Report from Dagstuhl Seminar 19331

Software Protection Decision Support and Evaluation Methodologies

Edited by

Bjorn De Sutter[1], Christian Collberg[2], Mila Dalla Preda[3], and Brecht Wyseur[4]

1 **Ghent University, BE,** bjorn.desutter@ugent.be
2 **University of Arizona – Tucson, US,** collberg@cs.arizona.edu
3 **University of Verona, IT,** mila.dallapreda@univr.it
4 **Kudelski Group SA – Cheseaux, CH,** brecht.wyseur@nagra.com

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 19331 "Software Protection Decision Support and Evaluation Methodologies". The seminar is situated in the domain of software protection against so-called man-at-the-end attacks, in which attackers have white-box access to the software that embeds valuable assets with security requirements such as confidentiality and integrity. The attackers try to compromise those by reverse-engineering the software and by tampering with it. Within this domain, the seminar focused mainly on three aspects: 1) how to evaluate newly proposed protections and attackers thereon; 2) how to create an appropriate benchmark suite to be used in such evaluations; 3) how to build decision support to aid users of protection tool with the selection of appropriate protections. The major outcomes are a structure for a white-paper on software protection evaluation methodologies, with some concrete input collected on the basis of four case studies explored during the seminar, and a plan for creating a software protection benchmark suite.

Seminar August 11–16, 2019 – http://www.dagstuhl.de/19331
2012 ACM Subject Classification Security and privacy → Software and application security
Keywords and phrases Benchmarks, Decision Support Systems, Evaluation Methodology, man-at-the-end attacks, metrics, predictive models, reverse engineering and tampering, software protection
Digital Object Identifier 10.4230/DagRep.9.8.1

1 Executive Summary

Christian Collberg
Mila Dalla Preda
Bjorn De Sutter
Brecht Wyseur

Overview and Motivation

The area of Man-At-The-End (MATE) software protection is an evolving battlefield on which attackers execute white-box attacks: They control the devices and environments and use a range of tools to inspect, analyze, and alter software and its assets. Their tools include disassemblers, code browsers, debuggers, emulators, instrumentation tools, fuzzers, symbolic execution engines, customized OS features, pattern matchers, etc.

To meet the security requirements of assets embedded in software, i.e., valuable data and code, many protections need to be composed. Those requirements include the confidentially of secret keys and software IP (novel algorithms, novel deep learning models, ...), and the integrity of license checking code and anti-copy protections. Attackers attack them through reverse engineering and tampering, for which they use the aforementioned tools and for which they often can afford spending time and effort on executing many, highly complex and time-consuming, manual and automated analyses. The need for composing many protections follows from the fact that advanced attackers can use all the mentioned tools and try many different approaches. In other words, to be effective, the deployed protections need to protect against all possible attack vectors.

As all protections come with overhead, and as many of them have downsides that complicate various aspects of the software development life cycle (SDLC), the users of a software protection tool cannot simply deploy all available protections. Instead, they have to select the protections and their parameters for every single asset in a program, taking into account non-functional requirements for the whole program and its SDLC.

The organizers of this workshop, and many experts in their network, consider the lack of automated decision support for selecting the best protections, and the lack of a generally accepted, broadly applicable methodology to evaluate and quantify the strength of a selected combination, the biggest challenges in the domain of software protection. As a result, the deployment of software protection is most often not trustworthy, error-prone, not measurable, and extremely expensive because experts are needed and they need a lot of time, increasing the time to market.

This situation is becoming ever more problematic. For example, connected intelligent vehicles are quickly being deployed in the market now and autonomous vehicles are going to be deployed in 3-5 years. Software protection evaluation and measurement research and development must match up that pace to provide enough technology support for controllable and scientific methods to manage the quality of automotive security as key part of vehicle reliability and safety. There is hence a huge need to make progress w.r.t. software protection decision support and evaluation methodologies, the topic of the proposed seminar.

Goals of the Seminar

Following a pre-seminar survey among the registered participants to focus the seminar and to select the highest priority objectives among the many possible ones, the primary goal of the seminar was determined to be the foundations of a white paper on software protection evaluation methodologies, to be used as a best practices guideline by researchers and practitioners when they evaluate (combinations of) defensive and/or offensive techniques in the domain of MATE software protection. This can also serve as a guideline to reviewers of submitted journal and conference papers in which novel techniques are proposed and evaluated. A secondary goal was the establishment of good benchmarking practices, including the choice of suitable benchmarks and the selection and generation thereof for use in future research in MATE software protection. A third goal was to collect feedback and ideas on how to push the state of the art in decision support systems.

Week Overview

Preparation

Prior to the seminar, the organizers set up a survey to collect the necessary information for a seminar bundle that provided background information about and to all participants. Moreover, they collected information regarding the potential outcomes that participants were most interested in, to which ones they could likely contribute, and which potential outcomes they considered most likely to make progress on. Furthermore, a reading list was presented to the participants with the goal of getting everyone on the same page as much and as soon as possible [1–8].

Whereas the schedule for the first two days was mostly fixed a priori, the schedule for later days was more dynamic, as it was adapted to the feedback obtained by the organizers during the early days, and to the outcomes of different sessions.

Monday

The first day was devoted to setting the scope of the seminar, and clarifying the seminar goals, strategy, and plan. In the morning, three overviews were presented of man-at-the-end software protection techniques in the scope of the seminar, as well as some attacks on them. These presentations focused on obfuscation vs. static analysis, (anti-)tampering in online games, and additional protections beyond the ones discussed in the first two presentations.

In the early afternoon, four deeper technical introductions were presented of four more concrete classes of defensive and corresponding offensive techniques that would serve as case studies throughout the seminar: 1) virtual machine obfuscation, 2) (anti-)disassembly, 3) trace semantics based attacks, and 4) data obfuscation. The strategy for the week was to brainstorm about these concrete techniques first, in particular on how the strength of these techniques are supposed to be evaluated, e.g., in papers that present novel (combinations of) techniques, or in penetration tests. Later, the concrete results for the individual case studies would then be generalized into best practices and guidelines for software protection evaluation methodologies.

