The Design and Implementation of Address Space Layout Randomization (ASLR) in the Windows Research Kernel (WRK)

Student: MAK Chung Leung

Supervisor: Prof. Lin GU

Introduction

Address Space Layout Randomization (ASLR), originally released by the PaX team in Linux, is a security countermeasure which randomizes program memory layouts so that the base addresses of different modules become non-fixed in virtual memory every time programs are run. In this way, crucial addresses cannot be located easily to induce the memory corruption which is the first step in hacking. See Figure 2.

Figure 1: PaX TuX the Mascot of ASLR

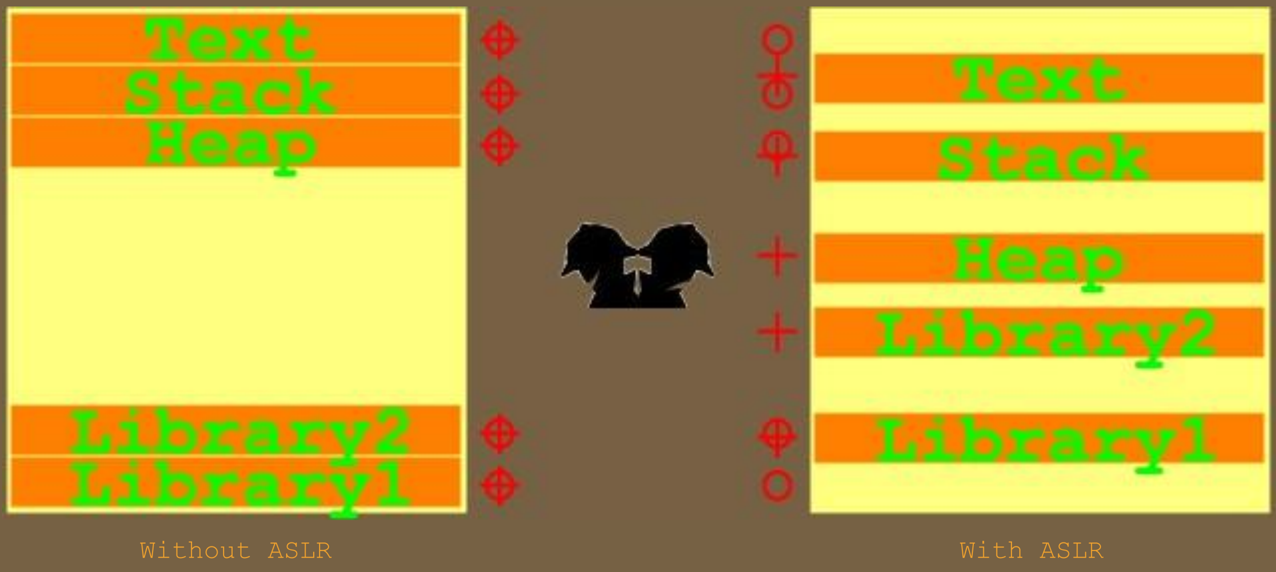

Figure 2: Virtual Memory Layouts

In this project, an ASLR was designed and implemented in a pre-Vista Windows kernel. Its effectiveness and possible improvements were studied.

Objective

1. To implement ASLR in the WRK of an MS Windows OS before Vista
2. To evaluate and analyze the efficiency of the ASLR implementation
3. To improve the implementation with different methods
4. To prove or disprove the statement:
   - ASLR in 32-bit architectures is ineffective

