Finding Focus in the Blur of Moving-Target Techniques

Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein | MIT Lincoln Laboratory

Moving-target (MT) techniques seek to randomize system components to reduce the likelihood of a successful attack, add dynamics to a system to reduce the lifetime of an attack, and diversify otherwise homogeneous collections of systems to limit the damage of a large-scale attack. In this article, we review the five dominant domains of MT techniques, consider the advantages and weaknesses of each, and make recommendations for future research.

Protecting critical systems and assets against cyberattacks is a continuous uphill battle. Defenders must protect a diverse landscape containing an unknown number of vulnerabilities of various types, whereas attackers need only find one or a few exploitable vulnerabilities to compromise a system. The problem is exacerbated by the uncomfortable reality that defenses of ever-increasing complexity are often outmaneuvered by simple, well-crafted exploits. Consequently, researchers and policymakers have increased focus on rebalancing the competition in favor of defense. One potentially promising approach is to create additional uncertainty for attackers by dynamically changing system properties in what is called a *cyber moving target* (MT).

Cyber MT techniques attempt to make systems less static, less homogeneous, and less deterministic to increase attackers' workload. MT techniques seek to randomize system components to reduce the likelihood of a successful attack, add dynamics to a system to reduce the lifetime of an attack, and diversify otherwise homogeneous collections of systems to limit the damage of a large-scale attack.[1] MT techniques are applicable to all aspects of defense including deception, protection, detection, and reaction. However, the majority of work to date has focused on deception and protection.

Many MT techniques have been proposed in academic literature (see the "Notable Moving-Target Techniques" sidebar); some were studied before the term *moving target* was coined, and others have MT implications without originally being intended for that area. Unfortunately, although many efforts have focused on the design and implementation of new MT techniques, little work has been done to evaluate their practical effectiveness, understand the full scope of their benefits, and enumerate and assess their gaps and weaknesses. This article presents a summary of several types of MT techniques, which we categorized into five major domains: dynamic networks, dynamic platforms, dynamic runtime environments, dynamic software, and dynamic data.

Overview

Movement for defensive purposes has long been studied and applied in the realm of physical confrontations, but its application to computing systems is comparatively new. We examine techniques for

1540-7993/14/$31.00 © 2014 IEEE

Notable Moving-Target Techniques

Many moving-target (MT) techniques and systems have been proposed and described in the literature, and a few have been implemented in the real world. One of our initial surveys identified more than 120 academic papers describing various cyber MT techniques,[1] and tens of new papers are being added to this body of literature every year. A comprehensive list of techniques would require many pages for the references alone, and even a cursory description of the most prominent techniques in each domain would require more space than is available. However, we provide a short description of one technique in each MT domain—dynamic networks, dynamic platforms, dynamic runtime environments, dynamic software, and dynamic data—and mention several others by name. Readers should refer to our survey for a longer list of MT techniques and their associated details.

Network address space randomization is a dynamic network technique that implements IP address randomization by modifying a Dynamic Host Configuration Protocol (DHCP) server to assign short IP leases.[2] A new IP address is assigned to each host as the previous short-term lease expires, with a goal of mitigating the propagation of hit-list worms. Other dynamic network techniques include Dynat for protocol obfuscation, DynaBone and Revere for dynamic routing, and Mute and Arcsyne for address hopping.

Moving-attack surface (MAS) is a dynamic platform technique that implements virtual server rotation for Web services.[3] MAS creates a pool of virtual webservers with a diversified software stack (for example, IIS on Windows versus Apache on Red Hat) and assigns each incoming Web request to one of the virtual servers at random. Other mechanisms, such as anomalous event detectors, random timers, and lifespan timers, also trigger virtual server rotations to mitigate attacks on vulnerable webservers. Other dynamic platform techniques include the Security Agility Toolkit for dynamically changing platform access levels, Genesis for diversification at the virtual machine level, multivariant execution for platform-level voting on applications, Talent for migration-based platform diversification, and several mechanisms to handle machine rotation.

Address space layout randomization (ASLR) is the most widely deployed dynamic runtime environment technique, having been implemented in a variety of mainstream server, desktop, and mobile platforms. The PaX team first implemented ASLR for Linux in 2001,[4] which was followed by integration into the Linux kernel in 2005 and into Windows and Mac OS X in 2007. ASLR implementations have steadily improved with newer operating systems: current 64-bit

cont. on p. 4

employing movement in computing systems, focusing on techniques available as research prototypes and commercial solutions. Although outside the scope of this analysis, many equally important management and analytic elements must also mature to guide effective use of MT techniques.

Management elements include command and control, decision support, and human interfaces required to properly direct the MT techniques.[2] Largely unspecified is the extent to and means by which humans will be involved in the command and control of these movements and what strategies, policies, and procedures should be in place to trigger such movement. Of particular note, organizations will need means to understand how MT techniques fit into overall defensive strategies. Whereas some techniques, such as address space layout randomization (ASLR), might be broadly applicable, others might require a connection to specific strategic objectives.

Consider a technique that migrates a vulnerable application across a diverse set of platforms. If attackers need to compromise all platforms in the set to achieve the objective (for instance, continuous denial of service), migration indeed increases the number of exploits and attacker effort. On the other hand, if the attack requires a few CPU cycles on any single platform (for instance, to trigger a kinetic effect), migrating across platforms actually makes attacks easier: adversaries need just one exploit from the full set of platforms.

