Report from Dagstuhl Seminar 14241

Challenges in Analysing Executables: Scalability, Self-Modifying Code and Synergy

Edited by
Roberto Giacobazzi[1], Axel Simon[2], and Sarah Zennou[3]

1 **Università degli Studi di Verona, IT,** `roberto.giacobazzi@univr.it`
2 **TU München, DE,** `Axel.Simon@in.tum.de`
3 **Airbus Group Innovations-Suresnes, FR,** `sarah.zennou@eads.net`

Abstract

This report summarizes the program and the outcomes of the Dagstuhl Seminar 14241, entitled "Challenges in Analysing Executables: Scalability, Self-Modifying Code and Synergy". The seminar brought together practitioners and researchers from industry and academia to discuss the state-of-the art in the analysis of binaries, the handling of the most challenging malware and the ever-lasting problem of scalability. The meeting created new links within this very diverse community and highlighted the broad interest in dealing with obfuscated code.

Seminar June 9–13, 2014 – http://www.dagstuhl.de/14241
1998 ACM Subject Classification B.2.2 Worst-case analysis, D.2.4 Formal methods, D.3.2 Macro and assembly languages, D.3.4 Debuggers and Interpreters, D.4.5 Fault-tolerance and Verification, D.4.6 Information flow controls and Invasive software, D.4.8 Modelling and prediction, D.4.9 Linkers and Loaders, F.3.2 Operational semantics and Program analysis, I.2.2 Program modification
Keywords and phrases Executable analysis, reverse engineering, malware detection, control flow reconstruction, emulators, binary instrumentation
Digital Object Identifier 10.4230/DagRep.4.6.48
Edited in cooperation with Ed Robbins

1 Executive Summary

Axel Simon

As a follow-up on the previous Dagstuhl Seminar 12051 on the analysis of binaries, the interest in attending this new seminar was very high. In the end, less than half the people that we considered inviting could attend, namely 44 people. In contrast to the previous seminar that ran for 5 days, this seminar was a four-day seminar due to a bank holiday Monday. Having arranged the talks by topic, these four days split into two days on the analysis of binaries and into (nearly) two days on obfuscation techniques.

The challenges in the realm of general binary analysis have not changed considerably since the last gathering. However, new analysis ideas and new technologies (e. g. SMT solving) continuously advance the state-of-the-art and the presentations where a reflection thereon. With an even greater participation of people from industry, the participants could enjoy a broader view of the problems and opportunities that occur in practice. Given the tight focus on binary code (rather than e. g. Java byte code), a more detailed and informed discussion ensued. Indeed, the different groups seem to focus less on promoting their own tools rather than seeking collaboration and an exchange of experiences and approaches. In this light, the seminar met its ambition on synergy. It became clear that creating synergy by combining various tools is nothing that can be achieved in the context of a Dagstuhl Seminar. However, the collaborative mood and the interaction between various groups give hope that this will be a follow-on effect.

The second strand that crystallized during the seminar was the practical and theoretic interest in code obfuscation. Here, malware creators and analysts play an ongoing cat-and-mouse game. A theoretic understanding of the impossibility of winning the game in favor of the analysts helps the search for analyses that are effective on present-day obfuscations. In practice, a full understanding of some obfuscated code may be unobtainable, but a classification is still possible and useful. The variety of possible obfuscations creates many orthogonal directions of research. Indeed, it was suggested to hold a Dagstuhl Seminar on the sole topic of obfuscation.

One tangible outcome of the previous Dagstuhl Seminar is our GDSL toolkit that was presented by Julian Kranz. We believe that other collaborations will ensue from this Dagstuhl Seminar, as the feedback was again very positive and many and long discussions where held in the beautiful surroundings of the Dagstuhl grounds. The following abstracts therefore do not reflect on the community feeling that this seminar created. Please note that not all people who presented have submitted their abstracts due to the sensitive nature of the content and/or the organization that the participants work for.

