Moving target defence: Economics and asymmetry

Received (in revised form): 21st November, 2019

Don Maclean
Chief Cybersecurity Technologist, DLT – A Tech Data Company, USA

Don Maclean is Chief Cybersecurity Technologist for DLT – A Tech Data Company and formulates and executes cyber security portfolio strategy, speaks and writes on security topics and socialises his company's cyber security portfolio. Don has nearly 30 years' experience working with US Federal agencies. Before joining DLT in 2015, Don managed security programmes for numerous US Federal agencies, including the Department of Justice (DOJ), Department of Labor (DOL), Federal Aviation Administration (FAA), Federal Bureau of Investigation (FBI) and the Treasury Department. This experience allowed him to observe the strengths and limitations of traditional cyber security defences, leading to his interest in innovative technologies such as those featured in this paper. In addition to his CISSP, PMP and CCSK certificates, Don holds a BA in music from Oberlin, an MS in information security from the Brandeis Rabb School, and is a recipient of the FedScoop 50 award for industry leadership. An avid musician, Don organises a concert for charity every year, and has been known to compete in chess and Shogi (Japanese chess) tournaments, both in person and online.

Don Maclean, Chief Cybersecurity Technologist, DLT – A Tech Data Company, 2411 Dulles Corner Park, Herndon, VA 20171, USA
Tel: +1 571-346-1854; E-mail: Don.Maclean@DLT.com

Abstract In cyberspace, warfare is asymmetric. It takes only a small army of well-trained hackers to inflict major damage on a much larger adversary. Ironically, the inequity stems from standardisation. When bad actors find a vulnerability in a popular application or operating system, they can exploit it on millions of systems, yielding exponential reward for linear effort. The hacker's advantage, then, is economic rather than technical. Unless and until we reverse this dynamic, the adversary will have the advantage. Moving target defence (MTD), also called polymorphic defence, has the potential to diminish the enemy's asymmetric advantage. This paper surveys the major MTD technologies currently on the market and under development, with special attention to dynamic runtime environments. In particular, it explores how each technology might reverse, or at least mitigate, the economic leverage the enemy now exerts when discovering and exploiting vulnerabilities.

KEYWORDS: moving target defence (MTD), polymorphic, address space layout randomisation (ASLR), instruction set randomisation (ISR), attack surface, honeypot, honeynet, diversity

THE PROBLEM

Before examining MTD solutions, let us first define the problems they seek to solve and the environment in which they must be implemented. This will clarify both the obstacles to and prospects for success of these technologies.

Security philosophy: Maginot mentality

In the 1930s, the French famously constructed the elaborate and expensive Maginot Line, 'defensive fortifications built before World War II to protect the eastern border of France but easily outflanked by German invaders'.[1] Although historians debate the issue, the line is synonymous with 'a defensive barrier or strategy that inspires a false sense of security'.[2]

These aspects of the line are relevant today. On a technical level, the Maginot Line incorporated state-of-the-art defensive technology, but still failed in the face of faster, more modern German weaponry. In cyber security today, attackers consistently outwit defenders by moving more rapidly and using innovative methods of attack.[3]

On a human level, it created a false sense of security that caught the French off guard when the Germans attacked. In our world, we place far too much trust in a perimeter defence strategy, while failing to appreciate the danger posed by our adversaries.[4]

Most importantly, the Maginot Line consumed vast quantities of resources — financial, intellectual, human — yet failed to provide adequate safety. Similarly, the cyber security defence industry thrives, but fails to offer enough defence.[5] Indeed, the industry needs to avoid truly effective solutions, or it would put itself out of business.[6]

The MTD approach adopts a radically different philosophy.[7] MTD technologies seek agility over impregnability by constantly varying the attack surface available to the adversary. This approach raises several key questions, which this paper will address. What is the definition of 'attack surface'? How is it measured? How do the attacker and defender view the attack surface? What aspects of the attack surface should change, when should they change, how should they be changed, and who should do the changes? Most importantly, can MTD truly reverse the economic and time asymmetry currently favouring the bad actors?

THEORETICAL MODELS

MTD technologies borrow from or emulate models from other disciplines, some of which is described briefly below. I have based my comments and evaluations largely around the attack surface model.

- *Biology*: In nature, animals such as the mimic cctopus, disguise themselves to deceive predators. Some MTD technologies overlap with deception ('honeypot') technologies to mislead attackers;
- *Genetic diversity*: In nature, genetic diversity limits the spread of disease. Much of our DNA is identical to our fellow humans, but there is enough diversity to ensure that an epidemic, while devastating, is not entirely apocalyptic;
- *Military*: Deception is a nearly universal tactic in warfare, and a static target (such as the Maginot Line) is a vulnerable target;
- *Game theory*: Approaches based on game theory are common,[8] because 'the opposition, dependency, and noncooperative features in network confrontation are highly compatible with the feature of game theory'[9];
- *Attack graph*: Approaches based on an attack graph are also widely studied, since they can 'describe complex attack sequence that causes system state transition by considering vulnerability, attack goals, and node connectivity in targeted system simultaneously'[10];
- *Attack surface*: Reduction or transformation of attack surfaces is a significant goal of many MTD technologies and research. A definition follows.