Likewise, analytics are essential for developing a deeper understanding of how to best leverage the techniques and for communicating relevant movement data to operators and management capabilities. For example, past work has looked at understanding adversaries' and defenders' predictability and tailoring MT techniques accordingly.[3] Metrics are also required to measure the techniques' effectiveness and assess real-time operational status. Existing entropy metrics might serve as a starting point for assessing the uncertainty presented by many techniques but provide a narrow and often misleading view of a movement's effectiveness.

In this article, we do not consider the applicability of more general apparatuses for developing and understanding MT techniques, such as game theoretic approaches, experimentation, modeling, and simulation; rather, we focus on the MT techniques themselves.

Techniques

The full spectrum of MT techniques operates over a variety of computer system aspects to change anything that can be changed. Some techniques attempt to hide a target from potential attackers, whereas others assume

cont. from p. 3

vanilla Linux kernels randomize the stack, heap, libraries, and main executable with 22, 13, 28, and 28 bits of entropy, respectively, and the newest 64-bit "high-entropy" mode in Windows 8 incorporates 33, 24, 19, and 17 bits of entropy, respectively. Other dynamic runtime environment techniques include address space layout permutation for a fine-grained version of memory randomization, DieHard(er) for heap allocator randomization, instruction-level memory randomization for internal stack and heap randomization, function pointer encryption for scrambling of computed branches, G-Free for return-oriented programming gadget removal, RISE and practical software dynamic translation for instruction encryption, and CIAS and RandSys for system call randomization.

Reverse stack is a dynamic software technique that creates two versions of a program executable during compilation, one with the stack growing in the reverse direction from the original.[5] An execution monitor compares the original program's outputs with its stack-reversed variant and searches for discrepancies to detect stack-based buffer overflow attacks. Other dynamic software techniques include proactive obfuscation for creating diversified application replicas, program differentiation for applying binary instruction-level modifications, and multicompiler techniques.

Redundant data diversity is a dynamic data technique that randomizes the user and group identifiers (UIDs and GIDs) in a Linux system.[6] This technique is implemented in the kernel by performing an XOR operation on UID and GID with a diversified bit string to detect privilege escalation or masquerading attacks that change the UID and GID values. Other dynamic data techniques include data and algorithm diversity techniques for fault tolerance, data randomization and other scrambling and encryption techniques to prevent data modifications, and a Web service data diversification technique to randomize webpage formats and document object models.

Appearing elsewhere in this issue are several other articles that relate to MT defenses. "Countering Intelligent Jamming with Full Protocol Stack Agility" explains a new protective dynamic networks technique designed to thwart intelligent jamming attempts. "Defense on the Move: Ant-Based Cyber Defense" is a defensive detector technique that incorporates aspects of dynamic software. "Managed Execution Environment as a Moving-Target Defense Infrastructure" describes an MT management and analysis system that involves aspects of all five technique categories for both protective and detective capabilities. "Security through Diversity: Are We There Yet?" is a study of dynamic software challenges and progress and serves as a more detailed inspection of that domain.

References

1. H. Okhravi et al., *Survey of Cyber Moving Targets*, tech. report 1166, MIT Lincoln Laboratory, Sept. 2013.
2. S. Antonatos et al., "Defending against Hitlist Worms Using Network Address Space Randomization," *Computer Networks*, vol. 51, no. 12, 2007, pp. 3471–3490.
3. Y. Huang and A. Ghosh, "Automating Intrusion Response via Virtualization for Realizing Uninterruptible Web Services," *Proc. 8th IEEE Int'l Symp. Network Computing and Applications* (NCA 09), 2009, pp. 114–117.
4. "PaX Address Space Layout Randomization," Pax, 2003; http://pax.grsecurity.net/docs/aslr.txt.
5. A.G. Salamat and M. Franz, "Reverse Stack Execution in a Multi-variant Execution Environment," *Workshop Compiler and Architectural Techniques for Application Reliability and Security*, 2008.
6. A. Nguyen-Tuong et al., "Security through Redundant Data Diversity," *Proc. IEEE Int'l Conf. Dependable Systems and Networks with FTCS and DCC* (DSN 08), 2008, pp. 187–196.

attackers will always find a way in and instead focus on limiting the damage.

Dynamically changing IP addresses, randomizing memory layouts, and temporarily encrypting the contents of memory can all be considered MT techniques. A few techniques have seen mainstream acceptance and deployment, such as the implementation of ASLR in modern operating systems. Others such as port knocking, though well-described and well-tested, are used in more of an ad hoc fashion. The vast majority of proposed techniques have been studied only in academic papers and have never been tested or deployed in the real world. A comprehensive list of all techniques is beyond this article's scope; the *Moving Target Defense* books provide a starting point for research in this area.[4,5]

We categorize MT techniques into five major domains, according to the software stack model illustrated in Figure 1. Other categorizations are possible, but the rationale behind our approach focuses on each technique's implementation. For example, although both the dynamic runtime environments domain and the dynamic platforms domain involve the operating system, techniques in the dynamic runtime environments domain actually modify operating system internals, whereas dynamic platforms techniques leverage the differences between multiple, unmodified operating systems.