2 Table of Contents

3 Overview of Talks

3.1 Binary-level analysis for safety-critical systems

Sebastien Bardin (CEA LIST, FR)

Joint work of Bardin, Sebastien; Védrine, Franck; Herrmann, Philippe

We present in this talk the work done at CEA LIST on binary-level analysis of safety-critical programs. Our efforts have been focused on Intermediate Languages for modelling the semantics of machine code instructions, generation of test data through dynamic symbolic execution and safe recovery of the Control-Flow Graph. We present our solutions, their limitations and a few case-studies on safety-critical programs from aeronautics and energy. We conclude by presenting some directions we want to explore and some ongoing work.

References

1 Sebastien Bardin, Philippe Baufreton, Nicolas Cornuet, Philippe Herrmann, Sebastien Labbé: Binary-Level Testing of Embedded Programs. QSIC 2013:11–20. IEEE
2 Sebastien Bardin, Philippe Herrmann: OSMOSE: automatic structural testing of executables. Softw. Test., Verif. Reliab. 21(1):29–54 (2011)
3 Sebastien Bardin, Philippe Herrmann, Jérôme Leroux, Olivier Ly, Renaud Tabary, Aymeric Vincent: The BINCOA Framework for Binary Code Analysis. CAV 2011: 165-170. Springer
4 Sebastien Bardin, Philippe Herrmann, Franck Védrine: Refinement-Based CFG Reconstruction from Unstructured Programs. VMCAI 2011:54–69. Springer

3.2 High-level semantics for low-level code

Frederic Besson (IRISA – Rennes, FR)

The "flat" memory model – where memory is as an array of bytes – is the archetype of a memory model that is not suitable for static analysis. The reason is that a local loss of precision has a dramatic (in the sense of drama) effect on the rest of the analysis. Memory models based on regions – address is an offset with respect to an abstract pointer – have proved valuable for analysing higher-level languages. For binary programs, this model is too abstract and fails to give a concrete semantics to certain programs. Borrowing ideas from symbolic execution, we show how to enhance this idealised model to capture low-level idioms. The semantics is executable and leverages SMT solvers. We have an implementation of this enhanced memory model for the CompCert C compiler. See http://www.irisa.fr/celtique/ext/csem/.

3.3 Verified Abstract Interpretation Techniques for Disassembling Low-level Self-modifying Code

Sandrine Blazy (IRISA – Rennes, FR)

Joint work of Blazy, Sandrine; Laporte, Vincent; Pichardie, David
Main reference S. Blazy, V. Laporte, D. Pichardie, "Verified Abstract Interpretation Techniques for Disassembling Low-level Self-Modifying Code," in Proc. of the 5th Int'l Conf. on Interactive Theorem Proving (ITP'14), LNCS, Vol. 8558, pp. 128–143, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08970-6_9

Static analysis of binary code is challenging for several reasons. In particular, standard static analysis techniques operate over control flow graphs, which are not available when dealing with self-modifying programs which can modify their own code at runtime. We formalize in the Coq proof assistant some key abstract interpretation techniques that automatically extract memory safety properties from binary code. Our analyzer is formally proved correct and has been run on several self-modifying challenges, provided by Cai et al. in their PLDI 2007 paper.

3.4 CopperDroid: On the Reconstruction of Android Malware Behaviors

Lorenzo Cavallaro (RHUL – London, GB)

Joint work of Fattori, Aristide; Tam, Kimberly; Khan, Salahuddin J.; Reina, Alessandro; Cavallaro, Lorenzo

Today mobile devices and their application marketplaces drive the entire economy of the mobile landscape. For instance, Android platforms alone have produced staggering revenues exceeding 5 billion USD, which unfortunately attracts cybercriminals with malware now hitting the Android markets at an alarmingly rising pace.