ATTACK SURFACE: TOWARD A DEFINITION

Attack surface

If the goal of MTD is to complicate the attacker's task by altering or reducing the attack surface, we must codify the term.

The concept of attack surface is intuitive, but at present it lacks a formal, widely accepted definition. For this paper, I will use the definition formulated by Manadhata and Wing.[11] Their complete formal definition, and its underlying theory, is extensive and rigorous, and would exceed the space constraints of this paper. Therefore, I offer a summary version of their approach here. Readers interested in more detail can refer to the 2011 article 'A Formal Model for a System's Attack Surface'.[12]

Manadhata and Wing define an attack surface in terms of methods (M), channels (C) and untrusted data items (I), which form a set of *resources*. A method is a technique for breaching a system, a channel is the means of communication through which attackers exfiltrate data or introduce malicious input. An untrusted data item may be ephemeral or persistent and is any input that seeks to compromise the target system. They further state that:

> 'A resource is part of the attack surface if an attacker can use the resource in attacks on the system [whose] contribution to the attack surface measurement reflects the likelihood of the resource being used in attacks. For example, a method running with root privilege is more likely to be used in attacks than a method running with non-root privilege.'[13]

From this definition, they derive the desired metric: the *damage potential-effort ratio*, which 'indicates the level of damage an attacker can potentially cause to the system and the effort required for the attacker to cause such damage'.[14]

Since the elements in a resource are countable, the result is a sound metric for measurement of the attack surface. The authors created the metric to measure individual systems or environments. It is, however, essential also to view the attack surface from the attacker's perspective. While they may attack individual systems, they develop, leverage, weaponise or monetise exploits that work across a vast collection of potential targets.

MTD DESCRIPTION AND FRAMEWORK

Activities and taxonomy

Like 'attack surface' the term 'moving target defence' has no current formal definition. It is still possible, however, to set forth a reasonable structure to clarify what it entails. As its name implies, one or more things in an MTD system must 'move'.

The term 'moving target defence' is a metaphor, not a descriptor. Some MTD technologies, such as address space layout randomisation (ASLR), do in fact move code or data within memory. Other technologies alter systems in other ways, but do not actually 'move' anything. It might be more accurate then, to discuss 'what to alter'. For consistency, however, I will use the term 'move', keeping in mind its metaphorical nature.

A simple and clear description of the requisite 'moving' activities came from Sengupta *et al.*,[15] who outline three criteria for 'moving':

What to move

Broad categories for what to move in MTD:[16]

- Data;
- Memory (code, data, and flow-control mechanisms such as pointers);
- Applications;
- Dynamic runtime environments;
- Networks and platforms.

More specific items include the following:

- Instruction sets;[17]
- Address space layouts;[18]
- IP addresses, port numbers, proxies;[19]
- Virtual machines;[20]
- Flow-control mechanisms such as pointers, stack and/or heap addresses;
- Keywords and tokens.

How to move it[21]

The MTD approach emphasises speed and agility, so viable MTD technologies must be easy to operate, ideally automatic. Moreover, the move operation should not be transparent to the attacker, except that the attacker realizes that the attack has failed.

When to move it[22]

Some technologies can move/alter an attack surface:

- On demand;
- On a predetermined schedule;
- In response to an attack or attack predecessors.

In addition, Cho *et al.*[23] categorised MTD technologies under the broad rubric of 'shuffling, diversity and redundancy' (SDR).

Examples of shuffling would include changing IP addresses, TCP/UDP port numbers and automated failover/switching of virtual machines. Diversity includes deployment of similar systems that perform equivalent functions in different ways or to determine discrepancies. Redundancy involves multiple, identical systems for both resiliency and security.

Implementation

Implementation considerations will also be explored, specifically:

1. *Availability and maturity concerns*: Many MTD approaches are either purely theoretical or are in the early stages of research. A large number are not available to the public, although there are some open-source projects in play, along with some commercially available products;
2. *Performance impact*: Nearly all MTD technologies affect the performance of underlying systems. The impact of market-tested commercially available products can be measured accurately from experience with them in production. The impact of theoretical or early-research methods is likely to improve with further development. Performance impact is an indirect cost of using an MTD system;
3. *Expertise required for implementation and operation*: Implementation and operation are costs of any system, security or otherwise, and both require some level of technical knowledge. The lower the requisite skill level, the lower the cost. Since the goal is to invert the economic asymmetry of defence vs attack, this is an essential indicator of viability. Given the low maturity of so many MTD technologies, however, these costs tend to be estimates, rather than evidential or statistical;
4. *Hybrid MTD technologies*: One approach to MTD is to combine multiple technologies to complicate the attacker's task. Some technologies lend themselves readily to such scenarios, others are more suitable for stand-alone implementation and operation.

CURRENT SCENARIO

The technical aspects of cyber security are complex and fascinating for both attackers and defenders. Consequently, practitioners treat the battle as a contest of technological prowess. It may be more relevant, however, to view the issue in economic terms — a game of costs.

