How to Make ASLR Win the Clone Wars: Runtime Re-Randomization

MAR 15TH, 2016

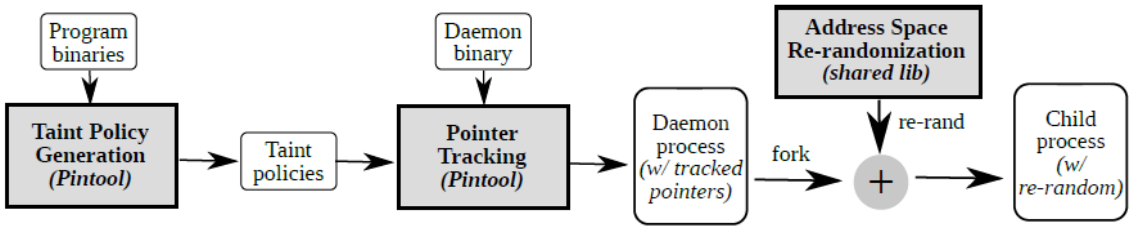
论文下载

Background

在Apache，Nignx和OpenSSH这类的服务程序里，都会用fork()系统调用来创建子进程。出于效率，还有内存资源共享等因素的考虑，在fork()调用之后不会继续调用execve()执行一个新的程序镜像。这就导致了，fork()出来的子进程和父进程的内存空间排布是一样。这就破坏了ASLR的保护机制。
这篇文章提出了一个RuntimeASLR的机制，让fork()出来的子进程内存空间地址重新随机化。

Implementation

RuntimeASLR的设计思路是，对Daemon进程用Pin辅助做污点跟踪，找到重新ASLR之后需要修复的指针；然后在fork()出子进程的时候，修复指针，并执行重新ASLR。



污点跟踪

标记污点的数据来源有三类：（1）操作系统初始化的指针，如：rsp；（2）PC指针；（3）特定syscall的返回值，如：mmap，mremap，brk。

作者用Pin来对这些被标记的数据做跟踪，记录下最后存储指针的内存地址的相对位置，以及相对应的opcode，oprand类型，oprand的size，flag bit标志位，以及oprand是不是已经被标记过了。把这些存到一个hashmap里。

为了防止在污点跟踪时记录的是数据，而不是指针。作者对一个Daemon进程跑多次Taint Trace，提取出重合的记录。

重新ASLR

作者写了一个Pin在detach的时候的一个Callback来执行重新ASLR。先调用mremap()来重新ASLR子进程的地址空间，然后根据之前污点跟踪的记录来修复指针，最后detach Pin。

