

Engineering Secure Software and Systems: 10th
International Symposium, ESSoS 2018, Paris, France, June
26-27, 2018, Proceedings (Lecture Notes in Computer
Science Book 10953)

Pages: 144

Publisher: Springer; 1st ed. 2018 edition (June 19, 2018)

Format: pdf, epub

Language: English

[**DOWNLOAD FULL EBOOK PDF**]

Volume 10953	Lecture Notes in Computer Science	Theoretical Computer
Science and General Issues	Series Editors	Lancaster
University, Lancaster, United Kingdom	David Hutchison	Carnegie Mellon
University, Pittsburgh, Pennsylvania, USA	Takeo Kanade	University of
Surrey, Guildford, United Kingdom	Josef Kittler	Cornell University,
Ithaca, New York, USA	Jon M. Kleinberg	ETH Zurich, Zurich,
Switzerland	Friedemann Mattern	Stanford University, Stanford, California,
USA	John C. Mitchell	Dept Applied Math & Computer Science, Weizmann
Moni Naor	Stanford University, Stanford, California,	Indian Institute of
Institute of Science, Rehovot, Israel	C. Pandu Rangan	TU Dortmund
Technology Madras, Chennai, India	Bernhard Steffen	University of
University, Dortmund, Germany	Demetri Terzopoulos	University of
California, Los Angeles, California, USA	Doug Tygar	Max Planck
California, Berkeley, California, USA	Gerhard Weikum	Commenced
Institute for Informatics, Saarbrücken, Saarland, Germany		Gerhard Goos, Juris
Publication in 1973	Founding and Former Series Editors:	David Hutchison
Hartmanis, and Jan van Leeuwen	Editorial Board	Carnegie Mellon
Lancaster University, Lancaster, UK	Takeo Kanade	University of Surrey, Guildford,
University, Pittsburgh, PA, USA	Josef Kittler	University of Surrey, Guildford,
UK	Cornell University, Ithaca, NY, USA	John C. Mitchell
Friedemann Mattern	ETH Zurich, Zurich, Switzerland	Weizmann Institute of
Stanford University, Stanford, CA, USA	Moni Naor	Indian Institute of Technology
Science, Rehovot, Israel	C. Pandu Rangan	TU Dortmund University,
Madras, Chennai, India	Bernhard Steffen	University of California, Los
Dortmund, Germany	Demetri Terzopoulos	Angeles, CA, USA
Angeles, CA, USA	Doug Tygar	University of California, Berkeley, CA, USA
Gerhard Weikum	Max Planck Institute for Informatics, Saarbrücken, Germany	

More information about this series at

[http://
www.](http://www.)

7407 Editors Mathias Payer, Awais Rashid and Jose M. Such
Engineering Secure Software and Systems 10th International Symposium, ESSoS
2018, Paris, France, June 26–27, 2018, Proceedings
Editors Mathias Payer Purdue University, West Lafayette, USA
Awais Rashid University of Bristol, Clifton, UK
Jose M. Such King's College London, London, UK ISSN
0302-9743 e-ISSN 1611-3349 Lecture Notes in Computer Science ISBN
978-3-319-94495-1 e-ISSN 978-3-319-94496-8
<https://doi.org/10.1007/978-3-319-94496-8> Library of Congress Control Number:
2018947337 © Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use. The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations. This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature. The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland.

Preface It is our pleasure to welcome you to the proceedings of the 10th International Symposium on Engineering Secure Software and Systems (ESSoS 2018), co-located with the conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2018). ESSoS is part of a maturing series of symposia that attempts to bridge the gap between the software engineering and security scientific communities with the goal of supporting secure software development. The parallel technical sponsorship from ACM SIGSAC (the ACM interest group in security) and ACM SIGSOFT (the ACM interest group in software engineering) demonstrates the support from both communities and the need for providing such a bridge. Security mechanisms and the act of software development usually go hand in hand. It is generally not enough to ensure correct functioning of the security mechanisms used. They cannot be blindly inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way. Building trustworthy components does not suffice, since the interconnections and interactions of components play a significant role in trustworthiness. Lastly, while functional requirements are generally analyzed carefully in systems development, security considerations often arise after the fact. Adding security as an afterthought, however, often leads to problems. Ad Hoc development can lead to the deployment of systems that do not satisfy important security requirements. Thus, a sound methodology supporting secure systems development is needed. The presentations and associated publications at ESSoS 2018 contribute to this goal in several directions: first, by improving methodologies for secure software engineering (such as flow analysis and policy compliance). Second, with results for the detection and analysis of software vulnerabilities and the attacks they enable. Finally, for securing software for specific application domains (such as mobile

