Lessons Learned in Network and Memory-Based Moving-Target Defenses

Richard Skowyra
richard.skowyra@ll.mit.edu
MIT Lincoln Laboratory

Samuel Jero
samuel.jero@ll.mit.edu
MIT Lincoln Laboratory

ABSTRACT

Moving-Target Defenses seek to introduce dynamism, randomness, and diversity into computer systems in an effort to make these systems harder to explore, predict, and exploit. Over the past decade a variety of work has explored applying these kinds of defenses to applications' runtime environments ,to the operating systems and architectures running the applications, and to networks.

In this paper, we report on lessons learned from seven years of building and evaluating moving-target defenses, primarily for process memory layouts and networks. We identify six major lessons learned from our experience that we believe to be broadly applicable to moving-target defenses, focusing around the importance and impact of threat models and characteristics of effective moving-target defenses. We then offer suggestions for the future direction of the field based on our experience.

ACM Reference Format:
Richard Skowyra and Samuel Jero. 2020. Lessons Learned in Network and Memory-Based Moving-Target Defenses. In *7th ACM Workshop on Moving Target Defense (MTD'20), November 9, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3411496.3421227

1 INTRODUCTION

That computer systems are frequently insecure and attacked by all manner of adversaries should come as no surprise to anyone. Common attacks today may exploit a variety of issues in the computation stack, from buffer overflows or use-after-frees in individual applications, to unpatched vulnerabilities in operating systems and drivers, or ineffective network segmentation. moving-target defenses seek to raise the bar for the attacker by introducing dynamism, or movement, into an otherwise static computer system. For example, instead of using identical memory layouts every time a binary executes, a moving-target defense may randomize the memory layout every time it is loaded or even at runtime. As this memory layout is critical to writing an exploit for a buffer overflow or use-after-free bug, this greatly increases the difficulty of attacking the system.

This core moving-target idea of defending systems by introducing dynamism, randomness, or diversity to make exploiting a system harder is broadly applicable across domains and not constrained to defending against one particular class of exploits or vulnerabilities. As a result, a wide variety of defenses relying on moving-target techniques have been developed over the last decade or more, including defenses applying randomization to process memory layout to protect against memory safety exploits [4, 11, 12, 20, 34], defenses applying dynamism to network addresses to protect against network mapping or denial-of-service [5, 13, 14, 36], defenses applying dynamism to packet headers to protect against traffic analysis [25, 31], and defenses applying diversity to the operating systems running some service to protect against vulnerabilities in particular operating systems [1, 18], as well as others [8, 30, 35].

While moving-target defenses have been successfully applied in a wide variety of use cases, there remain several challenges. One of the biggest is that moving-target defenses provide probabilistic protection. In other words, attackers are not prevented outright from exploiting the system; the dynamism or randomization introduced merely makes exploitation extremely unlikely. As a result, moving-target defenses may be rendered ineffective by attacks that can follow the dynamism at runtime [7], by information leakage attacks that can be used to determine the current configuration for exploitation [22, 33], or by attacks that use other information that is not perturbed by a given defense. As a result, moving-target defense is still an active area of research.

We have been building and evaluating moving-target defenses for more than seven years. Over the course of that time, we have built moving-target defenses for network packet headers [25] and network connectivity [8], as well as evaluated a variety of other moving-target defenses in these areas and developed attacks that circumvent them [1, 21, 26, 33]. Finally, we have taxonomized and examined a large portion of the existing work in this area [32].

In this paper, we reflect on our experiences over the last seven years and offer six lessons learned that we hope will be valuable for the community. These lessons center around a need to carefully consider the threat model and attacker capabilities when designing or deploying moving-target defenses, as well as identifying what information movement

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited. This material is based upon work supported by the Department of Defense under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
MTD'20, November 9, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8085-0/20/11.
https://doi.org/10.1145/3411496.3421227

is most effectively applied to when creating moving-target defenses. We then offer insights as to key directions for the future of the field.

