Summary of "Language-Based Information-Flow Security"[1]

CHRISTOPH MENDE, NIKOLAJ-JENS SCHWARTZ

1 INTRODUCTION

Language-Based Information-Flow Security is about the security of information processed by a program.

2 BACKGROUND

Standard security mechanisms like access control, firewalls, encryption or antivirus software can only control who has access to confidential data. They cannot control what a process with access to the data does with it. A process could for example leak confidential data to the outside world. Access control, firewalls and encryption only limit the group of people or processes with access, antivirus software can only detect known malicious behaviour and is thus ineffective against new attacks.

Language-based security is also different from language-based information-flow security in that its purpose is not to control the information-flow. Examples for language-based security are the JVM's bytecode verifier, sandbox model, and stack inspection. The bytecode verifier makes sure that the type system is respected and private fields cannot be directly accessed, but that doesn't mean that indirect access cannot leak information. The sandbox restricts an applet's access to other classes, but the applet can still leak information to the host it was downloaded from. Stack inspection is a dynmic check that only checks integrity, not confidentiality.

An application can leak information through various channels. If a channel leaks information that are outside of its actual purpose, these channels are called covert channels. Covert channels include implicit flows, termination channels, timing channels, probabilistic channels, resource exhaustion channels, and power channels. All of these channels expose some information — like the time it took to compute something, the RAM or CPU usage, or the power intake — that could be used to deduce information about what was computed.

One approach to controlling information-flow is mandatory access control, that assigns a security label to each data item. This label has to be computed simultaneously to each computation done by the application. This system had too much overhead and was too restricted though. Another problem of this approach is called label creep, which happens when more and more data items get assigned a higher security label until everything is in high security.

A more efficient approach is a static analysis of the program performed by type checking. This type check works similar to the security labels in the mandatory access control approach, but instead of being computed at runtime, the labels are assigned statically in this approach. Every expression in the program has its type and a label that defines the confidentiality of the data. The information-flow control is then done with a program-counter label that defines the sensitivity of the current context and of the expressions currently computed. If an expression tries to assign a value to a low-sensitivity variable during a high-sensitivity context, the system rejects the program as insecure.

Another approach is a noninterference policy. This policy also defines low and high-sensitivity data inside a program. Both may be modified during the execution as long as any change in high-sensitivity data does not influence low-sensitivity data. That is, modifying confidential data must not be visible to the outside.

3 BASICS OF LANGUAGE-BASED INFORMATION FLOW

REFERENCES

[1] A. Sabelfeld and A. C. Myers. 2006. Language-based Information-flow Security. *IEEE J.Sel. A. Commun.* 21, 1 (Sept. 2006), 5–19. DOI: https://doi.org/10.1109/JSAC.2002.806121