ISSN 2186-7437

NII Shonan Meeting Report

No. 2015-4

Low level code analysis and applications to computer security

Jean-Yves Marion, *Akira Mori,* *Mizuhito Ogawa*

March 2–5, 2015

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Low level code analysis and applications to computer security

Organizers:

Jean-Yves Marion (LORIA)
Akira Mori (AIST)
Mizuhito Ogawa (JAIST)

March 2–5, 2015

Description of the meeting

Software security is one of the very major issues of modern societies, since insecure systems can affect people (phishing, unauthorized payment, etc.), organizations (critical information leakage) and even states (cyber attacks). In regard to software security, a crucial problem is to have methods and to devise tools to analyze low level code for at least three reasons. First, low level codes are usually the one that is effectively run on a computer system. Low level code is usually obtained by a compilation chain where some parts of the chain maybe insecure. As a result it is necessary to guarantee low level code correction ensuring that programs run by the system cannot be attacked. Second, a lot of applications are available on Internet as binary codes. Those binary codes are quite obfuscated either to protect the intellectual property or because there are malicious software (malware). Therefore, it is necessary to have tools to analyze binary codes in order to detect malware and prevent an attack. It may look surprising that such important issues are so poorly addressed by current state-of-the-technology tools, while formal methods in general and automatic program verification in particular have made tremendous progress in the past decade, resulting in many impressive industrial applications, and demonstrating their strength both in bug finding and safety validation. There are two key problems here. First, security needs are more difficult to characterize (in terms amenable to automatic analysis) than safety needs. Security requirements change depending on the social context, and malicious behavior is not easy to define. For instance, malware often uses windows system calls for malicious behavior, but standard programs also use such capability. To clarify, not only theoretical observation form academia, but also empirical study in industry will be required. Second, security analyses must often be performed at the level of binary executables. Malicious behavior is often based on very fine low level details, which are invisible on the source code (e.g. memory layout, exception handling details, ambiguity in program semantics, bug in compilation), and malware are only available as executable files. This is a key difficulty, as most formal methods have been designed for high-level models or languages, and they rely on hypotheses that no longer hold at the executable code level. For instance,

even (static) disassembly is not possible for recent advanced polymorphic virus, like self-decryption. Thus, model generation of binary executable is already a challenging. The goal of the proposed meeting is to explore all aspects from theory to practice of low level code analysis, including communication between industry and academia. The main outcome of this meeting will be to trigger new interactions and enrich the various approaches. For instance,

- Security threat in practice. e.g., vulnerability, overflow attacks on x86 / Android.
- Malware and botnet.
- Static analysis of low level codes.
- Obfuscated programs, e.g., self-modifying code, packers, and obfuscation techniques.
- Low level code semantics.
- Model generation. e.g., control flow graph reconstruction, disassembly.
- CEGAR (Counter example guided abstraction refinement) and loop invariant generation.
- Testing and virtual binary emulation, dynamic analysis.
- Backend reasoning tools. e.g., SMT (SAT modulo theory), Model checkers.

By interacting industry and academia, and among different areas in academia, we will start to tackle the two key problems.

- *Analyzing security needs.* Learn concrete examples from industrial experiences, and manual pattern analysis of malware behavior in malware database. They would lead clear formal definitions of target properties.
- *Binary executable analysis.* Exchanging ideas of state-of-the-art research on binary executable analysis, and look for collaboration opportunity.

Program

March 1st (Sun) Arrival

18:00-21:00 *Reception*

March 2nd (Mon)

9:45-10:00 *Opening*

10:00-10:45 (1) Arun Lakhotia (University of Louisiana at Lafayette), VirusBattle: Automated Malware Analysis in the Large.

10:45-11:30 (2) Guillaume Bonfante (LORIA, Universite de Lorraine), On malwares chat on the web.

13:30-14:15 (3) Jean-Yves Marion (LORIA, Universite de Lorraine) Disassembly of X86 and related questions.

14:15-15:00 (4) Sebastien Bardin (CEA LIST), Binary-level analysis: from safety to security

15:30-16:15 (5) Todd McDonald (University of South Alabama), Developing an extensible deobfuscation framework.

March 3rd (Tue)

9:30-10:15 (6) Akira Mori (AIST), Methods for whole program analysis of binary code.

