On the Challenges of Effective Movement*

Thomas Hobson
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02420
thomas.hobson@ll.mit.edu

Hamed Okhravi
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02420
hamed.okhravi@ll.mit.edu

David Bigelow
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02420
dbigelow@ll.mit.edu

Robert Rudd
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02420
robert.rudd@ll.mit.edu

William Streilein
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02420
wws@ll.mit.edu

ABSTRACT

Moving Target (MT) defenses have been proposed as a game-changing approach to rebalance the security landscape in favor of the defender. MT techniques make systems less deterministic, less static, and less homogeneous in order to increase the level of effort required to achieve a successful compromise. However, a number of challenges in achieving effective movement lead to weaknesses in MT techniques that can often be used by the attackers to bypass or otherwise nullify the impact of that movement. In this paper, we propose that these challenges can be grouped into three main types: coverage, unpredictability, and timeliness. We provide a description of these challenges and study how they impact prominent MT techniques. We also discuss a number of other considerations faced when designing and deploying MT defenses.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Security and Protection*

General Terms

Security, Theory

Keywords

Moving Target, Randomization, Diversity, Cybersecurity Challenges, Metrics

*This work is sponsored by the Department of Defense under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

MTD'14, November 3, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2663474.2663480.

1. INTRODUCTION

Uniformity in design and configuration of cyber systems is known to greatly simplify resource management tasks. However, this same homogeneity also benefits modern attackers because a single attack vector can be leveraged on many systems to multiply the effects of the attack. Nowhere is this phenomenon more apparent than in the case of botnets where large numbers of systems of similar design and configuration are overtaken and used in large scale attacks against victim machines [1]. Botnets often form their large attack armies at very low cost by exploiting a single vulnerability over and over again.

The destruction caused by botnets and numerous other classes of attacks is largely facilitated by the uniform, static, and predictable nature of our systems. Moving Target (MT) defenses have emerged as a promising approach for mitigating these weaknesses inherent in today's systems by removing many opportunities for the attacker to leverage a single exploit in attacking many similar machines. MT techniques are designed to create uncertainty for attackers and raise the cost to attack protected systems by making them less homogeneous, less static, and less deterministic. Existing implementations of MT techniques have demonstrated the potential for substantially disadvantaging attackers.

Address Space Layout Randomization (ASLR) [20] is a widely used example of an MT technique that substantially raises the level of effort required by attackers to attack a given system. Prior to the advent of ASLR, development of attacks exploiting memory corruption vulnerabilities was a rather straightforward process. Attackers could identify, with a high degree of accuracy, the locations of useful code and data objects in memory by analyzing their own copies of the targeted binary in advance of the actual attack. These useful locations, and the exploits to attack them, would be the same for all systems running the vulnerable software at any moment in time. The widespread deployment of ASLR implementations, which randomize the layout of portions of memory, has significantly increased the difficulty of exploiting memory corruption vulnerabilities as attackers are now forced to expend greater effort to attack new machines: they must now exploit memory disclosure vulnerabilities or perform brute force attacks in order to locate the randomized target objects for each instance [29].

S	The static portion of the attack surface
D	The dynamic portion of the attack surface
D_t	The state of D at time t
$expl(X)$	A predicate asserting that X is exploitable
$\neg expl(X)$	A predicate asserting that X is not exploitable
$H(X)$	The entropy of X
$I(X;Y)$	The mutual information of X and Y
T	The amount of time it takes to attack the system if the attacker has all required information

Table 1: Notation used throughout the paper

Although many MT techniques, such as ASLR, have been shown to be effective against certain types of attack, challenges remain which limit their effectiveness. These challenges involve three related aspects of MT techniques: (i) coverage, (ii) unpredictability, and (iii) timeliness. Failure to confront challenges in these areas has kept many research prototypes sidelined and has limited the effectiveness of existing deployments such as ASLR. We believe that MT techniques continue to have the potential to disadvantage the adversary but contend that existing MT techniques have weaknesses stemming from deficiencies in one or more of these aspects. We intend that the challenges presented in this paper can serve as a foundation for evaluating the effectiveness of MT techniques, for defining metrics, and for building effective MT design principles.

We organize the remainder of the paper as follows. Section 2 specifies the exact definition of MT technique that we use in the remainder of this paper. In Section 3 we describe the three challenges facing effective movement. Section 4 presents case studies in Instruction Set Randomization and Software Diversity and illustrates the challenges in these classes of MT techniques. Other considerations for building successful MT defenses are briefly discussed in Section 5. Finally, we conclude in Section 6.

2. MOVING TARGET TECHNIQUES

A broad range of work may be considered to fall under the umbrella of MT defense, including approaches that provide one or more of diversity, dynamism, and non-determinism. From a systems perspective, Moving Target technologies may alter properties across all layers of the execution stack, including the data, software, runtime environment, operating platform, and network, in order to disadvantage an adversary [19]. In this paper we use the most literal definition of the term "Moving Target" and focus on techniques that perform some form of movement over time; or in other words, techniques that periodically alter one or more system properties.

This definition includes techniques such as ASLR or platform migration, techniques that randomize a program's location in memory at process load time or periodically change the platform of a running application, respectively. In contrast, we do not consider monitoring or voting techniques that explicitly lack movement but may otherwise be considered Moving Target under the broader definition incorporating pure diversity. As one specific example, consider the Reverse Stack technique [22] that executes two instances of a program with the same input data, one variant with an upward-growing stack and one with a downward-growing one, and monitors for deviations. Although this technique provides diverse variants (two), the defensive benefit is attained via monitoring overflow behavior rather than from movement or from the altering of properties over time. It is hence not considered in the context of effective movement.