devices and access control). The conference program featured two keynotes, as well as research and idea papers. In response to the call for papers, 26 papers were submitted. The Program Committee selected seven full-paper contributions, presenting new research results on engineering secure software and systems. In addition, three idea papers were selected, giving a concise account of new ideas in the early stages of research. ESSoS received four requests for artifact evaluation, out of which three were evaluated correctly and received the artifact evaluation badge. Many individuals and organizations contributed to the success of this event. First of all, we would like to express our appreciation to the authors of the submitted papers and to the Program Committee members and external reviewers, who provided timely and relevant reviews. Many thanks go to the Artifact Evaluation Committee for assessing the quality of the submitted artifacts to the Steering Committee for supporting this series of symposia, and to all the members of the Organizing Committee for their tremendous work. We are thankful to ACM SIGSAC/SIGSOFT and LNCS for continuing to support us in this series of symposia. The engineering of secure software and systems in an increasingly hyperconnected world is both a major challenge and a fundamental need to ensure that the digital fabric underpinning our society remains resilient in the face of a variety of threats. Bringing researchers from software and systems engineering and computer security to discuss challenges and solutions is an important important step in this regard. It has been our pleasure to chair the program for ESSoS 2018. Mathias Payer

Committee	Awais Rashid May 2018 Jorge Cuellar	Organization Siemens AG, Germany KU Leuven University, Belgium	Steering Mathias Payer
	Wouter Joosen (Chair) Fabio Massacci	University of Trento, Italy The Open University and LERO, UK IMDEA Software Institute, Spain	
	Bashar Nuseibeh Juan Caballero	University of Paderborn, Germany Royal Holloway University of London, UK	
	Eric Bodden Lorenzo Cavallaro	Organizing Committee Purdue University, USA University of Bristol, UK	Program Co-chairs Awais Rashid
	Mathias Payer Rashid Publication Chair UK	Jose M. Such Publicity Chair	King's College London, Raoul Strackx Local
Organizers	KU Leuven University, Belgium Sebastien Bardin	CEA, France Telecom SudParis, France	
	Gregory Blanc Web Chair	Annick Vandijck	KU Program Committee
Leuven University, Belgium	Yasmin Acar Luca Allodi	Leibniz University Hannover, Germany Technical University of Eindhoven, The Netherlands	
	Pauline Anthonysamy Rakesh Bobba	Google Inc., Switzerland Oregon State University, USA	
	Alvaro Cardenas Lorenzo Cavallaro	The University of Texas at Dallas, USA Royal Holloway University of London, UK	
	Tom Chothia Sven Dietrich	University of Birmingham, UK The City University of New York, USA	
	Mattia Fazzini Yanick Fratantonio	Georgia Institute of Technology, Atlanta, USA EURECOM, France	
	Seda Gurses Marina Krotofil	Princeton University, USA FireEye, Inc., USA	Per Larsen Martina Lindorfer
	University of California, Irvine, USA University of California, Santa Barbara, USA		Mira Mezini Nick
Nikiforakis	Technical University of Darmstadt, Germany Stony Brook University, New York, USA		
Michael Pradel	Technical University of Darmstadt, Germany		

cryptographic functions in highly optimized binary code and tests the identified functions for functional correctness, potentially also revealing backdoors. The research leading to this paper has received funding from the AMASS project (H2020-ECSEL no. 692474).