Attack Phases

A second major differentiating characteristic among domains is the phase (or phases) of an attack that they seek to disrupt. Attacks necessarily proceed through multiple phases; we have chosen a five-phase attack kill chain to illustrate the major phases involved. Other valid kill chains exist, and various attack types might combine

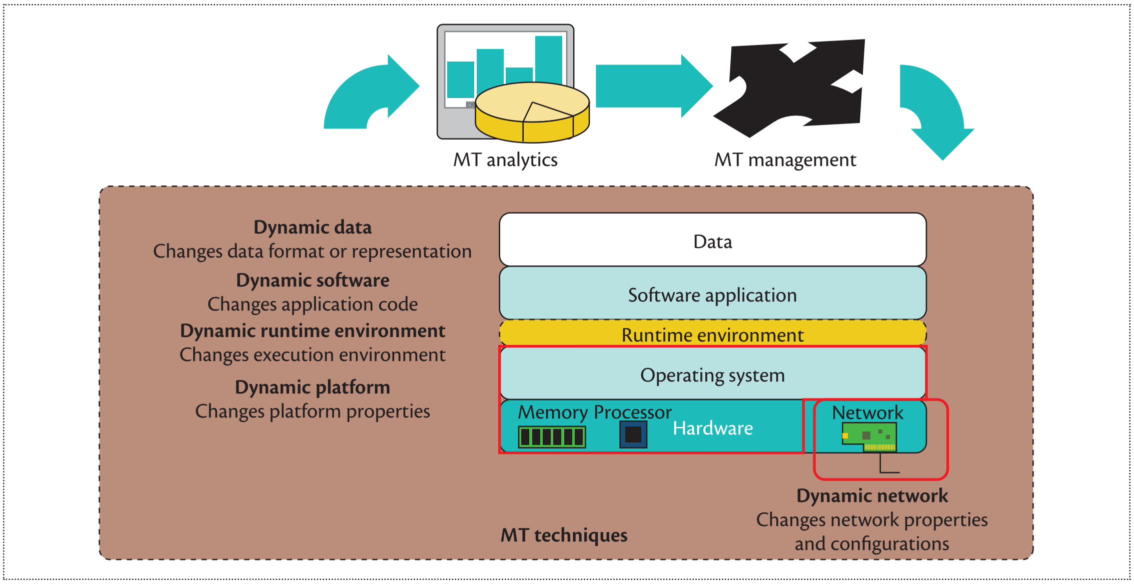

Figure 1. Moving-target (MT) techniques may be categorized into five different domains according to their place within the execution stack. Analytic and management components contribute to a feedback loop that can dynamically improve defense.

or divide some phases of our model, but we've found it sufficient and useful for discussion of MT domains and techniques:

- *Reconnaissance*. This phase encompasses the means by which attackers find the target and collect basic information on it. It's mainly limited to observation to learn more about a target. Examples include finding an appropriate spear-phishing target or determining a host's location using basic IP scanning.
- *Access*. This phase covers actions attackers take to collect detailed information about a chosen target. For example, attackers might launch probes against a webserver to determine the architecture, operating system, exact server application version, or configuration.
- *Development*. During this phase, attackers develop an attack that can exploit a vulnerability on a target. This phase can be performed offline without any involvement of the target.
- *Launch*. The launch phase involves attackers delivering the attack payload to the target to compromise it. The payload can be delivered via a variety of means, such as over a network or through infected media, and might come in many forms, such as an instance of malicious executable code or large volumes of otherwise benign data intended to overwhelm a service on the target.
- *Persistence*. If attackers mean to persist, this phase is used to ensure that they can maintain a foothold in the compromised system. One prominent example of ensuring persistence is installing other entry points (for example, backdoors) on the target.

Appropriate threat models are crucial to understanding the benefits and effectiveness of a given MT technique. In many cases, designers explicitly or implicitly provide a technique's threat model. Techniques that lack well-defined threat models are of questionable effectiveness, since a technique can't be evaluated without appropriate context to understand what it is trying to defend against.

Dynamic Networks

Techniques in the dynamic networks domain continuously modify network properties to increase the workload required and reduce the probability of success for network-borne attacks. Dynamic network modifications often focus on frequent changes to addresses and ports but might also include rotating protocols, using overlay routing networks, and changing the logical network topology.

Dynamic network MT techniques are primarily intended to hinder attacker reconnaissance but might also play a role in preventing the launch of an attack. Such techniques can prevent attackers from discovering an exploitable condition or invalidate the results of a previous scan before an attack can be developed. If attackers are aware of an exploitable condition, the attack might be undermined if the target is constantly shifting to new locations. An example of such a technique is Dynat,[6] which attempts to mitigate scanning attacks by obfuscating parts of network packet headers. Software-defined networking and overlay networks have also seen application in this area, such as with Revere,[7] which uses a redundant overlay network to dynamically distribute security updates without reliance on a single central authority for all nodes.