To better understand this slew of threats, in this talk I present CopperDroid, an automatic VMI-based dynamic analysis system to reconstruct the behavior of Android malware. Based on the key observation that all interesting behaviors are eventually expressed through system calls, CopperDroid presents a novel unified analysis able to capture both low-level OS-specific and high-level Android-specific behaviors. To this end, CopperDroid presents an automatic system call-centric analysis that faithfully reconstructs events of interests, including IPC and RPC interactions and complex Android objects, to describe the behavior of Android malware regardless of whether it is initiated from Java or native code execution. CopperDroid's analysis generates detailed behavioral profiles that abstract a large stream of low-level – sometimes uninteresting – events into concise high-level semantics, which are well-suited to provide effective insights.

Extensive evaluation on more than 2,900 Android malware samples, show that CopperDroid faithfully describes OS- and Android-specific behaviors and, through the use of a simple yet effective app stimulation technique, successfully triggers and discloses additional behaviors on more than 60 precent (on average) of the analyzed malware samples, qualitatively improving code coverage of dynamic-based analyses.

References

1 Aristide Fattori and Kimberly Tam and Salahuddin J. Khan and Alessandro Reina and Lorenzo Cavallaro. *CopperDroid: On the Reconstruction of Android Malware Behaviors.* Technical Report MA-2014-01 Royal Holloway University of London, Februrary, 2014

2 Alessandro Reina and Aristide Fattori and Lorenzo Cavallaro. *A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors.* Proceedings of the 6th European Workshop on System Security (EUROSEC), April, 2013

3.5 Practical Problems in Automated Static Analysis of Malware

Cory Cohen (Software Engineering Institute – Pittsburgh, US)

Operational malware analysts have different priorities and motivations than most academic researchers in the static analysis of binaries. This presentation will highlight some of those differences, and provide suggestions on how to promote adoption of research prototypes for operational use. It also presents some background on the current state of malware, examples of problem areas interesting to malware analysts, and discusses stack delta analysis algorithms as an example of these ideas.

3.6 Evaluating the strength of software protections

Bjorn De Sutter (Ghent University, BE)

Main reference ASPIRE EU FP7 Project – Advanced Software Protection: Integration, Research and Exploitation.
URL http://www.aspire-fp7.eu

An overview of the ASPIRE project is presented, focusing on the major goals of developing a protection tool flow, a reference architecture for protected applications, and decision support to select and apply protections. Then the challenge of modelling and evaluating the protection strength against attacks is discussed, and an overview is given of different types of evaluation metrics, ranging from software engineering metrics, compiler-based code analysis metrics, formal modelling metrics, tool-based metrics, and human experiments and human comprehension modelling. Their strong points and weak points are discussed, after which the talk concludes with a list of open challenges in the domain of protection strength evaluation.

3.7 Understanding Programs that Don't Want to be Understood

Saumya K. Debray (University of Arizona – Tucson, US)

Joint work of Yadegari, Babak; Johannesmeyer, Brian; Whitely, Benjamin

Malicious software are usually obfuscated to avoid detection and resist analysis. When new malware is encountered, such obfuscations have to be penetrated or removed ("deobfuscated") in order to understand the internal logic of the code and devise countermeasures. This talk discusses a generic approach for deobfuscation of obfuscated executable code. Our approach does not make any assumptions about the nature of the obfuscations used, but instead uses semantics-preserving program transformations to simplify away obfuscation code. We have applied a prototype implementation of our ideas to a variety of different kinds of obfuscation, including emulation-based obfuscation, emulation-based obfuscation with runtime code unpacking, and return-oriented programming. Our experimental results are encouraging and suggest that this approach can be effective in extracting the internal logic from code obfuscated using a variety of obfuscation techniques, including tools such as Themida that previous approaches could not handle.