3. CHALLENGES OF EFFECTIVE MOVEMENT

We propose that there are three primary challenges in developing effective MT defenses: coverage, unpredictability, and timeliness. Coverage refers to the movement of all elements of the attack surface defended by the technique. Unpredictability refers to the movements being performed in a sufficiently large space as to render guessing of the outcome infeasible. Timeliness refers to the synchronization of movements with attacker observations. Simply put, one needs to address three questions: are the right pieces being moved, is movement taking place in a large enough space, and is movement taking place at the correct time? All too often, techniques are developed that consider only one or two of these areas and ignore or dismiss the other(s); all too often, it is by means of one of these overlooked areas that the technique is ultimately bypassed. No truly effective MT defense may be constructed without a full consideration of these challenges.

In this paper we refer to effectiveness purely in terms of the security benefit offered by an MT technique. We do not focus on other hurdles to building successful MT defenses such as performance, compatibility, usability, or cost; that discussion is reserved for Section 5. Nor do we consider the composability of multiple techniques and the interrelation between their methods of addressing each challenge area, choosing instead to explore each challenge area on its own for the reasons detailed below.

On the surface, each challenge area seems to have a relatively straightforward application to movement. Coverage measures how various portions are subject to movement, and which components are ignored. Unpredictability refers to the range of movement and the likelihood that an attacker can either guess or predict how the movement is applied. Timeliness in movement is required to ensure that it happens at the correct time with respect to the attacker. Moreover, these properties can be precisely calculated.

In actuality, each area can only be accurately assessed in the context of a threat model. Without such a threat model, the conventional view that "more" of a property is better than "less" of it may be vacuous. For instance, consider coverage in the context of an MT defense against data

	Coverage	Unpredictability	Timeliness
Challenge	Exploitable elements of surface are subject to movement and no information is learned from static components	Current or future movements cannot be predicted by the attacker	Movement applied between attacker observation and subsequent attacker action
Formalism	$(I(D;S) = 0) \wedge \neg expl(S)$	$H(D) >> 0$	$\forall \tau \in (-\infty, t_0],\ I(D_{t_0+T}; D_\tau) = 0$

Table 2: Three challenges of effective movement

exfiltration. A technique that protects 80% of the data is generally seen as superior to one that protects 50% of the data, but if only 10% of the data is actually significant, we cannot say which technique is better merely on the basis of "more" and "less" protection. It may actually be the case that neither technique protects the critical data. The lack of a threat model has prevented us from recognizing this – and caused us to judge a technique based on a criteria that is measurable but has no relation to our actual needs.

Thus, although the challenges of effective movement can be discussed generally, their actual implementation and consideration in a technique can only make sense in the context of a threat model. The discussion in this section will therefore employ ASLR as an illustrative example where such is required. ASLR stands out as the most mature and widely deployed MT technique to date; implementations of varying potency are deployed in a variety of enterprise and consumer operating systems in both the desktop/server and mobile space. ASLR aims to complicate the development of practical code reuse attacks by making virtual memory addresses unpredictable. Since exploits commonly make use of absolute addresses [26], attackers must either exploit an additional memory disclosure vulnerability [28, 24] or craft an exploit that does not rely on an absolute address [7].

We do not mean to suggest that ASLR is particularly good or particularly bad in comparison to any other MT technique; we choose it as our motivating example for this section because of its widespread deployment and familiarity, and also because it exhibits a weak form of each challenge and is therefore suitable for discussion from both positive and negative viewpoints. Furthermore, it is not our intent to analyze strengths and weaknesses of ASLR (and other techniques) in any context other than that of effective movement, and nor is it our intent here to develop a new ASLR-like technique intended to address these challenges. We focus instead on the broader analysis of these challenges in order to give direction to future MT research and use ASLR only as an illustrative example.

Moreover, note that our goal is not to uncover or study new weaknesses for the techniques discussed in this paper; rather, we strive to categorize and formalize known weaknesses in MT techniques in order to develop a deeper understanding of them, and facilitate the development of stronger, more effective techniques.

Table 1 describes the notation used in the rest of the paper, and Table 2 summarizes the challenge areas explored in the following subsections.

As described in [13] and used in Table 1, $I(D;S)$ is the mutual information of S and D defined by:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

and $H(D)$ is the entropy D defined by:

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

3.1 Coverage

Coverage: *The extent to which all elements of a defended attack surface are subject to movement.*

We define an MT defense with full coverage as one that moves the entirety of the attack surface protected by the defense. Any residual static components of the protected system must not be sufficient for attacking the system. Furthermore, the attacker should not be able to use the static components to locate the dynamic components.

Since coverage tries to capture the extent to which the attack surface is subject to movement, a naïve formalism for it may be the ratio of the dynamic part to the entirety of the attack surface. If S denotes the static part of the attack surface and D denotes the dynamic part, the naïve measure would be $\frac{D}{D+S}$. However, this formalism is not accurate. Consider two systems, one where the entire attack surface is static and another one where only half of the attack surface is moving, but the static part leaks all the information necessary for attacking the dynamic part. These two systems are both insecure and easily exploitable, but the naïve metric would be very different for them.

The appropriate formalism for coverage must state that the amount of information that can be learned about the dynamic part from the static part is zero. In information theoretic notation, the requirement can be written as $I(D;S) = 0$. However, this is not enough because an MT technique which keeps the entirety of the attack surface static would still satisfy the requirement. It is also necessary that the static part itself is not enough to successfully attack the system. We write this requirement as $\neg expl(S)$.

In other words, a technique has full coverage iff:

$$(I(D;S) = 0) \quad \wedge \quad \neg expl(S) \tag{1}$$

A technique that moves the entirety of this attack surface has full coverage, while a technique that moves only part of this attack surface may either have partial or no coverage, depending on the specifics of the attack model. In some techniques, coverage is trivial to achieve because the attack surface consists of only a single indivisible component. When so, movement of that component is movement of the entire attack surface by definition, and movement is either complete or nonexistent. In other techniques, the attack surface

consists of several pieces that move independently of each other, and some portions may not move at all.