Whereas the morning presentations and most of the case studies focused mostly on defensive techniques, three presentations in the afternoon provided complementary insights about offensive techniques, ranging from more academic semantics-based attack techniques, over an industrial case study of deobfuscation of compile-time obfuscation, and offensive techniques in binary analysis.

Thus, the scene was set in terms of both defensive and offensive techniques, and all participants to a large degree spoke the same language before starting the brainstorm sessions in the rest of the week.

Tuesday

Tuesday focused mostly on the seminar track of software protection evaluation methodologies.

In the early morning, additional input was provided on existing, already studied aspects relevant to such methodologies. This included software protection metrics, empirical experiments to assess protections, and security economics. These presentations provided useful hooks for the next session, which consisted of parallel, small break-out brainstorm sessions (three groups per case study) on the first two case studies. In these brainstorm sessions, the goal was to provide answers to questions such as the following:

- What would a document similar to the SIGPLAN empirical evaluation checklist look like for papers presenting new VM-based protections?
- Which requirements or recommendations can we put forward with respect to the protected objects (i.e., benchmarks) and their treatment (i.e., how they are created, compiled, ...) for the evaluation?
- What aspects of the attack models and which assumptions should be made explicit, which ones should be justified, e.g., regarding attacker goals and attacker activities.
- How should sensitivity to different inputs (e.g., random generator seeds, configuration options, features of code samples, ...) be evaluated and discussed?
- What threats to validity should be discussed?
- What aspects of the protection should be evaluated (potency, resilience, learnability, usability, stealth, renewability, different forms of costs, ...)?
- Under what conditions would you consider the protection to be "real world" applicable?
- What flaws (e.g., unrealistic assumptions) have you seen in existing papers that should be avoided?
- What are (minimal) requirements / recommendations regarding reproducibility?
- What pitfalls can you list that we should share with people?

After the independent brainstorms in small groups and following lunch, the three groups per case study came together to merge the results of their brainstorms, after which the merged results were shared in a plenary session.

Later in the afternoon, additional ideas were presented on topics relevant for software protection evaluation methodologies. The covered topics were benchmark generation, security activities in protected software product life cycles, the resilience of software integrity protection (work in progress), and a (unified) measure theory for potency. These topics were presented after the initial brainstorms not to bias those brainstorms. Their nature was more forward looking, covering a number of open challenges as well as potential directions for future research. They offered the speakers a sound board to get feedback and could serve as the starting point of informal discussions later in the seminar.

While the practice is discouraged by the Dagstuhl administration, we still decided to organize an evening session on Tuesday. Afterwards, we realized that this made the seminar a bit too dense, but it did serve the useful purpose of introducing the participants to the seminar track on decision support tools for software protection early enough in the seminar to allow enough time for informal discussions with and between researchers active on this topic during the remainder of the week. This was especially useful to allow those academic researchers to check the validity of some of their assumptions about real-world aspects with the present practitioners from industry and with researchers from other domains.

Besides an overview of an existing design and implementation of a software protection decision support system, a hands-on walk through of a practical attack on a virtual machine protection (as in one of the case studies) was presented, as well as some ideas to make such protection stronger.

Wednesday

Early on Wednesday morning, the focus shifted towards decision support tools, with three presentations by practitioners in companies that provide software protection solutions. These presentations focused on the support they provide to help their customers use their tools.

Later in the morning, case studies 3 and 4 were discussed in another round of parallel, small group break-out brainstorm sessions.

In the afternoon, the social outing took place, which consisted of a visit to Trier and a wine tasting at a winery where we also had dinner.[1]

Thursday

On Thursday morning, another round of break-out sessions was organized to structure the outcomes of the first round. Based on inputs collected during the first three days, the organizers drafted a structure for a white paper on software protection methodologies. In 4 parallel sessions, the participants brainstormed on how to fit the results of the first round (i.e., bullet points with concrete guidelines and considerations for each case study) into that structure, and which parts of those results could be generalized beyond the individual case studies. In a plenary session, the results of these break-outs were then presented.

In addition, the specific topic of benchmarking was discussed, focusing on questions regarding the required features of benchmarks (e.g., should or should they not contain actual security-sensitive assets) as well as potential strategies to get from the situation today, in which very few benchmarks used in papers are available for reproducing the results, to a situation in which a standard set of benchmarks is available and effectively used in studies.

In the afternoon, several demonstrations of practical tools were given, including the already mentioned decision support system of which the concepts had been presented on Tuesday evening and the Binary Ninja disassembler that is rapidly gaining popularity. Two presentations were also given on usable security and challenges and capabilities of modern static analysis of obfuscated code. There provided additional insights useful for both designers of decision support tools and evaluation methodologies.

Friday

The last morning started off with a potpourri of interesting topics that did not fit well in the main tracks of general evaluation methodologies and decision support on the one hand, and benchmarking on the other. Given the availability of many experts in the domain of software protection, we decided that everyone that wanted to launch new ideas or collect feedback on them in the broad domain of the seminar should have that chance. So the day started with short presentations on the protection of machine learning as a specific new type of application, on security levels for white-box cryptography, and on hardware/software binding using DRAM.

Later in the morning, the seminar was wrapped up with a discussion of the outcomes so far, and an agreement on plans to continue the work on the software protection evaluation methodology white paper and the assembly of a benchmark collection.

References

1 S. Schrittwieser, S. Katzenbeisser, J.Kinder, G. Merzdovnik, and E. Weippl: Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Comput. Surv.*, **49**(1), 2016.

2 M. Ceccato, P. Tonella P, C. Basile, P. Falcarin, M. Torchiano, B. Coppens, and B. De Sutter: Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. *Empirical Software Engineering* 2018; **24**(1):240–286.