2 BACKGROUND

Moving-target techniques seek to disrupt or prevent the successful execution of a cyber-attack by adding dynamism, randomness, or diversity to an otherwise static system component. While they may be taxonomized in several ways, we find the categorization presented in [32] to be helpful for the purpose of this paper. Under their system, a defense falls into one of several categories based on the layer of the computation stack where dynamism is added:

- *Dynamic data* techniques change the representation of application data. This does not generally include encryption of data at rest or in transit, though cryptographic primitives may be used as part of a larger defensive technique.
- *Dynamic software* changes the application's code in memory, while preserving its higher-level execution semantics. For example, a multi-compiler may emit different binaries with equivalent functionalities.
- *Dynamic runtime environments* permute or add dynamism to the environment an application executes in, such as memory layout randomization.
- *Dynamic platforms* add dynamism to the underlying operating system while preserving (or adapting) its system call interface. Many virtualization-based approaches fall into this category, for example.
- *Dynamic networks* add dynamism to network structure (e.g., topology), addressing, or protocols. Examples include software-defined networking approaches that modify network topology over time, or permute network packet headers.

This paper will not focus on rigorously taxonomizing all referenced techniques. Nor will it argue that specific categories are more or less effective against certain threat models. However, many of the lessons discussed here are especially applicable to, or represent challenges naturally arising in the context of, particular categories of moving-target defense.

3 LESSONS LEARNED

Lesson 1: Attackers can use APIs too

One danger when considering attacker threat models, especially in enterprise environments, is assuming that the attacker does not have full access to the same system and network services that benign applications do (or not considering the full scope of those services). In particular, it is common for modern attack campaigns on enterprise networks to take advantage of system features such as Windows Powershell, LDAP services, DHCP, and Server Message Block (SMB) shares. All of these services are used by benign applications, yet each affords the attacker powerful reconnaissance and lateral movement tools.

As an example case study, consider the 2017 NotPetya attacks [27]. The malware was originally introduced via a software supply-chain attack that compromised a software update server. When a victim downloaded an update, they also became infected with the malware. From that original entry point, NotPetya queryed Active Directory domain controllers and DHCP servers for subnet configuration data in order to begin mapping the network. It conducted credential theft and remote process execution through valid Windows system administration tools. While it was able to spread via exploitation of SMB shares, credential theft was the primary vector for lateral movement [24]. With the exception of these potentially-unnecessary SMB exploits, the remainder of NotPetya's attack chain used existing APIs for the purposes they were intended for, albeit to malicious ends.

This case study is not unique. Similar techniques were used in the 2017 Equifax breach [16] and 2015 Anthem breach [17]. Enterprise environments require rich service APIs in order to support remote management, access control, system provisioning and maintenance, and consistency across distributed services. If an attacker can obtain sufficient privileges to utilize these APIs, there is little to distinguish an attacker from a legitimate system adminstrator or service. Disabling these APIs is also not a realistic option, as it would also impede system administrators trying to stop the spread of malware. Indeed, in the case of NotPetya, Microsoft advised that "if a threat actor has acquired the credentials needed for lateral traversal, you can NOT block the attack by disabling execution methods like PowerShell or WMI. This is not a good choke point because legitimate remote management requires at least one process execution method to be enabled [24]."

The lesson here is that moving-target defenses rely on the attacker needing capabilities unavailable through normal APIs. When malware can largely limit its activities to legitimate APIs and services, it is not clear what targets to move in order to disrupt the attack, that will not also disrupt legitimate applications. IP address randomization, for example, would be ineffective here, since the malware can simply query the relevant servers for configuration data also used by legitimate applications. Memory randomization to stop exploitation via memory corruption may mitigate the SMB exploits, but will do nothing to stop remote process execution via stolen credentials.

That said, we think this represents an opportunity for the moving-target community to reconsider how their defenses are designed, and the threat models that drive them. Attackers must still operate outside the bounds of a legitimate user (e.g., by stealing credentials), and it is at these points that moving-target defenses will be most capable at halting an attack with minimal impact on benign operators.

Lesson 2: moving-target defenses are most effective when hiding information only the attacker needs

moving-target defenses fundamentally try to hide some information from the attacker. We have observed that the choice

of that information is crucially important to the effectiveness and overhead of a defense. In particular, we have found that the moving-target defenses that are most effective and least costly focus on hiding information that is valuable to an attacker but of little to no value to legitimate users.