Defending a system and attacking a system both require the expenditure of resources, some combination of technical expertise, human effort and computing resources with their attendant expenses. When an attacker discovers an exploit, they gain access to millions of similar systems, allowing them to leverage their expenses exponentially. Moreover, they can recoup their costs or make a profit by selling their discovery — as packaged malware, as knowledge or as a server — to others.

Defenders, by contrast, leverage their

defence expenditures only across their own relatively small number of systems. When defenders discover an exploit, they might share it as a matter of civic responsibility, but generally do not recoup the cost of discovery, the occasional 'bug bounty' being the exception rather than the rule.

The attacker has a much stronger economic incentive than the defender. This disparity accounts for the untenable state of affairs in cyber security today. Bad actors simply have more leverage than the 'white hats'.

The goal of MTD is to reverse this dynamic, or at least minimise the disparity in economic incentives. When the attack surface changes constantly, the attacker loses the ability to leverage an exploit across multiple systems. They might gain access to a single machine or system but cannot reuse the exploit elsewhere. The defender does not seek the chimera of impregnability, only the minimisation of the value of the attacker's activities.

TECHNOLOGY SURVEY

MTD data technologies

Ward *et al.*[24] investigated or prototyped MTD methods applicable at the data level. At time of writing, none are available to the public for review or research. Most require a high level of expertise for implementation and all carry a performance penalty, ranging from trivial to significant. (Further development might mitigate these issues.)

Since these methods apply to data only as processed by specific applications, I see in them little promise for significant reversal of the asymmetry between the attacker and defender. I will therefore provide only a brief synopsis of these methods.[25]

Data diversity through fault tolerance

Data diversity through fault tolerance[26] uses multiple, independently developed copies of the same application. Given a specific input, they should generate the same, or semantically equivalent, output. A voting mechanism determines if a given output is unacceptable. If so, a warning or other action such as discarding the suspicious output, takes place. This technique straddles the line between the dynamic data and dynamic software categories, but Ward *et al.*[27] place it under the former, since the focus here is on evaluation of inputs.

SDR category: Diversity
Redundant data diversity

Like the previous technique, this method[28] uses multiple algorithms to create transformations of inputs — primarily inputs to system calls — and includes a voting mechanism to identify potentially illicit inputs.

SDR category: Diversity
Data randomisation

This method[29] randomises operands capable of unsafe memory reads or writes by XOR-ing them with a secret key before storage in memory and reversing the operation when the operand is used during execution. It uses multiple keys for each runtime instance, and the keys are different each time the programme executes.

SDR category: Shuffling
Diglossia

This technique[30] focuses on a single use case, ie sanitising SQL and NoSQL query input — a perennial problem in cyber security defence. Diglossia maps user-input queries to a shadow character set, then parses both the result and the original query, a method called dual parsing. If the two results are not syntactically isomorphic, the query is not processed.

The benefits of this method are its feasibility — far less broad than end-to-end software diversification, for example — and its applicability to a persistent issue. Many of the SQL injection issues, however, result from poor coding practices. Implementation of this method could allow such practices to continue, by relieving developers from the need to learn secure coding practices.

SDR category: Diversity, redundancy

NOMAD

This method[31] regularly updates HTML element names to thwart web bots. Like Diglossia, it has a narrow, but important focus. Unlike Diglossia, it would not compensate for poor coding practices.

SDR category: Shuffling

HERMES

HERMES[32] aims to prevent key theft by splitting keys across multiple virtual machines, using distributed RSA and threshold RSA. It would allow scheduled key rotation, customisable by the system administrator.

This technique has been tested, and hopefully code for it will soon be available. It has a specific focus on key protection, but broad applicability. Initial testing indicates that it is feasible in large-scale cloud environments.

SDR category: Shuffling

Content randomisation of Microsoft Office documents

This method[33] aims to defeat exploits embedded in MS Office documents by arbitrarily re-ordering components of object linking and embedding (OLE) and office open extensible markup language (OOXML). Bad actors continue to distribute malware through infected attachments and phishing campaigns, so this method addresses a key problem. It is limited, obviously, to MS Office documents, and it is easy to foresee bad actors finding ways to defeat or circumvent this approach.

SDR category: Shuffling

MTD software technologies

Here I will examine some general theoretical approaches as well as specific research efforts. Research at this level is thriving, so I have chosen efforts for which code is available or bear promise of effectiveness. Readers interested in more depth are referred to Cho *et al.*,[34] Zheng *et al.*[35] and Ward *et al.*[36]

Moving attack surfaces (MAS) for web services[37]

Huang and Ghosh[38] propose a method they call moving attack surfaces (MAS). MAS is a rotational scheme that uses a set of diverse but functionally equivalent virtual servers. Each server is configured differently and can be brought on- or off-line according to a schedule, or in response to an event (attack, attack indicator or other unforeseen circumstance). Selection of server is random to create uncertainty for the attacker.

The design entails two types of uncertainty: composition and reachability. The first forces the attacker to determine a software mix that may change during the attack, complicating their task at a minimum, and possibly defeating the attempt completely.