3.8 Static analysis of avionics software at Airbus

David Delmas (Airbus S.A.S. – Toulouse, FR)

Analysis of executables rely on static analyzers based on Abstract Interpretation: aiT WCET (for time-critical applications) and Stackanalyzer (for most avionics software). Run-time error analysis is performed at source code level using Astrée. These analyzers scale up to very large avionics software, while remaining sufficiently precise for industrial use. More local analyzes rely on tools such as Caveat, Frama-C and Fluctuat. Formal verification at source code level is still valid at binary level, provided a certified compiler is used, e. g. CompCert.

Considering raising security concerns and upcoming related standards for avionics, more binary analyses will be needed. Integration of third party binaries onto a software platform requires the verification of conformance to platform interface requirements. Detection of memory vulnerabilities and CFG reconstruction are of interest to audit third party software. Also, CFI/SFI sandboxing approaches, and related automatic verification, are of interest for some software platforms. Among new challenges is the security evaluation of some non-critical equipments, using standard connected PCs/tablets, and external Java bytecode.

For the future, timing analysis of multithreaded applications running on top of embedded OS and complex processors remains also an objective. Side-channel analyzes, e. g. information leakage through caches, is of interest. At source code level, formal verification of correct implementation of a given security policy on large complex systems is a long term challenge.

The main industrial requirements for such analyses are automation, scalability and precision. In safety-related domains, soundness is paramount.

3.9 Insight: A(nother) Binary Analysis Framework

Emmanuel Fleury (University of Bordeaux, FR)

We aim to have a full and efficient platform to easily try out novel algorithms or techniques. For this, we provide a full C++ framework designed for Unix systems (*BSD, Linux, MacOS X, ...) which contains a wide-spectrum binary format loaders (ELF, PE, Mach-O, ...), a decoder translating from assembly code (i386, amd64, ...) into our intermediate language, an interpreter to execute the program over a (potentially abstract) domain and several facilities to simplify, manipulate or transform the graph and the expressions extracted from the original program.

This talk introduces the Insight framework and includes a small demonstration of our interactive symbolic debugger.

3.10 Decompilation into Logic using HOL4

Anthony Fox (University of Cambridge, GB)

Joint work of Fox, Anthony; Myreen, Magnus

We present formal ISA models and tools that support the decompilation of machine-code into HOL4 functions. This decompilation is validated through the use of certificate theorems. We provide an overview of our ISA models and list some notable case studies. The HOL4 tools are demonstrated.

3.11 Similarity analysis in Big Code of executables

Roberto Giacobazzi (University of Verona, IT)

Data-sets in huge software enclaves, such as code, specifications, analyses, etc. put forward new and unconventional challenges to traditional Big-Data research. If Big-Data requires adequate infrastructures and abstractions for mining and learning information from huge data-sets, in Big-Code we need to include interpretation in order to be able to extract and represent the extensional meaning of programs. Any Big-Code analytics is therefore necessarily based on a form of interpretation and analysis, able to mine semantics and returning approximate information about programs behavior. We introduce a mixed syntactic/semantics approximation model based on symbolic automata for similarity analysis of large enclaves of binary executables. Following the structure of their control flow graph, disassembled binary executables are represented as symbolic automata, where nodes are program points and predicates represent the semantics of each basic block. Approximation is made by abstract interpretation, acting on these symbolic automata both at syntactic and semantic level. At syntactic level, the code of basic blocks is approximated by extracting their BinJuice. At semantic level the data information is abstracted in standard numerical domains. Simplification operations of the resulting abstract symbolic automata are discussed in order to extract common signatures of similar executables.

3.12 Large Scale Concurrent ISA Semantics (via the Sail DSL)

Kathryn E. Gray (University of Cambridge, GB)

There has been much work on formal models of ISA behaviour and on domain- specific languages for expressing them, but it has not addressed the relaxed- memory concurrency of multiprocessors such as IBM Power and ARM. On the other side, recent work by Sarkar et al. has established semantics for the latter but is not integrated with a large-scale ISA model. We discuss what is necessary to combine a large-scale ISA model with a realistic concurrency semantics and our work (in progress) to that end: using our Sail language to express a Power ISA model that we semi-automatically extract from the IBM documentation.

