Summary of Language-based Security

CHRISTOPH MENDE

ACM Reference format:
Christoph Mende. 2017. Summary of Language-based Security. 1, 1, Article 1 (March 2017), 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The paper Language-based Security[1] is about the threat of malicious code faced when executing source code from unknown sources. With critical infrastructure like transportation, communication, energy distribution, and health care and computer users downloading and running software with "little regard for the consequences" it is increasingly important to verify that the software does nothing undesirable.

2 SOME ISSUES IN SECURITY

The paper presents three basic safety policies that should be the minimum when running untrusted code. These are control flow safety, memory safety, and stack safety.

Control flow safety means that any jump or call must be to a valid function entry in the program's own code and any return must return to the point where the function was called. Jumps to random addresses or anything other than function entry points must not be allowed.

Memory safety is the safety that the code can only its own data. That is, it is only allowed to access its static data segment, the memory explicitly allocated to it on the live system heap and its stack frames. This data includes static or global variables in the code inside the static data segment, dynamically allocated variables inside the live system heap, and information about function calls such as the return address of the calling function, parameters, and local variables of the called function inside stack frames on the program's stack. Minor modification of the stack near its end is allowed.

A problem arises when trying to enforce these policies since some software needs to be trusted to some degree. One possibility is that the source code must be released so the user can compile the software with a trusted compiler on his own system, but that includes performance issues since the software needs to be compiled first. Alternatively we need to trust the compiler of the distributor and his distribution channels, but that has security issues since it we need some mechanism to provide trust to these. Because of this, approaches that do not require a trusted compiler are needed.

3 TRADTIONAL APPROACHES

Aside from trusted compilation, other approaches that do not require a trusted compiler include the kernel as a reference monitor, cryptography, and code instrumentation.

The kernel as reference monitor appraoch is an approach where the system kernel monitors the running code and enforces the security policies. Since the kernel is running privileged, it can access other data and restrict programs. This appraoch also limits performance for two reasons. One is that programs are no longer able to exploit properties of low-level data structes. Another reason is that kernel calls always have some overhead.

2017. XXXX-XXXX/2017/3-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Cryptography depends on the assumption that it cannot be broken in polynomial time. Additionally, cryptography can only ensure that the software comes from a trusted source and was not compromised, it cannot ensure that the software is safe to run.

4 LANGUAGE-BASED SECURITY

REFERENCES

[1] Dexter Kozen. 1999. Language-Based Security. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS '99)*. Springer-Verlag, London, UK, UK, 284–298. http://dl.acm.org/citation.cfm?id=645728.667701