Coarse memory randomization defenses, like ASLR [28] and DieHard [2], that randomize the location of the text and data sections of a process at load time are great examples of this. It turns out that there are almost no legitimate reasons for one process to care about the memory layout of another process. Debuggers, like `gdb`, are perhaps the one exception, but are rarely used on production machines and typically require the debugged program to be started with special flags or environments anyway. As a result, memory randomization defenses can freely randomize process address spaces without having to develop complex mechanisms to keep the rest of the system in sync (such mechanisms can also be used by an attacker, see Lesson 1). This reduces the overhead of the defense, as little metadata needs to be maintained, and improves the effectiveness, as the defense offers no way to identify randomized elements.

IP address randomization schemes, like RPAH [14], where the IP addresses used to communicate with legitimate devices on the network are automatically changed on a frequent basis, are examples of the opposite situation. Here, basically any device that communicates over the network needs to know the IP address of the hosts it communicates with. This includes essentially every legitimate device on a network: workstations, IP phones, printers, cameras, and a host of others. It also includes every router, which needs to make decisions and optimizations based on those addresses. As a result, these schemes require some mechanism to coordinate state across the whole system in a consistent manner and make it queryable by legitimate devices. This is frequently done using SDN and DNS, for IP hopping schemes [23, 36]. Generally, however, this type of coordination tends to be expensive and centralized and introduces a whole host of consistency challenges (see Lesson 6). Additionally, this query API is a possible channel for the attacker to learn and circumvent the randomization (see Lesson 1). As a result, these schemes tend to be expensive and offer easy paths for attackers to circumvent.

Another important realization is that this space is a spectrum. We have so far talked about defenses where the information being hidden is either of no value or extreme value to the rest of the system. Most defenses fall somewhere in the middle. Consider defenses that rerandomize a binary dynamically at runtime, like TASR [4]. While other processes do not care about the memory layout of the process, the process itself does. In particular, any given process is likely to have many code and data pointers in use and those would need to be updated if the location of code or data changes. This gets much worse if doing fine-grained randomization, where memory *within* the code and data segments is reordered, such that jumps within the code segment need to be changed. For defenses like these, metadata definitely needs to be maintained to allow the defense to quickly and accurately update the process image. Updating this metadata and the process image will definitely result in overhead. However, this metadata is specific to a single process, meaning that it does not introduce consistency challenges across many system components. Further, because the rerandomizing component is the only thing that needs this metadata and the new memory locations, no API is required for sharing this information.

Overall, we recommend those designing or deploying moving-target defenses to consider what information is being protected by a defense and focus on information that is of great value to an attacker while being of little use to legitimate elements of the system. We have observed that such systems usually have better effectiveness and reduced overhead.

Lesson 3: Threat model and system type matter

Threat models drive the development of defenses. A good threat model realistically considers what capabilities attackers have and what constitutes attacker success. The efficacy of a defense should be able to be quantitatively or analytically evaluated with respect to this threat model. Unfortunately, it is difficult to quantitatively demonstrate how much a defense makes it 'harder' to attack a system. The security community lacks standard metrics of success, and despite recent work on developing a science of security [10] this situation is unlikely to change in the near future. F Lacking standard metrics, some early moving-target defenses nonetheless provided a quantitative evaluation by claiming that some measurable quantity is linked to the attacker's difficulty, cost, or probability of success. This is dangerous if it implicitly weakens the attacker by restricting their capabilities, and can lead to a false sense of security. For example, early memory randomization schemes [3, 6] assumed that once memory layout was permuted, attackers would be limited to guessing the offsets of useful code. Thus, metrics based on probability and entropy were used to evaluate defenses. Not long after, the advent of memory-disclosure attacks demonstrated the limitations of such metrics, as attackers were no longer forced to guess memory offsets. The metrics implicitly limited attackers into a proscribed set of actions that were not reflective of their actual capabilities.