3.13 Dynamic Analysis: Knowing When to Stop

Paul Irofti (Bucharest, RO)

I made a lot of progress on the emulator since my last talk two years ago at the 12051 Seminar "Analysis of Executables: Benefits and Challenges". It is now a mature production-ready project (see the 'Other' document for a whitepaper on it) and I want to talk about the problems I faced, focusing on one in particular which is the stopping problem.

The classic scenario is that an executable gets loaded and emulated until the executable exit by itself. But there are times when the executable takes longer to be emulated than you'd want it to or, worse yet, the emulation process gets hogged somewhere due to anti-debugging techniques or bugs in the actual program. That's why most dynamic analysis solutions in the malware industry employ some sort of watchdog-like mechanism that forces a stop in emulation after a certain threshold is reached. These solutions involve time-based or emulated instruction-based thresholds that are either non-deterministic or unfair to certain machines (be it really fast machines or older, slower ones).

And so, I want to talk about a solution that provides a deterministic and fair on all systems mechanism of stopping the emulation process.

3.14 Generic Decoder Specification Language

Julian Kranz (TU München, DE)

Joint work of Kranz, Julian; Sepp, Alexander; Simon, Axel
Main reference A. Sepp, J. Kranz, A. Simon, "GDSL: A Generic Decoder Specification Language for Interpreting Machine Language," in Proc. of the 3rd Workshop on Tools for Automatic Program Analysis (TAPAS'12), ENTCS, Vol. 289, pp. 53–64, Elsevier, 2012; pre-print available from author's webpage.
URL http://dx.doi.org/10.1016/j.entcs.2012.11.006
URL http://www2.in.tum.de/bib/files/sepp12gdsl.pdf

Analysing binary code begins with the interpretation of a low-level binary input stream. This process consists of two steps: turning the byte stream into a sequence of instructions,

i. e. giving syntax to it, and giving meaning to the instructions, i. e. translating them into some kind of semantics representation. Addressing the first step, we present our functional language called GDSL which is geared to the simple and effective specification of binary decoders. To this end, the language is equipped with special syntax that allows the easy access to slices of bytes. As a proof of its practicability, GDSL ships with a complete decoder for Intel x86 implemented in GDSL. We demonstrate the effectiveness of the GDSL compiler by comparing our decoding performance to Intel's XED decoder. Regarding the translation to semantics, we present the minimalistic language RReil which is designed to be suited for binary analysis. Using a simple optimization, we achieve a translation output size of roughly only three semantic primitives per x86 instruction.

3.15 Formal Specification and Validation of ARM Machine-Code Analyses

Alexey Loginov (GrammaTech Inc.- Ithaca, US)

We will describe the foundations of extending GrammaTech's machine-code analysis to the ARM instruction-set architecture (ISA). We will start with our approach to the creation of a trustworthy specification of ARM instruction semantics. We will then describe our efforts on validating every step in the creation of intermediate representations of ARM binaries. The goal is to construct intermediate representations that enable sound, yet precise, static analysis. Our validation covers instruction decoding, disassembly, concrete instruction semantics, as well as abstract analyses. Finally, we will describe our approach to ISA-independent regression testing of static analyses, such as Value-Set Analysis (VSA). This approach enabled rapid adaptation of existing x86-only regression tests to testing static analyses of ARM binaries.

3.16 CoDisasm :a disassembly of self-modifying binaries with overlapping instructions

Jean-Yves Marion (LORIA – Nancy, FR)

Disassembly is a key task in software debugging and malware analysis. It involves the recovery of assembly instructions from binary machine code. It can be problematic in the case of malicious code, as malware writers often employ techniques to thwart correct disassembly by standard tools. Nonetheless, disassembly is a crucial step in malware reverse engineering. Correct disassembly of binaries is necessary to produce a higher level representation of the code and thus allow the analysis to develop high-level understanding of its behavior and purpose.

