RE-TRUST

PoliTO Prototypes Overview

***Paolo Falcarin*, Stefano DiCarlo**
Alessandro Cabutto, Nicola Garazzino, Davide Barberis

SoftEng
http://softeng.polito.it

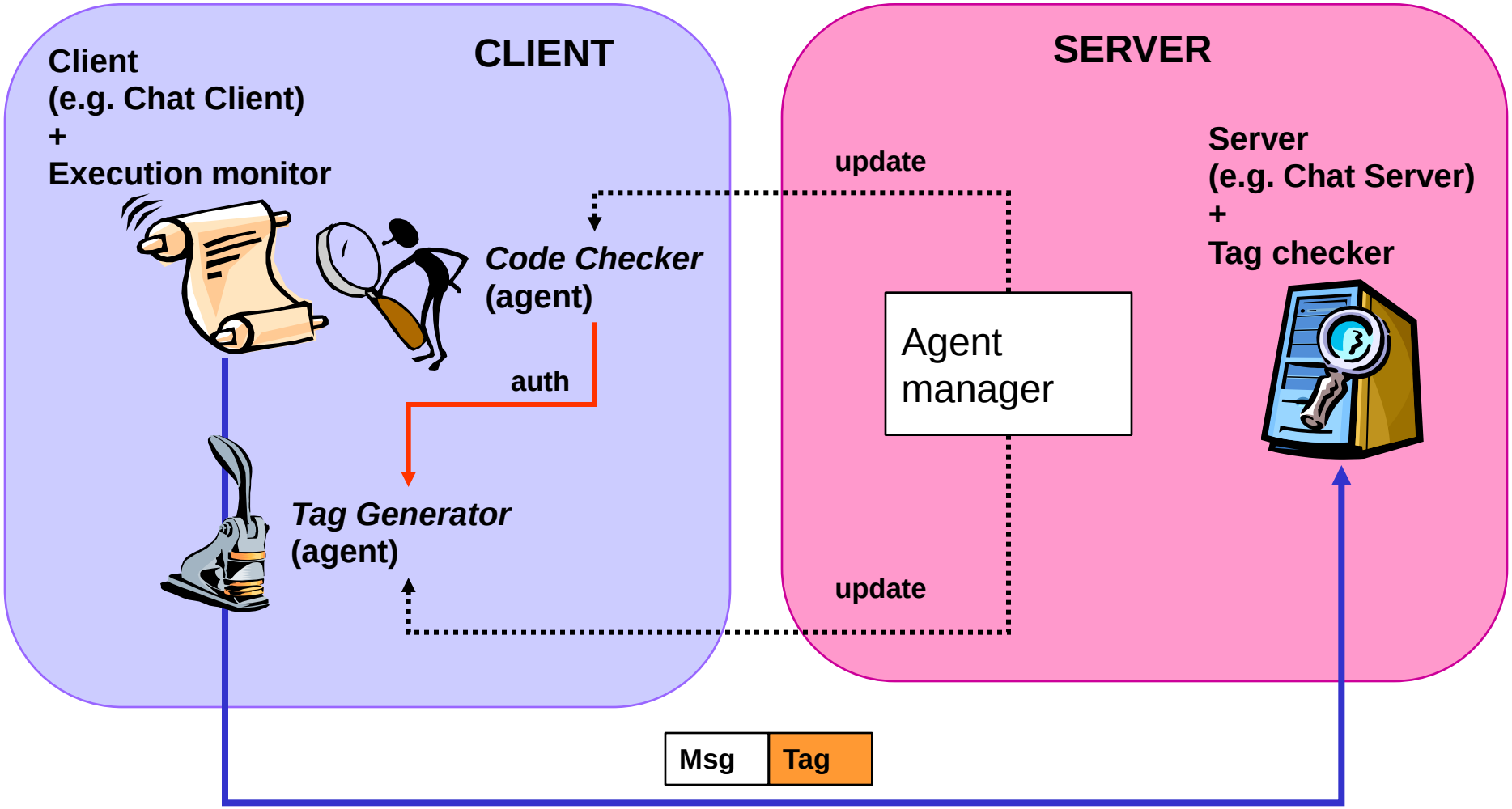
RE-TRUST meeting
Paris, Dec 2008

RE-TRUST Project

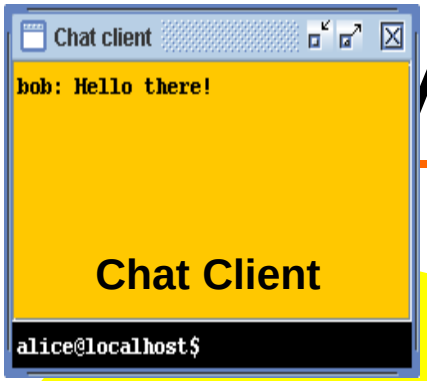
Many prototypes developed:

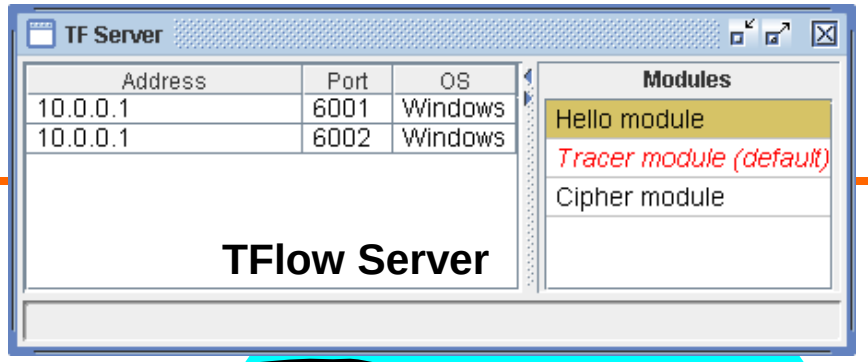
1. Mobile Code continuously replaced:
   a) Java Aspect on DynamicAOP –JVM
   b) Binary code on JVMTI interface– JVM5
   c) Binary code linked in EXE
2. Invariants checking
3. Control-Flow Checking

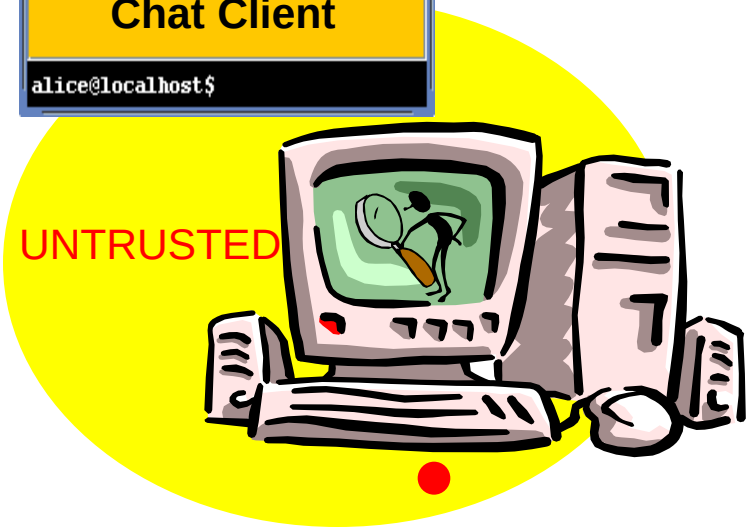
Prototype 1 a-b

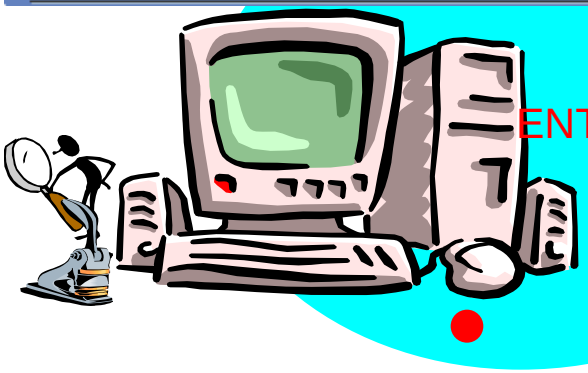


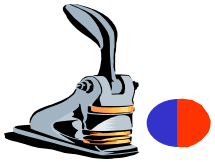
ype(s)