Similarly, some leakage-resilient memory randomization defenses evaluated the number of code reuse gadgets that were removed [11, 20, 34] or the granularity of randomization [12], as a ad-hoc security metric. The problem is that this is again making an implicit assumption about what an attacker can or cannot do. Using fine-grained randomization to make 99% of gadgets unavailable, for example, does not necessarily make an attack 99% harder or reduce the attacker's probability of success to 1%. It restricts the number of building blocks available to an attacker, but there is no guarantee that the removed gadgets were critical to attack success. Page-level randomization, for example, removes almost all gadgets available to the attacker by randomizing their location in memory. Yet, RelROP [33] attacks demonstrate that there can be enough gadgets in a single page of memory to launch a shell.

A related problem arises when moving-target defenses make implicit assumptions about what constitutes attacker success. Many moving-target defenses (e.g., memory randomization) inherently convert an attack on system integrity (e.g., memory corruption) into an attack on availability, by causing undefined behavior, trapping, or crashing a program in response to an attack. This is implicitly considered to be a better scenario for the defender when evaluating the technique. There are two issues with this approach.

First, it presupposes that the attacker's goal is not denial-of-service. Depending on the cost to restart a process and the overhead imposed by the defense, a moving-target defense could transform a normally non-crashing bug into valuable channel for denial-of-service attacks. Second, it assumes that an attack on availability is less harmful than an attack on integrity or confidentiality. This may be true in conventional computing (e.g., web servers), but in some environments unexpectedly crashing the process is entirely unacceptable. Cyber-physical systems, for example, use a software controller to sense and actuate physical processes. These include automotive braking controllers, flight control software, industrial plant controllers, and electrical grid management systems. In these, a software crash can result in physical damage or harm to human beings. If a moving-target defense increases the probability of an unexpected crash in the event of an attack, it may not improve the overall security of the system.

Overall, we recommend against attempting to informally quantify how a defense impacts an attacker's cost, difficulty, or probability of success. Many metrics make implicit assumptions about what attackers can or cannot do, and these may not hold up in reality. In particular, it is dangerous to claim that a moving-target defense makes a particular kind of attack 'too hard' for an adversary. Additionally, it is important to be clear about the impact of a defense not only on system integrity or confidentiality, but also on availability.

Lesson 4: Used improperly, moving-target defenses can help attackers

moving-target defenses add dynamism to a previously static system component. When done well, this effectively hides information needed for an attacker to succeed. However, if the impact of additional dynamism is not carefully considered, it may hinder legitimate users or even play to the advantage of the attacker. In particular, we have observed two kinds of detrimental effects that can arise from improper use of moving-target defenses.

First, care must be taken when using moving-target approaches that 'rotate' through one of several system configurations. TALENT [18], for example, provides platform diversity by moving a software container across several operating systems over time. The danger here is that depending on the threat model (see Lesson 3), such a defense may expose the union of all the vulnerabilities present in the rotated set of system components. For example, consider an application whose host OS is rotated through Windows, Linux, and Mac distributions. If an attacker needs to maintain constant persistence on the system, this may effectively hinder them by requiring a compromise of every OS instance. However, if the attacker only needs to cause a one-time effect, then a vulnerability against any one of the three operating systems will suffice. The defense has effectively aided the attacker by allowing them to leverage a wider attack surface. This phenomenon has been explored analytically in [19].

Second, the dynamism added by moving-target defenses may make it difficult or impossible for system administrators to reconstruct a past system state or to collect meaningful forensics. This is especially challenging when deploying a network-based moving-target system that adds dynamism to packet header fields [23, 25] or the network topology itself [8]. Network operators attempting to determine where an attack originated, what endpoints may be compromised, or how an attacker spread through the enterprise must be able to examine the network at past points in time. This is much easier if conventionally long-lived state, such as IP addresses or network routes, have changed only minimally over time. If they have been rapidly permuted by a moving-target defense, operators must trace those permutations backward in time in order to reconstruct the network state. Depending on the size of the network, the frequency of permutation (see Lesson 5), and the amount of time in question, a very large volume of meta-data may need to be maintained in order to support diagnostics that would otherwise be straightforward and low-cost.

Overall, we recommend that researchers working on moving-target defenses weigh the effects of the defense against what the attacker may be trying to achieve (see Lesson 3) and the impact of dynamism on system auditing and maintenance. Dynamism can aid, or hide, malicious activities when improperly used. If a defense adds software to a system, researchers should address how this changes attack surface. If a defense permutes information that is routinely logged, researchers should strive to provide a low-cost mechanism for tracing permutations backward in time.