However, such defensive techniques might hinder legitimate use of the system. Services that must reside at known network locations or remain otherwise reachable, and support well-defined public communications protocols (for example, a webserver), can't use defenses that hinder easy accessibility without also sacrificing their core reason to exist. Other services that utilize the network only as an untrusted communications channel have a wider range of options in their techniques.

A few dynamic network MT techniques have been employed in commercially available products. Others haven't been widely adopted, although practical implementations are available. Content delivery networks are intended primarily to improve content load times through distribution and localization but also offer dynamic protection against denial-of-service attacks. By distributing an increasing load over an increasing number of servers as demand rises, a weak form of address hopping occurs: requests in particular locales are shunted to new servers, and the system remains available. At the other end of the scale, port knocking has been well-known for more than a decade but hasn't been widely deployed. Advanced forms of port knocking use dynamically generated port sequences to signal that a connection is desired, a technique that, ironically, attackers have used to maintain access to a machine once it's compromised.

Weaknesses

The greatest weakness in dynamic network techniques is the lack of a well-defined threat model. If the technique creates a closed network—that is, prevents machines inside the network from connecting to those on the outside, and vice versa—the dynamic network will break the client–server model. Client–server models are fundamental to network communications and services, and any attempt to hide a public server or otherwise make it less accessible works against its very purpose. There are limits on how much obfuscation can be applied to network traffic without breaking the communications link.[6] Moreover, virtual private networks (VPNs) can be used to create a closed network with stronger security properties than dynamic networks when isolation is desirable.

On the other hand, if the technique maintains an open network in which machines can still communicate with the outside, systems remain vulnerable to client-side attacks in which the payload is delivered via a client-initiated session with a malicious server. Changing the network properties doesn't interfere with the establishment of such a connection, and thus it has little value in preventing client-side attacks. Drive-by downloads are an example of client-side attacks.

Moreover, the effectiveness of randomization depends on the amount of uncertainty created for attackers. The uncertainty in a random value can be measured via the entropy, and entropy depends on the number of possible values for a random variable, and their probabilities. Limited entropy is a problem in many dynamic network techniques, particularly with regard to the Internet. Both IP addresses and port numbers are subject to strict constraints due to network infrastructure, even aside from the maximum number of values in the current technical specifications. Even within the movement that is possible, the current network infrastructure isn't well-suited to keep up with a fast-changing topology, and considerable overhead is incurred in doing so on closed networks with adjusted infrastructure.

Research Directions

Priority in dynamic networks MT research should be given to approaches that treat the network as an untrusted communication channel between known parties. When private entities are forced to communicate over public interfaces, particularly when VPN use is precluded, information leakage is inevitable through side channels, even if all other aspects of the communication are obscured. Techniques that attempt to mitigate these issues benefit our widely connected environments.

As applied to broader public networks, current dynamic network techniques have an ill-defined threat model and lack clear definitions that remain valid when confronted with the existing client–server architecture. These techniques provide minimal defensive benefits for considerable infrastructure overhead, and there is not yet a clear path to overcome these difficulties. Nevertheless, new research in this area remains robust. We advise that researchers of new dynamic network techniques give careful thought to the intended use case; the fundamental threat model must be clarified before new research on the broader space of dynamic networks can be of full use.

Dynamic Platforms

Techniques in the dynamic platforms domain change the properties of the computing platform in an effort to disrupt attacks that rely on specific platform characteristics. The properties that can change include the operating system, processor architecture, virtual machine instance, storage systems, communication channels, and other low-level environmental factors. Techniques in this area might migrate applications from machine to machine or execute the same application in parallel in multiple architectural contexts. An example of the former technique is Talent, which regularly migrates live applications across a large pool of machines running different operating systems on different architectures.[8] *N*-variant systems are an example of the latter, comparing multiple simultaneous versions of the same application and restarting them from known good states if they diverge.[9]

Dynamic platforms techniques offer some defensive benefits against all five phases of the attack kill chain, but more protection is offered in the access, development, and persistence phases. Techniques in this domain are specifically intended to vary system properties exposed to attackers during the access phase of the kill chain. Furthermore, exploit development is rendered substantially more difficult for attackers when exploits for multiple diverse platforms are necessary to carry out an attack, particularly if an application is running in parallel on multiple instances simultaneously. Finally, when an attack is successful, the regular migration of applications to other platforms limits attackers' ability to maintain useful persistence.

Dynamic platform techniques have received substantial attention from the research community but have realized only limited deployment in the real world to date. A popular use case is to create transient platforms using live bootable media or disposable virtual machine instances, although application state is usually discarded after each session rather than persisted to a new platform.

> **“Address space randomization techniques have achieved the greatest maturity of any MT technique to date across all domains in the form of ASLR, which has been commercially deployed in the real world.”**

Weaknesses

Applications that require state to be maintained or synchronized across platforms present a significant challenge in the dynamic platform domain, because state is extremely difficult to extract from a running process in a platform-agnostic format unless the application is specifically designed to support it.[8] The advent of cloud computing might result in more applications designed to this paradigm, but a more serious problem arises when transferring state: if attackers are present, a full state transfer might allow them to persist in spite of the platform migration. A trade-off might exist between migrating enough state to be useful and not enough state to allow attackers to persist.