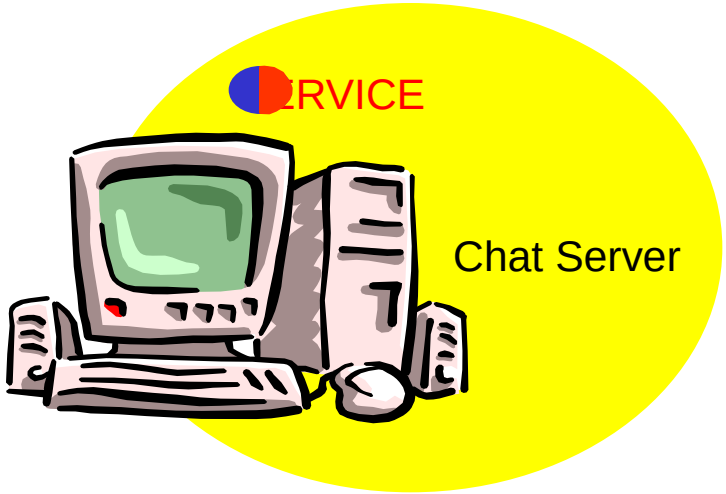

Chat client
bob: Hello there!
Chat Client
alice@localhost$


TF Server
Address
Port
OS
10.0.0.1
6001
Windows
10.0.0.1
6002
Windows
Modules
Hello module
Tracer module (default)
Cipher module
TFlow Server


UNTRUSTED


ENTRUSTER




RVICE
Chat Server

Integrity check

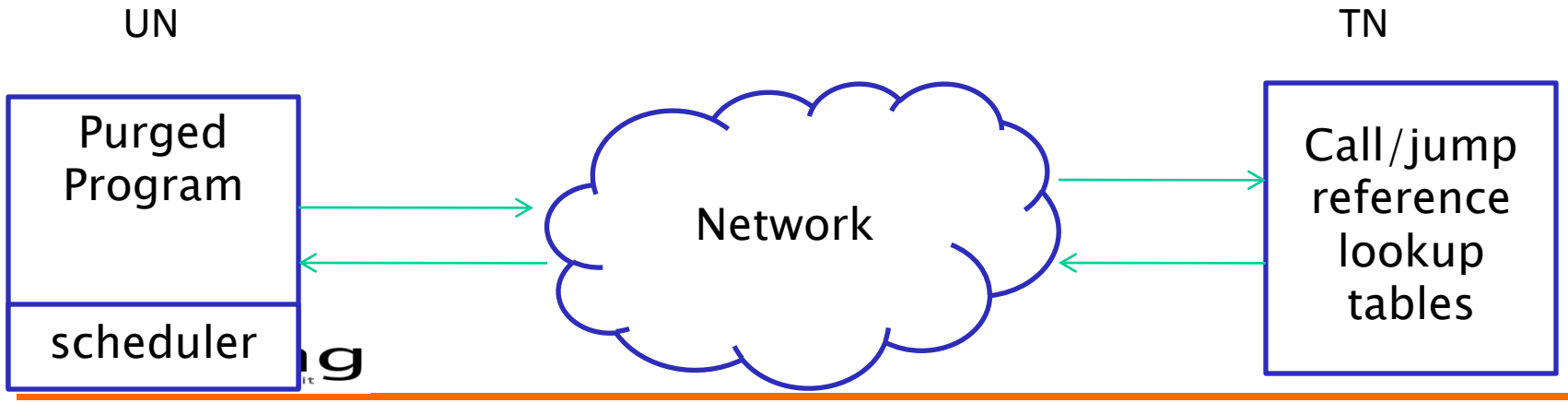
- Module contains
  - List of crypto hashes (each method)
  - Symmetric key
- Keyed hash recomputed each time a method is called
- Hash compared with “good” copy
- New Module checks previously deployed ones

Prototype 1a-b

- Execution interception
  - Module calculates proof
  - Seamless replacement at run-time
  - Slow start-up
- Transparent tag insertion
  - Call to socket write are intercepted and data buffer is tagged
- Client code:
  - Its image is checked in memory (JVMTI)
  - can be sandboxed (AOP)

Prototype 1c

- The application is deployed incomplete
- Some binary code blocks are downloaded and linked in memory at start-up
- The memory layout si decided by the server, and it changes at every run
- Useful to defeat static analysis



Program Instrumentation step

- PE header patch
  - make code segment writable at runtime;
- Disassembling the executable collecting information about control-flow
  - Instructions calls, jump, ret;
- Functions “to be protected” are purged
  - They won’t be available in client application;
- Patching of calls/jumps referring to purged functions:
  - they will now point to scheduler()

Runtime

- When scheduler is invoked ->it send caller address to trusted node
- The TN uses its lookup tables to find out if the caller needs some purged code to be sent to the UN
- The scheduler in UN
  - receives the necessary code
  - Allocate it in memory
  - then executes it

Remote Control Flow Checking

- Split the program integrity verification among the untrusted and the trusted node:
  - Program execution performed on the untrusted node
  - Control flow validation performed on the trusted node