Lesson 5: Timescale of movement must match threat model

Another important lesson we have learned is that it is essential for the time scale of the dynamism in a moving-target defense to match the threat model in order for that defense to be effective. Because moving-target defenses operate by hiding information from attackers, their nemeses are information leakage attacks. These attacks expose the current configuration of the information being hidden and, if acted on prior to an application of dynamism, allow an attacker to bypass the moving-target defense. Unfortunately, information leakage attacks are common and increasingly used by attackers [9, 29]. Effective moving-target defenses will take this into consideration in their threat models and use timescales of movement that minimize this risk of information leakage being weaponized.

Of course, the timescale of movement is a trade off between performance and security. Pick too fast of a timescale and

you waste time unnecessarily changing the system. However, if you pick too slow of a timescale, an attacker can easily defeat the system by using information leakage attacks to learn the current configuration of the system and then create an attack for that configuration. This too slow case is much harder to identify for because the performance looks good and untuned attacks are stopped. Ultimately, a moving-target defense's threat model should define how quickly attackers can weaponize an information leak and that should be the driving factor for the timescale of movement.

An excellent example of this is TASR [4], a memory randomization defense that rerandomizes a process's address space at runtime. The key insight of TASR, however, is less about how the rerandomization is actually done as *when* it is done. The most obvious thing to do would be to randomize the process's address space periodically, every n seconds, for instance. However, the key insight was that launching an information leakage attack against a remote network daemon and exploiting the results requires at least two I/O operations, one output and one input. In particular, one output operation is required to leak the process memory layout and one input operation is required to exploit it. TASR, therefore, chooses to rerandomize immediately after any input operation that follows one or more output operations, ensuring that an adversary is unable to exploit any information leakage attack they may launch. The careful consideration of information leakage attacks and threat model here results in a defense that effectively mitigates information leakage attacks with minimal overhead.

In contrast, many defenses focus exclusively on how to dynamically modify a system and do not really consider how often this should be done for meaningful protection. This significantly hurts their effectiveness because users are effectively given a knob to choose between performance and some abstract definition of security, with no real guidance about how to set said knob. For concreteness, imagine a defense that dynamically migrates applications between operating systems, like TALENT [18]. In this context, an information leakage attack looks like determining the current operating system being used to run the application. Fingerprinting an operating system in this manner is fairly easy, thanks to many tools like `nmap` [15], and looking up vulnerabilities and exploits for a given operating system version is not very challenging either. This means that information leakage is definitely possible and possible to weaponize quickly. Worse, because checkpointing and migrating applications is expensive, it needs to be done infrequently to achieve good performance. The needed balance between these competing requirements is simply not obvious without reference to a threat model.

Overall, we observe that most moving-target defenses have concentrated on mechanism, with policy like timescale of movement being extremely dependent on the threat model being considered. We also observe a striking lack of research into reasonable threat models for moving-target defenses that would attempt to answer this question. Instead researchers and administrators are largely left guessing about appropriate threat models and parameters, like timescale of movement.

Lesson 6: moving-target defenses must preserve system-wide consistency

Although moving-target defenses are based on dynamic movement, our experience has shown that when it comes to system-wide properties that other legitimate components depend on, consistency is absolutely critical. In fact, we find that ensuring this consistency and debugging transient inconsistent state takes up the majority of the engineering effort in building these kinds of defenses.

Ideally, one ensures system-wide consistency by avoiding dynamically changing any state that must be visible across the whole system, as discussed in Lesson 2. However, this is not always feasible, especially for network defenses that are fundamentally distributed in nature. For such defenses that involve system-wide state, the entire system must be architected so as to provide consistency and make identifying any inconsistencies that do occur as easy as possible.