Methodology

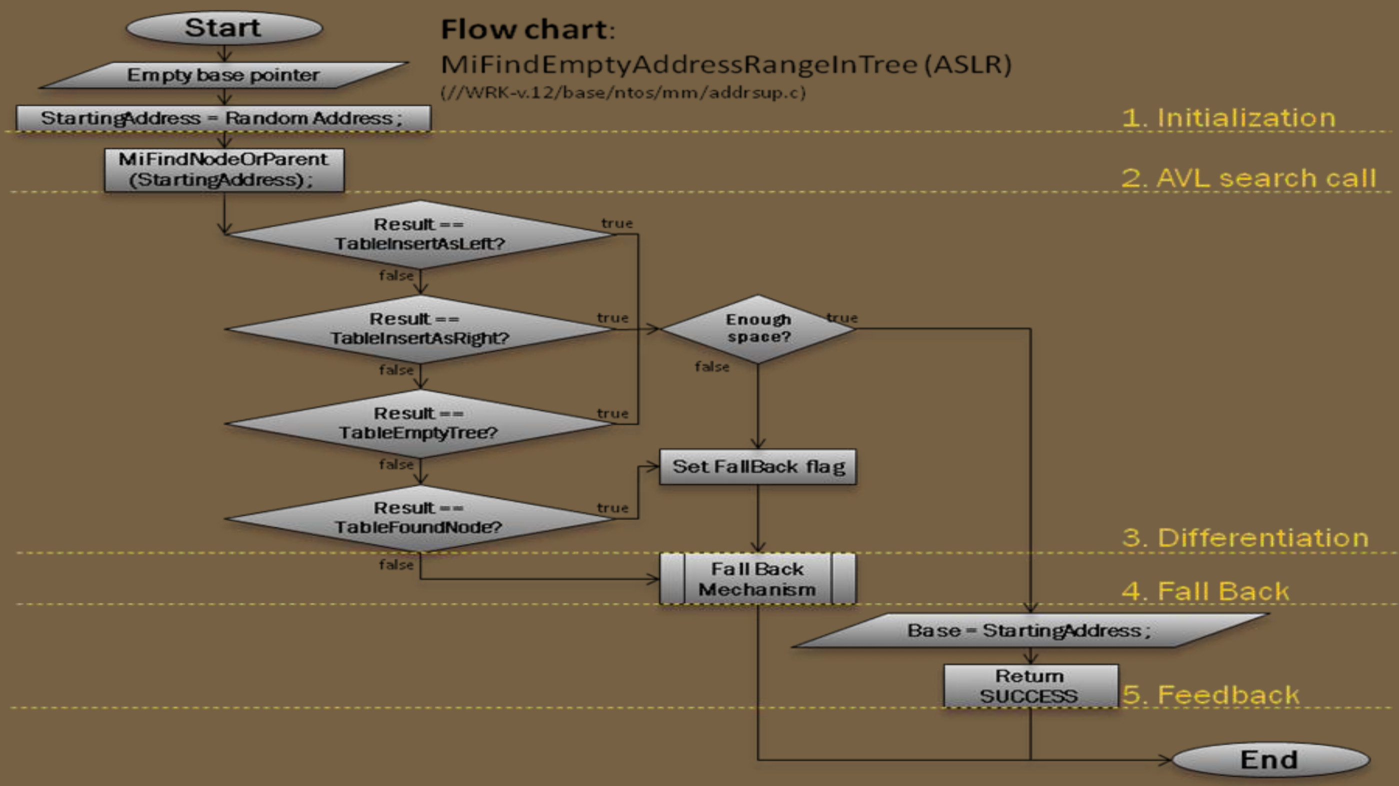
Flow chart:
MiFindEmptyAddressRangeInTree (ASLR)
(//WRK-v.12/base/ntos/mm/addrsup.c)
Start
Empty base pointer
StartingAddress = Random Address;
MiFindNodeOrParent (StartingAddress);
Result == TableInsertAsLeft?
Result == TableInsertAsRight?
Result == TableEmptyTree?
Result == TableFoundNode?
true
false
Enough space?
Set FallBack flag
Fall Back Mechanism
Base = StartingAddress;
Return SUCCESS
End
1. Initialization
2. AVL search call
3. Differentiation
4. Fall Back
5. Feedback

Figure 3: the Design of ASLR Architecture

Pseudo Random Number Generator (PRNG)

To generate random addresses, a PRNG is needed. Random numbers generated should distribute evenly. Otherwise, some addresses occur with higher probabilities, and can be used attacks to increase success rates. A high quality PRNG 'Mersenne Twister (MT)' was used. It is fast but cryptographically insecure.

