ASLR

Sebastian Kricner `http://tuxwave.net`

August 2010

Zusammenfassung

Was ist ASLR? Ist es die Lösung für die Probleme der IT-Sicherheit? Dieses Papier gibt einen kurzen Einblick über ASLR. Hierbei wird die Thematik vorwiegend auf Linux Basis betrachtet.

1 ASLR - Definition und Kurzbeschreibung

ASLR steht für Adress Space Layout Randomization.

ASLR ist seit Linux Kernel Version 2.6.8 standardmäßig integriert. Den Anfang machte zunächst der PaX Patch, welcher bereits seit Jahren verfügbar ist. ASLR war und ist Bestandteil des PaX Patches. D.h handelt es sich bei der ASLR Implementation um einen Teil dessen, was PaX implementiert.

Auch seit Windows Vista ist ASLR implementiert. Je nach Konfiguration wird ASLR gleichzeitig mit DEP (Data Execution Prevention) eingesetzt. Zu beachten ist, dass es auch weitere Maßnahmen gibt, wie GS. GS ist ein Stack-Schutz auf Compilerbasis. Auch GCC bietet einen Stackschutz, sofern beim Vorgang des Compilens die Option aktiviert ist.

2 Funktion

ASLR randomisiert den Adressbereich bei der Ausführung eines Prozesses. Hierbei werden folgende Adressbereiche randomisiert:

- Stack
- Heap [1]
- VDSO & DSO

Dynamic Shared Objects & Virtual Dynamic Shared Objects
DSO sind Funktionen aus Programmbibliotheken, welche in den Arbeitsspeicher geladen werden. VDSO können hingegen Syscalls des Kernels als linkbare Bibliotheken zur Verfügung stellen.

Nicht betroffen sind:

- Code Segment
- BSS Segment
- Datensegment

3 Wirkungen von ASLR

Angriffe auf Sicherheitslücken werden erschwert. Es ist nicht mehr möglich bei einem Angriff einen festen Wert für das Überschreiben des Instruction Pointers zu verwenden. Auch ret2libc Angriffe sind kaum mehr möglich, da ebenfalls die Adressen der libc Funktionen im Speicher randomisiert sind. In Kombination mit einem nicht ausführbaren Stack wird der mögliche Umfang des Nutzen für den Angreifer erheblich minimiert.

Da bei einem Angriff auf einen Pufferüberlauf mit einem nicht gültigen Instruction Pointer ein Absturz der Software zwangsmäßig ist, ist es aufgrund der Randomisierung statistisch sehr unwahrscheinlich beim ersten Angriff diese Sicherheitslücke zu nutzen. Dies bedeutet, jedoch, dass auch ASLR keinen vollständigen Schutz gegen Stack Korrumpierung gibt.

Da es nicht möglich ist einen beliebig großen Adressbereich für die Randomisierung zu nutzen,

1. Der Heap war in den ersten Kernel Versionen mit ASLR Implementierung nicht randomisiert.

hat dies die Folge, dass die Speicherbereiche bei der Ausführung des Programms sich in einem beschränkten Umfang unterscheiden. Für 32 Bit Systeme ist die Variation der Adresse eingeschränkt. Bei 64 Bit Systemen ist die Variation entsprechend größer und bietet daher in Kombination mit ASLR weitreichende Sicherheit. Zu beachten ist jedoch, dass die 64 Bit CPU im 64 Bit Modus laufen muss. D.h das Betriebssystem muss für die jeweilige 64 Bit Architektur ausgelegt sein.

Diese Beschränkung hierzu geht sowohl aus der Datenwortbreite der CPU, als auch aus der Adressierung mittels Paging hervor (Teile der Datenwortes werden für verschiedene Lookup Tables genutzt).

3.1 Bedeutung der Wirkungen

Zunächst ist festzustellen, dass es Möglichkeiten gibt, Angriffe mehrmals durchzuführen, wenn das entsprechende Programm immer wieder neu gestartet wird. D.h ist ein kurzfristiger Erfolg eines Angriffes zeitlich davon abhängig, wie schnell sich die nächste Angriffsmöglichkeit bietet.