Our experience building DFI [8] provides a good example. DFI is a network moving-target defense designed to dynamically configure the network with only the connectivity needed at any moment in time. For example, a workstation with no users logged in will only have connectivity to the authentication server, but as soon as a user logs in, connectivity will be added to services appropriate for the user's role in an organization. In a traditional network, by contrast, each device's connectivity is static and represents the maximum connectivity that device could ever need. DFI is built using a variety of sensors to collect authentication events and network identifier binding information (e.g., IP-hostname or MAC-IP mappings) and then a Software Defined Network (SDN) to update the connectivity. All of this information is visible across the system and needed to be kept consistent. In particular we had 4 mappings that needed to be kept consistent both across our defense and with the rest of the network infrastructure: switch port to MAC address, MAC address to IP address, IP address to hostname, and hostname to logged in users. Additionally, the flow rules in multiple SDN switches needed to be kept consistent with our defenses understanding of allowed connectivity.

Although we were able to quickly prototype an initial concept for DFI, it took well over a year of engineering effort to work out all the consistency bugs and get the system working reliably for sustained periods of time. Just one of the bugs we faced was where a user would log in, but the first packets from the user would arrive before we got the notification that they logged in, resulting in their traffic being blocked until they logged out and logged back in. This actually involved two consistency issues: first, flow rules were being kept in the switches after they were inconsistent with the traffic DFI would currently allow and, second, network traffic was beating the notification that they logged in to our system. For the first, we added logic such that when connectivity changes in the network flow rules that are now

inconsistent with that connectivity are removed from the switches. For the second, we experimented with a variety of different login notification options and eventually found one that was faster and fast enough that traffic did not get blocked in practice. This was, of course, just a single consistency related bug, among many, that we encountered during development.

These issues are not unique to DFI or networks, but are a general feature of moving-target defenses that modify system-wide state. moving-target based systems that change the operating system on which a program runs, like TALENT [18], would be another good example. In such systems, the entire state of the program and any operating system interactions must be tracked such that it can be moved to a new operating system correctly. Correctly tracking, preserving, and replicating things like open file descriptors is a major challenge and a large engineering effort. Bugs in this will result in state inconsistencies that will be extremely hard to track down and fatal to the running program.

The key take away from this discussion is that the challenges in implementing a moving-target defense are often not about what dynamism to introduce into the system or how to introduce it, but rather how to preserve consistency so that as dynamism occurs the rest of the system continues to operate correctly.

4 FUTURE DIRECTIONS

Stepping back to consider the field of moving target research as a whole and looking towards the future, we see two major directions for future work.

First, we argue that the community needs to recognize that what dynamism is applied to is much more important than exactly how that dynamism is created. In particular, we argue that future work needs to focus on identifying elements of a system to which it would be most valuable to apply dynamism. As discussed earlier (Lesson 2), these are usually elements of the system that provide data that attackers need but not legitimate components. Further, because outside components do not need access, there is no need for APIs that attackers can take advantage of (Lesson 1) and the implementation and performance of the system improves (Lesson 6).

Much moving target work has been confined to demonstrating that dynamism can be applied to a given element or incrementally improving the dynamism of a common element (e.g., IP Addresses, Memory Address Space). While these are good contributions, for our community to advance, we need to begin to understand where moving target defenses provide value and how to slice major security problems in such a way that our defenses are able to maximally impede attackers while having minimal impact on legitimate users.

While we propose that focusing on applying dynamism to elements of the system that are crucial to the attacker, but not to legitimate components, is a good marker, it is also only an initial marker. There are undoubtedly other markers for problems and places that moving target defenses will be effective. Much research is yet needed.

Second, it is essential for research in this area to consider well-defined and reasonable threat models. A number of our lessons learned (1, 3, 4, and 5) highlight the importance of having a correct threat model before designing or deploying moving target defenses and the dangers that come from not doing so. While it can be tempting to design a moving target defense under the assumption that any dynamism is beneficial and leave threat models for determining what defenses apply where, our experience indicates that these defenses are most effective when designed with a realistic threat model in mind. Additionally, research is also needed on appropriate threat models for moving target defenses. A promising starting point may be systematic examination of threat intelligence and case studies of real-world attacks. Novel defenses may need to leverage deep understanding of both modern attacker tactics, and the complex and distributed systems that are being attacked.

5 CONCLUSION

Moving target defenses introduce dynamism into computer systems in an effort to make these systems harder to exploit. Over the past decade a variety of work has explored applying these kinds of defenses to runtime environments to protect process memory layout, to the operating systems and architectures running target applications, and to networks.