Memory Allocation Mechanism

In Windows, Virtual Address Describer (VAD) tree, which is just an AVL tree, is used to search for memory space. Originally, the search starts from the lowest address. ASLR can be deployed through replacing the lowest address with a random address.

Fall Back

If the random address generated is occupied, the memory allocation mechanism enters the Fall Back stage where empty memory space is searched upward; when the search reaches the highest address, it rolls back to the lowest address to continue the search upward; when the starting point is encountered again, the whole virtual address space is completely covered.

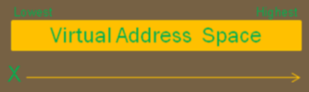

Figure 4: Original Search

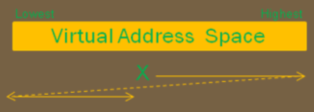

Figure 5: ASLR Search

Result

Complexity

The ASLR complexity
= O (AVL search for the random address
  + Fall Back)
= O((log N) + N)
= O(N)
where N is the number of VAD nodes

The complexity is the same as the original architecture.

Performance

The performance was measured by the time needed to run 10k merge-sorts. Without ASLR, it spent 27 minutes; with ASLR, it spent 37 minutes, which was 37% more.
The architecture is not the reason for the decrease in performance. The reason is that a faster memory allocation mechanism called 'VAD bitmap' was not modified to adapt to the ASLR.

Address Distribution

10k address samples were collected. Using MT, there was a uniform address distribution.

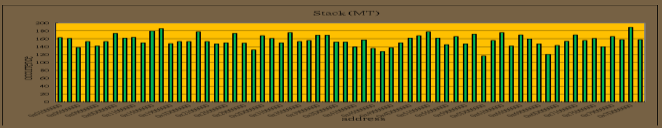

Figure 6: Address Distribution

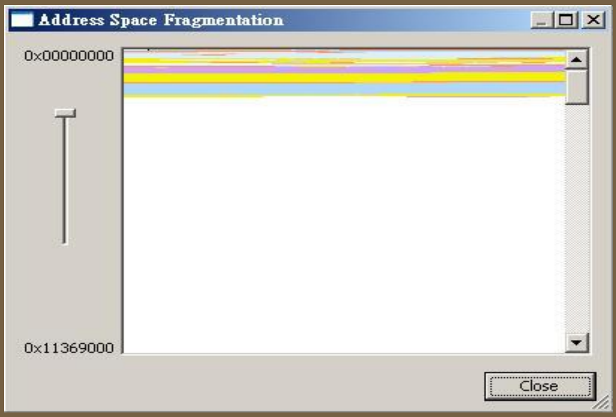

Figure 7: Memory Layout (Original)

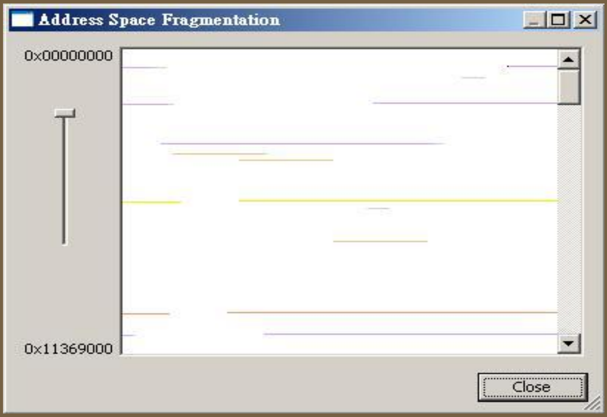

Figure 8: Memory Layout (ASLR)

Conclusion

In conclusion, an ASLR was successfully designed and implemented in the WRK. Its efficiency was evaluated. Also, the random address quality was improved by 'MT'. Other possible improvements were proposed without implementations. However, they may not be able to bring substantial improvements to the ASLR in 32-bit architecture. Therefore, the statement 'ASLR in 32-bit architectures is ineffective' is still correct when ASLR protects server systems against massive network attacks.