ISR key aligned CFI

Anonymous Submission

1. RELATED

We can split CFI systems found in the literature into two main CFI approaches: Systems that rely on just binaries and do not require recompiling and those that require to be present in compilation time and act as another level in the build toolchain. Systems in the first category typically begin by disassembling the executable and then attempting to identify transfer control edges by constructing a CFG; some of those systems attempt to locate only forward edges while others look into return addresses as well. Systems from the second category since they have access during the build time, can generally produce more complete CFG since they have access to the source code but on the flip side require rebuilding of all modules in the software and that is not possible in closed source modules or shared libraries where source code is not freely available. Specifically for the shared libraries, this is another key point where work found in the literature forks since some systems support having shared libraries while others do not.

In the table below "Alignment" refers to protection in jumping in the middle of instructions due to x86 CISC dense architecture. "Shared libraries" represents the ability of the system to work with modules not necessarily protected by the technique and/or loaded at a later time. Specifically for Safedispatch [2] while it is not a general CFI system and targets vtables and hence C++ code; it was mentioned extensively in the related work of most CFI systems; while it captures only JOP style attacks targeting vtables it claims that this is one of the most important attack vectors, this claim is supported by Tice et al. [3] were they report that vtable jumps correspond to 95-99% of all Indirect Control Transfers.

Everyone assume additional protections such as stack canaries and DEP or WOX and some form of ASLR.

2. REFERENCES

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS. ACM, 2005.

[2] D. Jang, Z. Tatlock, and S. Lerner. Safedispatch: Securing c++ virtual calls from memory corruption attacks. In *NDSS*, 2014.

[3] C. Tice, T. Roeder, P. Collingbourne, S. Checkoway, Ú. Erlingsson, L. Lozano, and G. Pike. Enforcing forward-edge control-flow integrity in gcc & llvm. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[4] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song, and W. Zou. Practical control flow integrity and randomization for binary executables. In *Security and Privacy (SP), 2013 IEEE Symposium on*, 2013.

[5] M. Zhang and R. Sekar. Control flow integrity for cots binaries. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

Table 1: Comparison Table

			Protects						
Name	Relocation Tables	Compiling	Vtable	Indirect	Direct	Ret	SharedLibraries	Alignment	Performance
CCFIR [4]	X	X	-	✓	DEP	✓	✓	✓	3.6%/8.1%
binCFI [5]	two versions	X	✓	✓	DEP	✓	✓	✓	12%-45%
ForwardEdgeCFI [3]	-	✓	✓	✓	DEP	X	✓	✓	2%/8%
CFI [1]	✓	X	✓	✓	DEP	✓	✓	✓	15%/46%
Safedispatch	-	✓	✓	-	DEP	-	X	✓	2.1%