Side Channel Finder (Version 1.0)[*]
Poster Proposal

Alexander Lux Artem Starostin

Computer Science Dept., TU Darmstadt, Germany
{lux,starostin}@mais.informatik.tu-darmstadt.de

The Problem of Side Channels A cryptographic mechanism based on algorithms which are proven to be secure may become vulnerable after it is implemented in some programming language and run on an actual computer system. *Side channel attacks* are based on the fact that by observing the implementation's behavior which is not modeled by the underlying cryptographic algorithm the attacker can infer confidential data, e.g., a secret key. Therefore, when developing a cryptographic mechanism it is desirable to check whether its actual implementation opens up side channels.

One possibility to launch a side channel attack is to exploit the variance in the running time of a crypto-algorithm implementation. First studies of timing attacks on cryptographic schemes, including Diffie-Hellman and RSA, date back to mid 1990s [5]. Since then, they have been practically demonstrated [4], optimized [8], and evaluated [9]. We present the *Side Channel Finder* in the version 1.0 (short SCF 1.0), a static analysis tool for detection of potential timing channels in Java implementations of cryptographic algorithms. To the best of our knowledge, SCF 1.0 is the first timing channel analysis tool for Java.

The Tool The purpose of SCF 1.0 is to support a programmer of a crypto-algorithm implementation in assessing his code for that it is not vulnerable to timing channel attacks. The tool lets the programmer specify which input of an implemented algorithm constitutes a secret that must not be leaked, especially not through timing channels. These specifications are a part of a security policy which assigns security levels *high* and *low* to object fields, method parameters, and return values. A policy is stored in an automatically generated XML-document accompanying the source code.

[*]Work is supported by the Center for Advanced Security Research Darmstadt (CASED).

SCF 1.0 then analyzes the given program code by checking whether the control flow potentially depends on the confidential inputs.

Implementation Techniques The analysis which SCF 1.0 performs is based on a carefully crafted security type system for information flow. Most importantly, this type systems checks whether (i) the high-classified data is moved to low-classified locations, and (ii) the execution would branch on a value with high security level. These branches might appear due to conditional statements, loops, or polymorphic method calls.

Applications We have applied SCF 1.0 to analyze several existing implementations of cryptographic algorithms. Our studies include well known open-source libraries *FlexiProvider* (version 1.6p7) [2] and *GNU Classpath* (version 0.98) [1]. For example, we have found timing channel vulnerabilities in the FlexiProvider's implementation of the IDEA encryption scheme.

Future Work Currently we see two directions for improvements in the Side Channel Finder. First of all, in the next months we plan to implement automatic program transformations for elimination of discovered timing channels. Here we have a choice between a number of techniques, namely, cross-copying [3], unification [6], or conditional assignment [7]. Secondly, we are working in the direction of making the analysis more fine-grained. This would be based on a refined model for timing channel security where the branching on the secret data would be allowed provided the branches take the same amount of time. For that, we have to decide how precise this time has to be measured with possibilities ranging from counting the number of executed statements — through considering single expression evaluation steps — to careful estimation of the timing behavior of the compiled code instructions.

Bibliography

[1] GNU Classpath. `http://www.gnu.org/software/classpath/`, 2009.

[2] FlexiProvider. `http://www.flexiprovider.de`, 2010.

[3] J. Agat. Transforming out Timing Leaks. In *27th POPL*, pages 40–53, 2000.

[4] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[5] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *16th CRYPTO*, LNCS 1109, pages 104–113. Springer-Verlag, 1996.

[6] B. Köpf and H. Mantel. Transformational Typing and Unification for Automatically Correcting Insecure Programs. *Intl Journal of Information Security*, 6(2–3):107–131, 2007.

[7] D. Molnar, M. Piotrowski, D. Schultz, and D. Wagner. The Program Counter Security Model: Automatic Detection and Removal of Control-Flow Side Channel Attacks. In *8th ICISC*, LNCS 3935, pages 156–168, 2005.

[8] W. Schindler. On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods. In *8th PKC*, LNCS 3386, pages 85–103, 2005.

[9] F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *28th EUROCRYPT*, LNCS 5479, pages 443–461, 2009.