

# On the Effectiveness of Address-Space Randomization

## Summary:

The paper discusses the strengths and weaknesses of the Address Randomization techniques in systems having operating systems like Linux and OpenBSD having a 32-bit architecture which has some limitations due to less number of bits for address randomization.

The author in paper discusses weaknesses of ASLR and shows with an example of a return-to-libc attack that how the ASLR can be disrupted. PaX ASLR used in the Linux and OpenBSD operations is used while performing the experiments in this paper.

The attack used in the experiment of this paper is not the usual return-to-libc attacked in fact it is a chained return-to-libc attack. Traditional Return-to-libc attack which has both the information related to stack and text segment of libc is not used to exploit the address space in the system. But in the technique used in this paper uses only address information placed by the target program on the stack. A simple experiment to show that a derandomized attack takes a long time such as 216 seconds to exploit the Linux PaX ASLR system. The author also explains the related work done on write or execute pages a security feature to counterattack the various attack like return-to-libc, as under the “write or execute” the pages in various types of memory segment is writable or executable pages but not both. But due to this patch, there are high-performance penalties.

The author then also discusses the improvement that can be made in the ASLR to fortify the chances of preventing the program from various attacks. The authors suggest improvements such as by upgrading the system architecture to 64-bit s it gives more address space( for example 40 bits mentioned in this paper) for randomization on 64-bit architecture rather than a 32-bit architecture which gets only 16 bits. To increasing randomization frequency of the address space(randomisation need to be done for every attack on the address space) with each attack as it will further slow the process of attack being launched on the address space. The author also suggests finer granularity randomization, both at compile-time and runtime. At compile-time, the finer granularity can be achieved by randomizing the variables and functions by doing simple modifications to compiler and linker or by introducing random padding into stack frames. At runtime, the finer granularity can be achieved by randomizing more than 16 bits of address space, reordering functions. The fourth way which the author discusses to improve the address space randomization is to use a watcher which is a system service that monitors the actions of attacks and shut them down after some threshold. At last, the author suggested using overflow mitigation systems such as StackGuard, Propolice and PointGuard which are independent of the Address space layout randomization and which can disrupt many exploits including the buffer overflow is another technique discussed by the author which can be used with ASLR to defeat many attacks.

---

# Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks

## Summary:

In this paper, the author explains how a binary code injection attack can be disrupted by a prototype based on Valgrind x86-to x86 binary translator. The author discusses the implementation and limitation of RISE and discusses how the RISE disrupts the various types of attacks. The RISE is an obfuscation technique used to protect the program by scrambling the binary code by updating instruction with a constant unique seed for each program execution. Without the knowledge of the key, the attack code will be processed and when it gets descramble the code will crash.

When using RISE as a binary-to binary translator, to avoid the execution of malicious code injected in the form of data in the currently executing code adoption of a method such as code shepherding is used. There are two security policies which come under the method of code shepherding such as code origin policies and restricted control transfers. These policies enforce security by optimizing interpreters.

One advantage of RISE over avoiding the buffer overflows that is not necessary to apply it on the whole system like PaX. But it can be applied to the selective processes running in the system. The author of the paper considers 14 types of attacks and verifies the impact on Valgrind with RISE and Valgrind without RISE. Valgrind with RISE avoids number of attacks including the Synthetic heap, stack overflow, Bind NXT and SAMBA trans2 attacks.

In terms of performance, Valgrind with RISE performs 5% slower than the Valgrind without RISE, as there is more instruction in the former than the latter.

---

# Building Diverse Computer Systems

## Summary:

The authors raise the point on how diversity is overlooked in the mass market of computer systems and compares it to the dominant species of society. The authors try to convince as in our society how diversity is important such importance of diversity is also needed in computer systems. The author discusses various methods for achieving diversity based on randomizations with a focus on computer security.

