PAIRS: Control Flow Protection using Phantom Addressed Instructions
Mohamed Tarek Ibn Ziad, Miguel A. Arroyo, Evgeny Manzhosov, Vasileios P. Kemerlis*, Simha Sethumadhavan
{mtarek,miguels,evgeny,simha}@cs.columbia.edu, *vpk@cs.brown.edu
Columbia University, *Brown University

Many critical programs of the current software stacks like shared libraries, OS kernels, virtual machine monitors and many others are written in memory unsafe languages such as C or C++. Moreover, plenty of environments are based on the legacy code that was not written with memory safety in mind. These facts in conjunction with systems that accept inputs from an unverified origin provide a fertile ground for attacks that use a memory corruption [1,2] as a starting point. For example, the use of buffer overflow to corrupt the return addresses on the stack in code reuse attacks.

Both industry and academia proposed different methods to address the exploitation of memory safety violations via software-only [5] or hardware-assisted solutions [3,4]. On one hand, software-only solutions typically suffer from high performance overheads and increased memory footprint. For example, Address Sanitizer, which is a Memory error detector provided by Google has average slowdowns of 73%. Additionally, it increases memory footprint by 237%, and binary size by 150%, on average. On the other hand, hardware-based solutions typically have much lower performance overheads, but they require the installation of new hardware and recompilation of the existing software. Moreover, current state-of-the-art hardware solutions are very energy inefficient. For instance, CHERI [4] increases the size of pointers at least by a factor of two, hence making all pointer and memory operations more expensive.

As complete memory safety has prohibitive performance and energy overheads, a more realistic approach is to prevent the malicious consequences of memory safety violations rather than detecting the violations themselves. One example for such approaches is ARM Pointer Authentication (PAC) technology. ARM PAC uses the (currently) unused bits of 64b addressing scheme to store authentication codes for memory pointers. Unfortunately, PAC is very expensive for 32b addressing modes, since there are no spare bits to be used for the authentication code.

To mitigate these issues, we propose Phantom Addressed Instructions that are Randomly Selected (PAIRS) [6]. The idea of PAIRS is to randomize the execution of the program between two phantom copies of the program, which forces an attacker to bet on the copy she targets for the code reuse attack. We call the copies phantom copies since they do not exist in the program binary. PAIRS is based on the microarchitectural modifications that allow creating a phantom copy of the executed program in the virtual address space. Thus, every instruction of the original program has two addresses and the attacker must guess what address will be fetched next - the phantom or the original. Since every code reuse attack is based on chaining multiple pieces of the program together, the attacker must guess all the switches correctly, hence the probability of a successful attack decreases exponentially.

To summarize, benefits of PAIRS are: (1) no recompilation of the source code, (2) PAIRS is suitable for both 32b and 64b systems, and (3) no performance and energy overheads. We evaluated PAIRS on a modified gem5 simulator and compared it against state-of-art techniques like PAC and Software-based Execution Path Randomization (SEPR). PAIRS has no overhead compared to the unmodified baseline and outperforms both PAC and SEPR [6]. All of the above makes PAIRS an excellent solution for deployment in legacy, embedded/IoT or server-class systems.

References