1 Introduction In recent years cryptography became almost universal in applications, both commercial and open-source. Every application securely communicating with its cloud server relies on a full suite of symmetric and asymmetric encryption algorithms. Hence, running a closed-source application employing cryptography requires one to trust the software developer to (a) consistently use cryptography, (b) have chosen a good crypto-library and use it correctly, and (c) not to have built-in a backdoor that leaks sensitive information. Especially when dealing with software running in a sensitive environment it is necessary to thoroughly inspect the software for potential vulnerabilities and threats. We focus on the correctness analysis of the software using the binary code as unwanted functionality can be introduced during the compilation process [1], the original source code or libraries are not available, or one cannot re-produce bit-matching binary images from the supplied sources. For similar reasons, i.e. no source available or highest level of trust, any late stage customization has to be performed on the machine-code level. In the following, we focus on cryptography routines because they are often exposed to the network and present a major part of the attack surface of an application. Also, this specific domain allows us to use a lot of domain-specific knowledge in the attempted automated analysis. Once some piece of software fails the verification, information about vulnerabilities becomes public, or encryption needs to be added due to changing security requirements, the software needs to be patched or replaced. In order to save on the re-certification/verification effort, we propose to automatically address the shortcomings in the binary file. Put differently, we do not only want to analyze binaries, but also fix certain problems automatically or even add encryption to binaries that do not currently employ encryption, e.g. adding HTTPS to a legacy application. While one source of vulnerabilities are sophisticated, deliberate backdoors, we believe a significant portion of flaws are accidental. Developers typically are not very knowledgeable in cryptography. A common pattern these days is that developers use the first seemingly working code-snippet from some blog or stackexchange answer for their code. There are numerous problems when using cryptography in this naïve way. The first problem is that developers may use a home-grown implementation of their desired cipher or some not very well maintained library. Such home-grown implementations will not contain hardening against side-channel attacks and will likely contain unknown bugs or weaknesses [2]. The second problem is connected to the use of statically linked libraries. Here, one issue is that the user of the software cannot easily update the version of the library used, once a vulnerability has been found. Further, the user has no guarantees (and is generally unable to check) whether the software supplier has used the latest version of the library when building the release. As security issues are regularly found and fixed in libraries, this is a major problem. Lastly, even the best crypto library may be used incorrectly. There are numerous topics around encryption, such as key management or safe storage and erasure of the key in memory that a developer can get wrong. In the past numerous flaws concerning cryptography have been found in software. Here are some of the more interesting cases. Our first example has been revealed by the Chaos Computer Club of Germany (CCC) in software used by the German government to spy on suspected criminals. On close analysis of the software, which is also known as the federal trojan horse, the CCC discovered that the trojan used the same hard-coded AES-key to secure the data transfer to the control server for years. The key can now be found on the internet [3]. Hence, unauthorized parties would have been able read the data transferred. The second example is the well-known Heartbleed bug [4]. This bug may still be present in applications that were statically linked with any of the vulnerable OpenSSL versions. Here, we would want an automated upgrade of the library linked into the affected applications to fix the issue. A third example shows another source for weaknesses when using cryptographic functions: the random number generator. The best crypto implementation does not help if the key is predictable. This was the case in Debian's OpenSSL package, that was able to generate only 32 768 different keys [5]. As before, we would like to be able to address this bug by replacing the random generator used in the application by a trusted one. These are just a few

examples, but they illustrate a need to focus on cryptography as part of a software security analysis and life cycle. They also show that binary analysis is needed because merely analyzing the input and output of the application may not reveal a faulty crypto implementation. We are now ready to present our vision in Sect. 2, before arguing for its feasibility in Sect. 3 by presenting already existing work we can build on. Finally, we conclude this idea paper in Sect. 4.

2 The Vision

As we have shown in Sect. 1, the current treatment of cryptography in the life-cycle of executable programs is far from optimal. Put shortly, the user has no efficient way of checking whether the employed algorithms or library versions are up-to-date, whether the cryptographic primitives are used correctly, or whether the advertised algorithms are used at all. Even worse, once vulnerabilities are discovered there is no easy way to fix the affected executables. Finally, there is no way to add additional functionality to the application.

Our vision is to address these shortcomings with a platform for analyzing, rewriting, and embedding cryptographic functions in binaries. By 'binary' we mean a compiled program, where we don't have access to the source code or any debug information. We understand the term cryptography here in its wider sense to also include random number generators needed to generate keys, cipher modes (counter mode, cipher feedback mode, etc.), and cryptographic protocols like TLS. We envision our platform to have the following features:

Analyze cryptographic functions in the binary for correctness and potential backdoors. The analysis should reveal if the algorithm computes the correct result, has potential information leaks, and if it is vulnerable to known attacks.

Replace cryptographic functions in the binary with alternate implementations.

Insert cryptographic functionality, which is most useful for legacy binaries.

While these features seem trivial, achieving them is far from it. We will use a few scenarios to illustrate common problems and how our platform would help.

Unknown Binary Using Crypto. Assuming we bought an application that employs the Advanced Encryption Standard (AES) according to the data sheet. In this situation we need to check whether AES really is being used and whether there are (known) vulnerabilities, weaknesses, or leaks with this version. Hence in this scenario we could use the analysis capabilities of our platform. In case we are not satisfied with the quality of the implementation, we can then replace the existing AES implementation with calls to our own trusted library.

Legacy Application Without Patches. Another important situation is the maintenance of legacy applications without vendor support. Be it a statically linked application or an application dynamically linked to some outdated version of some cryptographic library that contains known vulnerabilities. In order to develop our own patch for the application, our platform can be used. In the case of an statically linked application, the platform will supply the means to replace the original library version with an updated one. If the application has a dynamic dependency one might be able to find a drop-in replacement if the interface did not change. If, however, there was some interface change, our platform will help with its insert and replacement functionality to add adapter code so the application can use the interface incompatible new version of the library.