[1] For some reason, most of us don't remember the rest of the evening in enough detail to report on it reliably.

3 B. Cataldo, D. Canavese, L. Regano, P. Falcarin, and B. De Sutter: A Meta-model for Software Protections and Reverse Engineering Attacks. *Journal of Systems and Software* 150 (April): 3–21, 2019

4 B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray: A generic approach to automatic deobfuscation of executable code. In: *Proc. IEEE Symposium on Security and Privacy*, pp. 674–691 (2015)

5 T. Blazytko, M. Contag, C. Aschermann, and T. Holz: Syntia: synthesizing the semantics of obfuscated code. *Proc. of the 26th USENIX Security Symposium (SEC'17)*, pp. 643–659. 2017

6 S. Banescu, C. Collberg, and A. Pretschner: Predicting the Resilience of Obfuscated Code Against Symbolic Execution Attacks via Machine Learning. *Proc. of the 26th USENIX Conference on Security Symposium (SEC'17)*, pp. 661-678, 2017

7 C. Basile et al.: D5.11 ASPIRE Framework Report. Technical Report ASPIRE project. https://aspire-fp7.eu/sites/default/files/D5.11-ASPIRE-Framework-Report.pdf

8 M. Ceccato et al.: D4.06 ASPIRE Security Evaluation Methodology – Security Evaluation. Technical Report ASPIRE project. https://aspire-fp7.eu/sites/default/files/D4.06-ASPIRE-Security-Evaluation-Methodology.pdf

2 Table of Contents

3 Overview of Talks

3.1 On the resilience of software integrity protection techniques (work in progress)

Mohsen Ahmadvand (TU München, DE)

In this talk we present our ongoing work on a methodology for measuring the resilience of software integrity protection techniques. Our methodology is comprised of catalogs of attacks, defences, and metrics. Attacks aid attackers to detect and/or to disable protections. Defences, on the other hand, hinder specific attacks and hence raise the bar. Metrics aim to capture the effectiveness of attacks or defences. We use a combination of empirical and analytical evaluations to measure the difficulty that is added by different combination of defences against attacks. Lastly, we present some preliminary results of machine learning based attacks on different composition of protections.

3.2 Automated Deobfuscation: A Tour on Semantic Attacks

Sébastien Bardin (CEA LIST, FR)

Joint work of Sébastien Bardin, Richard Bonichon, Jean-Yves Marion

MATE attacks aim at taking advantage of a program once access to its executable code is granted. Typical goals include stealing critical assets (e.g., cryptographic keys or proprietary code) or software tampering (e.g., bypassing security checks). Obfuscation aims at defending against such attacks by turning the initial program into a very-hard-to-understand equivalent code. Obfuscation has thus become highly important in IP protection, leading to a arm race between obfuscation and deobfuscation techniques.

Recently, semantic analysis coming from source-level safety analysis have been proven to be highly efficient against standard code protections, leading Schrittwieser et al. asking whether "Obfuscation can keep pace with progress in code analysis". Notably, Symbolic Execution combines both the standard advantages of semantic methods (automatic inference of values and triggers) with the robustness of dynamic analysis (allowing to bypass advanced protections such as packing and self-modification).

In this talk, we will review recent advances in automated semantic deobfuscation, with a special emphasis on Symbolic Execution and SMT solvers, together with an overview of their strengths, limitations and potential mitigation.

References

1 S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner. Code obfuscation against symbolic execution attacks. In *Annual Conference on Computer Security Applications, ACSAC 2016*, 2016.
2 Sébastien Bardin, Robin David, and Jean-Yves Marion. Backward-bounded DSE: targeting infeasibility questions on obfuscated codes. In *2017 IEEE Symposium on Security and Privacy, SP*, 2017.

3 D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin. Automatically identifying trigger-based behavior in malware. In Wenke Lee, Cliff Wang, and David Dagon, editors, *Botnet Detection: Countering the Largest Security Threat*, volume 36 of *Advances in Information Security*, pages 65–88. Springer, 2008.

4 J. Salwan, S. Bardin, and M.-L. Potet. Symbolic deobfuscation: from virtualized code back to the original. In *5th Conference on Detection of Intrusions and malware & Vulnerability Assessment (DIMVA)*, 2018.

5 S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl. Protecting software through obfuscation: Can it keep pace with progress in code analysis? *ACM Comput. Surv.*, 49(1), 2016.

6 B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray. A generic approach to automatic deobfuscation of executable code. In *Symposium on Security and Privacy, SP*, 2015.

3.3 An Expert System for Software Protection

Cataldo Basile (Polytechnic University of Torino, IT)

This presentation presents the current status of the Decision Support System for Software Protection developed during the EC-funded ASPIRE project and now maintained by the Security Group of the Politecnico di Torino. Moreover, we present open issues and several hints for new research and collaborations to be discussed during this seminar. In addition to the presentation that discusses concepts of the decision support system, a live demonstration was presented as well.

3.4 Hardening VM Semantics

Tim Blazytko (Ruhr-Universität Bochum, DE) and Moritz Contag (Ruhr-Universität Bochum, DE)

We discuss limitations of current VM-based obfuscation schemes and introduces automated attacks that reveal the core semantics of VM instructions. Afterwards, we propose hardening techniques which defeat the latter.