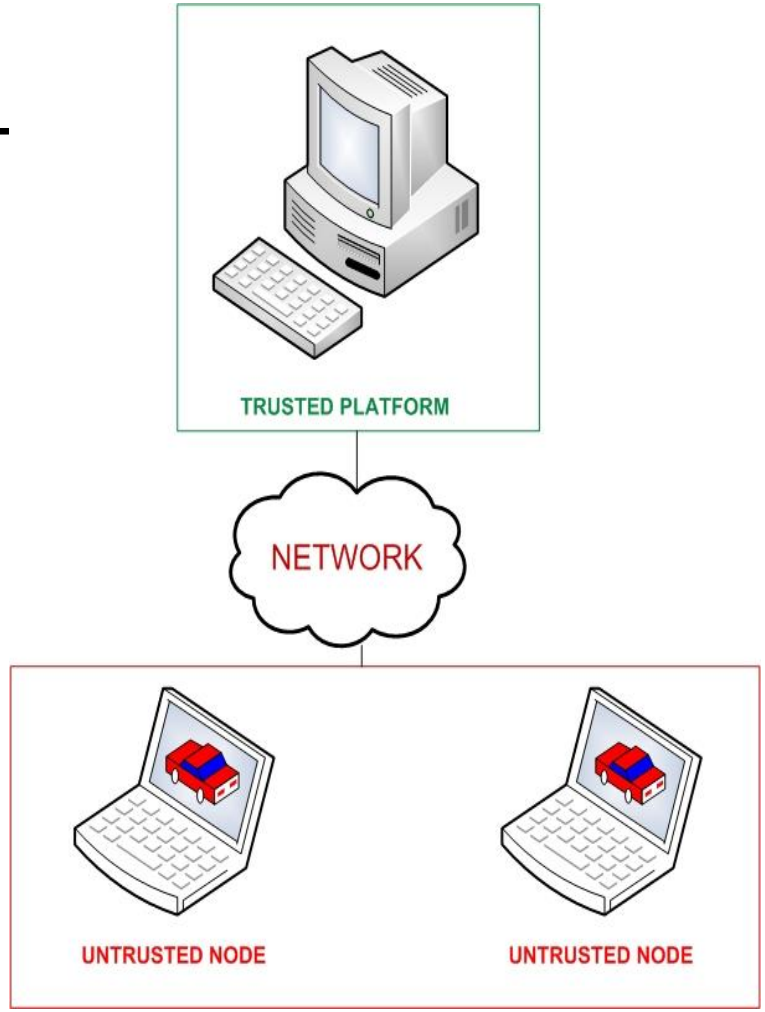
Remote Control Flow Checking

- Basic flow:
  - The target application collects information (traces) about executed instructions
  - Traces are transmitted from the untrusted node to the trusted node
  - The trusted node validates the control flow of the application
  - Any violation is detected as an attack

- The trusted node is in charge of:
  - Monitoring the flow of instructions received from the untrusted node (correct sequence of basic blocks)
  - Validating the checksum of each basic block (correct instructions opcode)

Invariants Prototype: Car Race

- Language: C++
- Graphic Library: Open GL
- IDE: Visual Studio 2003
- One Server, 2 players



Prototype: Car Race

- There are 10 check points.
- Every check points has a particular picture with DRM

Car Race: Security

- Security:
  - Secure Protocol
  - Invariants Checks
  - Mobile Code
  - Mutual Authentication
- Invariants Check
  - To protect algorithm
  - To protect DRM

Car Race: Security

- Mobile Code
- Mutual Authentication → A client needs the server authentication, in fact without this “ACK”, the trusted platform uses a wrong key to crypt information.

Initial Analysis

SOftEng
http://softeng.polito.it

Open issues with mobility

- How to protect mobile modules?
  - Obfuscation
- How often a module is replaced?
  - It depends on time needed to understand it and implement an attack
- How to measure this Time-2-Break-It?
  - Metrics
  - Empirical Evaluation

Obfuscation Metrics

- Source Code complexity
  - Potency and other metrics (Collberg et al.)
  - Depth of Parse Tree (Goto et al.)
  - DeObfuscation Time (Udupa et al.)
- Binary Code complexity
  - Confusion Factor (Linn et al.): % of code that cannot be disassembled
- Compare Obfuscations (Anckaert et al.)
  - Code & control flow metrics
  - Data and data-flow metrics

