Address Space Layout Randomization

Radu-Constantin Onuțu
University of Bucharest, Romania

Abstract

Address Space Layout Randomization (ASLR) is a security technique that aims to mitigate memory corruption attacks by randomizing the locations of key memory regions such as the stack, heap, and shared libraries. However, as demonstrated by Shacham *et al.* [4], the limited entropy available on 32-bit architectures (often restricted to only 16 bits) renders ASLR susceptible to brute-force attacks. In their work, the authors present a derandomization attack that leverages return-to-libc techniques to bypass ASLR in a matter of minutes, particularly when targeting services like the Apache web server on PaX ASLR Linux systems. This survey revisits their methodology and results, discussing the limitations of low-entropy ASLR and exploring potential enhancements such as migration to 64-bit architectures, frequent re-randomization, and compile-time function reordering. This survey highlights the critical role of combining ASLR with additional protections such as stack canaries, W$\bigoplus$X memory pages, and improved randomization techniques to create a more resilient defense against modern exploit techniques.

1 Introduction

Address Space Layout Randomization (ASLR), first widely deployed in OpenBSD 3.4 in 2003 [3], emerged as a countermeasure against memory corruption exploits (such as buffer overflows [1]) by randomizing the base addresses of memory regions (stack, heap, libraries) to thwart predictable attacks. The theoretical foundation of ASLR is that attackers must either guess these randomized addresses or leak address information through side channels.

However, Shacham *et al.* [4] challenged ASLR's efficacy, particularly on 32-bit systems, by demonstrating a brute-force derandomization attack that bypassed PaX ASLR within minutes. Their work exposed two critical weaknesses: (1) the limited entropy (16 bits) allows systematic guessing of library bases, and (2) defensive mechanisms intended to counteract brute-forcing can introduce denial-of-service (DoS) risks.

This survey provides a comprehensive analysis of Shacham *et al.*'s attack methodology, the implications for modern systems, and the interplay between ASLR and buffer overflow vulnerabilities. We also examine enhanced ASLR implementations—including increased entropy in 64-bit architectures, runtime re-randomization, and integration with control-flow integrity mechanisms—while advocating for a multi-layered approach to security.

2 Technical Description of the Problem

2.1 Entropy Constraints in 32-bit ASLR

On 32-bit architectures, PaX ASLR typically allocates 16 bits of randomness to library base addresses. This limitation arises from the need to balance sufficient randomization with practical constraints such as contiguous memory allocation for libraries and avoiding fragmentation. With only $2^{16} = 65.536$ possible addresses, an attacker can feasibly brute-force the correct base address in a matter of minutes.

2.2 Buffer Overflow Vulnerabilities

Buffer overflow vulnerabilities occur when a program writes more data to a memory buffer than it can hold, thereby overwriting adjacent memory regions. This classic vulnerability, often enables an attacker to overwrite control data such as return addresses or function pointers. In systems where ASLR is employed, the attacker's task becomes more complex because the memory layout is randomized. However, as Shacham *et al.* demonstrate, the low entropy inherent in many 32-bit ASLR implementations substantially lowers the barrier for a successful attack, allowing adversaries to reliably predict the locations needed for a return-to-libc attack even when exploiting a buffer overflow.

2.3 Return-to-libc Attack Mechanics

The return-to-libc attack reuses legitimate library functions to perform malicious actions. For example, an attacker who overwrites a return address with the address of `system()` can trigger the execution of arbitrary commands once ASLR is bypassed. Shacham *et al.* detail an attack comprising two main phases:

1. **Brute-force Derandomization:** The attacker systematically guesses the base address of `libc` by exploiting memory disclosures or through iterative probing.

2. **Payload Execution:** Once the `libc` base is determined, the attacker calculates function offsets (e.g., for `system()`) and hijacks the control flow to execute a payload.

2.4 32-bit Compatibility Mode Vulnerabilities

Even on 64-bit systems, 32-bit applications often inherit the reduced entropy of their architecture due to inherent address space limitations. Consequently, legacy software running in compatibility mode remains vulnerable to the same brute-force attacks unless specifically hardened.

3 Possible solution

The study by Shacham *et al.* clearly illustrates that the low entropy in 32-bit ASLR implementations is a significant weakness. To address these issues, several mitigation strategies have been proposed:

1. **Migration to 64-bit Architectures:** The larger address space in 64-bit systems inherently provides higher entropy, significantly reducing the feasibility of brute-force attacks.

2. **Frequent Re-randomization:** Periodically re-randomizing memory layouts during a process's lifetime can further complicate an attacker's efforts to reliably target a fixed address.

3. **Compile-Time Function Reordering:** Randomizing the layout of functions within binaries can reduce the predictability of code locations, thus complementing ASLR.

4. **Layered Security Mechanisms:** ASLR should be combined with other defenses, such as stack canaries (to detect buffer overflows) and W$\bigoplus$X policies (to prevent execution of data), to provide a multi-layered defense against exploitation.