3.5 An Introduction to Security Economics

Richard Clayton (University of Cambridge, GB)

This talk gave a very brief overview of the field of security economics as it has evolved over the past twenty years. Technical analysis of security failures allows us to work out which part of a system failed; security economics helps us understand why the system was built that

way in the first place – and hence how we can redesign it to be more resilient in the future. The key economic ideas are: Incentives and Liability – if Alice is being protected by Bob, then Bob will be far more motivated if he loses out, rather than Alice, should the system fail. Externalities and Negative externalities – does a system naturally move towards a monoculture; are you dumping costs onto other people? Moral Hazard – are you encouraging bad behaviour? Asymmetric Information – Akerlof's "Market for Lemons" applies to security solutions just as much as to the market in used cars. Conflict Theory can be explained in relation to the Island of Anarchia whose flood defences are as good as the "least efforts" of the laziest family building their section of the sea wall; their ability to repel the Athenian Navy depends on the skill of their "best shot", but their trade surplus as a group will depend on "sum of efforts". This leads to an insight into software development – security depends on the worst effort of the sloppiest programmer (who writes a buffer overflow), on the best efforts of the security architect (so hire the best you can afford) and the sum of efforts of the testers (the more testing you do, the fewer bugs you should ship). The talk finished with some observations from a Security Economics perspective on CAPTCHAs, which have been broken by simple "hacks" rather than by advances in AI or graphics; on a Connect 4 competition – where it was realised that the aim was to win the competition rather than to play excellent Connect 4; and finally the story of "DVD Jon" whose DeCSS program "broke" the DVD Content Scrambling System at the end of the last century.

3.6 Introduction to the virtual machine obfuscation case study

Christian Collberg (University of Arizona – Tucson, US)

In this talk I will give an overview of obfuscation by virtual machine generation. To virtualize a function F in order to protect some asset A, we 1) create a unique and random virtual instruction set I specific to F; 2) translate the F into the virtual instruction set I (the bytecode array); and 3) construct an interpreter that can execute programs written in I. This interpreter consists of an execution stack, a dispatch unit which issues the next instruction, and one instruction handler per virtual instruction. We will discuss numerous ways to attack a virtual machine by reverse engineering the instruction set, the dispatch, or the instruction handlers. We will further discuss ways to protect the virtual machine against such attacks using diversification of the instruction set, obfuscating instruction handlers and dispatch units, or turning parts of virtual machines into dynamically generated code.

3.7 Software Protection Benchmark Generation

Christian Collberg (University of Arizona – Tucson, US)

Researchers in software protection and malware analysis face a similar problem: what programs should they test their techniques on? Often, two different papers, solving similar problems, will perform evaluation on vastly different sets of benchmark programs. Hence, it becomes difficult to compare their results. Sometimes, researchers use performance

benchmarks as security benchmarks. This is problematic, since they were not designed with security in mind. For example, there is no specific "asset" to protect and hence no clear meaning of when an attack has succeeded. In this talk we will discuss a how to automatically generate security benchmark suites. The idea is that, to evaluate a new protection idea we should 1) generate random benchmark programs, 2) run those through the protection tool, and 3) attack these randomly generated challenges through a choice of reverse engineering tools, Generating random programs turns out to be a challenging problem. A random program P should, at minimum, fulfill the following requirements. P may contain different types of assets; P should have simple I/O behavior, making it easy to automate attacks; P should have "interesting" internal structure; P should terminate within a time bound; P should not be guessable from its I/O behavior; P should resemble a real program; Finally, it should be obvious when an attack has succeeded. In this talk we will present two types of random program generators we have constructed: namely generators of random hash functions, and generators of random CAPTCHA programs.

3.8 Introduction to the (anti-)disassembly case study

Bart Coppens (Ghent University, BE)

This talk has the purpose of bringing the audience to the same level of background knowledge and terminology with regards to disassembly and anti-disassembly of binary programs. I start with techniques for unobfuscated programs. First, I talk about the very basic difference between linear sweep and recursive descent disassembly techniques. Then I explain how the resulting disassembly can be leveraged to reconstruct the original programs at ever-higher levels of abstractions. In particular, I talk about control flow recovery and how the quality of the disassembly can influence the resulting control flow graph. Next, I talk about anti-disassembly techniques, and how these can influence both the quality of the resulting disassembly as well as the quality of the reconstructed control flow graph. Finally, I talk about some advanced disassembly techniques whose purpose it is to deal correctly with binaries that have had such anti-disassembly techniques applied to them.

3.9 Securing workflows for industrial Use Cases

Jorge R. Cuéllar (Siemens AG – München, DE)

In industrial Use Cases it is often more important to secure the integrity of the process (manufacturing, testing, collaborative design, cloud applications, Industrial IoT-based Control Systems, Supply Chain, etc) than the confidentiality of the workflow itself. Different users (employees of different companies, notification bodies, governmental stakeholders or NGOs) collaborate, using smart devices like handhelds, to execute a workflow in a predefined form. The proposal combines the use of Petri-Nets for modelling the workflows and a logic (divided into 4 different layers: PKI, Trust reasoning, "Snippet"-reasoning, Accountability) that can be used to exchange information (tokens, similar to ACE/OAUTH tokens) and to reason locally

about the tokens in order to secure the integrity of the workflow. The PKI layer determines which certificates to verify for which secrets or public keys, the tust lacer determines which parties may claim which assertions, the snippet layer verifies the single transactions of the Petri Net and the accountability layer provides a method for a judge to find which server is responsible for an incorrect decision.

3.10 Introduction to the data obfuscation case study

Mila Dalla Preda (University of Verona, IT)

Data are important components of programs and their values, evolution and structure provides important information for program understanding. With the term data obfuscation we refer to those protection techniques that target data. In particular, data obfuscation techniques often modify the encoding of data in order to prevent direct analysis and hide the content of data. In this introduction, we presented a short overview to ensure that all participants had a basic understanding of the subject.

3.11 Empirical Software Protection Experiments

Bjorn De Sutter (Ghent University, BE)

We present an opverview of goals, pitfalls, issues and best practices in empirical experiments for determining the strength of software protections. This includes the technical aspects such as which protections are combined in the treatment of the objects, which tasks are given to the subjects, but also methodological aspects such as learning effect testing, statistical methods, threats to validity, etc.

3.12 Extra protections and attack in seminar scope

Bjorn De Sutter (Ghent University, BE)

We present an overview of attacks and protections in the scope of the seminar to complement the first two talks. This includes some of the tools that attackers use such as disassemblers, emulators, and debuggers, as well as a short overview of automated methods for deobfuscation. It also includes protections against all kinds of attacker-activities, such as anti-tampering, anti-debugging, anti-taint-tracking, etc.