The reachability aspect increases the overall uncertainty for the attacker, since they will not know which server, at any given time, is responding to a request, or when it will go offline and be replaced by a functional equivalent.[39]

This approach bears promise in that it increases the attackers' cost and reduces their 'return on investment'; however, it also increases the defenders' costs. Determining and testing equivalent configurations, procuring software from multiple vendors and the operation of complex systems and troubleshooting all add to both the capital outlay and operating costs for the defender. It is unclear at this time if this approach will be worth the effort. Further research, testing and production deployment may clarify the answer.

SDR category: Diversity

ChameleonSoft[40]

Azab *et al*. propose a system they have dubbed ChameleonSoft. Inspired by biological and genetic diversity, ChameleonSoft seeks a lofty goal: encryption of software *behaviour*. Design principles include:

> 'Separating functional roles and runtime role players; devising intrinsically-resilient composable online programmable building blocks; separating logic, state and physical resources; and employing functionally-equivalent, behaviorally-different code variants.'[41]

ChameleonSoft is based on the authors' concept of cell-oriented architecture (COA).

System functions are assigned to 'cells', which are 'dynamically composable into organisms that are bound to functional roles at runtime'. Such construction supports online programmability, hot code swapping and automated recovery.

This approach entails significant changes to software development and deployment. Such changes are necessary in view of the current untenable state of cyber security, but it is difficult to recommend adoption of such wholesale system re-design without much more research and testing. Even so, their work appears fascinating, and their research may well bear fruit.

SDR category: Shuffling, diversity, redundancy

Web application diversity

In Taguinod *et al.*,[42] the authors outline a method for web application diversity. Acknowledging the need for low-level diversity techniques such as address space layout randomisation (ASLR) and instruction set randomisation (ISR), they point out that many breaches exploit weakness in high-level languages such as personal home page (PHP) and structured query language (SQL). Their research therefore entails automated language translation from Python to PHP and use of multiple SQL dialects in lieu of a single version.

Their research did not examine the circumstances that would dictate moving to PHP or to an alternate SQL dialect. They also acknowledge a central issue with their approach: 'we anticipate this approach to be resource and time intensive as it is essentially creating two implementations of one web application'.

SDR category: Diversity

End-to-end software diversification

This highly ambitious method seeks to randomise software at all levels: scripting, application programming interface (API) calls, keywords and syntax in HTML and HTTP and more. Given its focus on alteration of applications and protocols, it is not clear why the authors[43] classified it as a data method. The coordination across multiple systems raises concerns about feasibility, but research in this field could produce beneficial results for specific cases, eg randomising API calls. (Note: Ward *et al.* classify this method as a dynamic data technique, but in view of its focus on code rather than data, I regard it as a software technique.)

SDR category: Diversity

MDMS multitier diversification in web-based software[44]

Allier *et al.* developed a prototype blogging system called MDMS. MDMS can run in numerous configurations, facilitating diversification. It runs on Linux and Windows alike and can use multiple versions of Java virtual machine (JVM) and deploys on top of the RingoJS framework, which enables diversification through 'sosies', or multiple versions of a given application.

Multiple versions of an application are deployed simultaneously, so that the attacker faces multiple attack surfaces during an attempted intrusion. The authors' work was a proof of concept, and they recognised significant practical hurdles, such as load balancing and performance impact, before this method sees wide acceptance.

SDR category: Diversity

Security agility for dynamic execution environments

Fraser *et al.*[45] took an interesting approach to the MTD problem. They created a software toolkit to make applications aware of underlying security policies and respond dynamically to changes. The goal is to modify security policy automatically in response to attempted intrusions. This technique would require deployment on all systems, as well as a policy controller.[46]

This is an interesting way to implement MTD. The policy controller is a central point of failure, however, and accurate identification of intrusion behaviour is an ongoing difficulty in cyber security.

SDR category: Shuffling

Genprog: A generic method for automatic software repair[47]

This system is tangential to MTD, but is noteworthy nonetheless. Software patching, or lack thereof, is a chronic problem for system administrators. An intrusion succeeds against at least one victim, and then the developer must produce a patch to mitigate the vulnerability. That delay leaves all affected systems open to attack. While there are many systems for *applying* patches automatically, Genprog aims to *create* patches automatically. In the authors' words, Genprog 'uses existing test cases to automatically generate repairs for real-world bugs in off-the-shelf legacy applications'.[48]

Genprog's goals are worthy, but if a fix fails, systems could crash, or security could be compromised unexpectedly. Fixing the problem would fall squarely on the operators, it would seem, so Genprog could create more problems than it solves. The goal of speedy, near-real-time patching is, however, worthy of investigation.

SDR category: Diversity

MTD dynamic runtime environment technologies

Dynamic runtime environment technologies, along with moving target defence (MTD) networking approaches, has been the subject of much research and development: papers, projects, and commercially available products abound. A full list with even brief descriptions would far exceed the available space, so I have constrained this portion of the survey to technologies that are currently available, either as open source projects or on a commercial basis or bear exceptional promise for success.

