A Brief Survey of ASLR

Address Space Layout Randomization (ASLR) 是神马?

- 一些攻击，比如 return-oriented programming (ROP)之类的代码复用攻击，会试图得到被攻击者的内存布局信息。这样就可以知道代码或者数据放在哪里，来定位并进行攻击。比如可以找到 ROP 里面的 gadget。而 ASLR 让这些内存区域随机分布，来提高攻击者成功难度，让他们只能通过猜猜猜来进行不断试错的攻击(理想状况下)。图 1 举了个例子。

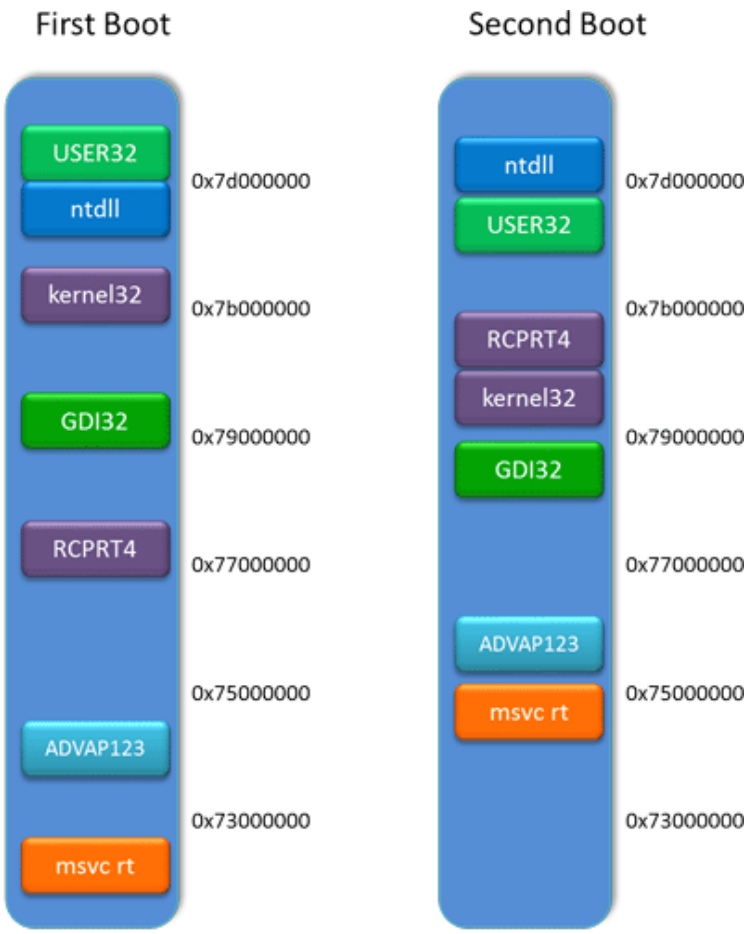

图 1: ASLR 示例

(from https://insanitybit.files.wordpress.com/2012/06/aslr.png)

ASLR 有啥问题?

- 在出现了某些漏洞，比如内存信息泄露的情况下，攻击者会得到部分内存信息，比如某些代码指针。传统的 ASLR 只随机化整个 segment，比如栈，堆，或者代码区。这时候攻击者可以通过泄露的地址信息来推导别的信息，比如另外一个函数的地址，等等。这样整个 segment 的地址都可以推导出来，进而得到更多信息(见图 2)，大大增加了攻击利用的成功率。而且，在 32 位系统中，由于随机的熵值不高，攻击者也容易通过穷举猜出地址。

