive-scale software diversity as a defense mechanism. In *Proc. the 2010 New Security Paradigms Workshop*, September 2010, pp.7-16.

[39] Jackson T, Homescu A, Crane S, Larsen P, Brunthaler S, Franz M. Diversifying the software stack using randomized NOP insertion. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Jajodia S, Ghosh A K, Subrahmanian V S, Swarup V, Wang C, Wang X S (eds.), Springer, 2013, pp.151-173.

[40] Hobson T, Okhravi H, Bigelow D, Rudd R, Streilein W. On the challenges of effective movement. In *Proc. the 1st ACM Workshop on Moving Target Defense*, November 2014, pp.41-50.

[41] Spinellis D. Reflection as a mechanism for software integrity verification. *ACM Transactions on Information and System Security*, 2000, 3(1): 51-62.

[42] Msgna M, Markantonakis K, Naccache D, Mayes K. Verifying software integrity in embedded systems: A side channel approach. In *Proc. the 5th International Workshop on Constructive Side-Channel Analysis and Secure Design*, April 2014, pp.261-280.

[43] Basili V R, Selby R W. Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering*, 1987, SE-13(12):1278-1296.

[44] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proc. the 14th ACM Conference on Computer and Communications Security*, October 2007, pp.552-561.

[45] Pappas V, Polychronakis M, Keromytis A D. Practical software diversification using in-place code randomization. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Jajodia S, Ghosh A K, Subrahmanian V S, Swarup V, Wang C, Wang X S (eds.), Springer, 2013, pp.175-202.

[46] Cui A, Stolfo S J. Symbiotes and defensive mutualism: Moving target defense. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.99-108.

[47] Zhuang R, Zhang S, Bardas A, DeLoach S A, Ou X, Singhal A. Investigating the application of moving target defenses to network security. In *Proc. the 6th International Symposium on Resilient Control Systems*, August 2013, pp.162-169.

[48] Al-Shaer E. Toward network configuration randomization for moving target defense. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.153-159.

[49] Dunlop M, Groat S, Urbanski W, Marchany R, Tront J. MT6D: A moving target IPv6 defense. In *Proc. Military Communications Conference*, November 2011, pp.1321-1326.

[50] Jafarian J H, Al-Shaer E, Duan Q. OpenFlow random host mutation: Transparent moving target defense using software defined networking. In *Proc. the 1st Workshop on Hot Topics in Software Defined Networks*, August 2012, pp.127-132.

[51] Groat S, Dunlop M, Urbanksi W, Marchany R, Tront J. Using an IPv6 moving target defense to protect the Smart Grid. In *Proc. IEEE PES Innovative Smart Grid Technologies*, January 2012.

[52] Clark A, Sun K, Poovendran R. Effectiveness of IP address randomization in decoy-based moving target defense. In *Proc. the 52nd IEEE Conference on Decision and Control*, December 2013, pp.678-685.

[53] Zheng J, Namin A S. The impact of address changes and host diversity on the effectiveness of moving target defense strategy. In *Proc. the 40th Annual Computer Software and Applications Conference*, June 2016, Volume 2, pp.553-558.

[54] Wang H, Xi Z, Li F, Chen S. WebTrap: A dynamic defense scheme against economic denial of sustainability attacks. In *Proc. IEEE Conference on Communications and Network Security*, October 2017.

[55] Yeung F, Cho P, Morrell C, Marchany R, Tront J. Modeling network based moving target defense impacts through simulation in Ns-3. In *Proc. IEEE Military Communications Conference*, November 2016, pp.746-751.

[56] Huang Y, Ghosh A K. Introducing diversity and uncertainty to create moving attack surfaces for web services. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.131-151.

[57] Bardas A G, Sundaramurthy S C, Ou X M, DeLoach S A. MTD CBITS: Moving target defense for cloud-based IT systems. In *Proc. the 22nd European Symposium on Research in Computer Security*, September 2017, pp.167-186.

[58] Stoll C. The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage (1st edition). The Bodley Head Ltd, 1989.

[59] Kreutz D, Ramos F M V, Veríssimo P E, Rothenberg C E, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2015, 103(1): 14-76.

[60] MacFarland D C, Shue C A. The SDN shuffle: Creating a moving-target defense using host-based software-defined networking. In *Proc. the 2nd ACM Workshop on Moving Target Defense*, October 2015, pp.37-41.

[61] Wang K, Chen X, Zhu Y F. Random domain name and address mutation (RDAM) for thwarting reconnaissance attacks. *PLOS ONE*, 2017, 12(5): Article No. e0177111.

[62] Skowyra R, Bauer K, Dedhia V, Okhravi H. Have No PHEAR: Networks without identifiers. In *Proc. the 2016 ACM Workshop on Moving Target Defense*, October 2016, pp.3-14.