### **Few of the randomizations methods mentioned in paper by the author:**

1. Adding or deleting nonfunctional code  
Insert or delete a no-ops instructions at random locations in the compiled code which. This technique has the ability to disrupt the Kocher's timing attack on RSA.
2.
  - a. Reordering code-  
Basic blocks can be reordered in a random way and stored at different locations but the order of execution is not affected. But this technique does not provide security against a large class of viruses.
  - b. Optimizations for parallel processing:  
Code is divided into blocks of instructions and executed on multiple processors simultaneously. We need to select the code which is intended to be run on a single processor. But the amount of randomness that can be produced with the method is limited to the no of processors being used.
  - c. Instruction scheduling:  
Padding on stack frame by a random amount  
Randomize the locations of global variables and also offset assigned to local variables.
3. Memory layout:
  - 1) Padding each stack frame with random gaps in memory.
  - 2) Randomizing the global and local variables
  - 3) Assign the memory to the stack frame in randomized manner not in a contiguous manner.

Some Other Transformations that can be done for Randomizations

Process Initialization

Dynamic libraries and System calls

Unique name of system files

The magic number in certain files

Randomized Runtime checks

---

# Preventing overflow attacks by memory randomization

## Summary:

The author in the paper talks about two approaches for preventing overflow attacks by Memory Randomization. Both the approaches are implemented at the user's end.

1. Fine-grained stack randomization to avoid stack smashing overflow attack: Done by randomizing few of binary instructions of the stack.
2. Fine-grained heap randomization

The Stack randomization is not done at the time of compilation rather it done at the time of installation of the program at the user end and it done at various levels. Below are the randomization techniques used

Addition of dummy variables during a sequence of program instruction compilation

Padding the stack frame.

Randomizing code at the level of assembly instruction level with help of assembler

But the difference between the approach in the recent research randomization and the one which they have implemented in this papers is the approach taken towards the randomizations of the stack frame. Fine-grained stack randomization techniques used in this paper, which was performed on various binaries where the routines are randomized independently of other routines in the binary

Advantages of this technique that it is low cost and provides an easy way out for the applying randomization on binaries instead of large scale applications such as linker/compiler/loader.

A technique for fine-grained heap randomization is also presented by the author in this paper. The library patching technique used in this solution adds random pads above and below every chunk of memory allocated off the heap. The values for these random pads are generated at run-time due to which each instantiation of a program allocates.

---

# Online videos on WebAssembly(Just for reference)

---

## **Video - WebAssembly: Binary in Plain English by Milica Mihajlija at DACHfest 2018**

Questions Answered:

What it is?

Js is not ideal for all task so we need something new that is WebAssembly to increase the performance.

WASM is used to increase the performance on the web.

WASM is in Binary format so fast to load and fast to execute

Another high-level language is compiled to web assembly. Such as C++,C and Rust

Web assembly is not assembly language as it is not for any specific system but for the browsers.

Not a replacement to js but used with js to increase the performance of the online applications.

Why do we need it?

For Performance

Online Game need a high degree of performance

The more rich application came to the web which cannot be handled by the js so we need something more that is WebAssembly.

How you can we benefit from it?

WebAssembly brings speed(small binaries), portability() and flexibility

Speed

Js - parse, compile and optimize the code as it is being executed on-page.

WebAssembly- delivered as binary so the decoding happens much faster.

WASM Statically typed so no overhead of which types are going to be used

No garbage collection in WASM

20% slower than native code

Portability-

Whenever a code is running on the browser it has to be compatible with device processor architecture and the operating system. WASM needs only compilation step and the source code will run in every browser

Flexibility-

Lot more developer can code on the Web.

How does it work:

LLVM compiler outputs Web Assembly

Emerscripten- based on LLVM for compiling C and C++ to WASM

Rust has its own compiler called rustc to convert the source code from Rust to WASM.

WebAssembly does not access with DOM, WebGL, Web audio other APIs so to work with this we need the emerscripten and rustc will give output in form of WASM+js+html. So that these apis can be accessed from the js file.

Web Assembly functions can be called from JS code.

Buffer Overflow Attack:

A buffer is a place where the data is stored informs of the blocks.

When the buffer is overflown with data, the overflown data can move into another buffer.

Moreover this

There are two types of buffer overflow attacks:

- 1) Stack overflow attack
- 2) Heap overflow attack