Another problem is that dynamic platform techniques inevitably increase the attack surface of the systems being protected. Extra code used to control and manage migrations creates more code that can be attacked. Perhaps more subtly, certain attack models let attackers benefit from additional platforms, such as when attackers seek to control an application running on a particular platform. A vulnerability might exist only on a certain platform, and attackers need only control the application for a limited period of time to complete a successful attack. As additional platforms are included in the dynamic platform system, it becomes increasingly likely that a vulnerable platform will be exposed to attackers and fall to a successful compromise.

Finally, an implicit assumption of this domain is that a diverse pool of platforms actually exists. In actuality, a limited number of useful operating systems, architectures, and platforms exist, and creating new ones is a nontrivial task. Two platforms might provide sufficient diversity for certain techniques in this domain, but the available variety remains a concern.

Research Directions

Scalability concerns must be addressed to open up a path to wide adoption of dynamic platform techniques. Specifically, automated mechanisms to create platform replicas are needed to populate platform pools in an affordable and scalable manner. Performance is also an issue, because movement between platforms is a heavyweight task that can impose significant costs each time it occurs. Perhaps the greatest problem is the lack of a method to universally transfer application state in a platform-independent manner; there may be lessons to apply from the high-performance and distributed computing communities.

Dynamic Runtime Environments

Techniques in the dynamic runtime environments domain randomize the environment in which the application operates and can be divided into two subdomains: address space randomization (ASR) and instruction set randomization (ISR). Techniques in this domain aim to prevent attackers from exploiting software vulnerabilities in order to compromise a machine by exploiting an application's vulnerabilities. ASR techniques have achieved the greatest maturity of any MT technique to date across all domains in the form of ASLR, which has been commercially deployed in the real world. ASLR has received significant attention in the literature and varying levels of support in modern operating systems and software, and ASLR implementations of varying potency are now standard on all major desktop operating systems as well as certain mobile systems.

ASR randomizes the layout of an application's virtual memory at runtime and is usually implemented through modifications to the core operating system, sometimes requiring additional support in the application. For example, an ASR technique might change the base addresses of certain memory segments including the stack, heap, and shared libraries. Others might apply intrasegment randomization such as shuffling stack frames inside the stack.

Methods in the ASR subdomain seek to transform a deterministic memory layout into a randomized one. This prevents the attacker from being able to use a known memory address to redirect control flow or to read out a particular piece of data. Dynamic libraries represent an early implementation of a technique in this domain, because their construction allows any given library to be placed at a different virtual address within each process's memory space.

ISR randomizes the actual instructions in an application and might take place in the operating system, the application, or even the hardware. This stops attackers from predicting how the program will execute. One example technique in this space involves encrypting each instruction with a key chosen at load time, and then decrypting it immediately prior to execution.[10]

The techniques in this domain assume that attackers already have an exploitable vector into the program and thus aim to prevent attackers from completing the attack. They complicate the attack development and launch phases by making it more difficult to exploit memory corruption vulnerabilities. Buffer-overflow exploits are a notoriously common form of memory corruption and can be used to inject malicious code or hijack the control flow. When techniques in this domain aren't applied, known instructions can be inserted at known locations (that is, code injection) or control might be redirected to existing instructions, also at known locations (such as code reuse or return-oriented programming). When techniques in this domain are applied, attackers can only guess the correct form of an injected instruction or the correct location of an existing or injected instruction. Although successful attacks can't be blocked completely, the randomization aspects of dynamic runtime techniques reduce the likelihood of success and help prevent large-scale attacks against multiple systems.

ASR Weaknesses

Despite the strong potential for ASR techniques to substantially increase the difficulty for attackers, several weaknesses in current ASLR implementations have limited their effectiveness.[11] One major weakness is that only a portion of the application's memory space is randomized in standard configurations; other portions remain static. For example, randomization of the dynamic libraries is common, but the base program image isn't randomized by default. Attackers can develop meaningful payloads using the static program image alone. A related problem with ASLR is that relative addresses in program segments often remain unchanged. It's common practice to randomize only the base address of an entire memory segment and leave the individual portions of that segment in the same position relative to each other, thus allowing attacks to bypass ASLR defenses with the use of relative addresses.

ASR techniques typically operate with the assumption that the contents of memory aren't known to attackers. Attackers who can exploit a complementary memory disclosure vulnerability might be able to determine the randomized locations of objects and use those locations in later attack stages. Currently, no ASR techniques can resist an arbitrary memory disclosure attack.

Limitations in available entropy present an additional weakness: the area over which an address is randomized is often too small in practice due to the defended system's architectural limitations. Small memory spaces are vulnerable to brute-force attacks, in which addresses can be guessed until discovered, or heap-spraying attacks, which can fill large portions of the address space with malicious objects. In many cases, attackers can complete brute-force attacks within minutes due to the relatively small address space of 32-bit operating systems[11]; modern 64-bit operating systems often offer improved but still surprisingly low entropy.

ISR Weaknesses

To date, performance overhead has been the greatest barrier to the adoption of ISR techniques. Without pervasive hardware support, most ISR techniques must rely on software emulation and, as such, incur substantial overhead. Non-MT defenses such as nonexecutable memory, which prevents a memory page from being both writable and executable, also protect against code injection. Nonexecutable memory is well-supported in hardware and has seen widespread adoption, suggesting that similar hardware support for ISR techniques applied to popular architectures might lead to similar adoption and usage.