Ein erfolgreicher Angriff in einem kurzen Zeitraum ist nicht abwegig, da viele Serverprogramme bei einem Absturz neu gestartet werden. Sei es ein Webserver, DNS Server uvm. Insbesondere, wenn eine hohe Verfügbarkeit von den Serverdiensten gewünscht wird.

Auf dieser Grundlage, dass auf 32 Bit Systemen die Randomisierung eingeschränkt ist, ist ein Bruteforce Angriff mit Überschreiben des Instruction Pointers mit verschiedenen Adresswerten möglich. Da der Instruction Pointer in der Regel auf eingeschleusten Schadcode zeigen soll, wird die Wahrscheinlichkeit des Erfolges erhöht, liegt ein großer Puffer vor, in dem der Schadcode eingeschleust wurde, so dass der Instruction Pointer in einen Bereich des üblichen NOP Schlittens zeigt.

D.h liegen folgende Konditionen vor:

- 32 Bit System
  - 32 Bit OS auf 64 Bit CPU ohne 64 Bit Extensions, wie PAE.
- großer Puffer
- Programm wird wieder neu gestartet

ist ein Angriff trotz ASLR erfolgreich möglich.

3.2 Was ist mit NX, non executable Stack oder DEP?

Wird DEP, NX eingesetzt, ist es nicht mehr möglich den Stack direkt für das Ausführen von Schadcode zu nutzen.

Jedoch ist es immer noch möglich anderweitig Sicherheitslücken zu nutzen. Da der Codesegment, Datensegment, BSS weiterhin statisch sind, lassen sich deren Speicher bzw. Funktionen für Angriffe nutzen.

4 Resultat

ASLR bietet erhöhten Schutz gegen Angriffe durch Stack Korrumpierung, kann diese jedoch nicht vollumfänglich verhindern und kann daher nicht als Lösung für Fehler in Software herhalten. Gerade wegen ASLR besteht das Risiko, dass Nutzer möglicherweise Sicherheitslücken nicht beheben, da ASLR als Lösung angesehen wird. ASLR in Kombination mit anderen Maßnahmen minimiert das Risiko ebenfalls, könnte jedoch in bestimmten Fällen selbst zu Sicherheitslücken führen.

5 Sonstiges

ASLR lässt sich über das `/proc` System aktivieren/deaktivieren: `/proc/sys/kernel/randomize/va/space`

Optionen sind:

1 ASLR aus

2 Randomisiere Stack, DSO & VDSO, mmap

3 Randomisiere alles (mit Heap)

Von der Deaktivierung ist abzuraten, andernfalls sind die Chancen eines Angriffs auf eine Sicherheitslücke praktisch 100%, es sei denn, es werden noch andere Maßnahmen getroffen. Es gibt einige Admins, die ASLR deaktivieren, aus dem Grund, dass ASLR einen minimalen Performanceverlust darstellen solle, jedoch wird die Randomisation lediglich am Anfang der Ausführung eines Programms durchgeführt. So liegt praktisch kein Performanceverlust vor.

Überprüfen lässt sich, ob ASLR aktiviert ist, indem man z.B den Base Pointer ausliest, des jeweiligen gleichen Programs mehrmals ausgeführt. Dass auch die Adressen der libc Funktionen randomisiert werden, ist schön per ldd zu erkennen.

Durch mehrmaliges aufrufen der Speicher-Map, lässt sich veranschaulichen, wie ASLR randomisiert. z.B über `/proc/self/maps` oder über die jeweilige PID. Hier ist auch zu erkennen, welche Speicherbereiche randomisiert werden.

Hat ein Angreifer Zugriff auf Informationen über den Speicher eines Prozesses, kann dieser anhand der Informationen einen Angriff planen, um erfolgreich zu sein.