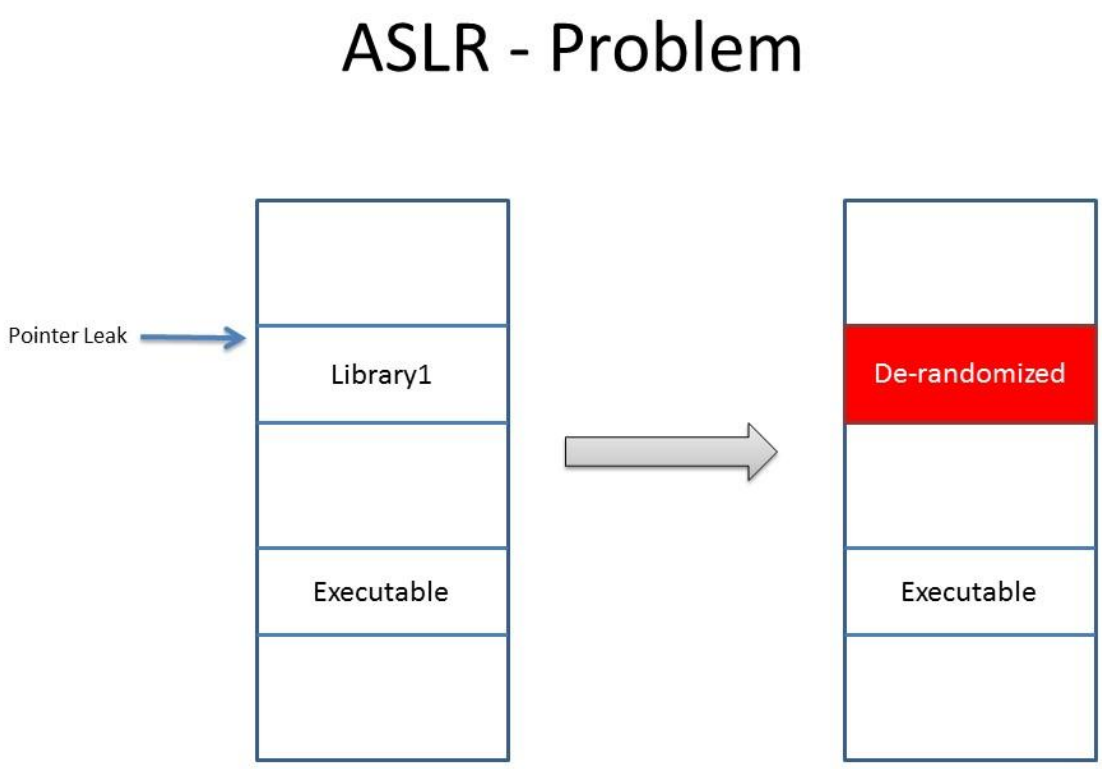

图 2: 目前 ASLR 如何不靠谱

如何改进?

- 主要的改进方法，一方面是防止内存信息泄露，另一方面是增强 ASLR 本身。本文主要讨论后者。

1. 随机化的粒度可以改进。粒度小了，熵值增加，就很难猜出 ROP gadget 之类的内存块在哪里。[1](ASLP)在函数级进行随机化，[2](binary stirring)在 basic block 级进行随机化，[3](ILR)和[4](IPR)在指令级。[3]将指令地址进行随机化；而[4]把指令串进行重写，来替换成同样长度，并且相同语义的指令串。

2. 随机化的方式可以改进。[5](Oxymoron)解决了库函数随机化的重复问题: 原先假如每个进程的 library 都进行 fine-grained 的 ASLR，会导致 memory 开销很大。该文用了 x86 的 segmentation 巧妙地解决了这个问题；并且由于其分段特性，[6](JIT-ROP)之类的攻击也很难有效读取足够多的 memory。[7](Isomeron)利用两份 differently structured but semantically identical 的程序 copy，在 ret 的时候来随机化 execution path，随机决定跳到哪个程序 copy，有极大的概率可以让 JIT-ROP 攻击无效。

3. 随机化的时间(timing)可以改进。假如程序中存在能泄露内存的漏洞，那这种传统的、一次性的随机化就白费了。所以需要运行时动态 ASLR。[8]解决了 fork 出来的子进程内存布局和父进程一样的缺陷。其思路是在每次 fork 的时候都进行一次随机化。方法是用 Pin 进行 taint 跟踪，找到 ASLR 之后要修复的指针并进行修复。为了降低把数据当成指针的 false positive，一个 daemon 进程会跑多次来提取出重合的部分。

   [9](Remix)提出了一种在运行时细粒度随机化的方法。该方法以 basic block 为单位，经过一个随机的时间对进程(或 kernel module)本身进行一次随机化(图 3)。由于函数指针很难完全确认(比如被转换成数据，或者是 union 类型)，该方法只打乱函数内部的 basic blocks。该方法另一个好处是保留了代码块的局部性(locality)，因为被打乱的 basic blocks

位置都很靠近。打乱后，需要 update 指令，以及指向 basic block 的指针，来让程序继续正确运行。假如需要增加更多的熵值，可以在 basic blocks 之间插入更多的 NOP 指令(或者别的 garbage data)。

另一种方法[10]是用编译器来帮助定位要 migrate 的内存位置(指针)，并且在每次有输出的时候进行动态随机化。该方法对于网络应用比如服务器，由于其是 I/O-intensive 的应用，可能会导致随机化间隔极短而性能开销巨大。