Adding Encryption. A further use case we envision for our platform is to add encryption to executables. This could be done by adding transient encryption/decryption when saving/loading files but could also mean securing network connections in non-standard ways. For example we may want to enable a legacy application to encrypt its network traffic. The encryption could be added to the data passed to the send function of libc and the decryption to the data returned by the receive function. We could provide standard packages that also take care of key derivation between peer, for example as part of the connect call. Similarly employing steganography or adding direct support for onion routing techniques would benefit from our platform.

Weakening Encryption. While not in our primary focus, our platform could even be used to weaken a cryptographic implementation. This could be useful in case of reverse engineering, i.e. malware analysis. An important part of reverse engineering is the network traffic, which may be difficult to analyze if it is TLS encrypted with certificate pinning. Our framework would save the analyst a lot of time by simply leaking the encryption key to a file.

All of the discussed scenarios can be realized manually with dedicated personnel. Our platform, however, should automate the work involved as much as possible. While this is no easy feat, we think it is viable and give an overview

of already available building blocks in the following section.

3 Available Already

Quite a number of building blocks for our vision are already in place. On the one hand there is a large body of research and tools that deal with binary analysis and manipulation, on the other hand we started working towards our platform for Linux/x86-64 and gained first, encouraging results.

In the following sub-sections, we give a brief overview of available tools and techniques that help realizing our vision. Starting from tools helping with binary rewriting and analysis, we refer to techniques used for specifying machine code that needs replacement before describing our own contributions.

3.1 Supporting Tools

Binary Rewriting.

We benefit from a large body of work in runtime manipulation of machine code and binary rewriting. We use DynamoRIO [6] to inject code and modify the control flow of binaries during runtime. While this is ideal for prototyping we would eventually want to rewrite the binaries to persist changes in it. Rewriting binaries is challenging because the control flow can be unpredictable due to exceptions and signals and because some parts of the code may be only revealed at runtime due to Just-In-Time compilation or encryption. Thankfully many of these problem are already addressed, e.g. in Zipr++ [7] and RL-Bin [8].

Binary Analysis.

There are two approaches to analyzing a binary. One is to observe the binary during its normal operation at runtime. That is suitable to understand the normal operation of the binary, but not to find a backdoor. The latter can be found for example by symbolic analysis, a technique to reach all program locations. The downside is that such analysis is inherently slow. Since we expect the crypto routines to be part of the normal operation of the binary our prototype uses runtime analysis. We use DynamoRIO to record a trace that we later analyze. A trace means the sequence of all machine code instructions that were executed during a single run of the binary under observation.

There are a few symbolic analysis frameworks for machine code, including S2E [9], angr [10], and libtriton [11]. We also showed that it is feasible to use KLEE [12] for symbolic analysis if the machine code is first lifted to LLVM intermediate language. A task that is non-trivial in itself and we completed only for simple binaries.

Specification.

We specify functions and their input/output behavior with the help of model programs. That means we have a model of the intended functionality as a specification and are searching for functions that, when given the same input, will return the same value. Hence our specification does not use pairs of inputs and outputs, but a reference implementation of the function we are looking for. To our knowledge this specification approach is novel in the field.

Another specification

method is to use seed functions [13]. Seed functions are functions in the binary one wants to remove by removing the function and all functions that depend on it. These functions can specify a function in a specific binary, but cannot describe the same functionality over all binaries. An alternative is dual slicing [14] where a feature is defined by the difference in two program executions. So the program is started with two different parameters and the function calls that are present in only one trace are the functions of interest.

3.2 First Results

Using

these building blocks we built a prototype demonstrating parts of our vision.

Identification. Our specification is given in the form of an implementation of the crypto functions we are looking for. We use these to find the functions of interest in the trace we recorded. The naïve approach of testing all functions called in the trace brute-force does not scale. We therefore employ domain-knowledge to narrow the search: A candidate function can often be identified by its use of certain constants (SBox in AES), specific CPU instructions (AES-NI), or heuristically by a density of bit-level operations in the code. To test the latter we use a machine-learning approach that is able to identify functions containing cryptographic operations with high confidence. A different approach is implemented in CryptoHunt [15] and the Software Analysis Workbench [16], where the authors translate the binary program into logical formulas that can be compared to a given reference implementation with an SMT solver.

Function

identification is further complicated because we also need the order of parameters of the function in order to replace or invoke it. Again a brute-force attempt would be very slow, so we again use domain knowledge: The parameters we are interested in (plaintext, key, ciphertext) are pointers to memory buffers of at least 16 byte length. That significantly reduces the number of parameters we need to test.