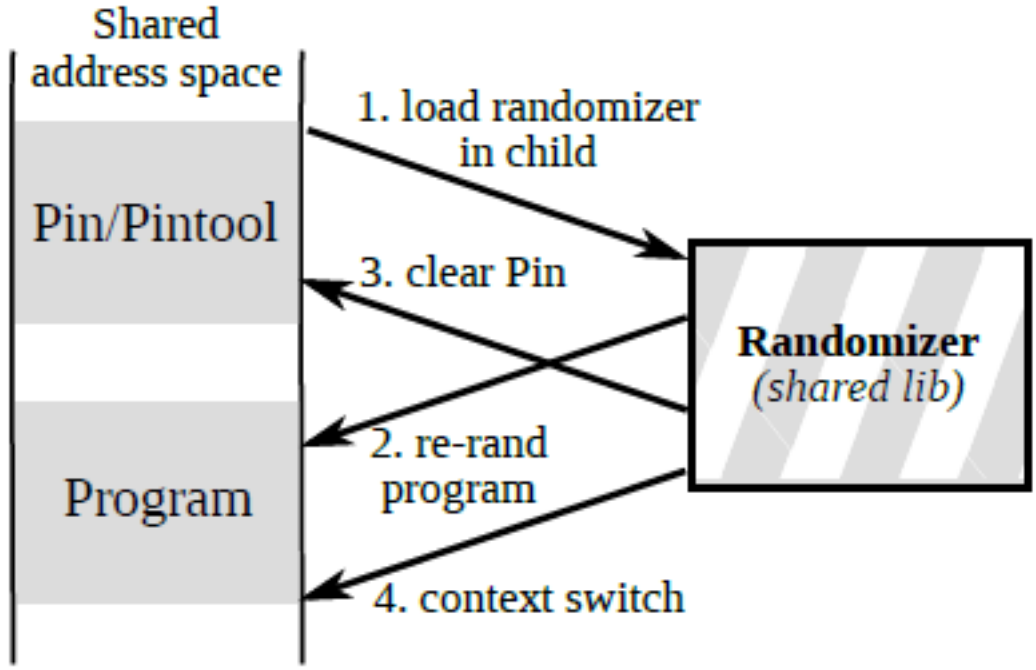
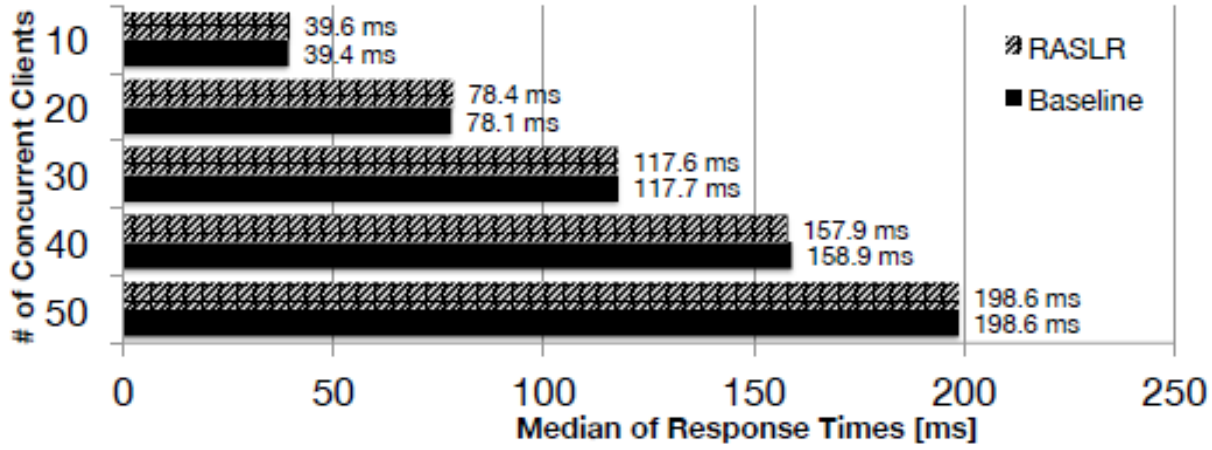


Fig. 4: The workflow of re-randomization.

Evaluation

- 服务程序的相应时间：



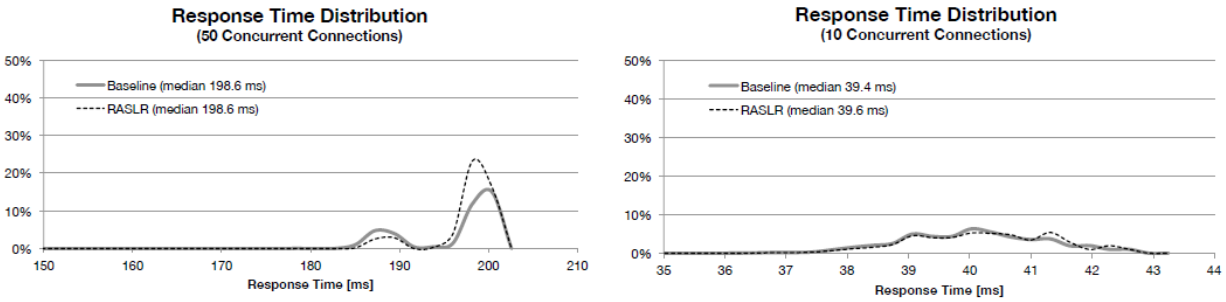
Response Time Distribution

- 实际的Overhead：

Benchmark	Original (seconds)	RUNTIMEASLR (seconds)	Overhead (times)
gcc	1.13	12,783	11,312
mcf	2.64	24,708	9,359
hmmer	2.28	19,004	8,335
libquantum	0.06	1,491	24,850
xalancbmk	0.08	1,932	24,150
soplex	0.02	217	10,850
lbm	2.28	2,468	1,028
sphinx3	1.61	12,661	7,863

TABLE V: Pointer tracking on SPEC CPU2006 benchmarks. Pointer tracking is completely detached in child process, and thus does not affect the performance of child (worker) process.

- 并发的时候的响应时间：



(a) Always 50 concurrent connections

(b) Always 10 concurrent connections

Fig. 9: Nginx response time distribution with/without our pointer tracking `Pin` instrumentation.