Related to the lack of hardware support, some ISR techniques rely on low-overhead methods such as a simple XOR operation to "encrypt" instructions. Use of XOR, or similar weak encryption methods, allows key recovery if attackers can read just one known instruction. Key recovery allows attackers to inject correctly encoded instructions of their own without difficulty.

Research Directions

In the dynamic runtime environment domain, researchers must focus attention on the composition of comprehensive techniques that randomize all components of memory and thus prevent attackers from preying on the residual static components. They should also focus on increasing the spatial independence of objects within a given memory segment. This avenue of investigation will not only provide better protection against relative address–based attacks but also limit the usefulness of memory disclosure attacks: disclosing one object's location wouldn't necessarily disclose all other objects' locations. The assumption of memory secrecy in general must be examined, and new models that consider attackers with visibility into memory must be developed. Finally, researchers should reexamine ISR techniques in light of new encryption hardware making its way into mainstream architectures, possibly providing an opportunity to revitalize the concepts with fewer performance penalties.

Dynamic Software

Techniques in the dynamic software domain modify the application such that the internal state is no longer deterministic relative only to the input, while ensuring that functionality is unaffected. Diversification is achieved by substituting equivalent program instruction sequences for each other, changing the instructions' order and format, rearranging the internal data structure layout, and otherwise altering formerly static properties of the application. Transformation of this nature reduces the applicability of specific instruction-level exploits for a given piece of software and forces attackers to guess which software variant is in use. For example, Figure 2 shows two different but functionally equivalent instruction sequences.

These techniques can be applied on a gross or fine-grained scale, either creating a corpus of semantically equivalent binaries for mass distribution or using a single application with its own internal randomization capability. External randomization can be applied at compile time or through binary rewriting. Several techniques in this area propose a voting system in which two or more diversified binaries execute in parallel with identical input while a trusted execution monitor detects aberrant behavior in any of the variants.[12] Other techniques, such as GenProg,[13] attempt to dynamically patch software vulnerabilities as they're discovered, rewriting application code to prevent exploitation.

Dynamic software techniques seek to disrupt an attack's development and launch phases. Development is hindered due to the uncertainty as to which code is being executed in any given software instance. Code injection and reuse become complicated due to the many variants. For an attack to succeed in all cases, attackers must discover a nonrandomized path to the application or use an exploit that doesn't rely on specific code instructions and a static internal data structure layout. Probabilistic attacks might still yield positive results for attackers, but a trusted execution monitor can guard against that possibility.

Sequence 1	Sequence 2

`xor eax, eax`
`shl ebx, 0x3`
`pop edx`
`jmp edx`
	`mov eax, 0x0`
	`imul ebx, ebx, 0x8`
	`ret`

Figure 2. Functionally equivalent instruction sequences in x86 assembly. Each uses different operations to produce the same intended result over the span of the entire sequence, but intermediate steps are not equivalent and cannot be used to produce the same unintended result by an attacker reusing the existing code.

Techniques in dynamic software aren't widely deployed in the real world, and those techniques that do exist are largely limited to academic and research environments and aren't available for wide-scale experimentation.

Weaknesses

A major weakness in the dynamic software domain is that it's very difficult to ensure (automatically or otherwise) that translated software provides functionality equivalent to the original. Heavyweight binary translation and emulation impose significant performance overhead, lack scalability, and might create unexpected side effects even while appearing to operate as expected. Compiler-based approaches allow greater confidence in the correctness of programs but can slow down performance and create runtime inconsistencies between program variants (even if not changing the overall result) that complicate dynamic comparisons.[12] Furthermore, compiler-based approaches naturally require access to source code, which in turn necessitates continuous vendor collaboration in the case of commercial software.

Techniques that employ an execution monitor supervising multiple variant applications are common in dynamic software and allow for greater security and detection capability with regard to that application. The price is increased performance overhead, especially in resource usage. Using an execution monitor also expands the attack surface and imposes a second point of failure.

Finally, many programs are crafted (and compiled) for maximum performance. Semantically equivalent programs created for these applications will result in degraded performance. Real-time and

Format 1	Format 2

<Age=23;
Gender=Male;
ID=132573;
Salary=$75000;>
	<ID=00132573;
	Gender=M;
	Salary=75K;
	Age=10111;>

Figure 3. The same data presented in two different formats. Diversity in data formatting can be used to complicate exploit development for an attacker.

high-performance applications might not be able to tolerate such degradation in highly diversified binaries.

Research Directions

Compiler-based randomization is both comprehensive and compatible with many techniques across other domains and is well-studied in other contexts. As such, focus on these techniques will likely yield the best results in this domain, although practical implementation depends on appropriate vendor support. Further research on binary-based translations or distribution channels that make it easy for vendors to distribute a diverse set of semantically equivalent binaries would allow diversified software to be deployed on a static basis among the user base. Such distribution, although not as beneficial as software that re-randomizes upon each invocation, is still a great improvement over the status quo.

