Practical Challenges of Type Checking in Control Flow Integrity

Reza Mirzazade farkhani
Northeastern University
Boston, MA
reza699@ccs.neu.edu

Sajjad Arshad
Northeastern University
Boston, MA
arshad@ccs.neu.edu

Saman Jafari
Northeastern University
Boston, MA
jafari1149@ccs.neu.edu

I. Abstract

Lack of memory management in unsafe programming languages such as C/C++ has been introducing significant threats to the applications. As a result, there has been a continuous arms race between the development of attacks and countermeasures. Generally speaking, memory corruption attacks are categorized into two types; code injection and code reuse. The most prevalent and practical defense mechanisms against these attacks are non-executable memory ($W \oplus X$) and Address Space Layout Randomization (ASLR). However it has been shown that such defenses can be bypassed by motivated attackers [1]. Therefore, new protections such as Control Flow Integrity (CFI) have been introduced by researchers [2].

CFI is introduced to enforce the application's control flow to adhere to the statically generated Control Flow Graph (CFG). The effectiveness of CFI depends on the ability to construct an accurate CFG. Creating a CFG needs a precise static analysis and points-to analysis. Precise points-to analysis is an undecidable problem and it leads to over approximation in the CFG of the program. These over approximations let the adversary to perform attacks despite the presence of CFI [3]. Recently, a type matching method has been proposed in order to solve the aforementioned problems in a practical way [4], [5].

Type checking only allows control transfers if the types of the caller and the callee match. Similar to CFI, type matching attempts to enforce the control flow of the program during runtime to stick to the branches recognized in the statically generated CFG, using label-based control-flow approach. In other words, the type of a function pointer that is used to call a function and the type of the calling function are compared at runtime before jumping to the function. The control flow transfer is disallowed if the types do not match. It is very common in real-world applications that different functions and function pointers have the same type. This leads to type collision for type matching.

In this study, we find out that runtime type checking, indeed, faces numerous practical challenges for deployment. For example there are some types such as *void* * that can be matched with any other type. Therefore, restricting based on type is not effective because the caller and callee can have different types. There are also variadic functions and function pointers in which the type of caller is not known beforehand. Deciding based on the number and type of arguments is not possible statically. In C++, virtual methods return types can be covariant which is another challenge for type checking.

In order to prevent type collision, a type diversification technique is required. Resolving collisions requires global type diversification which complicates dynamic loading of libraries and separate compilation. Our preliminary investigation on popular web servers such as Nginx, Apache and Lighttpd shows that there are a substantial number of functions and function pointers that have same type, which produces type collision and consequently over approximation. If the shared libraries are added to this list, the number of collision rises significantly. Diversifying all of these collisions requires having all the source code at same time for preserving the functionality. In addition to being performance-heavy, this prevents separate compilation.

During this research, we have studied the implications of creating a restrictive CFI with type matching and propose some solutions to improve the accuracy of the CFG. Previous efforts have shown that even context-sensitive points-to analysis is not accurate enough for creating CFG [3]. Combining the result of points-to analysis and type checking can result in a more precise CFG. In other words, by pruning the CFG with type matching, a more precise CFG would be available. This purging decreases the chance of a practical attack on CFI, but it faces numerous practical deployment challenges.

References

[1] Shacham, Hovav, et al. "On the effectiveness of address-space randomization." Proceedings of the 11th ACM conference on Computer and communications security. ACM, 2004.

[2] Abadi, Martn, et al. "Control-flow integrity." Proceedings of the 12th ACM conference on Computer and communications security. ACM, 2005.

[3] Evans, Isaac, et al. "Control jujutsu: On the weaknesses of fine-grained control flow integrity." Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015.

[4] van der Veen, Victor, et al. "A tough call: Mitigating advanced code-reuse attacks at the binary level." Security and Privacy (SP), 2016 IEEE Symposium on. IEEE, 2016.

[5] Tice, Caroline, et al. "Enforcing Forward-Edge Control-Flow Integrity in GCC & LLVM." USENIX Security Symposium. 2014.