We have presented six lessons learned from our seven years experience in the building and evaluation of moving target defenses. These lessons center around a need to carefully consider the threat model and attacker capabilities when designing or deploying moving target defenses and identifying what information dynamism is most effectively applied to when creating moving target defenses. We also suggest that future work on moving target defenses needs to consider carefully exactly what element of the system dynamism is applied to and what an appropriate and correct threat model is for the target system.

REFERENCES

[1] Kevin Bauer, Veer Dedhia, Richard Skowyra, William Streilein, and Hamed Okhravi. 2015. Multi-variant execution to protect unpatched software. In *2015 Resilience Week (RWS)*. 1–6.

[2] Emery D Berger and Benjamin G Zorn. 2006. DieHard: probabilistic memory safety for unsafe languages. *ACM SIGPLAN notices* 41, 6 (2006), 158–168.

[3] Sandeep Bhatkar, Daniel C DuVarney, and R Sekar. 2005. Efficient Techniques for Comprehensive Protection from Memory Error Exploits.. In *USENIX Security Symposium*.

[4] David Bigelow, Thomas Hobson, Robert Rudd, William Streilein, and Hamed Okhravi. 2015. Timely rerandomization for mitigating memory disclosures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 268–279.

[5] Sang-Yoon Chang, Younghee Park, and Bhavana Babu Ashok Babu. 2018. Fast IP hopping randomization to secure hop-by-hop access in SDN. *IEEE Transactions on Network and Service Management* 16, 1 (2018), 308–320.

[6] Cristiano Giuffrida, Anton Kuijsten, and Andrew S Tanenbaum. 2012. Enhanced operating system security through efficient and fine-grained address space randomization. In *USENIX Security Symposium*. 475–490.

[7] Enes Göktas, Benjamin Kollenda, Philipp Koppe, Erik Bosman, Georgios Portokalidis, Thorsten Holz, Herbert Bos, and Cristiano Giuffrida. 2018. Position-independent code reuse: On the effectiveness of ASLR in the absence of information disclosure. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. 227–242.

[8] Steven R Gomez, Samuel Jero, Richard Skowyra, Jason Martin, Patrick Sullivan, David Bigelow, Zachary Ellenbogen, Bryan C Ward, Hamed Okhravi, and James W Landry. 2019. Controller-Oblivious Dynamic Access Control in Software-Defined Networks. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 447–459.

[9] Mordechai Guri. 2015. ASLR - What it is, and what it isn't. https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/

[10] Cormac Herley and Paul C Van Oorschot. 2017. Sok: Science, security and the elusive goal of security as a scientific pursuit. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 99–120.

[11] Jason Hiser, Anh Nguyen-Tuong, Michele Co, Matthew Hall, and Jack W Davidson. 2012. ILR: Where'd my gadgets go?. In *IEEE Symposium on Security and Privacy*. 571–585.

[12] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning. 2006. Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software. In *22nd Annual Computer Security Applications Conference (ACSAC)*. 339–348.

[13] Henry CJ Lee and Vrizlynn LL Thing. 2004. Port hopping for resilient networks. In *IEEE 60th Vehicular Technology Conference*, Vol. 5. 3291–3295.

[14] Yue-Bin Luo, Bao-Sheng Wang, Xiao-Feng Wang, Xiao-Feng Hu, Gui-Lin Cai, and Hao Sun. 2015. RPAH: Random port and address hopping for thwarting internal and external adversaries. In *IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. 263–270.

[15] Gordon Fyodor Lyon. 2009. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure.

[16] Nick Marinos and Michael Clements. 2018. Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach. *United State Government Accountability Office, Report to Congressional Requestors* (2018).

[17] California Department of Insurance, New Hampshire Department of Insurance, Indiana Department of Insurance, North Dakota Department of Insurance, Maine Bureau of Insurance, South Carolina Department of Insurance, and Missouri Department of Insurance. 2016. Multistate Targeted Market Conduct and Financial Examination of Anthem Insurance Companies, Inc. and its Affiliates. https://doi.idaho.gov/DisplayPDF?id=2034&cat=DocketIndex

[18] Hamed Okhravi, Adam Comella, Eric Robinson, and Joshua Haines. 2012. Creating a cyber moving target for critical infrastructure applications using platform diversity. *International Journal of Critical Infrastructure Protection* 5, 1 (2012), 30–39.