Dynamic Data

Techniques in the dynamic data domain change the internal or external representation of an application's data in such a way as to ensure that the semantic content is unmodified, but unauthorized use or access is hindered. This is accomplished by changing the format, syntax, encoding, and other properties of the data representation. Attackers' infiltration attempts might be rendered detectable when valid data is presented in an improper format, and exfiltration attempts might not yield data in a useful format.

Some techniques in this domain are rooted in methods originally designed to guard against data corruption, such as the data diversity technique described by Paul Ammann and John Knight,[14] which is configured to run computations on multiple distinct data representations and vote on the results to detect corrupted (malicious) input. Data encryption techniques, such as data randomization, encrypt portions of the application in memory to hinder data extraction and infiltration.[15] For example, Figure 3 shows the same data represented in two different ways.

Similar to those in the dynamic software domain, dynamic data techniques seek to complicate the development and launch phases of an attack. Attack development is impaired due to the difficulty in crafting an appropriate payload for multiple data representations. An exploit that depends on a particular data format is less likely to succeed.

Almost all modern applications employ well-specified data layouts for their internal use, whereas I/O is largely limited to a selection of standard or custom data formats, as appropriate. No dynamic data techniques are deployed in the real world, and a survey of existing literature yields only a few examples, most of which focus on memory encryption or limited randomization of certain data (for instance, user identifiers).

Weaknesses

Techniques in this domain suffer from a lack of diversity in allowable data encodings, because most standard binary formats support one canonical representation. The continuing desire within the computing community to standardize and facilitate easy communication means that new binary formats aren't encouraged unless they fulfill an unmet need or offer substantial improvements over an old method. Even in text, where many canonical versions of the same information are possible, standard layouts are encouraged to improve interoperability.

The use of techniques that operate on a diversified set of data formats also results in an expansion of the attack surface. Each additional data format implicitly carries with it the need for new parsing capabilities and a new set of error-checking code. Furthermore, the proliferation of new data formats will likely lead to a loss of compatibility, with standard data manipulation utilities that don't understand the nonstandard formats—again, requiring the application to add new utility code instead of using well-tested standard libraries.

All this new code might harbor new vulnerabilities, particularly if I/O is involved. Even when multiple formats exist, their number might be insufficient to thwart attackers. Encryption methods provide sufficient protection of an application's internal data state, but the continued lack of a practical homomorphic encryption scheme requires that all data be decrypted back to its original representation before any processing can be performed.[15] This presents a window of vulnerability to attackers. From a more practical perspective, dynamic data techniques impose an increased burden on both application development and runtime performance due to the need to process and monitor the diverse data representations.

Research Directions

Because of the expanded attack surface and increased code complexity when operating over diversified data formats, future research should be directed at finding effective methods of data encryption for data residing in main memory. Applications that operate on only one

Table 1. Primary attack phases disrupted by techniques in the five domains.

MT domains	Attack phases				
	Reconnaissance	Access	Development	Launch	Persistence
Dynamic networks	■			■	
Dynamic platforms		■	■		■
Dynamic runtime environments			■	■	
Dynamic software			■	■	
Dynamic data			■	■	

canonical representation of data are more robust and secure than those operating on many diversified data representations, but the data need only be in its canonical form when actual operations are taking place. If the data is transformed when it isn't being actively manipulated, the window of vulnerability is reduced, and attackers can neither exfiltrate nor inject data in a useful form, so long as the encryption scheme is robust.

Further research into text-based external communications might also provide a means of attack detection and perhaps outright prevention. Text is no less vulnerable to the diversified data problems of binary formats, but the community has traditionally been willing to make extra allowances for textual data in parsing and manipulation, perhaps because it's human understandable.

Discussion

Based on our analysis of the MT techniques' strengths and weaknesses, we have identified three major properties for an effective defense—MT techniques should be comprehensive, timely, and unpredictable. Comprehensive defenses demand the inclusion of all components that could be used in a given attack phase. As previously discussed, if an application randomizes the location of its library code but leaves the program image at a fixed location, it has failed to significantly improve its defensive posture. The application remains vulnerable to a code reuse attack in which adversaries simply ignore the randomized code and target the fixed code.

MT defenses' movements should also be timely with respect to adversary observation and attack points. If adversaries have an opportunity to observe the result of a movement, and that knowledge presents them with an opportunity to launch an attack, another movement must be made before they can complete the attack. In addition, if the MT defense relies on environmental diversity, it must expose attackers to this diversity within the relevant attack period.

Consider a dynamic platforms technique such as Talent,[8] configured to migrate an application among three platforms. The defense is predicated on forcing attackers to exploit vulnerabilities on multiple, diverse platforms. If the migration time is less than the attack time, the technique has achieved its objective: attackers must have a second vulnerability available on the new platform to continue the attack. However, if the migration time is greater than the attack time, the MT technique actually diminishes security because attackers have a choice of three platforms in which to locate a vulnerability, rather than only a single, fixed platform.

Effective movements must also be unpredictable. If attackers can predict the next movement, then the movement provides no additional security. If attackers can predict the movement with high probability, or if only a narrow range of movement is available, the movement provides only a small amount of additional security.