Testing Cryptographic Implementations. Knowing the exact interface we

can test the cryptographic implementation in the binary. We support two test modes: Firstly, we support supervised encryption, which means that we check after every invocation of the encryption routine if the returned result is correct. This is done by running in parallel a trusted implementation. At this point we could also replace the entire crypto routine with the trusted implementation. Currently this check is done at runtime using DynamoRIO. The second test mode is to run the encryption function against a list of well-known input-output pairs. In case of AES such pairs are provided by the NIST [17]. This works during runtime by waiting until the encryption function is first invoked and then repeatedly invoking just the encryption function with the chosen inputs and comparing the outputs to the specified ones. Any deviation is an indication that the encryption is not implemented correctly. This is essentially a form of differential testing [18].

Symbolic Analysis. We already use symbolic analysis to find so-called logic bombs in binaries. A logic bomb is a malicious action hidden in the binary and triggered on certain conditions. In terms of crypto this could for example be a backdoor leaking the key. A first attempt working on source code was already published [19] and we are currently busy porting this to the machine code level.

Replacing the Encryption Algorithm. The identification of the encryption algorithm and its parameters is the first important step to allow replacing the encryption algorithm. Our framework could be trivially extended to make the replacement at runtime with DynamoRIO. This is because we already intercept the call to crypto functions for testing. Instead of running both functions and comparing the result, one could simply return the result of the reference implementation and never invoke the original function. While the runtime manipulation of the binary is perfect for testing, it is not desirable as a permanent solution due to the overhead. Therefore we need the binary rewriting tools outlined in Sect. 3.1 to persist changes in the binary itself.

Inserting Encryption. To insert encryption we need to specify insertion points, e.g. function calls to `libc`. `libc` is a standard C library used by virtually every Linux application. We can replace these calls with a transparent wrapper to encrypt data before it is passed to `libc` and decrypting data returned by `libc`. This could be done when writing data to a file or to a network socket. Of course this would be further complicated by adding key management and exchange. To protect the keys in memory from the original application we can use novel CPU-backed technologies that isolate certain parts of memory, such as Intel SGX.

3.3 Evaluation Our current prototype implements the complete testing toolchain for AES: It has the ability to record traces, find the addresses and parameters of the crypto functions, and test the crypto function using NIST vectors. We implemented several models for AES using two modes (ECB and CTR) and various keylengths.

We can not only process the small toy examples we created for numerous ways to implement AES, but also the `aesencrypt2` sample program from the `mbedtls` library [20]. All these example were compiled with GCC optimization level 3 and without symbols. The `aesencrypt2` example is 180 kb in size and contains more than 7000 assembler instructions.

4 Conclusion We have presented our vision on how to address the challenges posed by cryptography in the life-cycle of executable programs. In order to automate the process of testing and adapting executables as much as possible, we propose to build a platform capable of analyzing, replacing, and inserting cryptographic functions with the goal of achieving a high level of automation. For this, we rely on a mix of techniques known from binary analysis and rewriting, program verification, model-based testing, and compiler construction.

Our envisioned platform will help analysts find flawed cryptographic implementations and replace them by trusted ones or even insert encryption functionality into executable programs. We have made promising first steps towards our vision by implementing parts of the platform on the Linux-x86-64 platform and applying it to different applications relying on AES. Our lessons learnt led us to new approaches for speeding up solving the identification problem of functions and parameters, which we are implementing right now. We are also working on improving the symbolic analysis to make it more scalable and applicable to larger executables. Finally, we want to use rewriting techniques to make permanent changes to binaries as the next step in going after our vision.

References

1. Thompson, K.: Reflections on trusting trust. *Commun. ACM* 27(8), 761–763 (1984) [Crossref](#)
2. Egele, M., Brumley, D., Fratantonio, Y.,

Kruegel, C.: An empirical study of cryptographic misuse in Android applications. In: CCS 2013, pp. 73–84 (2013) 3. CCC: Analyse einer Regierungs-Malware. Technical report, Chaos Computer Club (2011) 4. Codenomicon, Google-Security: CVE-2014-0160. Available from MITRE, CVE-ID CVE-2014-0160, 3 Dec 2013 5. Bello, L.: CVE-2008-0166. Available from MITRE, CVE-ID CVE-2008-0166, 9 Jan 2008 6. Bruening, D., Zhao, Q., Amarasinghe, S.: Transparent dynamic instrumentation. ACM SIGPLAN Not. 47(7), 133–144 (2012) [Crossref](#) 7. Hiser, J., Nguyen-Tuong, A., