ASLR was a security game changer. Could expanding the concept be one, too?

By John Matthew Holt and Apostolos Giannakidis

As long as there has been a commercial internet, cybersecurity has largely been predicated on the concept of guess work. Looking for patterns or matching words and phrases that were close enough to known or variations of known vulnerabilities/attacks have been the norm. Heuristic-based solutions are, by definition, trial and error.

But with a seemingly never-ending increase in vulnerabilities, hackers and attacks, the cyber security status quo is simply not sufficient to provide the level of certainty needed to instill confidence and improve reliability in an app driven, interconnected world. We need higher accuracy, less complexity and more simplicity as more devices, applications and individuals join the internet.

The inspiration for one powerful new solution can be found in the past - in a seemingly unrelated technology.

Attacking the OS to get to an app

Think back to 2001. Hackers were still largely amateurs who created malware or penetrated systems for fun or academic credibility. The first criminal charges were filed under the US Digital Millennium Copyright Act (DMCA) against a Russian programmer while he was attending Def Con. And, Microsoft was the target of one of the first DoS attacks.

In the summer of that year, we were still nearly a year away from Bill Gates declaration that security would become a core focus of Microsoft and two years from the formation of Anonymous.

2001 was also the year that a new approach to OS security appeared: Address Space Layout Randomization or ASLR. First introduced as patch to the Linux kernel by the PaX Project and later added as a default OS feature by OpenBSD in 2003, ASLR is now a standard component of most mainstream operating systems, from MS Windows and Apple OS to Android and even Nintendo 3DS.

An idea whose time had come

Before ASLR was introduced, awareness of attacks against OS components was generally limited to the radar screen of security experts and hackers. In those days, address locations were well known or easy to predict.

Hackers, however, had discovered a highly effective way of launching attacks. To be successful, all they had to do was use known addresses or guess the location of a process or function somewhere in the OS memory.

At the time, just about every attack was a Zero Day. Once inside the OS, the most common approach was a buffer overflow attack to allow hackers to expand beyond set parameters within the OS and inject malicious code into more vital areas.

The approach of the PaX Project was deceptively simple: randomize the address of OS processes. If a malicious attacker entered a wrong address location, the target application would crash, the exploit would be halted and, in some cases, a system admin might be alerted.

Same vulnerabilities; same attacks; new approaches

In the intervening years, the effectiveness of ASLR and other improved security techniques have forced malicious hackers to find new ways to exploit known (and unknown) vulnerabilities in applications. ASLR is highly effective at stopping code injections where hackers attempt to use the address space of OS processes to exploit a flaw, but ASLR has not removed code injections from a hacker's toolkit.

Code Injections are listed as #6 on MITRE's list of CVEs with 2067 related vulnerabilities. If you visit CVEDetails.com, you'll find a listing of more than 700 known exploits dating back 20 years.

Code injection attacks exploit specific vulnerabilities and the most common approach to protecting against them is to identify and fix/patch the vulnerability. As with so many vulnerabilities, that is not always practical for a wide variety of reasons such as the risk of breaking an in-production application or the time and expense of refactoring a legacy app.

If a vulnerability cannot be mitigated, the application functionality/service associated with the vulnerability can be disabled or removed. For the obvious reasons, this approach is rarely feasible.

Applications today are protected more often than not by ineffective Web Application Firewalls and other tools that rely heavily on instrumentation or filters to guess if a call is a malicious attack or a permissible action. Such heuristic-based approaches often produce false positives at an unacceptably high rate that only increases as the volume and sophistication of attacks rise.

Randomizing the address space of OS processes makes sense since any code injection payload at this level references memory addresses. However, at the JVM level, code injection payloads do not reference memory locations. That means randomizing the address space at the Java Virtual Machine (JVM) level would not mitigate such attacks.

However, randomizing the namespace within the JVM does. In fact it can virtually remove the entire class of attacks - code injections - from the list of likely exploits.

Introducing Name Space Layout Randomization - NSLR

Name Space Layout Randomization or NSLR is the equivalent of ASLR for Java-based applications. NSLR has been developed by Waratek, a runtime application security company that uses application virtualization to improve app protection and operations.

NSLR hardens the JVM by randomizing the JDK namespace (Java packages), which makes code injection exploits so difficult to execute that they become unfeasible.

A malicious attacker who attempts to introduce a Code Injection exploit in Java needs to know the exact names of classes and packages to be invoked. Making the Java class packages non-deterministic, or randomized, makes any exploit unsuccessful; in effect, it defeats any code injection exploit.

For example, with NSLR enabled, the *java.lang.System* class will be randomized and renamed to something like *java$85rbuLjHNERijUhN.lang.System*. Any exploit that tries to invoke *java.lang.System* will automatically fail. For attackers to successfully execute an attack they would need to know the randomized package name. For additional protection Waratek randomizes the package each time the JVM boots.

(The Waratek Runtime Application Security Platform is based on patented application virtualization technology where an entire application stack operates as a guest inside a virtual container sitting on top of a host JVM. Because of this architecture, it is impossible to access or exploit a randomized host package name from the guest JVC.)

Attempts to brute force the system and retrieve the randomized package name is not feasible, either. Waratek's standard configuration includes NSLR with a minimum level of security at 96-bit names, which would require several thousand years to crack the encryption. Names can be randomized up to up to 1024 bits.

A code injection attack attempted in an NSLR- enabled JVM will generate a ClassNotFoundException, ending the attack.

NSLR is the kind of emerging security solution that could be the replacement for heuristics we've all be waiting to arrive.

____

John Matthew Holt is CTO and Co-Founder of Waratek and **Apostolos Giannakidis** is Lead Security Engineer at Waratek