Coverage is best measured as a binary state, and a technique either has full coverage or no (effective) coverage. When the attacker can achieve their goal via only a small portion of the attack surface, exposure of that small portion is equivalent to not having coverage, regardless of how many other portions are subject to the technique. In some cases, a unique portion of the attack surface must be protected for the technique to have full coverage, and the challenge lies in recognizing which portion must be protected for that threat model. In other cases, when many independent portions of the attack surface are all of equal value to the attacker, a single exposed portion becomes the target of choice and the application of movement to all other portions is entirely wasted.

In the context of ASLR, the main threat model considers code reuse attacks [17, 30, 25, 4]. These are attacks in which the adversary executes existing otherwise benign code in the program for malicious purposes. The attack surface consists of all executable code segments, and thus full coverage requires one to ensure that the locations of every code segment in the target application's address space are randomized and not known to the attacker. Failure to meet this coverage criteria can severely limit the effectiveness of ASLR because those unrandomized code segments quickly become the focus of attacks.

Most modern Linux implementations of ASLR randomize the locations of the heap, stack, and memory mapped pages (which includes dynamically linked libraries), as illustrated in Figure 1. Randomization of the program image is less common since many application binaries are not compiled as position-independent executables (PIE), despite the availability of compile-time options to do so. This lack of randomization allows an attacker to re-use any portion of the main program image with certainty, since ASLR is never applied to that component in the first place. Applications on Windows operating systems face similar challenges; notably, there have been widely-used libraries that were not randomized because the /DYNAMICBASE (or equivalent) option was not applied during compilation. These libraries thus became the targeted surface for many attacks [5].

Our definition of coverage in this context is not meant to incorporate any judgment on the quality of a movement. We only care that a given portion of the attack surface is subject to movement and thus labeled as having coverage, whereas the actual quality of the movement is measured by unpredictability. Therefore, again in the context of ASLR, we do not consider the fact that all libraries are loaded in a specific order and at specific positions relative to each other, to be a coverage issue. Even though libraries are not randomized relative to each other, the fact that they are randomized at all means that ASLR has coverage with regard to libraries, and subsequent analysis of the quality of that movement is more properly a component of unpredictability.

As implied by the previous statement, coverage is naturally a prerequisite of unpredictability and timeliness. If a portion of the attack surface is not moving, it is 100% predictable and timeliness has no meaningful definition. If an attack requires only that non-moving portion of the attack surface, the technique is fatally flawed, and even when an attack requires multiple portions of an attack surface, any uncovered portion is an immediate freebie for the attacker.

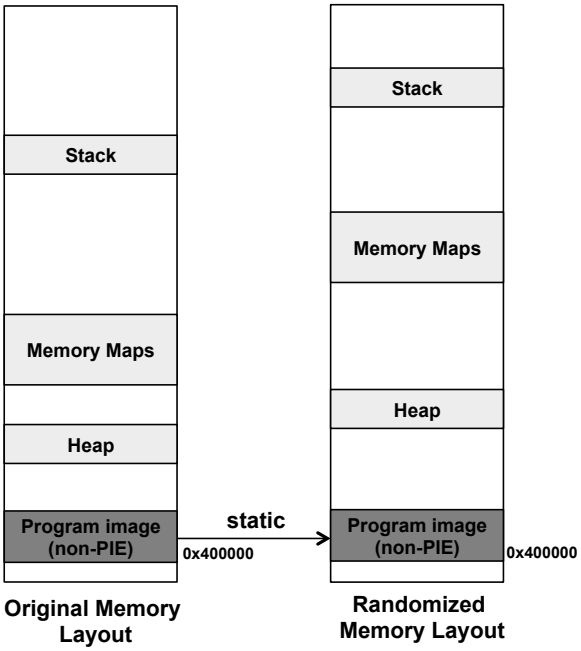

Figure 1: 64-bit x86 Linux non-PIE memory layout (not to scale and not all components included). Program image sufficient for code reuse attacks but remains in static location.

However, the fact that coverage is necessary by no means makes it sufficient. Simply because a portion (or all) of the attack surface is moved does not mean that it is being moved to an appropriate location, and nor does it mean that it is being moved at the correct time with respect to the attacker.

3.2 Unpredictability

Unpredictability: *The extent to which the outcome of current or future movements of the attack surface are indeterminable by an attacker.*

Unpredictability can be defined as the amount of information an attacker has or can guess about the dynamic part of the attack surface in a given MT technique. In other words, an MT technique is unpredictable if the space of possible movements is sufficiently large, where "sufficiently" depends upon the threat model. A system that has two possible states should probably not be regarded as unpredictable under any possible threat model. Formally, a technique is unpredictable iff:

$$H(D) >> 0 \tag{2}$$

In other words, unpredictability requires that the movement exposes a large entropy to the attacker. The unpredictability of a technique can be measured in a relatively straightforward manner. At its most basic, and assuming the use of a secure and correctly invoked random number generator, unpredictability can be regarded as the probability of the attacker correctly guessing where something has moved. This can be expressed as $\frac{1}{n}$, where n is the number of possible movement locations. This type of probability may be calculated as a whole or on a piece-by-piece basis depending on the threat model and the technique in question, and

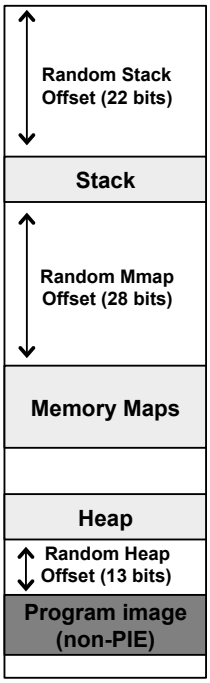

Figure 2: Entropy by section in 64-bit x86 architectures.

may be complicated by implementation concerns, but generally remains a simple calculation. The method by which to combine multiple piece-by-piece unpredictability calculations into a single measurement is sometimes less clear, but is certainly mathematically possible if both the threat and the attack surface can be adequately quantified.

Unlike coverage, unpredictability works on a "more is better" model, assuming that the movement is not otherwise compromised. No defense can entirely ignore the possibility of a lucky guess by the attacker, or of a brute force attack to exhaust the possible movement range, but the likelihood of such a compromise is reduced proportionally as the range of movement possibilities is expanded.

In the context of ASLR, unpredictability varies by component and certain components are in turn composed of multiple pieces. As shown in Figure 2, in current x86 64-bit Linux kernels, the stack, heap, and mapped regions are randomized with 22, 13, and 28 bits of entropy, respectively. Thus in theory, and temporarily setting aside the problem of leaked information, the probability of guessing the location of a particular library (contained in the mapped regions) is $\frac{1}{2^{28}}$. However, libraries are generally loaded in a static order and with static offsets to each other. Therefore, if one knows the location of one particular library in a process, the location of every other library is 100% predictable.

In other words, if the libraries are moved as one cohesive block with static offsets from one another, they can be regarded as a single component when evaluating the unpredictability of a movement. The security of one portion of the component becomes dependent upon the security of other portions of the component. One proposed improvement for ASLR, Address Space Layout Permutation (ASLP) [11], attempts to eliminate this dependence amongst libraries by randomizing each library separately.

Another situation that can reduce the apparent entropy is when components of the surface contain references to one another but the components have varying levels of entropy. In such cases the predictability of components with high entropy is often reduced to the predictability of components with low entropy [8]. In the context of ASLR, increasing the independence of memory sections can minimize this entropy reduction effect.

Unpredictability only applies when coverage is present, but has no direct impact (or dependence) on the timeliness discussed in the next section. Indirectly, unpredictability depends entirely on the secrecy of the movement and in forcing the attacker to guess randomly rather than intelligently. If that secrecy is compromised, even when the actual movement otherwise remains entirely unpredictable, the attacker need not guess at all.

3.3 Timeliness

Timeliness: *The extent to which a movement can be applied between the time at which an attacker makes an observation and time at which an attack is completed.*

We define a timely MT defense as one that completes movement in the period between an attacker observation point and a subsequent attacker interaction point. An observation point is any point in which the attacker has the opportunity to learn about the outcome of the most recent movement and may be passive or covert. A subsequent interaction point is any point in which the attacker has the opportunity to execute an action based upon information learned at the observation point. Timeliness requires that information learned at an observation point must be of no value during the next interaction with an attacker. The duration of time to perform a movement must be shorter than the time between an observation point and the next interaction point.

Timeliness can be more formally defined as the amount of information an attacker learns about the state of the dynamic part of the system based on all of the information available about its past states. In other words, if the amount of time it takes to successfully attack the system is T, the amount of information the attacker has about the state of the system at time of attack given all the past history of the system should be zero. Formally, a technique is timely iff:

$$\forall \tau \in (-\infty, t_0],\ I(D_{t_0+T}; D_\tau) = 0 \tag{3}$$

Another way of interpreting Eq. 3 is that the system has to move randomly after the attacker has the chance to observe (in the period $t = -\infty$ to t_0) and before there has been an opportunity to act on that information ($t = t_0 + T$).

Choosing the correct time to move is just as critical for a technique as is its coverage and unpredictability, but the timeliness of movement is often significantly harder to judge than the other two. It is likely that multiple instances of movement will be required during the duration of a defensive activity, since a single movement is of limited utility unless the attacker is similarly restricted to a single attack, and the timing of this movement is critical. Even more than coverage and unpredictability, timeliness as applied to a specific technique depends entirely upon the threat model. Movement must always be synchronized with the attacker's actions.

Unfortunately, an attacker's actions may not be detectable by the defender. Even aside from the difficulties inherent in detecting the initial launch stages of a new attack, the "actions" taken by the attacker may be limited to passive observation. When the attacker only observes the target in order to judge the perfect moment to strike, the defender can observe no signal until the very moment that the attack is launched, at which point it may be too late to take effective defensive action. Without any sign that an attack is imminent, or even any indication that it is being targeted, movement can only be triggered at defined system events based on the threat model. Ironically, if a threat model has been clearly defined and movement is performed during the appropriate events, the defender may never see even an indication of an attack because the attacker is prevented from ever getting a clean shot.

While a faster movement may often correlate with a more effective movement, it is misleading to measure timeliness purely in terms of speed. Rather, it must be measured in terms of the relative time that it takes to perform a movement (following an observation point) versus the time that it takes an attacker to execute subsequent steps in an attack sequence. While an ideal movement is performed between an observation point and the very next attack, moving at any point within the attack sequence may provide some security benefit by disrupting later stages of an attack.

In the context of ASLR, timeliness in movement means eliminating the interval in which a potential attacker can observe or learn information about a memory address in a target process and launch an attack using that observed address, or reducing the window of opportunity for a brute force attack. A perfectly timed ASLR implementation would be one in which rerandomization takes place in synchronization with the attacker's observation of addresses, and in synchronization with brute force attacks to prevent the probability of success from rising above a defined threshold. A lack of timeliness in ASLR is most apparent in the case of a direct [28, 24] or indirect memory disclosure attack [23] in which the adversary leaks the addresses of a running process and proceeds to use those leaked addresses to craft the next stage of the attack, as shown in Figure 3.

In the case of a traditional brute force attack, timely movement can provide a slight benefit to security by eliminating the information the attacker has learned from previous unsuccessful guesses. As Shacham et al. [26] have demonstrated, rerandomization can only double the number of expected guesses versus a technique that never rerandomized; we consider this to be primarily a challenge of unpredictability. However, continuing with the example of code reuse attacks, there also exist attacks in which the attacker can learn portions of an address, such as by a bytewise guessing attack [3], or even the entire address by memory disclosure. In the face of this method, timely movement can revert the probability of a successful attack from highly probable (perhaps even 100%), back to the original level of unpredictability, which is the amount of uncertainty in memory locations.

In all cases, the attacker model is critical to an appropriate understanding of timeliness. Without an understanding of the actions of an attacker, or at least an idea of their steps, there is no way to determine whether any particular schedule of movement is at all reasonable. Lacking this understanding, the only approximation of timeliness is to schedule movement frequently and hope that the time required to carry out an attack is greater than the movement interval.

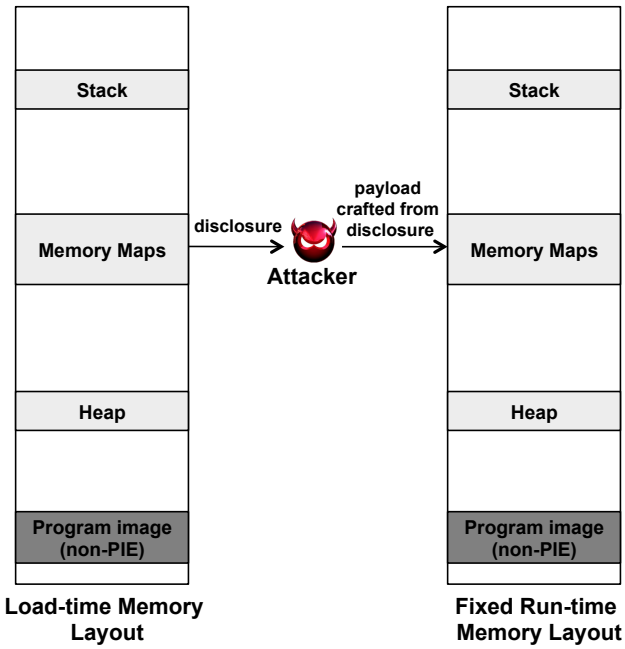

Figure 3: Memory locations are fixed at process load time, while memory disclosures occur at run time. A disclosure at run time reveals location for the duration of the process.

Lack of timeliness is the greatest flaw that we have seen in existing moving target techniques, and indeed the lack of a defined threat model is pervasive throughout MT defenses in general. Techniques that are intended to prevent attacker persistence have a good implicit understanding of timeliness due to the characterization of their threat model: their timeliness factor is directly related to the amount of time that they are willing to risk an attacker maintaining persistence within their system [18, 6]. Many other techniques make an attempt at timeliness by including movement based on timers or detected actions, but do not explicitly tie such movement to the attacker. This is of particular importance in assessing a technique: coverage is usually mentioned and unpredictability can be explicitly measured in most cases, but timeliness is more subtle. The inclusion of time-based and action-based rerandomization can easily convince the reader that a technique is timely, but unless the threat model is fully defined, one cannot be sure that the movements are actually relevant.

4. CASE STUDIES

Thus far, we have employed ASLR in order to help illustrate the challenges of effective movement with a real-world example. ASLR is by no means the only such example, and we now wish to form a more complete picture of the challenges by considering two classes of technique in toto. First, we will consider the category of Instruction Set Randomization (ISR) which has seen multiple independent implementations. We will then consider the category of Software Diver-

sity, which is an even broader space with multiple techniques developed along entirely different lines, although we choose to focus on a narrower subset for discussion purposes. Table 3 summarizes these challenges as applied to these three technique types.

While we have defined formal ways to describe the goals for MT techniques in addressing these challenges, and it is feasible to use these formalisms to gain intuition about whether or not a technology meets these goals, it is much more difficult to quantitatively measure a technology's effectiveness in the challenge areas. This is particularly true for coverage, given the difficulty of defining the attack surface [14], and for timeliness, given the difficulty of determining all potential attacker observation points. Metrics to quantitatively measure the effectiveness of technologies in terms of these challenge areas have not yet been developed and are sorely needed for proper evaluation, but can only be defined in terms of a threat model. Therefore, the case studies in this section are discussed and formally described rather than quantitatively measured.

4.1 Instruction Set Randomization

Instruction Set Randomization (ISR) techniques [10, 21, 2] aim to prevent code injection attacks. Code injection attacks work by exploiting memory corruption vulnerabilities in a victim process to insert code into that process, and then redirect the control flow to that injected code. Threat models for ISR techniques assume that the attacker already has the ability to inject code and redirect control flow, and ISR defenses randomize the instruction set of a process, thus complicating an attacker's attempt to insert valid code. An early ISR proposal by Kc et al. [10] suggests encoding standard x86 instructions by performing XOR operations with a secret key. Prior to execution of each encoded instruction, the processor decodes the instruction by XORing it with the key. A code injection attack that injected unencoded x86 instructions would likely be untranslatable or interpreted as non-sensible instructions during the decoding process, thus thwarting attackers. In such ISR techniques the instruction encoding is the system property subject to movement.

An ISR technique with full coverage strives to encode all executable instructions. Wherever instructions are permitted to execute without encoding, the attacker has an opportunity to use them in a code injection attack. Instructions that are statically compiled into the binary may be easily encoded, but it is not immediately clear how to best handle self-modifying or just-in-time compiled code. Such compatibility concerns commonly lead to coverage concessions. For example, it is difficult to encode instructions that are written into the process stack upon signal delivery. One ISR technique developed by Portokalidis et al. [21] offers compatibility with such signal jump vectors (trampolines) by executing those instruction sequences without performing any decoding, but this compromise in coverage presents an opportunity for the attacker.