Address space layout permutation

Kil *et al.*[49] have proposed address space layout permutation (ASLP), an enhanced address-space randomisation technique. They assert that ASLP provides 29 bits of entropy in a 32-bit architecture. It randomly relocates code and data segments, as well as the stack, heap and other memory regions, and they indicate minimal performance impact. Ward *et al.*[50] confirm the performance claims, noting that 'experimental results show an approximately 20% increase in executable size and memory footprint'.[51]

> 'This technique, and others like it, are extremely promising, as they bear the potential to mitigate whole classes of memory-based attacks, although the authors note that it does not yet offer stack frame randomization to guard against return-to-libc attacks.'[52]

SDR category: Diversity, shuffling
Dieharder

Novark and Berger[53] studied the heap allocators in Windows, Linux, FreeBSD, and OpenBSD, to understand their effect on security. Their allocator, DieHarder uses address randomisation, heap space and replication to randomize the heap itself.

The strategy used has three main elements: address randomisation, heap spacing and replication. Randomisation is done with a different seed for each instance and the heap is replicated in multiple locations in memory. A voting mechanism, like that of Data Diversity Through Fault Tolerance, identifies discrepancies and suspicious behaviour. Ward *et al.*[54] note a large performance penalty (50–100 per cent) and a substantial increase in memory consumption, in view of the heap replication.

This approach may provide some useful insights and techniques, but given the performance penalty and memory consumption, it will require more development before emerging as a viable method in production.

SDR category: Diversity, shuffling
Function-pointer encryption

Zhu and Tyagi[55] seek to protect against function pointer overwrites by encrypting each function pointer with a simple encryption scheme (*fp XOR *random_ number*), with the random number chosen differently for each program execution. They assert two benefits to this approach:[56]

> '(1) orthogonality of key space, (2) zero incremental knowledge gain for the adversary between two attacks on two different program runs.'

Ward *et al.*[57] evaluated the performance penalty at 4 per cent and stated an unquantified but 'likely small' impact on memory consumption.

This approach is elegant and simple and addresses a significant attack vector — buffer overflows.

SDR category: Shuffling
In-place code randomization

Pappas *et al.*[58] have taken on a major challenge: fighting return-oriented programming (ROP) attacks. ROP attacks are pernicious, because they use existing, legitimate code to perform malicious behaviour. The authors combine multiple techniques at the machine-code level. For instance, they replace instruction sequences with equal-length code of identical functionality; Ward *et al.*[59] cite the example of replacing addition with negative subtraction. Flow control structures are also altered with no effect on programme execution. The authors tested their system against well-known ROP attacks and against systems specifically designed to elude in-place code randomisation and claim to have defeated all such exploits.

Testing by Ward *et al.*[60] showed no memory overhead and 'negligible' performance impact.

This technology is extremely promising, given its minimal system impact.

SDR category: Shuffling
Morphisec

Morphisec is commercially available, so technical specifics are less readily available than for research projects.[61] The product randomises memory and retains a copy of non-randomised memory for troubleshooting and exploit identification.[62,63]

SDR category: Shuffling
Dynaguard

Dynaguard[64] is a highly specific technique for protection against a particular class of attack, namely blind return-oriented programming (BROP). A BROP attack seeks to defeat a protection mechanism called the stack canary. A stack canary is a random number placed on the stack, and it warns of a potential buffer overflow. Since both parent and child processes use the same stack canary, a BROP attack can reveal the canary to bypass various protections. Dynaguard's method is basically to alter the canary value of the child process.[65]

Although this method appears to have very low overhead in terms of performance and memory consumption, its specific nature suggests that it is a continuation of the ongoing 'cat-and-mouse' game between attackers and defenders, rather than a solution to a broad class of problems.

SDR category: N/a

ASLR-GUARD

Lu *et al.*[66] have developed ASLR-GUARD, in response to information leaks that let attackers identify the memory locations of code gadgets, despite the presence of ASLR. They assert that ASLR-GUARD can 'render leak of data pointer [*sic*] useless in deriving code address by separating code and data, provide a secure storage for code pointers'. The method protects against any type of exploit that reuses code maliciously. Testing by Ward *et al.*[67] showed a minimal performance impact (<1 per cent), a small increase in executable file size (6.26 per cent), and an increased load time (31 per cent).

This is a potentially strong technique, and further research may yield improvements in the load time issue.

SDR category: Shuffling, diversity

Polyverse

Full disclosure: Polyverse is a client of my company, DLT Solutions. Consequently, I will simply quote third-party evaluations and descriptions, to avoid unintentional bias.

Ward *et al.* describe Polyverse as follows:[68]

> 'Polyverse provides three different products. The first product is a compiler-based randomization technique. This provides an install-time randomization that scrambles the program binary generated from the source code without affecting the semantics of the program. The scrambling can be performed by simply pointing the Linux package manager at the proper repository (a one-line command). The second product applies a similar randomization to closed source applications where the source code is unavailable (primarily for the Windows operating system). This technique employs binary rewriting to apply a boot-time randomization to the layout and instructions of close-source binaries. The third product is a rapid cycling technology that can be applied to continuously running services (eg web servers) to periodically restore their environment to a pristine, good state.'