3.13 Introduction to the trace-semantics-based attack case study

Bjorn De Sutter (Ghent University, BE)

As an introduction to the breakout sessions on evaluation techniques, case study 3 comprises two attack techniques: generic deobfuscation by Yadegari et al. and Syntia by Blazytko et al. An overview of these techniques is presented, and some potential issues with the evaluation are enumerated.

References

1 B. Yadegari, B. Johannesmeyer, B. Whitely and S. Debray. *A Generic Approach to Automatic Deobfuscation of Executable Code*. 2015 IEEE Symposium on Security and Privacy, San Jose, CA, 2015, pp. 674-691.

2 Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Thorsten Holz. *Syntia: synthesizing the semantics of obfuscated code*. In Proceedings of the 26th USENIX Conference on Security Symposium (SEC'17), Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, Berkeley, CA, USA, pp. 643-659.

3.14 Metrics for Software Protection Evaluation

Bjorn De Sutter (Ghent University, BE)

We present an overview of opportunities and challenges for using quantitative metrics for evaluation of software protections. This includes a discussion of some existing metrics from the domain of software engineering, pitfalls in using them on protected software. We also discuss the relation between attacker effort of individual steps of an attack path and features such as potency and resilience.

3.15 A (unified) measure theory for potency?

Roberto Giacobazzi (University of Verona, IT)

We observe that there exists a potentially infinite set of metrics for measuring the potency of code obfuscation. Any code attack can be encoded into an(abstract) interpreter-based attack. Because there are infinitely many interpreters, this implies that there are infinitely many metrics, one for each attack. This means that looking at standard SW engineering-like metrics is clueless in this field. All these metrics have anyway some common aspects: They are not measure of complexity in the sense of Blum's and they all try to measure the level of uncertainty that an attacker (i.e., an abstract interpreter) gets out of the performed analysis. I think that we should shift the measure of potency from SW engineering-like metrics to entropy. The presentation shows the main challenges in this direction.

3.16 Security Activities in Protected SW Product Life Cycle

Yuan Xiang Gu (Irdeto – Ottawa, CA)

This talk is to aim a better understanding what the best industrial security practices may be looking for from this seminar. First, we discuss three aspects of economics of security for a protected SW product:
1) Challenges to design a secure System
2) Do security right at early stage
3) SW security debt

And then, we clarify 9 kinds of security activities during a protected SW product life cycle from early stages of requirements, architecture design and implementation design, to code stages of implementation, testing and assurance, to the post stages of deployment, monitoring, update and renew. By these discussions, it is very clear that during different development and maintenance stages of a protected SW product, security activities require different kinds of security guidelines and evaluation approaches and supports to make right decision. Also, based on our experience from our own practices in past more than 20 years, we present some real constrains which security technologies, methods and approaches should be compliant to and suitable for real adaption and uses.

3.17 Security Problems of AI/ML Applications

Yuan Xiang Gu (Irdeto – Ottawa, CA), Mila Dalla Preda (University of Verona, IT), and Roberto Giacobazzi (University of Verona, IT)

This presentation is to introduce a new research subject on AI/ML security. AI/ML technology is getting much more adaptions for many applications in past 10 years. Recently, more and more high-stake applications start to adapt AI/ML as well. There are a couple of driving forces for AI/ML's recent success, but security is not a real important play factor yet. On the other hand, researchers are focusing on very narrow AI/ML security on the adversarial ML problem that is a special and serious issue. Our recently research shows that AI/ML security has much broader security scope well beyond adversarial ML problem. Moreover, we suggest that software protection can be adapted to address many security problems of AI/ML application systems. This talk just gives some highlights of our findings and would like to raise awareness by sharing them with a list of open questions and suggestions for how to move this new research forward.

3.18 (State of) The Art of War: Offensive Techniques in Binary Analysis

Christophe Hauser (USC – Marina del Rey, US)

Joint work of Christophe Hauser, Audrey Dutcher, Siji Feng, John Grosen, Christopher Kruegel, Mario Polino, Christopher Salls, Yan Shoshitaishvili, Nick Stephens, Giovanni Vigna, and Ruoyu Wang
Main reference Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Krügel, Giovanni Vigna: "SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis", in Proc. of the IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pp. 138–157, IEEE Computer Society, 2016.
URL https://doi.org/10.1109/SP.2016.17

Finding and exploiting vulnerabilities in binary code is a challenging task. The lack of high-level, semantically rich information about data structures and control constructs makes the analysis of program properties harder to scale. However, the importance of binary analysis is on the rise. In many situations binary analysis is the only possible way to prove (or disprove) properties about the code that is actually executed. In this paper, we present a binary analysis framework that implements a number of analysis techniques that have been proposed in the past. We present a systematized implementation of these techniques, which allows other researchers to compose them and develop new approaches. In addition, the implementation of these techniques in a unifying framework allows for the direct comparison of these approaches and the identification of their advantages and disadvantages. The evaluation included in this paper is performed using a recent dataset created by DARPA for evaluating the effectiveness of binary vulnerability analysis techniques. Our framework has been open-sourced and is available to the security community.

3.19 Hardware / Software Binding Using DRAM PUFs

Stefan Katzenbeisser (Universität Passau, DE)

Joint work of Stefan Katzenbeisser, Wenjie Xiong, Andre Schaller, Jakub Szefer

Low-end computing devices are becoming increasingly ubiquitous, especially due to the widespread deployment of Internet-of-Things products. There is, however, much concern about sensitive data being processed on these low-end devices which have limited protection mechanisms in place. This paper proposes a Hardware-Entangled Software Protection (HESP) scheme that leverages hardware features to protect software code from malicious modification before or during run-time. It also enables implicit hardware authentication. Thus, the software will execute correctly only on an authorized device and if the timing of the software, e.g., control flow, was not changed through malicious modifications. The proposed ideas are based on the new concept of Dynamic Physically Unclonable Functions (PUFs). Dynamic PUFs have time-varying responses and can be used to tie the software execution to the timing of software and the physical properties of a hardware device. It is further combined with existing approaches for code self-checksumming, software obfuscation, and call graph and register value scrambling to create the HESP scheme. HESP is demonstrated on commodity, off-the-shelf computing devices, where a DRAM PUF is used as an instance of a Dynamic PUF.