In formal terms of coverage as defined in Eq. 1, D represents the encoded instructions whereas S represents the unencoded instructions. Based upon known attacks it does not appear that any information about the encoding of D can be obtained from S, thus $I(D;S) = 0$. However, we cannot say that $\neg expl(S)$ because it is unclear whether the unencoded instructions alone are sufficient for successful attacks. Thus, $(I(D;S) = 0) \wedge \neg expl(S)$ cannot be assured and full coverage is not guaranteed. In order to satisfy Eq. 1, ISR techniques must apply the encoding to all instructions, including those generated at runtime.

Unpredictability in ISR is concerned with the guessability of the encoding of the instruction set. In assessing unpredictability we often initially evaluate the entropy of the encoding key: with a 32-bit key, we would expect an attacker to perform 2^{31} guesses on average before discovering the correct key. Although this level of entropy would indeed be sufficient against most threats today, it is difficult to achieve in practice. Sovarel et al. [27] demonstrated attacks against ISR techniques using XOR encoding that use bytewise guessing in order to significantly reduce the number of expected guesses required by an attacker. Using this approach an attacker could guess a 32-bit key in as few as 512 ($2^7 \times 4$) attempts on average. In the case of an automatically respawning web server, where an attacker may make hundreds or thousands of guesses in a short amount of time, this is insufficient unpredictability. In terms of Eq. 2 for this attack model, $\neg(H(D) >> 0)$ for $H(D) = 7$ bits. Satisfying Eq. 2 requires the use of stronger encryption schemes that can in practice provide the ostensible level of entropy.

In the context of ISR, timeliness considers when to change the encoding. To be effective, techniques must change the encoding before the attacker is able to determine the key and act according to that knowledge. A technique that offers more granular encoding is more likely to achieve effective timeliness against a broader range of attackers. In the case of the bytewise key guessing attack, an ideal ISR technique would change the encoding after each guess, rendering any information gained from previous guesses useless. While ISR techniques to date may not support re-encoding at arbitrary points in a program, some offer sufficient granularity to defeat this particular attack. Each attempted guess causes the process to crash and thus any re-encoding that is more granular than crash-time provides sufficient timeliness.

For example, the RISE technique [2] thwarts this attack by changing keys each time the attacker attempts a guess (and induces a crash), including at the time of process forking. Any information about the encoding that was obtained from a crash-inducing guess provides no benefit to the attacker about the new encoding. In terms of Eq. 3, we define t_0 as crash time and T as the time to inject encoded instructions following the crash. Under this attack model, information about the encoding can only be learned at time t_0 and any information learned at time t_0 provides no information about future encodings as the key is immediately changed, thus $\forall \tau \in (-\infty, t_0],\ I(D_{t_0+T}; D_\tau) = 0$. Conversely, a second published technique [10] changes the encoding only at compilation time rather than program invocation time, therefore not satisfying Eq. 3 and thus not timely in the face of such guessing attacks. Under broader threat models, including against guessing attacks that do not crash the process, techniques must allow for more granular re-encoding, such as after every instruction, in order to satisfy Eq. 3.

4.2 Software Diversification

Software diversity aims to increase the cost of exploit development by randomizing aspects of a program's implementation [12]. In widely-used traditional software, identical binary code may run on millions of machines at once. This allows an enormous economy of scale for the attacker: a single exploit can be used to compromise all machines running

MT Defense	Coverage	Unpredictability	Timeliness
ASLR	Program image not compiled as position independent executable	13 bits of entropy for heap on 64-bit Linux	Randomized at load time, disclosures at runtime
ISR	Exception for unencoded instructions during signal delivery	XOR encoding permits key guessing in as few as 512 attempts on average	Encoding at compilation time, bytes of encoding key disclosed at runtime
Software Diversity	Code generated at runtime not diversified	Having an insufficient pool of variants	Only changing variants when user downloads an update

Table 3: Example deficiencies in popular categories of MT defenses (not exhaustive).

this code. Software diversity can reduce this economy of scale by reducing the likelihood that a single exploit can be used against large numbers of machines. Many existing implementations of software diversity focus on compiler-based approaches [12, 9], which leverage the compiler's ability to produce large amounts of functionally equivalent, but internally different, variants of an input program. It is these internal implementation details that are subject to movement.

We will here focus on one specific software diversification technique: NOP insertion, as described by Jackson et al. [9]. NOP insertion is a compiler-based diversification pass which probabilistically inserts a single no-operation (NOP) before each instruction of a program according to the user's parameter p_{nop}. This changes the code size and layout and thus restricts the attacker's ability to predict the location of useful code segments such as gadgets used in Return-Oriented Programming (ROP). This attack model can be defined in terms of the notation of Table 1 where our attack surface is the code segment of a program binary. We can partition this segment into two portions: D, the portion of the binary that is affected by NOP insertions, and S, the portions of the binary that is not affected by NOP insertion.

Implementations of software diversification with full coverage must ensure that all parts of the program are diversified. In the case of diversity implemented by inserting NOPs into programs, full coverage requires that NOPs be inserted with a pre-specified rate into all code executed by an application, including code in the program image, statically or dynamically linked libraries, and code generated at runtime. A failure to diversify any of these code portions may lead to a situation where an attack on the non-diversified portion can still be performed, undermining the effectiveness of the movement. Compiler-based software diversification evades most coverage issues by virtue of being implemented as a compiler pass: ensuring that the program image and libraries are comprehensively diversified can be achieved by compiling them with the diversifying compiler. Handling code generated at runtime, such as that created during just-in-time compilation, is more difficult, since every runtime code generator would need to implement a diversification element. This is, however, a difficulty in implementation and not a fundamental limitation of diversification techniques. By applying NOP-insertion indiscriminately to all code, including code generated at runtime, Eq. 1 can be satisfied. If all code is affected by NOP-insertions then $S = \emptyset$, making $(I(D;S) = 0) \wedge \neg expl(S)$ trivially true as $I(D;\emptyset) = 0$ and $\neg expl(\emptyset)$ are both true.