[63] Kampanakis P, Perros H, Beyene T. SDN-based solutions for moving target defense network protection. In *Proc. IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, June 2014.

[64] Chowdhary A, Pisharody S, Huang D. SDN based scalable MTD solution in cloud network. In*Proc. the 2016 ACM Workshop on Moving Target Defense*, October 2016, pp.27-36.

[65] Kil C, Jun J, Bookholt C, Xu J, Ning P. Address Space Layout Permutation (ASLP): Towards fine-grained randomization of commodity software. In *Proc. the 22nd Annual Computer Security Applications Conference*, December 2006, pp.339-348.

[66] Casola V, de Benedictis A, Albanese M. A moving target defense approach for protecting resource-constrained distributed devices. In *Proc. the 14th International Conference on Information Reuse and Integration*, August 2013, pp.22-29.

[67] Zeitz K, Cantrell M, Marchany R, Tront J. Designing a micro-moving target IPv6 defense for the Internet of things. In *Proc. the 2nd International Conference on Internet-of-Things Design and Implementation*, April 2017, pp.179-184.

[68] Kumar A, Aggarwal A. Lightweight cryptographic primitives for mobile ad hoc networks. In *Proc. International Conference on Recent Trends in Computer Networks and Distributed Systems Security*, October 2012, pp.240-251.

[69] Okhravi H, Riordan J, Carter K. Quantitative evaluation of dynamic platform techniques as a defensive mechanism. In *Proc. the 17th International Symposium on Research in Attacks, Intrusions and Defenses*, September 2014, pp.405-425.

[70] Hamlet J R, Lamb C C. Dependency graph analysis and moving target defense selection. In *Proc. the 2016 ACM Workshop on Moving Target Defense*, October 2016, pp.105-116.

[71] Green M, MacFarland D C, Smestad D R, Shue C A. Characterizing network-based moving target defenses. In *Proc. the 2nd ACM Workshop on Moving Target Defense*, October 2015, pp.31-35.

[72] Zaffarano K, Taylor J, Hamilton S. A quantitative framework for moving target defense effectiveness evaluation. In *Proc. the 2nd ACM Workshop on Moving Target Defense*, October 2015, pp.3-10.

[73] Taylor J, Zaffarano K, Koller B, Bancroft C, Syversen J. Automated effectiveness evaluation of moving target defenses: Metrics for missions and attacks. In *Proc. the 2016 ACM Workshop on Moving Target Defense*, October 2016, pp.129-134.

[74] Evans D, Nguyen-Tuong A, Knight J. Effectiveness of moving target defenses. In *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Jajodia S, Ghosh A K, Swarup V, Wang C, Wang X S (eds.), Springer, 2011, pp.29-48.

[75] Xu J, Guo P Y, Zhao M Y, Erbacher R F, Zhu M H, Liu P. Comparing different moving target defense techniques. In *Proc. the 1st ACM Workshop on Moving Target Defense*, November 2014, pp.97-107.

[76] Manadhata P K. Game theoretic approaches to attack surface shifting. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Jajodia S, Ghosh A K, Subrahmanian V S, Swarup V, Wang C, Wang X S (eds.), Springer, 2013, pp.1-13.

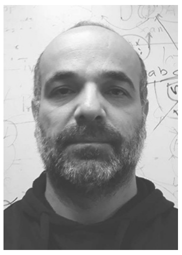
[77] Zhu Q Y, Başar T. Game-theoretic approach to feedback-driven multi-stage moving target defense. In *Proc. the 4th International Conference on Decision and Game Theory for Security*, November 2013, pp.246-263.

[78] Zheng J J, Namin A S. A Markov decision process to determine optimal policies in moving target. In *Proc. ACM SIGSAC Conference on Computer and Communications Security*, October 2018, pp.2321-2323.

[79] Carter K M, Riordan J F, Okhravi H. A game theoretic approach to strategy determination for dynamic platform defenses. In *Proc. the 1st ACM Workshop on Moving Target Defense*, November 2014, pp.21-30.

[80] Maleki H, Valizadeh S, Koch W, Bestavros A, van Dijk M. Markov modeling of moving target defense games. In *Proc. the 2016 ACM Workshop on Moving Target Defense*, October 2016, pp.81-92.

Jianjun Zheng received his first Master's degree in computer science and his second Master's degree in statistics in 2004 and 2013, respectively, both from Texas Tech University, Lubbock. He is currently a Ph.D. candidate in the Department of Computer Science, Texas Tech University, Lubbock. His research focuses on modeling moving target defense and network defense strategy optimization.



Akbar Siami Namin received his Ph.D. degree in computer science from University of Western Ontario, London, Canada, in 2008. He is currently an associate professor in the Department of Computer Science at Texas Tech University, Lubbock. His research interests include software engineering, cyber and software security, analytical reasoning, and machine learning. He has published over 80 research papers in premier software engineering and security venues. His research is funded by National Science Foundation of USA.