In this paper, we focus on the disassembly of self-modifying binaries with overlapping instructions. Current state-of-the-art disassemblers fail to interpret these two common forms of obfuscation, causing an incorrect disassembly of large parts of the input.

We have developed a standalone disassembler called CoDisasm that implements this approach, together with a plug-in for the popular reversing engineering tool IDA called

BinViz to visualize the code waves generated by CoDisasm and to visualize overlapping instructions. Our approach substantially improves the success of disassembly when confronted with both self-modification and code overlap in analyzed binaries. Experimental results on about five hundred malware samples show that our approach correctly recovers large parts of the code. To our knowledge, no other disassembler thwarts both of these obfuscations methods together.

3.17 Worst Case Execution Time for Safty Critical Systems

Florian Martin (AbsInt – Saarbrücken, DE)

Joint work of Martin, Florian; Daniel Kästner, Markus Pister, Gernot Gebhard, Christian Ferdinand

All contemporary safety standards require to demonstrate the absence of functional and non-functional safety hazards. In real-time systems this includes demonstrating the absence of critical timing hazards.

To meet this verification objective it is necessary to show the correctness of the timing behavior with ad- equate confidence. Adequate confidence means that the evidence provided can be trusted beyond reason- able doubt. There are two main sources of doubt: the logical doubt associated with the validity of the reasoning and the epistemic doubt associated with uncertainty about the underlying assumptions. A fundamental timing property is the per-task worst-case execution (WCET). It is an ingredient for determining all higher-level timing concepts like worst-case response times, and system-wide end-to-end times. This talk gives an overview of the challenges in ensuring timeliness of real-time software focusing on the worst-case execution time problem. It describes the principles of abstract interpretation-based WCET analysis and summarizes the confidence argument for applying it in the certification process of safety-critical software, addressing both logical and epistemic doubt.

3.18 Obfuscation as Incomplete Approximation

Isabella Mastroeni (University of Verona, IT)

Main reference R. Giacobazzi, N. Jones, I. Mastroeni, "Obfuscation by Partial Evaluation of Distorted Interpretation," in Proc. of the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation (PEPM'12), pp. 63–72, ACM, 2012.
URL http://dx.doi.org/10.1145/2103746.2103761

We present a novel approach to automatically generating obfuscated code P' from any program P whose source code is given. Start with a (program-executing) interpreter *interp* for the language in which P is written. Then "distort" *interp* so it is still correct, but its specialization P' w.r.t. P is transformed code that is equivalent to the original program, but harder to understand or analyze. Potency of the obfuscator is proved with respect to a general model of the attacker, modeled as an approximate (abstract) interpreter. A systematic approach to distortion is to make program P obscure by transforming it to P' on which (abstract) interpretation is incomplete. Interpreter distortion can be done by making residual in the specialization process sufficiently many interpreter operations to defeat an attacker in extracting sensible information from transformed code. Our method is applied to: code flattening, data-type obfuscation, and opaque predicate insertion. The technique is language independent and can be exploited for designing obfuscating compilers.

3.19 Sendmail crackaddr – Static Analysis strikes back

Bogdan Mihaila (TU München, DE)

Joint work of Mihaila, Bogdan; Sepp, Alexander; Simon, Axel
Main reference A. Sepp, B. Mihaila, A. Simon, "Precise Static Analysis of Binaries by Extracting Relational Information," in Proc. of the 18th Working Conference on Reverse Engineering (WCRE'11), pp. 357–366, IEEE, 2011.
URL http://dx.doi.org/10.1109/WCRE.2011.50

The "sendmail crackaddr" bug from 2003 is an example for a vulnerability that is difficult to prove using static analysis. In the course of analyzing the simplified version of this famous example we discovered that it is surprisingly easier than expected to separate the vulnerable from the non-vulnerable example. We show that it can be solved using abstract interpretation and show what invariants are inferred by our binary analysis framework: Bindead. Though the results are promising, there is still some work necessary to apply the same methods to the original example.