Unpredictability in software diversity is related to the attacker's ability to predict the location of useful code segments such as ROP gadgets. Therefore, it is ultimately determined by the ability of the attacker to infer the location of gadgets in one variant by their knowledge about the location of gadgets in another variant. The unpredictability of gadgets in a NOP-compiled program varies with the value of p_{nop} as defined above, the total number of variants distributed, and the pattern of distribution. At $p_{nop} = 0$ or $p_{nop} = 1$, no unpredictability is achieved, whereas the maximum variance occurs at $p_{nop} = 0.5$. Unpredictability is also directly proportional to the number of variants produced, and the uniformity of their distribution. Ideally, every user would run a different variant of each program, but there may be practical problems in creating an arbitrarily large number of variants. As the number of variants increases, the cost of their generation and maintenance also increases, and reproduction of individual user errors is rendered problematic.

Due to the number of instructions in even a small program, the limit on the number of possible variants is extremely large to the point of being unlimited for practical purposes. For example, a small program containing about 1,000 instructions would have 2^{1000} possible variants. If $p_{nop} = 0.5$ then all variants are equally likely and the entropy is $H(D) = -\sum_{2^{1000}} \frac{1}{2^{1000}} log_2(\frac{1}{2^{1000}}) = 1000$ bits, which satisfies Eq. 2. However, calculation of the unpredictability may be somewhat more complex depending on the attack model. If the attacker need locate only one particular instruction, the location of that instruction in the un-diversified binary impacts the unpredictability. The location of each instruction in the binary is dependent upon the instructions that come before it, and thus an "early" instruction has much lower unpredictability than a "later" instruction. To best satisfy Eq. 2, NOP-insertion techniques must provide a sufficient level of unpredictability for all instructions in a program, support a large pool of variants, and insert NOPs according to the maximum variance quantified by $p_{nop} = 0.5$.

Timeliness is related to the length of time that it takes an attacker to profile a single variant; a timely implementation should replace one variant with another at intervals too short to allow the attacker to make use of any discovered gadgets. The NOP insertion technique outlined in [9] does not describe a timeline over which new variants are obtained, but does describe an "app-store"-like distribution model. Assuming usage similar to app stores common to modern mobile platforms, a user could get an individualized variant whenever they download or update an applica-

tion. In terms of timeliness, this implementation is vulnerable to targeted attacks: an attacker has until the next time a new version of the software is released (and longer, if the user does not immediately upgrade) to carry out the attack. A higher level of timeliness could be achieved by creating a more frequent replacement cycle, applied as a response to opportunities for probing the variant. An implementation that switched variants on each crash would likely frustrate the construction of exploits because attacker probes commonly cause applications crashes, although more subtle probings are also possible.

We can restate these timeliness requirements in terms of Eq. 3: let T be the time it takes an attacker to execute an attack and T_r be the time between successive variant replacements. If we assume that variants are independent it is clear that $\forall \tau \in (-\infty, t_0], I(D_{t_0+T_r}; D_\tau) = 0$ because one cannot gain any information on $D_{t_0+T_r}$ from D_{t_0}. Variant replacement must have occurred between $[t_0, t_0 + T_r]$ and variants are independent. By choosing a T_r such that $T_r < T$ we can satisfy Eq. 3. Thus, under the crash-inducing adversarial model, variant replacement would best happen at process load time rather than upon downloading or updating, to best satisfy Eq. 3.

Note that in many cases, it is straightforward to determine whether or not equations 1, 2, and 3 are satisfied, but it is significantly more difficult to calculate the actual values for them. For instance, it is arguable whether $I(D; S)$ equals zero for a given technique and an attacker model, but in cases that it is not, it much harder to compute its value because one has to consider all possible ways that information about D can be leaked by S.

5. OTHER CONSIDERATIONS

Aside from ensuring the effectiveness of the movement, there are a number of other considerations when designing and deploying MT techniques. Since these considerations do not impact the *effectiveness* from a security perspective, we do not consider them as part of the challenges of effective movement. However, they are important for achieving a practical defense that is capable of widespread adoption, so we briefly discuss them here for the sake of completeness.

Low overhead is frequently argued to be one of the most important criteria for achieving widespread adoption. Overhead can be defined in different ways depending on the context of the system, but a few typical measures include performance overhead, memory/storage overhead, and communication (network) overhead. Previous work has suggested that if the processing overhead of a technique in general is higher than a few percentage points, the technique has a lower chance of seeing widespread adoption [29], and the history of widely deployed techniques certainly provides more evidence for this belief. Widely deployed defenses including ASLR and even non-MT techniques such as non-executable data (NX) have negligible performance overheads. In order for MT techniques to gain widespread adoption, they must avoid incurring high overhead.

Moreover, since MT is inherently a probabilistic defense, its overhead should be correspondingly less than the overhead of a stronger, deterministic defense. For example, recompilation of a C program with SoftBound + CETS [15, 16, 29] allows for complete memory safety with an average of 116% performance overhead. All else being equal, a probabilistic randomization defense must have a 116% overhead performance ceiling as there would be no reason to substitute guaranteed protection for probabilistic protection. Note that the thresholds set by deterministic defenses are absolute upper bounds; for widespread adoption, the acceptable overhead is usually very small, as previously discussed.