Ward *et al.* state that Polyverse has 'negligible' impact on performance and only a slight impact on load time. They evaluate Polyverse's weaknesses as follows:

> 'Two of the Polyverse products implement one-time randomization. Such techniques are vulnerable to information leakage, in which an attacker may be able to discover the location or content of relevant code to construct an attack. Unlike traditional ASLR, however, in which the disclosure of one address gives sufficient information for an attacker to infer the entire program's address space, under Polyverse, an attacker would require far more information to be leaked.'[69]

SDR category: Shuffling

Randomised instruction set emulation (RISE)

Barrantes *et al.* developed randomised instruction set emulation (RISE), which seeks to incapacitate injected code attacks, using a machine emulator to create diversified instruction sets. Legitimate code is encrypted at the byte level by XOR-ing them with a random key re-generated or chosen upon programme execution.[70] The primary goal is to thwart return to libc attacks. RISE runs atop the Valgrind IA32-to-IA32 binary translator, which adds considerable execution overhead: 400 per cent, per Ward *et al*.[71]

It is difficult to see how the performance penalty can be reduced enough to make this a viable technique.

SDR category: Shuffling

MTD dynamic networking and platforms

N-variant systems, proposed by Cox *et al*.[72] are an extension of N-version programming.[73] The system works by automatically creating and executing diversified programme variants, and a monitor determines if a single input results in equivalent outputs on all variants. It guards against attacks that inject malicious code or manipulate flow-control mechanisms. The authors' performance testing results are given in Table 1.[74]

The testing results look very strong; this method deserves further investigation, research and development.

SDR category: Diversity, redundancy

TALENT

The clumsily named system Trusted Dynamic Logical Heterogeneity[75] system (TALENT) appears to be anything but clumsy in operation. It leverages the agility of containers and a portable checkpoint compiler to migrate applications across platforms (in about one second, according to the authors) to elude attackers.

This is a significant result that could have implications beyond security. Fast migration across platforms could be useful in many other use cases.

SDR category: Shuffling, diversity

Reconnaissance deception system

Reconnaissance deception system, developed by Achleitner *et al*.,[76] aims to defeat malicious scanning of networks. The system uses software-defined networking (SDN) to implement five defence techniques:

1. *Dynamic address translation*: Rewrites packet headers in real time to hide the real host addresses and make the overall address space of a network appear larger.
2. *Route mutation*: Alters the topology of different network views so that a scanner cannot detect the real network topology.
3. *Vulnerable host placement*: Places vulnerable hosts in virtual subnets to increase the

Table 1: N-variant performance statistics

Configuration		1	2	3	4	5	6
Description		Unmodified Apache, unmodified kernel	Unmodified Apache, N-variant kernel	2-variant system, address partitioning	Apache running under Strata	Apache with instruction tags	2-variant system, instruction tags
Unsaturated	Throughput (MB/s)	2.36	2.32	2.04	2.27	2.25	1.80
	Latency (ms)	2.35	2.40	2.77	2.42	2.46	3.02
Saturated	Throughput (MB/s)	9.70	9.59	5.06	8.54	8.30	3.55
	Latency (ms)	17.65	17.80	34.20	20.30	20.58	48.30

duration a malicious scanner needs to identify them.

4. *Honeypot placement*: Increases the number of potential targets to confuse and slow the attacker.
5. *Dynamic detection of malicious flows*: Detects malicious flows trying to establish connections to honeypots or protected hosts.[77]

This technology more properly belongs in the deception category but is presented here out of general interest.

SDR category: Diversity

Open flow random host mutation (OFRHM)[78]

> 'OFRHM transparently mutates IP addresses with high unpredictability and rate, while maintaining configuration integrity and minimizing operation overhead, in which the OpenFlow controller frequently assigns each host a random virtual IP that is translated to/from the real IP of the host.'[79]

This technology is very promising, although it appears to extend the 'arms race' with the attacker, rather than to address a fundamental issue.

CONCLUSION

In general, MTD is an area of emerging research and development. Some areas, particularly the area of dynamic data, are in the very early stages, while others, such as dynamic network defences, are more mature. Dynamic runtime environments, which can defeat a wide range of attacks exploiting memory and flow-control mechanisms, have great promise, and some commercial products are currently available.

Assessing the effectiveness of these technologies is clearly a key issue. Despite the profusion of work in this area, some analysts do not regard the MTD with favour. David Evans and Anh Nguyen-Tuong, authors of *Effectiveness of Moving Target Defenses*,[80] are two such sceptics. Their work deserves serious attention, even if they are the minority voice at the time of writing.

In addition, combining MTD defences could increase the defenders' options dramatically and is an area of ongoing research. Hybrid defences could increase the attackers' costs by orders of magnitude.

Regardless of the state of the technology, I assert that the goal of MTD should not be to seek impregnability — an illusion like that of the Maginot Line — but to create an economic asymmetry that favours the defender. To win the war in cyberspace, we must first convince ourselves that we can win. There are many tools to enable victory, and I believe MTD technology is among them.