3.20 Decision processes @ Guardsquare

Eric Lafortune (Guardsquare – Leuven, BE)

Guardsquare develops software to protect mobile apps against reverse engineering and tampering. Its users are engineers who need to integrate and configure the software to process their apps, and then evaluate the results. The current approach is driven by the technology. Required configuration to make sure processed Android apps continue to work is facilitated by the widespread use of our open-source software ProGuard. Configuration to actually harden the apps follows the same conventions. Based on our experience, configuration to harden iOS apps works at a higher declarative level. External penetration testers typically provide feedback.

3.21 Binary Ninja Demonstration

Peter Lafosse (Vector 35 – Melbourne, US)

A hands-on demonstration was given of the disassembler tool Binary Ninja, a tool growing in popularity for reverse engineering of binaries.

3.22 Usable Security

Katharina Pfeffer (SBA Research – Wien, DE)

Usable security aims to investigate how IT systems can be designed so that end-users and software developers use them correctly and the attack surface is minimized. In this talk we present 2 research projects on usable security and discuss how the insights gained and the methodology used can help defending MATE attacks and develop appropriate metrics for prevention evaluation.

3.23 Case Study in Deobfuscation: Compile-Time Obfuscation

Rolf Rolles (Mobius Strip Reverse Engineering – San Francisco, US)

Software obfuscation has always been a controversially discussed research area. While theoretical results indicate that provably secure obfuscation in general is impossible, its widespread application in malware and commercial software shows that it is nevertheless popular in practice. Still, it remains largely unexplored towhat extent today's software

obfuscations keep up with state-of-the-art code analysis, and where we stand inthe arms race between software developers and code analysts. The main goal of this survey is to analyze the effectiveness of different classes of software obfuscation against the continuously improving de-obfuscation techniques and off-the-shelf code analysis tools. The answer very much depends on the goals of the analyst and the available resources. On the one hand, many forms of lightweight static analysis have difficulties with even basic obfuscation schemes, which explains the unbroken popularity of obfuscation among malware writers. On the other hand, more expensive analysis techniques, in particular when used interactively by a human analyst, can easily defeat many obfuscations. As a result, software obfuscation for the purpose of intellectual property protection remains highly challenging.

3.24 Protecting software through obfuscation: Can it keep pace with progress in code analysis?

Sebastian Schrittwieser (FH – St. Pölten, AT), Stefan Katzenbeisser (Universität Passau, DE), Johannes Kinder, Georg Merzdovnik, and Edgar Weippl

Main reference Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, Edgar R. Weippl: "Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?", ACM Comput. Surv., Vol. 49(1), pp. 4:1–4:37, 2016.
URL https://doi.org/10.1145/2886012

Software obfuscation has always been a controversially discussed research area. While theoretical results indicate that provably secure obfuscation in general is impossible, its widespread application in malware and commercial software shows that it is nevertheless popular in practice. Still, it remains largely unexplored towhat extent today's software obfuscations keep up with state-of-the-art code analysis, and where we stand inthe arms race between software developers and code analysts. The main goal of this survey is to analyze the effectiveness of different classes of software obfuscation against the continuously improving de-obfuscation techniques and off-the-shelf code analysis tools. The answer very much depends on the goals of the analyst and the available resources. On the one hand, many forms of lightweight static analysis have difficulties with even basic obfuscation schemes, which explains the unbroken popularity of obfuscation among malware writers. On the other hand, more expensive analysis techniques, in particular when used interactively by a human analyst, can easily defeat many obfuscations. As a result, software obfuscation for the purpose of intellectual property protection remains highly challenging.

3.25 Software Protection, Cloakware Style

Bahman Sistany (Irdeto – Ottawa, CA)

In this talk, we introduce Irdeto's Cloakware Software Protection and go into some detail about the Obfuscation engine and various obfuscations and transformations that it offers. We present a summary of how guidance is given to users on the use of Irdeto's Software

Protection to protect their assets. We discuss why there is a need to rank code/data entities in terms of level/kind of protection needed and how performance and size considerations are taken into account. We also cover how heuristics may be used to identify and rank entities as low and high security and how we can draw further inferences about the whole code units and use it as a basis for creating training datasets for Machine Learning based security application. Finally some early research results in ML models and their datasets were presented.

3.26 Security levels for white-box crypto

Atis Straujums (whiteCryption – Riga, LV)

We present a potential list of levels, assigned to our white-box algorithms based on their strength. The level criteria don't define a strict ordering but nevertheless help quickly assess the strength of protection and decide whether to spend any more resources on improving it for each algorithm.

3.27 Cheating in Online Games

Stijn Volckaert (KU Leuven – Ghent, BE)

In this talk, I presented an overview of the most prevalent types of cheats used in competitive online games. I discussed the functionality of these cheats and explained which techniques cheat coders use to construct them. I then shifted to cheat protection tools (so-called anti-cheats). After reviewing the overall architecture of an anti-cheat, I zoomed in on my own anti-cheat tool, ACE, which I built to protect Unreal Engine 1 games. I briefly talked about ACE's most prominent features and then reflected on things that have an have not worked when rolling out new functionality.

3.28 Modern Static Analysis of Obfuscated Code

John Wagner (Vector 35 – Melbourne, US)

Static analysis tools have improved significantly in recent years. This talk is an exploration of how modern static analysis tools analyze binary code and its impact on deobfuscation techniques. Various obfuscation techniques are discussed, including those that have been defeated by modern tools, those that are easier to defeat using the scripting features of these tools, and those that are still very difficult to analyze.