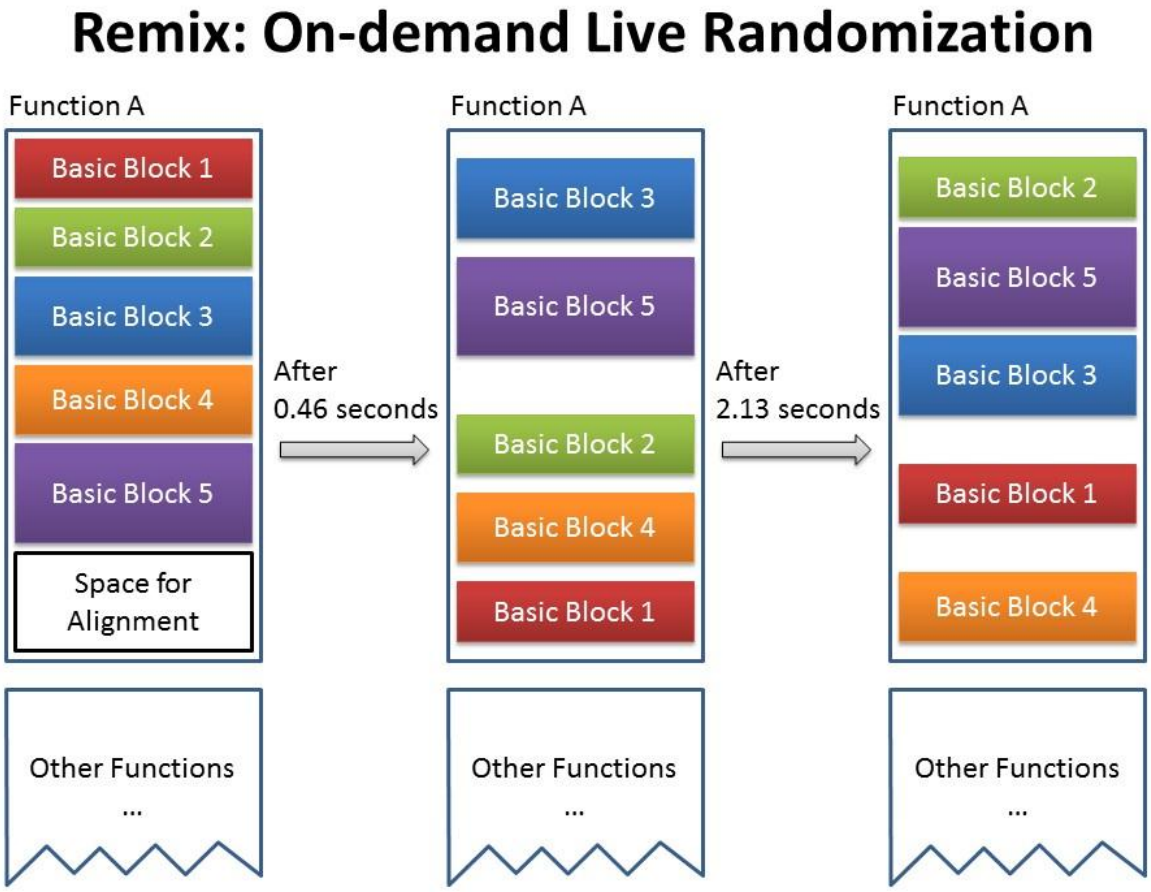

图 3: 似乎是更靠谱的 ASLR

(摘自 http://ww2.cs.fsu.edu/~ychen/paper/Remix_slides.pdf)

作者简介:

Yue CHEN, Florida State University 博士在读。个人主页是 http://yuechen.me
欢迎交流。似乎也没什么别的可以写了。

参考文献:

[1] C. Kil, J. Jun, C. Bookholt, J. Xu, and P. Ning. Address Space Layout Permutation (ASLP): Towards Fine-Grained Randomization of Commodity Software. In Proceedings of the 22nd Annual Computer Security Applications Conference, 2006.

[2] R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin. Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code. In Proceedings of the 19th ACM Conference on Computer and Communications Security, 2012.

[3] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson. ILR: Where'd My Gadgets Go? In Proceedings of the 33rd IEEE Symposium on Security and Privacy, 2012.

[4] V. Pappas, M. Polychronakis, and A. D. Keromytis. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. In Proceedings of the 33rd IEEE Symposium on Security and Privacy, 2012.

[5] M. Backes and S. Nurnberger. Oxymoron: Making fine-grained memory randomization practical by allowing code sharing. In Proceedings of the 23rd USENIX Security Symposium, 2014.

[6] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi. Just-in-time Code Reuse: On the Effectiveness of Fine-grained Address Space Layout Randomization. In Proceedings of the 34th IEEE Symposium on Security and Privacy, 2013.

[7] L. Davi, C. Liebchen, A.-R. Sadeghi, K. Z. Snow, and F. Monrose. Isomeron: Code randomization resilient to (just-in-time) return-oriented programming. In Proceedings of the 22nd Network and Distributed Systems Security Symposium, 2015.

[8] K. Lu, S. Nurnberger, M. Backes and W. Lee. How to Make ASLR Win the Clone Wars: Runtime Re-Randomization. In Proceedings of the 23rd Network and Distributed Systems Security Symposium, 2016.

[9] Y. Chen, Z. Wang, D. Whalley and L. Lu. Remix: On-demand live randomization. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, 2016.

[10] D. Bigelow, T. Hobson, R. Rudd, W. Streilein, and H. Okhravi. Timely rerandomization for mitigating memory disclosures. In ACM SIGSAC Conference on Computer and Communications Security, 2015.