Development and Industrial Application of Multi-Domain Security Testing Technologies

Innovation Sheet

Static Binary Code Analysis for Vulnerability Detection

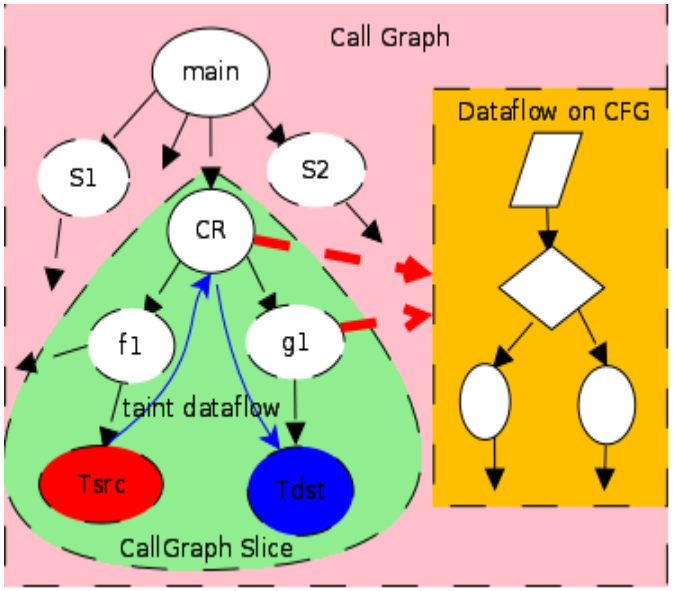
ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Static Binary Code Analysis
Description

The technique statically analyzes the binary executable code of application to detect low-level vulnerabilities. The main features are:

1. Vulnerable Sink Identification:
   - By performing an intraprocedural dataflow analysis, the technique identifies vulnerable functions (e.g. *buffer overflow prone functions)*.
2. Vulnerable Execution Path Identification (Program slicing):
   - Given a pair of taint input function (Tsrc) and vulnerable function (Tdst), calculate a callgraph based slice rooted at common function (CR).
   - Intraprocedural and interprocedural dataflow on the control flow graph of each function (f_is and g_is) to calculate data dependencies.
   - Output: statically computes a path such that taint input can flow to vulnerable functions.



Static Binary Code Analysis
State of the art

- Vulnerable functions identification:
  - Focusing on known vulnerable library functions is the most common way to detecting vulnerabilities.
    - D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken, "A first step towards automated detection of buffer overrun vulnerabilities," in Proc. of the symp. NDSS 02. The Internet Society, 2000, pp. 3–17.
    - J. Newsome and D. X. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in Proc. of NDSS 2005, San Diego, California, USA. The Internet Society, 2005.
  - Another way is to calculate differences between patched and un-patched versions of a given application OR full function coverage.
    - D. Brumley, P. Poosankam, D. Song, and J. Zheng, "Automatic patchbased exploit generation is possible: Techniques and implications," in Proc. of the 2008 IEEE Symposium S&P. Washington, DC, USA: IEEE, 2008, pp. 143–157.
    - P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in NDSS, 2008.

Static Binary Code Analysis
State of the art

- Static Vulnerable Path Identification:
  - Not many tools/techniques to perform binary static taint analysis on real-world applications. Parfait (Scholz et al) tool operates at the C source level and work by Tripp et al performs the similar analysis, but on the source code.
    - Scholz, B., Zhang, C., Cifuentes, C.: User-input dependence analysis via graph reachability. In: IEEE Int. Workshop SCAM '08, Los Alamitos, CA, USA (2008)25–34
    - Tripp, O., Pistoia, M., Cousot, P., Cousot, R., Guarnieri, S.: Andromeda: accurate and scalable security analysis of web applications. In: Proc. of the 16th international conference FASE. FASE'13, Berlin, Springer-Verlag (2013) 210–225
  - Reps et al. proposed an algorithm for "precise interprocedural program chopping", but this algorithm relies on specific intermediate program representations (PDG and SDG), which is very expensive. LoongChecker tool is very close to our work, but it performs an interprocedural data-dependence analysis with a VSA (Value-Set Analysis) on all functions, which is again expensive.
    - Reps, T., Rosay, G.: Precise interprocedural chopping. In: Proceedings of the 3rd ACM symposium FSE. SIGSOFT '95, NY, USA, ACM (1995) 41–52
    - Cheng, S., Yang, J., Wang, J., Wang, J., Jiang, F.: Loongchecker: Practical summary-based semi-simulation to detect vulnerability in binary code. In: Proc. 10th Int. Conf. on TrustCom, IEEE (2011) 150–159

Static Binary Code Analysis

Advances beyond the state of the art

- Identifying GENERIC vulnerable functions
- Directly applicable on BINARY code

In general, most of the approaches on vulnerability analysis rely on well known vulnerable functions for the analysis. The idea of vulnerable functions (buffer overflow vulnerability) is formalized and an efficient algorithm is presented to detect such functions in the large binary code. The algorithm is implemented in a tool which is released as open source.

[Deliverable D3.WP2, Section 2.2.3]

- Pure static analysis of taint flow.
- Directly on BINARY code

Performing a taint analysis has largely been targeted as dynamic approach. We present a pure static approach for performing taintflow analysis. There are very few proposals in the literature that perform static taint analysis and this number goes way down when we consider binary code analysis. The existing approaches on binary taint analysis are very complex and not scalable. We propose technique to perform static taintflow analysis which is scalable and able to analyze real world applications. The whole approach is implemented in a tool and tested on several applications.

[Deliverable D5.WP2, Section C-IV]

Static Binary Code Analysis
Exploitation and application to case studies

- The proposed work have been presented in international conferences and various technical meetings/discussions/seminars.
- The approach has been applied to Metso case study.
- The approach has been implemented as a tool to make it available for larger usage.