3.29 Kudelski Decision Support

Brecht Wyseur (Kudelski Group SA – Cheseaux, CH)

Kudelski has been developing Software Protection Tools internally. First for use on its own products – to protect Digital TV applications on a wide variety of devices. Now this has become also a product offering where Kudelski is helping its customers to protect its applications. In this presentation, we elaborate on the constraints and requirements that have been taken into account and how this has been managed in the product design and development processes.

3.30 Seminar introduction

Brecht Wyseur (Kudelski Group SA – Cheseaux, CH)

At the start of the seminar, we presented an opening introduction presentation, setting the scene and aligning on the seminar objectives. We also did a tour de table.

4 Working groups on Software Evaluation Methodology White Paper

As described in the executive summary, different groups brainstormed on evaluation methodology best practices for different use cases, to serve as the initial input to a white paper. The goal of this white paper is not to prescribe how exactly evaluations should be done, but rather what aspects are relevant to consider explicitly in evaluations, which assumptions might make sense and which might not. As a good way to convey, we consider the following potential structure of a software protection paper:

1. Abstract & Introduction
2. Attack Model – Background – Related Work
3. Technical Contribution
4. Evaluation
5. Discussion
6. Availability
7. Conclusions

Our guidelines are not bound to such a structure, nor do we put forward this structure. It simply allows us to put some structure in the many relevant aspects, as we can then formulate advice on what aspects to consider and discuss in each section.

For each of those supposed sections, we can then formulate advice that would be relevant for (almost all) papers in the domain of MATE software protection, or advice that would only be relevant for papers focused on either offensive or defensive techniques, or advice that would only be relevant for specific classes of techniques, such as trace-based techniques or static analyses, or for specific tools, such as obfuscators or disassemblers. So on top of the aforementioned structure, we structure the white paper itself into the following parts:

1. Introduction – to the white paper.
2. Methodology – explaning the methodology followed to get to the white paper.
3. General Principles
4. Defensive Techniques
5. Offensive Techniques
6. Benchmarks – specific advice on the use of benchmarks.
7. Appendices – one per separate class of techniques.

As this is an evolving field, the white paper would be a living document.

For the four case studies discussed during the seminar, the initial input for the corresponding appendices was collected, and potential plans were agreed upon to continue the necessary work towards an initial publishable white paper. Here, we list, as an example, a number of items for two of them: trace-based attack techniques on the one hand, and (anti-)disassembly techniques on the other. Principles or practices considered to be clearly more generally applicable are marked with an asterisk, even when some examples are given to clarify them that clearly only apply to the technique at hand. We do not repeat such generally applicable concepts in the second use case.

4.1 Class 1: (Anti-) Disassembly

4.1.1 Abstract & Introduction

- * Explicitly mention the specific goals of the defense or attack methods that your technique tries to overcome or to mitigate: disassembly, hiding code, identifying function starts, letting the disassembler produce incomplete information (such as incomplete CFGs) or letting it produce wrong information (i.e., incorrect graphs), identifying all basic blocks within a function, hindering dataflow analysis, call graph reconstruction, increase false-positive or false-negative rates of function starts, branches, function “ends”, etc
- * Explicitly mention the advantages of your technique.
- * Discuss upfront negative side-effects and impacts, as well as limitations
- * Discuss upfront the maturity of the proposed technique (e.g., tested on state-of-the-art combinations of protections or on single play version of one obfuscation, multiple platforms or not)

4.1.2 Attack Model – Background – Related Work

- * Describe the attack goals against which you defend, e.g., extracting instructions, function starts, modifying code, symbolic execution, dataflow analysis, ...
- * Discuss which attacks are out of scope, e.g., instruction tracing.
- * Make assumptions explicit, e.g., regarding dynamic analysis being difficult or impossible on some target, regarding the disassembler working on (the basis of) executable files on disks, memory dumps, traces, regarding defender and attacker capabilities, limitations on existing state-of-the-art techniques that you will use or build on.
- * Discuss relevant, concrete (real-world) scenarios in which the proposed technique will be demonstrated or is claimed to be useful, i.e., pushes the state of the art.
- Discuss related work that and the extent to which it targets cases that your technique can handle or is compatible with, such as overlapping instructions, polyglot code, dynamic jumps, self-modifying code, architectures designed against static disassembly (next instruction depends on current one – see Malbolge), opaque predicates, edit distance as a measure.

4.1.3 Technical Contribution

- * Explicitly discuss technical constraints, such as where in the build process is this applied (compile time, link time, post-link, runtime, etc)? What platform (incl. OS) / architecture / compilers / compiler options and other features are required or are the techniques limited to (e.g., data in code or not).
- * Explicitly discuss any diverges in the experimental setup / prototype implementation from the relevant scenarios that were discussed in the attack model section.

4.1.4 Evaluation

- * Explicitly mentioned the baseline you are comparing against.
- * Measure performance on standard benchmarks.
- * Advice in the form of "You can use metrics X, Y, and Z, for stealth, potency, performance, cost, resilience, ... but not A, B, and C because they are useless".
- Consider scalability and sensitivity regarding program size, amount of indirection in it, amount of aliasing in code when data flow analysis is used,...
- Ideally use multiple disassemblers and not just the tools out-of-the-box, but, e.g., with additional scripts that make up for the fact that the existing tools might not include some simple heuristics they would have improved their performance on your binaries but that were irrelevant before.
- Ideally do not evaluate on on binaries with only your new protection, but in combination with relevant protections taken from existing state of the art (commercial or academic).
- You should look ahead a bit in the cat-and-mouse-game: not all possible attacks on a new defense should be tested, but an analysis of some basic new attacks or small adaptations to existing attacks should be discussed, and ideally already be evaluated (e.g., by writing and running IDA Pro plugins that mimic simple heuristics that an attacker-improved version of IDA Pro would include once attackers get to know your knew protection).
- As long as the benchmarks span the relevant ranges as needed to assess scalability and sensitivity, any benchmarks will typically do, such as SPEC (i.e., no assets required in software).
- Provide at least 1 set of ground truth experiments, account for false positives & false negatives.
- Moving towards a standard set of benchmarks would be good.