References

1. Maginot Line (n.d.), available at https://www.merriam-webster.com/dictionary/Maginot Line (accessed 17th April, 2020).
2. *Ibid*., note 1.
3. Lei, C., Hong-Qi, Zhang, Jonglei, T., Zhang, Y-C. and Liu, X-H. (July 2018), 'Moving Target Defense Techniques: A Survey', *Security and Communication Networks*, No. 2, p. 2.
4. Sengupta, S., Chowdhary, A., Sabur, A., Huang, D., Alshamrani, A. and Kambhampati, S. (2019), 'A Survey of Moving Target Defenses for Network Security', p. 1.
5. Morgan, S. (June 2019), 'Global Cybersecurity Spending Predicted to Exceed $1 Trillion From 2017–2021', Cybercrime Magazine, available at https://cybersecurityventures.com/cybersecurity-market-report/ (accessed 17th April, 2020).
6. Bradley, T. (October 2019), 'The Standard Cybersecurity Model Is Fundamentally Broken', *Forbes*, available at https://www.forbes.com/sites/tonybradley/2019/10/07/the-standard-cybersecurity-model-is-fundamentally-broken/#42df83581189 (accessed 17th April, 2020).
7. *Ibid*., note 7, p. 5.
8. Cai, G.-L., Wang, B.-S., Hu, W. and Wang, T.-Z. (2016), 'Moving target defense: State of the art and characteristics', *Frontiers of Information Technology & Electronic Engineering*, Vol. 17, No. 11.
9. *Ibid.,* note 2, p. 7.
10. *Ibid*., note 2, p. 5.
11. Manadhata, P. K. and Wing, J. M. (2011), 'A Formal Model for a System's Attack Surface', *Advances in Information Security Moving Target Defense*, pp. 1–2.

12. *Ibid*., note 11.
13. *Ibid.,* note 11, section 1.1.2.
14. *Ibid.,* note 11, section 1.1.2.
15. Ibid., note 4.
16. Ward, B. C., Gomez, S. R., Skowyra, R. W., Bigelow, D., Martin, J. N., Landry, J. W. and Okhravi, H. 'Survey of Cyber Moving Targets, Second Edition', Lincoln Laboratory, MIT.
17. Cho, J-H., Sharma, D., Alavizadeh, H., Yoon, S., Ben-Asher, N., Moore, T., Kim, D., Lim, H. and Nelson, F. (2019), 'Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense', *IEEE Communications Surveys & Tutorials*, Vol. 22, No. 1, pp. 709–745.
18. *Ibid.,* note 17.
19. *Ibid.,* note 17.
20. *Ibid.,* note 17.
21. *Ibid*., note 15.
22. *Ibid*., note 15.
23. *Ibid*., note 17.
24. *Ibid.,* note 16.
25. Interested readers can refer to Ward *et al*. (*ibid*., note 16) for more detail.
26. Ammann, P. E. and Knight, J. C. (1988), 'Data diversity: An approach to software fault tolerance', *IEEE Transactions on Computers*, Vol. 37, No. 4, pp. 418–425.
27. *Ibid*., note 16.
28. Nguyen-Tuong, A., Evans, D., Knight, J. C., Cox, B. and Davidson, J. W. (2008), 'Security through redundant data diversity', *IEEE International Conference on Dependable Systems and Networks*, pp. 187–196.
29. Cadar, C., Akritidis, P., Costa, M., Martin, J. P. and Castro, M. (2008), 'Data randomization', Technical Report TR-2008-120, Microsoft Research.
30. Son, S., McKinley, K. and Shmatikov, V. (2013), 'Diglossia: Detecting code injection attacks with precision and efficiency', Proceedings of the 2013 ACM Conference on Computer and Communications Security, pp. 1181–1191.
31. Vikram, S., Yang C. and Gu, G. (2013), 'Nomad: Towards non-intrusive moving-target defense against web bots', Proceedings of the 2013 IEEE Conference on Communications and Network Security, pp. 55–63.
32. Pattuk, E., Kantarcioglu, M., Lin, Z. and Ulusoy, H. (2014), 'Preventing cryptographic key leakage in cloud virtual machines', in USENIX Security, USENIX Association, pp. 703–718.
33. Smutz, C. and Stavrou, A. (2015), 'Preventing exploits in Microsoft Office documents through content randomization', Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses, Vol. 9404, Springer-Verlag, New York, pp. 225–246.
34. *Ibid.,* note 17.
35. Zheng, J. and Namin, A. S. (January 2019), 'A survey on the moving target defense strategies: An architectural perspective', *Journal of Computer Science and Technology*, Vol. 34, No. 1, pp. 207–233.
36. *Ibid.,* note 16.
37. Huang, Y. and Ghosh, A. K. (2011), 'Introducing Diversity and Uncertainty to Create Moving Target Attack Surfaces for Web Services', in Jajodia, S., Ghosh, A., Swarup, V., Wang, C. and Wang, X. (eds), *Moving Target Defense: Advances in Information Security*, Vol. 54, Springer, New York.
38. *Ibid*., note 37.
39. *Ibid*., note 37.
40. Azab, M. and Eltoweissy, M. (2012), 'ChameleonSoft: Software Behavior Encryption for Moving Target Defens', *Mobile Networks and Applications*, Vol. 18, No. 2, pp. 271–292.
41. *Ibid*., note 40.
42. Taguinod, M., Doupé, A., Zhao, Z.and Ahn, G-J. (2015), 'Toward a moving target defense for web applications', IEEE International Conference on Information Reuse and Integration (IRI), pp. 510–517
43. *Ibid*., note 16.
44. Allier, S., Barais, O., Baudry, B., Bourcier, J., Daubert, E., Flurey, F., Monperrus, M., Song, H. and Tricoire, M. (2015), 'Multitier diversification in Web-based software application', *IEEE Software, Institute of Electrical and Electronics Engineers,* Vol. 32, No. 1, pp. 83–90
45. Fraser, T., Petkac, M. and Badger, L. (2002), 'Security agility for dynamic execution environments', AFRL-IF-RS-TR-2002-229 Final Technical Report, DARPA (2002), pp. 1–15
46. *Ibid*., note 16.
47. Le Goues, C., Nguyen T., Forrest, S. and Weimer, W. (2012), 'Genprog: A generic method for automatic software repair', *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 54–72.
48. *Ibid.,* note 47.
49. Kil, C., Jun, J., Bookholt, C., Xu, J. and Ning, P. (2006), 'Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software', 22nd Annual Computer Security Applications Conference (ACSAC06).
50. *Ibid.,* note 16.
51. *Ibid.,* note 16, p. 73.
52. *Ibid.,* note 49, p. 9.
53. Novark, G. and Berger, E. D. (2010), 'DieHarder', Proceedings of the 17th ACM Conference on Computer and Communications Security – CCS 10.
54. *Ibid*., note 16, p. 77.
55. Zhu, G. and Tyagi, A. (2004), 'Protection against indirect overflow attacks on pointers', Proceedings of the 2nd IEEE International Information Assurance Workshop.
56. *Ibid*., note 55, p. 1.
57. *Ibid.,* note 16, p. 87.
58. Pappas, V., Polychronakis, M. and Keromytis, A. D. (2012), 'Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization', IEEE Symposium on Security and Privacy.
59. *Ibid*., note 16, p. 100.
60. *Ibid*., note 16, p. 101.
61. *Ibid*., note 16.
62. *Ibid*., note 16, p. 103.