References

1 B. Mihaila, A. Sepp and A. Simon. *Widening as Abstract Domain.* In G. Brat, N. Rungta and A. Venet, editors, NASA Formal Methods, volume 7871 of LNCS, pages 170–186, Moffett Field, California, USA, May 2013. Springer.
2 A. Sepp, B. Mihaila and A. Simon. *Precise Static Analysis of Binaries by Extracting Relational Information.* In M.Pinzger and D. Poshyvanyk, editors, Working Conference on Reverse Engineering, Limerick, Ireland, October 2011. IEEE Computer Society.

3.20 Through the Lens of Abstraction

Aditya Thakur (University of Wisconsin – Madison, US)

Joint work of Thakur, Aditya; Reps, Thomas

This talk explores the use of abstraction in two areas of automated reasoning: verification of programs, and decision procedures for logics.

Establishing that a program is correct is undecidable in general. Program- verification tools sidestep this tar-pit of undecidability by working on an abstraction of a program, which over-approximates the original program's behavior. The theory underlying this approach is called abstract interpretation, and is around forty years old. However, harnessing abstraction to develop a scalable and precise abstract interpreter still remains a challenging problem.

This talk also exposes the use of abstraction in the design and implementation of decision procedures. I call such an abstraction-centric view of decision procedures Satisfiability Modulo Abstraction. Abstraction provides a new language for the description of decision procedures, leading to new insights and new ways of thinking.

The common use of abstraction also brings out a non-trivial and useful relationship between program verification and decision procedures.

3.21 Turbulence – an assembly level obfuscator

Axel Tillequin (Airbus Group – Suresnes, FR)

Turbulence is an assembly level obfuscator used inside Airbus Group for IP protection. It has been developed since 2005. It seamlessly integrates into the gcc toolchain and manages to automatically determine how many obfuscating iterations are needed by reaching a fixed point on the distribution of its internal set of obfuscating transforms. Open questions are related to finding a good measure of complexity of some obfuscated code in order to provided an adaptive approach of reaching an uniform complexity by orienting the obfuscator on badly obfuscated parts.

3.22 Interactive static analysis

Franck Vedrine (CEA – Gif-sur-Yvette, FR)

Only few abstract interpreters have an interactive interface. In this talk we present the concepts behind the interactive interface of the Fluctuat static analyzer for C programs [1], and possible usages. While already useful for source-level analysis, we do think that interactivity is even more interesting for binary-level analysis, where it is not so evident for a user to define an analysis scenario or to insert annotations at a dedicated point. The implementation into CFGBuiler is future work.

References

1 Franck Vedrine, Eric Goubault, Sylvie Putot, Tristan Le Gall: *Interactive Analysis in FLUCTUAT*. Tools for Automatic Program Analysis – TAPAS 2012, Deauville, France 2012

3.23 Jackdaw: Automatic, unsupervised, scalable extraction and semantic tagging of (interesting) behaviors

Stefano Zanero (Politecnico di Milano University, IT)

Joint work of Polino, Mario; Scorti, Andrea; Maggi, Federico; Zanero, Stefano

When analyzing (malicious) software, hybrid static-dynamic program analysis techniques help the analyst in finding interesting behaviors. One of the key requirements of these methods is a catalog of patterns or specifications of such interesting behaviors, which need to be created manually.

Due to the rising number of complex malicious software and the growth of their potential, unknown yet interesting behaviors, automatic techniques are needed to build their specifications, present them to the analyst, and create a catalog of matching rules and relevant implementations (e. g., variants).