10:15-11:00 (7) Quan Thanh Tho (VNU-HCMC, HCMUT), Our group in HCMUT (Those who are working on malware)

11:00-11:45 (8) Nguyen Minh Hai (VNU-HCMC, HCMUT), Pushdown Model Generation for Malware Deobfuscation

13:30-14:15 (9) Sandrine Blazy (University of Rennes) Towards a formally verified obfuscating compiler

14:15-15:00 (10) Julian Kranz (Technische Universitat Muchen), IR processing using GDSL

15:30-16:15 (11) Frederic Besson (University of Rennes), Semantics for low level code : From C to binary (and back)

16:15-17:00 (12) Tachio Terauchi (JAIST), Information Flow Analysis and Applications to Computer Security

March 4th (Wed)

9:30-10:30 (13) Thomas Dullien and Tim Kornau (Google), Analysis of binaries at Google: The unreasonable success of nonsemantic methods

10:30-11:15 (14) Roberto Giacobazzi (University of Verona), Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables

11:15-11:45 (15) Mizuhito Ogawa (JAIST), Constant propagation for Binary CFG rebuilt

11:45-11:50 *Closing*

13:00-21:00 *Excursion and Banquet*

March 5th (Thu) Departure

Abstract

1. Arun Lakhotia (University of Louisiana at Lafayette), VirusBattle: Automated Malware Analysis in the Large.

   While the growth in unique malware, as defined by their hashes, is growing exponentially, it is apparent that without similar growth in malware authors the diversity must be machine generated. The large volume of malware must then contain evidence that connects them. To extract the information one needs an ability to analyze very large collection of malware. In the Software Research Lab, we have been exploring methods of extracting information from malware by peeking through their obfuscations. The research has led to the development of VirusBattle, a system that automatically unpacks malware, performs semantic analysis of disassembled malware to create semantic hashes, and builds a repository of malware indexed using a variety of hashes. The system is highly scalable and highly parallelizable. In a lab environment, with a cluster of three VMs with shared 32 cores, the system can process over 1,000 malware samples in an hour. The information extracted from malware can be used to make a variety of queries. For instance, one can search for similar malware in a collection, or search for similar procedures, or even similar code blocks. The system also provides other finer level details that may be useful for developing other analyses. VirusBattle is available as a free web-service from api.virusbattle.com.

2. Guillaume Bonfante (LORIA, Universite de Lorraine), On malwares chat on the web.

   The fact that new malware appear every day demands a strong response from anti-malware forces. One way to do it is to observe communications between the malware and its Command and Control. Indeed, it is usually a mandatory task of malware, and thus a weakness. We propose a new approach based on dynamic analysis and finite state automata.

3. Jean-Yves Marion (LORIA, Universite de Lorraine) Disassembly of X86 and related questions.

   This talk introduces brief overview of High-security lab at LORIA, and our tool for x86 disassembly, intending malware. Our tool combines dynamic trace analysis (based on Intel PIN) and static analysis at detected conditional branches, and focuses on decomposing binary into waves, which is effective to analyze packer behavior. Experimental results are also shown.

4. Sebastien Bardin (CEA LIST), Binary-level analysis: from safety to security

   We present some recent on-going work on the BINSEC platform, an open-source platform developed within the BINSEC project (French research project gathering CEA, INRIA, LORIA, University of Grenoble and Airbus Group) dedicated to binary-level security analysis. We focus the presentation on two points. First, while most semantic binary-level analyses rely on a flat memory model, we reuse the "low level region" semantics developed by Besson, Blazy and Wilke for low level C programs. Experimental results show that the low level region semantics can indeed represent many legitimate binary-level behaviours, while still providing a clean region-based memory model. Second, we describe the simplification mechanisms integrated into the platform in order to reduce the size of the produced IR. Compared to other proposals, our simplifications are done at the function-level (rather than block-level) and they are interprocedural (yet efficient, through automatic function summarization). Experimental results shows that our simplification stage does remove most of the computations on flags and temporary variables introduced at the translation stage, yielding a smaller and simpler IR to analyze.