Empirical Evaluation of Obfuscation

- Complexity of reverse engineering binary code
  - Asking a group of 10 students to perform static analysis, dynamic analysis and change tasks on several C (compiled) programs.
  - They found that the subjects' ability was significantly correlated with the success of reverse engineering tasks they had to perform.
  - I. Sutherland, G. E. Kalb, A. Blyth, and G. Mulley. An empirical examination of the reverse engineering process for binary files. *Computers & Security, 25(3):221–228, 2006.*
- Complexity of Java id-renaming source code obfuscation:
  - Controlled experiment of Master and Ph.D students to crack 2 apps (one clear , one obfuscated)
  - Calculate statistical effect size of the applied obfuscation
  - M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. Towards experimental evaluation of code obfuscation techniques. In *Proc. of the 4th Workshop on Quality of Protection. ACM, Oct 2008*

References

- [H. Chang and M. Atallah, "Protecting software code by guards" Proc. of ACM Workshop on Security and Privacy in Digital Rights Management, 2002
- B. Horne, L. Matheson, C. Sheehan, and R. E. Tarjan, "Dynamic Self-Checking Techniques for Improved Tamper Resistance" Proc. of ACM Workshop on Security and Privacy in Digital Rights Management, 2001
- Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. Jakubowski, "Oblivious hashing: Silent Verification of Code Execution" Proc. of 5th International Workshop on Information Hiding (IHW 2002), 2002
- B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (Im)possibility of Obfuscating Programs" Proc. of CRYPTO 2001, 2001
- D. Aucsmith, "Tamper resistant software: An implementation" in *Information Hiding, Lecture Notes in Computer Science 1174*, R. J. Anderson, Ed.: Springer-Verlag, 1996.
- The Trusted Computing Group. On-line at https://www.trustedcomputinggroup.org
- R. Sailer, X. Zhang, T. Jaeger, and L. VanDoorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture" Proc. of 13th USENIX Security Symposium, 2004, pp. 223-238.
- M. Jakobsson, K. Reiter, "Discouraging Software Piracy Using Software Aging". ACM Workshop on Security and Privacy in Digital Rights Management, Philadelphia, USA, November 2001.
- Kennell, R., Jamieson, L. H., Establishing the Genuinity of Remote Computer Systems. Proceedings of the 12th USENIX Security Symposium, 2003
- Maña, A., López, J., Ortega, J., Pimentel, E., Troya, J.M., A Framework for Secure Execution of Software. International Journal of Information Security, Vol. 3(2), 2004
- Sander, T., Tschudin, C. F., Towards Mobile Cryptography. IEEE Symposium on Security and Privacy, 1998
- A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms. In 20th ACM Symposium on Operating Systems Principles (SOSP-05), Brighton, UK, October 2005.
- P. Falcarin, R. Scandariato, and M. Baldi: Remote Trust with Aspect Oriented Programming. In IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 06), 2006
- Garay, J.A., and Huelsbergen, L.: Software Integrity Protection Using Timed Executable Agents. In Proc. ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'06), pp. 189-200, 2006.

References

- B. Anckaert, M. Madou, B. D. Sutter, B. D. Bus, K. D. Bosschere, and B. Preneel. Program obfuscation: a quantitative approach. In *QoP '07: Proc. of the 2007 ACM Workshop on Quality of protection, pages 15–20, New York, NY, USA,* 2007. ACM.
- B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. *Lecture Notes in Computer Science,* 2139, 2001.
- M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. Towards experimental evaluation of code obfuscation techniques. In *Proc. of the 4th Workshop on Quality of Protection. ACM, Oct 2008*
- C. Collberg, C. Thomborson, and D. Low. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering, 28,* 2002.
- H. Goto, M. Mambo, K. Matsumura, and H. Shizuya. An approach to the objective and quantitative evaluation of tamper-resistant software. In *Third Int. Workshop on Information security (ISW2000), pages 82–96. Springer, 2000.*
- C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Computer and Communications Security Conference (CCS-03), pages* 290-299. ACM, 2003.
- A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement. Pinter, London, 199*
- I. Sutherland, G. E. Kalb, A. Blyth, and G. Mulley. An empirical examination of the reverse engineering process for binary files. *Computers & Security, 25(3):221–228, 2006.*
- S. Udupa, S. Debray, and M. Madou. Deobfuscation: reverse engineering obfuscated code. *Reverse Engineering, 12th Working Conference on, Nov. 2005.*