4.1.5 Discussion

- * Discuss threats to validity
- * In case there were divergences between real-world attack model and experimental setup, the potential impact thereof should be discussed.

4.1.6 Availability

- * Anything that matters for reproducibility and to let others build on your results should be discussed, including the following items
- * Our code is available under license XXX.
- * Our benchmarks are standard, and available at ZZZ.
- * Output data is available at ...
- * Intermediate results are available ...
- * Scripts for summarizing results from raw data are available ...

4.1.7 Conclusions

4.2 Class 2: Trace-based Attack Techniques

4.2.1 Attack Model – Background – Related Work

- Explicitly describe and define the attack goals (e.g., finding keys, finding data value, modifying data value, ...) in relation to legitimate software protection or malware. For example, define what it means to "deobfuscate" in the context of your paper, and why that is chosen as a goal (e.g., undo VM-based protection or analyse code at the bytecode level).
- Be explicit about the defenses and features thereof that you consider in scope or not, as illustrated in the next bullets.
- Depending on the type of tracing (including data flow analysis on the fly or not), different types of tracing tools might be necessary, so limitations should be discussed if any exist that relate to anti-emulation protections that can break the use of tools.
- The granularity of traces should be discussed.
- Stealth is very important to defend against these attacks (hide fragments or operations that are relevant), so consider what stealth measures you can handle and how.
- Taint-tracking is difficult to do because it is easy to thwart techniques, so if your method depends on it, discuss this explicitly.
- Hiding boundaries between relevant and irrelevant code is important and easy (e.g., by inlining sensitive code) for many techniques, e.g., because the handle short sequences in traces that need to be identified first. So benchmarks should span a wide range in terms of stealth and complexity, but of course that also means we need a good definition of stealth first.
- It is okay to assume that a trace can always be collected, given the capabilities of modern virtualization technologies.
- It is okay to assume that deterministic replay of an execution on one input is possible. Making the trace artificially different for different inputs can be a real problem, so don't assume delta-techniques are trivial.
- If relevant, take into account in practice, deployed obfuscations might not be limited to small parts (that are easily identifiable).
- * The paper should explicitly cover whether or not the attacker is assumed to know the obfuscator internals, and whether or not the paper assumes security through obscurity (preferably not).

4.2.2 Evaluation

- * Use microbenchmarks to measure performance (CPU and memory runtime overhead, code size overhead)
- * Also measure performance on standard benchmarks, for example taken from a competition.
- * When analyzing obfuscations, it is important to measure the variability they introduce in the execution and report averages and confidence intervals.
- * Check scalability by measuring performance on benchmarks that covers a range (includes program size & complexity, trace size & complexity), measure time, memory consumption and anything else needed. Identify bottlenecks if there is an issue with scalability.

- * Both evaluation for specific protection-attack combinations (i.e., single protection) and on more real-world relevant cases (with more protections combined) can be useful and are ideally included. If you only chose one, explain why it's enough.

4.2.3 Discussion

- * Threads to validity should include how easy or difficult it is to port the proposed techniques to different platforms or setups and to deploy them in other scenarios.
- Coverage requirements should be discussed. What are the expectations in terms of coverage and how does it affect the validity of the attack?

Participants

- Mohsen Ahmadvand
TU München, DE
- Sébastien Bardin
CEA LIST, FR
- Cataldo Basile
Polytechnic University of Torino, IT
- Tim Blazytko
Ruhr-Universität Bochum, DE
- Richard Bonichon
CEA LIST – Nano-INNOV, FR
- Richard Clayton
University of Cambridge, GB
- Christian Collberg
University of Arizona – Tucson, US
- Moritz Contag
Ruhr-Universität Bochum, DE
- Bart Coppens
Ghent University, BE
- Jorge R. Cuéllar
Siemens AG – München, DE
- Mila Dalla Preda
University of Verona, IT
- Bjorn De Sutter
Ghent University, BE
- Laurent Dore
EDSI – Cesson-Sevigne, FR
- Ninon Eyrolles
Paris, FR
- Roberto Giacobazzi
University of Verona, IT
- Yuan Xiang Gu
Irdeto – Ottawa, CA
- Christophe Hauser
USC – Marina del Rey, US
- Stefan Katzenbeisser
Universität Passau, DE
- Eric Lafortune
Guardsquare – Leuven, BE
- Peter Lafosse
Vector 35 – Melbourne, US
- Patrik Marcacci
Kudelski Security – Cheseaux, CH
- J. Todd McDonald
University of South Alabama – Mobile, US
- Christian Mönch
Conax – Oslo, NO
- Leon Moonen
Simula Research Laboratory – Lysaker, NO
- Jan Newger
Google Switzerland – Zürich, CH
- Katharina Pfeffer
SBA Research – Wien, DE
- Yannik Potdevin
Universität Kiel, DE
- Uwe Resas
QuBalt GmbH, DE
- Rolf Rolles
Mobius Strip Reverse Engineering – San Francisco, US
- Sebastian Schrittwieser
FH – St. Pölten, AT
- Bahman Sistany
Irdeto – Ottawa, CA
- Natalia Stakhanova
University of Saskatchewan – Saskatoon, CA
- Atis Straujums
whiteCryption – Riga, LV
- Stijn Volckaert
KU Leuven – Ghent, BE
- John Wagner
Vector 35 – Melbourne, US
- Andreas Weber
Gemalto – München, DE
- Brecht Wyseur
Kudelski Group SA – Cheseaux, CH
- Michael Zunke
SFNT Germany GmbH – München, DE