We propose Jackdaw, an automatic behavior extractor and semantic tagger. Our system first exploits jointly static control-flow analysis and dynamic data- flow analysis on malware samples to find interesting, connected sequences of API calls that are potential behaviors. Then, it maps these building blocks to their implementation(s), taking care of capturing and modeling the distinct characteristics of each variant's implementation. Finally, it associates semantic information to the behaviors, so as to create compact and descriptive summary that help the analysts in the first phases of reverse engineering. To do this, it matches relevant code against Web knowledge bases.

We tested Jackdaw on 1,272 distinct variants drawn from 17 families. We compared the behaviors extracted automatically by Jackdaw against a ground truth of 44 behaviors created manually by expert analysts: Jackdaw matched 77.3% of them. We also discover 466 novel behaviors, among which manual exploration reveals interesting findings. Manual analysis confirms also that the semantic tags that Jackdaw attaches to the behaviors are meaningful.

3.24 Binary analysis and manipulation at Airbus Group Innovations

Sarah Zennou (Airbus Group – Suresnes, FR)

Joint work of Zennou, Sarah; Biondi, Philippe; Mehrenberger, Xavier; Tillequin, Axel

This talk presents research activities at the research center of Airbus Group that are linked to the topics of the seminar: malware classification, scalable static analyses and obfuscation.

Participants

- Davide Balzarotti
EURECOM – Biot, FR
- Sébastien Bardin
CEA LIST, FR
- Frederic Besson
IRISA – Rennes, FR
- Sandrine Blazy
IRISA – Rennes, FR
- Juan Caballero
IMDEA Software Institute – Madrid, ES
- Lorenzo Cavallaro
RHUL – London, GB
- Aziem Chawdhary
University of Kent, GB
- Cory Cohen
Software Engineering Institute – Pittsburgh, US
- Mila Dalla Preda
University of Verona, IT
- Bjorn De Sutter
Ghent University, BE
- Saumya K. Debray
Univ. of Arizona – Tucson, US
- David Delmas
Airbus S.A.S. – Toulouse, FR
- Thomas Dullien
Google Switzerland, CH
- Emmanuel Fleury
University of Bordeaux, FR
- Anthony Fox
University of Cambridge, GB
- Roberto Giacobazzi
University of Verona, IT
- Kathryn E. Gray
University of Cambridge, GB
- Paul Irofti
Bucharest, RO
- Yan Ivnitskiy
Trail of Bits Inc. – New York, US
- Andy M. King
University of Kent, GB
- Tim Kornau
Google Switzerland, CH
- Julian Kranz
TU München, DE
- Colas Le Guernic
Direction Generale de l'Armement, FR
- Junghee Lim
GrammaTech Inc.- Ithaca, US
- Alexey Loginov
GrammaTech Inc.- Ithaca, US
- Federico Maggi
Politecnico di Milano Univ., IT
- Jean-Yves Marion
LORIA – Nancy, FR
- Florian Martin
AbsInt – Saarbrücken, DE
- Isabella Mastroeni
University of Verona, IT
- Bogdan Mihaila
TU München, DE
- Magnus Myreen
University of Cambridge, GB
- Gerald Point
University of Bordeaux, FR
- Edward Robbins
University of Kent, GB
- Bastian Schlich
ABB AG Forschungszentrum Deutschland – Ladenburg, DE
- Alexander Sepp
TU München, DE
- Axel Simon
TU München, DE
- Aditya Thakur
University of Wisconsin – Madison, US
- Axel Tillequin
Airbus Group – Suresnes, FR
- Franck Védrine
CEA – Gif-sur-Yvette, FR
- Aymeric Vincent
University of Bordeaux, FR
- Xueguang Wu
TU München, DE
- Brecht Wyseur
NAGRA Kudelski Group SA – Cheseaux, CH
- Stefano Zanero
Politecnico di Milano Univ., IT
- Sarah Zennou
Airbus Group – Suresnes, FR