Direct cost is another important factor that determines the practicality of any defense in general, and MT defenses in particular. There are various different direct costs to consider: development cost, deployment cost, operational cost, maintenance cost, and others. These costs can only be evaluated in the context of a particular environment and may directly impact the deployable practicability of many techniques.

The utility of an MT defense is also contingent upon the extent to which it impairs or otherwise negatively impacts normal functionality of a system. This is sometime referred to as a system's *mission*. If an MT technique impairs the functionality of the system because of the nature of the movement that is being applied or even because of the overhead it incurs, there will be a much smaller chance of its adoption.

The expertise required on the operator side of an MT technique is yet another important consideration. Requiring expert knowledge for proper operation is a hurdle to the deployment of MT techniques. Ideally, an MT technique should be seamless to the user or administrator of a system while providing complexity for the attackers.

Finally, dependency of the MT defense on other system components, its compatibility with those components, and being able to apply it to some parts of the system, but not others (modularity) are other considerations for practical MT defenses.

6. CONCLUSION

MT research has exhibited the potential to substantially disadvantage the adversary but at the same time has exposed numerous shortcomings in the effectiveness of MT techniques. We have identified three primary challenges to achieving effective defense: (i) coverage, which describes the challenge of including the entire attack surface in the movement, (ii) unpredictability, which refers to the range of movement and the likelihood that an attacker can guess or predict a movement, and (iii) timeliness, which is concerned with the challenge of synchronizing movements with attacker observations.

We have shown how these challenges manifest themselves in some of the more well-known MT categories of ASLR, ISR, and Software Diversity. Notably, all MT defenses examined exhibited weaknesses across all three challenges. We do not believe that these weaknesses are fundamental to the techniques themselves but that they are indicative of the major challenges that must be addressed when building effective MT defenses. We hope that the identification and discussion of these challenges can help build a foundation for future research in developing and evaluating effective MT defenses.

7. REFERENCES

[1] P. Barford and V. Yegneswaran. An inside look at botnets. In M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, pages 171–191. Springer US, 2007.

[2] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, CCS '03, pages 281–289, New York, NY, USA, 2003. ACM.

[3] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh. Hacking blind. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, 2014.

[4] S. Checkoway, L. Davi, A. Dmitrienko, A. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *Proc. of the 17th ACM CCS*, pages 559–572, 2010.

[5] X. Chen. Aslr bypass apocalypse in recent zero-day exploits, 2013.

[6] DoD. Lightweight portable security, 2014.

[7] T. Durden. Bypassing pax aslr protection, 2002.

[8] W. Herlands, T. Hobson, and P. Donovan. Effective entropy: Security-centric metric for memory randomization techniques. In *Workshop on Cyber Security Experimentation and Test*, 2014.

[9] T. Jackson, A. Homescu, S. Crane, P. Larsen, S. Brunthaler, and M. Franz. Diversifying the software stack using randomized nop insertion. In *Moving Target Defense*, pages 151–173. 2013.

[10] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM.

[11] C. Kil, J. Jun, C. Bookholt, J. Xu, and P. Ning. Address space layout permutation (aslp): Towards fine-grained randomization of commodity software. In *Proc. of ACSAC'06*, pages 339–348. Ieee, 2006.

[12] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz. Sok: Automated software diversity. In *Proceedings of the 35th IEEE Symposium on Security and Privacy*, 2014.

[13] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

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

[15] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic. Softbound: Highly compatible and complete spatial memory safety for c. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 245–258, New York, NY, USA, 2009. ACM.

[16] S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic. Cets: Compiler enforced temporal safety for c. In *Proceedings of the 2010 International Symposium on Memory Management*, ISMM '10, pages 31–40, New York, NY, USA, 2010. ACM.

[17] Nergal. The advanced return-into-lib(c) exploits (pax case study). *Phrack Magazine*, 58(4):54, Dec 2001.

[18] H. Okhravi, A. Comella, E. Robinson, and J. Haines. Creating a cyber moving target for critical infrastructure applications using platform diversity. *Elsevier International Journal of Critical Infrastructure Protection*, 5:30–39, Mar 2012.

[19] H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy*, 12(2):16–26, Mar 2014.

[20] PaX. Pax address space layout randomization, 2003.

[21] G. Portokalidis and A. D. Keromytis. Fast and practical instruction-set randomization for commodity systems. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 41–48, New York, NY, USA, 2010. ACM.

[22] B. Salamat, A. Gal, and M. Franz. Reverse stack execution in a multi-variant execution environment. In *Workshop on Compiler and Architectural Techniques for Application Reliability and Security*, 2008.

[23] J. Seibert, H. Okhravi, and E. Soderstrom. Information leaks without memory disclosures: Remote side channel attacks on diversified code. In *Proc. of the 21st ACM CCS*, 2014.

[24] F. J. Serna. cve-2012-0769, the case of the perfect info leak, 2012.

[25] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proc. of ACM CCS*, pages 552–561, 2007.

[26] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *Proc. of ACM CCS*, pages 298–307, 2004.

[27] A. N. Sovarel, D. Evans, and N. Paul. Where's the feeb? the effectiveness of instruction set randomization. In *14th USENIX Security Symposium*, volume 6, 2005.

[28] R. Strackx, Y. Younan, P. Philippaerts, F. Piessens, S. Lachmund, and T. Walter. Breaking the memory secrecy assumption. In *Proceedings of EuroSec '09*, 2009.

[29] L. Szekeres, M. Payer, T. Wei, and D. Song. Sok: Eternal war in memory. In *Proc. of IEEE Symposium on Security and Privacy*, 2013.

[30] M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning. On the expressiveness of return-into-libc attacks. In *Proc. of RAID'11*, pages 121–141, 2011.