Threat models and use cases provide the necessary context for evaluating sufficient thresholds of unpredictability. For example, in the case of an automatically respawning webserver, 16 bits of entropy in an ASLR implementation require only minutes to break through brute force. On the other hand, if a process must be manually restarted after each guess (because incorrect guesses likely cause segmentation faults), even a low amount of total entropy might not provide attackers with a significantly high probability of success in a reasonable amount of time.

Table 1 provides a summary of the five domains broken down by the attack phases predominantly disrupted by techniques in that domain.

MT techniques, though currently underdeveloped, are important tools for defense against cyberattacks. Those few in active deployment that are still nascent have demonstrated the potential to increase the difficulty level for attackers. In combination, techniques from each of the five domains can guard against all phases of the attack kill chain and thus dramatically improve our cyber posture, though this requires a deep understanding and mitigation of their weaknesses and shortcomings.

Techniques in each of the five MT domains have various strengths and weaknesses. Some are most appropriate for a specific mission in which needs are specialized and security is emphasized over performance. Others are well-suited for integration into our general-purpose computing environments, and the community should work to ensure that systems and applications have the appropriate mechanisms built in to support these. Many techniques need further development, and all require dedicated analysis.

However, the field is in a strong position to increase attacker workload, with both a growing research base and widespread deployments on which to build. ■

Acknowledgments

We thank William Leonard and Mark Rabe for their significant contributions to this work. 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 authors and are not necessarily endorsed by the US government.

References

1. "Cybersecurity Game-Change Research & Development Recommendations," Networking and Information Technology Research and Development Program, May 2010; www.nitrd.gov/pubs/CSIA_IWG_%20Cybersecurity_%20GameChange_RD_%20Recommendations_20100513.pdf.
2. M. Carvalho et al., "Command and Control Requirements for Moving-Target Defense," *IEEE Intelligent Systems*, vol. 27, no. 3, 2012, pp. 79–85.
3. R. Colbaugh and K. Glass, "Predictability-Oriented Defense against Adaptive Adversaries," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics* (SMC 12), 2012, pp. 2721–2727.
4. S. Jajodia et al., eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, Springer, 2011.
5. S. Jajodia et al., eds., *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, Springer, 2012.
6. D. Kewley et al., "Dynamic Approaches to Thwart Adversary Intelligence Gathering," *Proc. DARPA Information Survivability Conference and Exposition II* (DISCEX 01), vol. 1, 2001, pp. 176–185.
7. J. Li, P. Reiher, and G.J. Popek, "Resilient Self-Organizing Overlay Networks for Security Update Delivery," *IEEE J. Selected Areas in Communications*, vol. 22, no. 1, 2004, pp. 189–202.
8. H. Okhravi et al., "Creating a Cyber Moving Target for Critical Infrastructure Applications Using Platform Diversity," *Int'l J. Critical Infrastructure Protection*, vol. 5, no. 1, 2012, pp. 30–39.
9. B. Cox et al., "N-Variant Systems: A Secretless Framework for Security through Diversity," *Proc. 15th Usenix Security Symp.*, 2006, pp. 105–120.
10. G.S. Kc, A.D. Keromytis, and V. Prevelakis, "Countering Code-Injection Attacks with Instruction-Set Randomization," *Proc. 10th ACM Conf. Computer and Communications Security* (CCS 03), 2003, pp. 272–280.
11. L. Szekeres et al., "SoK: Eternal War in Memory," *Proc. IEEE Symp. Security and Privacy*, 2013, pp. 48–62.
12. B. Salamat et al., "Runtime Defense against Code Injection Attacks Using Replicated Execution," *IEEE Trans. Dependable and Secure Computing*, vol. 8, no. 4, 2011, pp. 588–601.
13. C. Le Goues et al., "GenProg: A Generic Method for Automatic Software Repair," *IEEE Trans. Software Eng.*, vol. 38, no. 1, 2012, pp. 54–72.
14. P.E. Ammann and J.C. Knight, "Data Diversity: An Approach to Software Fault Tolerance," *IEEE Trans. Computers*, vol. 37, no. 4, 1988, pp. 418–425.
15. C. Cadar et al., *Data Randomization*, tech. report MSR-TR-2008-120, Microsoft Research, Sept. 2008.

Hamed Okhravi is a member of the technical staff at MIT Lincoln Laboratory. His research interests include systems security, science of security, security metrics, and operating systems. Okhravi received a PhD in electrical and computer engineering from the University of Illinois at Urbana-Champaign. Contact him at hamed.okhravi@ll.mit.edu.

Thomas Hobson is a member of the technical staff at MIT Lincoln Laboratory. His research interests include systems security, software security, and vulnerability analysis. Hobson received an MS in information security from Carnegie Mellon University. Contact him at thomas.hobson@ll.mit.edu.

David Bigelow is a member of the technical staff at MIT Lincoln Laboratory. His research interests include operating systems, storage, real-time systems, high-performance computing, and reliability. Bigelow received a PhD in computer science from the University of California, Santa Cruz. Contact him at dbigelow@ll.mit.edu.

William Streilein is an assistant group leader in the Cyber Systems and Technology Group at MIT Lincoln Laboratory. His research interests include machine learning and modeling and simulation, especially as applied to problems in cybersecurity, security metrics, and cyber moving target. Streilein received a PhD in cognitive and neural systems from Boston University. Contact him at wws@ll.mit.edu.