5. Todd McDonald (University of South Alabama), Developing an extensible deobfuscation framework.

   Obfuscation tools, methods, and techniques for low and high level code are well developed, finding implementations in both commercial products and research/open source projects. Malware writers use these tools at their disposal as well to frustrate analysis or detection of their software. Apart from commercial or open source disassemblers and decompilers, deobfuscation tools tend to be more of an art than science in terms of maturity. We consider the software engineering and pedagogic aspects of developing a framework for obfuscated code analysis from the ground up, considering not only the expertise and skills required of students with various levels of reverse engineering background, but the technical complexities of creating a framework that can be built, extended, and updated with new research techniques while leveraging current results from the research community.

6. Akira Mori (AIST), Methods for whole program analysis of binary code.

   We show our tool development for a whole binary code analysis on IA-32. It consists of both static and dynamic analysis. We apply typical static analyses based on SSA, and symbolic evaluation to decide the destinations of indirect jumps. Static analyses are designed to be context sensitive, in the sense 1-CFA (tagged copy). We also extend SSA to so-called fibered

SSA by adding hyper edges of projection functions. Some demos are also shown.

7. Quan Thanh Tho (VNU-HCMC, HCMUT), Our group in HCMUT (Those who are working on malware)

   Among various approaches for binary analysis, BE-PUM (Binary Emulation for PUshdown Model Generation) is the sole tool supports producing pushdown model. It is achieved by combining concolic approach which pushdown model. In one hand, concolic performs under-approximation when processing complicated instructions. In the other hand, pushdown model precisely simulates the operational environment of an executed program. As compared to similar emerging tools, BE-PUM introduces visible difference when generating control flow graph (CFG); especially when encountering indirect jumps. Moreover, pushdown model allows BE-PUM to handle self-modification code, a technique widely used by sophisticated viruses.

8. Nguyen Minh Hai (Industrial University of HCMC), Pushdown Model Generation for Malware Deobfuscation

   Further the previous talk on BE-PUM, in this talk we discuss pushdown model generation for malware deobfuscation. Nowadays, malware usually obfuscates itself to hide their real harmful intentions and behaviors. Such techniques pose a real challenge for anti-virus tools since the obfuscated malicious actions are difficult to be captured using the typical approaches. By generating the pushdown model from the original code using BE-PUM, we successfully handle some popular obfuscations techniques, including indirect jumps, encryption, and exceptions. Real virus samples using those techniques and close observation of how BE-PUM processes them will be also given in details.

9. Sandrine Blazy (University of Rennes) Towards a formally verified obfuscating compiler

   We have been working on CompSert, which is a modestly optimized C compiler with proofs in a proof assistant Coq. Thus, it will not introduce during compile time. In this talk, we investigate introduction of obfuscations at C source code level, such as opaque predicates, integer encoding, variable renaming/encoding, and control flow flattering. Their preservation of semantics is verified in the certified compiler framework CompSert.

10. Julian Kranz (Technische Universitat Muchen), IR processing using GDSL

    Analyzing binary code begins with the interpretation of a low level binary stream. To this end, the machine instructions are decoded and translated into an intermediate representation (IR) that serves as an input to an analyzer. We present our functional language GDSL that is geared towards the simple and effective specification of instruction decoders and semantic translators. The GDSL toolkit ships with front-ends for various architectures that translate machine code into the minimalistic IR called RReil. We show that applying a few classic program optimizations to this IR leads to a considerable size reduction and, thereby, allows for a faster

analysis. Moreover, fusing several IR statements also recovers parts of the original high-level computation and, as a consequence, allows for a simpler analyzer.

11. Frederic Besson (University of Rennes), Semantics for low level code : From C to binary (and back)

    Formal semantics of low level code is a difficult problem, since it relates compiler / CPU implementation details and complex memory models. There are several recent works focusing on C-semantics. We have been working on CompSert, which is a modestly optimized C compiler with proofs in a proof assistant Coq. Thus, it will not introduce during compile time. In this talk, we investigate the formal semantics of assembler by converting (and back) C semantics by the certified framework CompSert.

12. Tachio Terauchi (JAIST), Information Flow Analysis and Applications to Computer Security

    Information flow analysis and its applications to computer security I will give a brief overview of my past work on quantitative information flow. I will also give a brief overview on a work on tamper resilience against side channel attacks.