Each of these strategies comes with its own trade-offs in terms of performance and complexity; however, when combined, they contribute to a significantly more resilient security architecture.

4 Experiments and Results

To evaluate the effectiveness of ASLR on PaX ASLR-protected Linux, Shacham *et al.* conducted an attack targeting an Apache web server. The objective was to bypass ASLR using a brute-force approach and ultimately gain a remote shell on the system.

The attack comprised three primary steps:

1. **Guessing the Address:** The attacker repeatedly guessed the address of a `libc` function (e.g., `usleep` or `system`) by leveraging information leakage and iterative probing.

2. **Exploiting a Buffer Overflow:** By exploiting a buffer overflow vulnerability, the attacker overwrote the return address to redirect control flow.

3. **Executing the Payload:** After successfully determining the correct memory address, the attacker executed arbitrary shell commands using a return-to-libc attack.

Due to the limited 16-bit entropy in PaX ASLR, the attack was completed within a feasible timeframe. The recorded results were as follows:

Average	**Min**	**Max**
216 s	29 s	810 s

Table 1: Total time of a brute-force attack

Additionally, the attack was both low-bandwidth and stealthy:

Average	**Max**
6.4 MB	12.8 MB

Table 2: Traffic generated by the brute-force attack

These experimental results underscore that, on 32-bit architectures, ASLR does not provide robust protection against brute-force attacks. The small randomization space can be systematically searched within minutes, and when combined with buffer overflow exploits, the defense mechanism is significantly undermined.

5 Limitations of monitoring systems

A proposed enhancement to ASLR, advocated by PaX developers, involves coupling randomization with a crash detection and reaction mechanism (or "watcher") to thwart brute-force attacks. This system would monitor segmentation faults (e.g., from incorrect address guesses) and trigger mitigations such as process termination or administrator alerts. However, Shacham et al. demonstrate fundamental limitations in this approach, highlighting tensions between security and availability.

The efficacy of a watcher hinges on its response strategy:

- **Administrator Alerts**: Human intervention is impractical given the attack's speed—Shacham's derandomization succeeds in ≤ 216 seconds (on average), far faster than diagnostic or remediation workflows.
- **Automated Process Termination**: While halting a compromised daemon prevents exploitation, it also enables trivial denial-of-service (DoS) attacks. For instance, inducing repeated crashes in Apache could force persistent downtime, incurring significant financial losses.

Method	Dictionary
Brokerage operations	\$6,450,000
Credit card authorization	\$2,600,000
Ebay	\$225,000
Amazon.com	\$180,000
Package shipping services	\$150,000
Home shopping channel	\$113,000
Catalog sales center	\$90,000
Airline reservation center	\$89,000
Cellular service activation	\$41,000
On-line network fees	\$25,000
ATM service fees	\$14,000

Table 3: Total cost of one hour of downtime [2]

In summary, while the concept of a crash watcher offers an additional layer of defense, its practical limitations render it insufficient to counteract ASLR bypass attacks effectively. Instead of preventing exploitation, an overly aggressive watcher may inadvertently increase the system's susceptibility to denial-of-service attacks. This reinforces the argument for employing a comprehensive, multi-layered security strategy that does not rely solely on reactive measures like crash detection.

6 Conclusion

The findings presented in Shacham et al.'s paper demonstrate that ASLR is not a strong standalone defense mechanism, particularly on 32-bit architectures. Due to limited entropy (16-bit randomness), an attacker can systematically brute-force memory addresses in a matter of minutes or even seconds, effectively bypassing ASLR and exploiting buffer overflow vulnerabilities. This highlights a fundamental weakness in ASLR's implementation on 32-bit systems, as it merely delays exploitation rather than preventing it entirely.

Since brute-force attacks remain feasible, ASLR alone does not provide sufficient protection against memory corruption exploits. A more effective long-term solution is to migrate to 64-bit architectures, where higher entropy significantly increases the difficulty of address-guessing attacks. Additionally, ASLR should be combined with other security mechanisms, such as stack canaries, W$\bigoplus$X (Write XOR Execute) protections, and compile-time function randomization, to create a more resilient defense against modern exploitation techniques.

In conclusion, while ASLR is a useful mitigation technique, it should not be relied upon as the sole protection against memory corruption attacks. Strengthening security measures and adopting architectures with greater entropy is crucial for defending against sophisticated attack methods in modern systems.

References

[1] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 49(14), nov 1996. URL: `http://www.phrack.org/phrack/49/P49-14`.

[2] David A Patterson et al. A simple way to estimate the cost of downtime. In *LISA*, volume 2, pages 185–188, 2002.

[3] The OpenBSD project. OpenBSD Innovations, 2003. Last accessed 14 February 2025. URL: `https://www.openbsd.org/innovations.html`.

[4] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307, 2004.