63. Interested readers are referred to 'Advanced Endpoint Detection: Prevent the Most Dangerous Cyberattacks', Morphisec, available at www.morphisec.com (accessed 17th April, 2020).
64. Petsios, T., Kemerlis, V. P., Polychronakis, M. and Keromytis, A. D. (2015), 'DynaGuard, Proceedings of the 31st Annual Computer Security Applications Conference on – ACSAC 2015.
65. *Ibid*., note 64, p. 1.
66. Lu, K., Song, C., Lee, B., Chung, S. P., Kim, T. and Lee, W. (2015), 'Aslr-guard: Stopping address space leakage for code reuse attacks', Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, pp. 280–291.
67. *Ibid*., note 16, p. 141.
68. *Ibid*., note 16, pp. 150–152.
69. *Ibid*., note 16.
70. Barrantes, E., Ackley, D., Forrest, S. and Stefanovic, D. (2005), 'Randomized instruction set emulation', *ACM Transactions on Information Systems Security*, Vol. 8, No. 1, pp. 3–40.
71. *Ibid*., note 16, p. 164.
72. Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A. and Hiser, J. (August 2006), 'N-Variant Systems: A Secretless Framework for Security through Diversity', 15th USENIX Security Symposium, Vancouver, BC.
73. Chen, L. and Avizienis, A. (1995), 'N-Version Programming: A Fault-Tolerance Approach to Reliability Of Software Operation', Twenty-Fifth International Symposium on Fault-Tolerant Computing, Highlights from Twenty-Five Years', Pasadena, CA, p. 113.
74. *Ibid*., note 73, p. 12.
75. Okhravi, H., Comella, A., Robinson, E. and Haines, J. (2012), 'Creating a cyber moving target for critical infrastructure applications using platform diversity', *International Journal of Critical Infrastructure Protection*, Vol. 5, No. 1, pp. 30–39.
76. Achleitner, S., La Porta, T., McDaniel, P., Sugrim, S., Krishnamurthy, S. and Chadha, R. (2016), 'Cyber deception: Virtual networks to defend insider reconnaissance', Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats, ACM, pp. 57–68.
77. *Ibid*., note 76, p. 59
78. Jafarian, J. H., Al-Shaer, E. and Duan, Q. (2012), 'Openflow random host mutation', Proceedings of the First Workshop on Hot Topics in Software Defined Networks – HotSDN 12.
79. *Ibid*., note 78.
80. Evans, E. and Nguyen-Tuong, A. (2011), 'Effectiveness of Moving Target Defenses', in Jajodia, S., Ghosh, A. K., Swarup, V., Wang, C. and Wang, X. S. (eds), *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Springer, New York, pp. 29–46.