[19] Hamed Okhravi, James Riordan, and Kevin Carter. 2014. Quantitative evaluation of dynamic platform techniques as a defensive mechanism. In *International Workshop on Recent Advances in Intrusion Detection*. 405–425.

[20] Vasilis Pappas, Michalis Polychronakis, and Angelos D Keromytis. 2012. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In *2012 IEEE Symposium on Security and Privacy*. 601–615.

[21] Robert Rudd, Richard Skowyra, David Bigelow, Veer Dedhia, Thomas Hobson, Stephen Crane, Christopher Liebchen, Per Larsen, Lucas Davi, Michael Franz, et al. 2017. Address Oblivious Code Reuse: On the Effectiveness of Leakage Resilient Diversity. In *NDSS*.

[22] Jeff Seibert, Hamed Okhravi, and Eric Söderström. 2014. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 54–65.

[23] Dilli Prasad Sharma, Dong Seong Kim, Seunghyun Yoon, Hyuk Lim, Jin-Hee Cho, and Terrence J Moore. 2018. FRVM: Flexible random virtual IP multiplexing in software-defined networks. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 579–587.

[24] Mark Simos. 2018. Overview of Petya, a rapid cyberattack. https://www.microsoft.com/security/blog/2018/02/05/overview-of-petya-a-rapid-cyberattack/

[25] Richard Skowyra, Kevin Bauer, Veer Dedhia, and Hamed Okhravi. 2016. Have no phear: Networks without identifiers. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. 3–14.

[26] Richard Skowyra, Kelly Casteel, Hamed Okhravi, Nickolai Zeldovich, and William Streilein. 2013. Systematic analysis of defenses against return-oriented programming. In *International Workshop on Recent Advances in Intrusion Detection*. 82–102.

[27] Karen Sood and Sean Hurley. 2017. NotPetya Technical Analysis - A Triple Threat: File Encryption, MFT Encryption, Credential Theft. https://www.crowdstrike.com/blog/petrwrap-ransomware-technical-analysis-triple-threat-file-encryption-mft-encryption-credential-theft/

[28] The PaX Team. 2001. Address Space Layout Randomization. https://pax.grsecurity.net/docs/aslr.txt

[29] Jacob Thompson. 2020. Six Facts About ASLR on Windows. https://www.fireeye.com/blog/threat-research/2020/03/six-facts-about-address-space-layout-randomization-on-windows.html

[30] Jue Tian, Rui Tan, Xiaohong Guan, and Ting Liu. 2018. Enhanced hidden moving target defense in smart grids. *IEEE Transactions on Smart Grid* 10, 2 (2018), 2208–2223.

[31] Yulong Wang, Qingyu Chen, Junjie Yi, and Jun Guo. 2017. U-tri: Unlinkability through random identifier for SDN network. In *Proceedings of the 2017 Workshop on Moving Target Defense*. 3–15.

[32] Bryan C Ward, S Gomez, Richard Skowyra, David Bigelow, Jason Martin, James Landry, and Hamed Okhravi. 2018. Survey of cyber moving targets. *Massachusetts Inst. Technol. Lexington Lincoln Lab., Lexington, MA, USA, Rep* 1228 (2018).

[33] Bryan C Ward, Richard Skowyra, Chad Spensky, Jason Martin, and Hamed Okhravi. 2019. The Leakage-Resilience Dilemma. In *European Symposium on Research in Computer Security*. 87–106.

[34] Richard Wartell, Vishwath Mohan, Kevin W Hamlen, and Zhiqiang Lin. 2012. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 157–168.

[35] Yulong Zhang, Min Li, Kun Bai, Meng Yu, and Wanyu Zang. 2012. Incentive compatible moving target defense against VM-colocation attacks in clouds. In *IFIP international information security conference*. 388–399.

[36] Zheng Zhao, Fenlin Liu, Daofu Gong, Lin Chen, Fei Xiang, and Yan Li. 2017. An SDN-based IP hopping communication scheme against scanning attack. In *9th International Conference on Communication Software and Networks (ICCSN)*. 559–564.