13. Thomas Dullien and Tim Kornau (Google), Analysis of binaries at Google: The unreasonable success of nonsemantic methods

    Google performs analysis of executable code for many different purposes - from the analysis of closed-source components for security flaws to automated analysis of executables. This presentation will discuss our experiences in performing binary analysis at Google with particular focus on the almost unreasonable success that we had in applying non-semantic and statistical methods to the problem. Frederick Jelinek famously quipped about language processing that "every time I fire a linguist, the performance of my speech recognizer goes up" - indicating that a pragmatic approach to improving performance sometimes requires resorting to statistical methods, especially when data is abundant.

14. Roberto Giacobazzi (University of Verona), Abstract symbolic automata: Mixed syntactic/semantic similarity analysis of executables

    The use of mixed syntactic/semantic representation of code in similarity analysis is becoming a good practice because pure semantic similarity is too complex and often undecidable while pure syntactic similarities are often imprecise and prone to false negatives due to code obfuscation techniques. We introduce a model for mixed syntactic/semantic approximation of programs based on symbolic finite automata (SFA). The edges of SFA are labeled by predicates whose semantics specifies the denotations that are allowed by the edge. We introduce the notion of abstract symbolic finite automaton (ASFA) where approximation is made by abstract interpretation of symbolic finite automata, acting both at syntactic (predicate) and semantic (denotation) level. We investigate in the details how the syntactic and semantic abstractions of SFA relate to each other and contribute to the determination of the recognized language. Then we introduce a family of transformations for simplifying ASFA. We apply this

model to prove properties of commonly used tools for similarity analysis of binary executables. Following the structure of their control flow graphs, disassembled binary executables are represented as (concrete) SFA, where states are program points and predicates represent the (possibly infinite) I/O semantics of each basic block in a constraint form. Known tools for binary code analysis are viewed as specific choices of symbolic and semantic abstractions in our framework, making symbolic finite automata and their abstract interpretations a unifying model for comparing and reasoning about soundness and completeness of analyses of low level code.

15. Mizuhito Ogawa (JAIST), Constant propagation for Binary CFG rebuilt

    BE-PUM, which reconstructs a CFG from obfuscated binary, is based on dynamic symbolic execution. A common problem on symbolic execution is loop handling, and unlimited unfolding leads non-termination of CFG rebuilt. Its ultimate solution is loop invariant generation, which is usually a hard problem. As a practical solution, we apply constant propagation for the loop unrolling control. The idea comes from observation on a typical reason of failure to find the loop exit by dynamic symbolic execution. When a *loop counter* is set to a constant, dynamic symbolic execution cannot find a satisfiable instance leading to the loop exit except for unfolding the loop until the constant matches to the exit condition. If we know the loop counter is a constant, it is safe to unfold as long as the original binary terminates. We also report very preliminary experiments.

Participants list

There are 19 participants from 8 countries. Detailed classification is, France 5, Japan 4, USA 3, Switzerland / Vietnam 2, Germany / Italy / Malaysia 1. They are mostly from academia, and 3 from industry, including Google. Unfortunately, we missed participants from China, due to the budget problems. Although the number is not very high, the atmosphere has been kept intimate. We think the number was quite adequate.

- Sebastien Bardin (CEA LIST, France)
- Frederic Besson (Inria, France)
- Sandrine Blazy (University Rennes - Inria, France)
- Guillaume Bonfante (LORIA - Universite de Lorraine, France)
- Thomas Dullien (Google, Switzerland)
- Roberto Giacobazzi (University of Verona, Italy)
- Nguyen Minh Hai (Industrial University of Ho Chi Minh City, Vietnam)
- Tomonori Izumida (AIST, Japan)
- Tim Kornau-von Bock und Polach (Google, Switzerland)
- Julian Kranz (Technische Universitat Munchen, Germany)
- Arun Lakhotia (University of Louisiana at Lafayette, USA)
- Jean-Yves Marion (LORIA - Lorraine University, France; *Organizer*)
- Jeffrey Todd McDonald (University of South Alabama, USA)
- Akira Mori (National Institute of Advanced Industrial Science and Technology, Japan; *Organizer*)
- Mizuhito Ogawa (Japan Advanced Institute of Science and Technology, Japan; *Organizer*)
- Tachio Terauchi (Japan Advanced Institute of Science and Technology, Japan)
- Quan Thanh Tho (Ho Chi Minh University of Technology, Vietnam)
- Sarah Zennou (Airbus Group Innovations, USA)
- Abdullah Mohad Zin (Universiti Kebangsaan Malaysia, Malaysia)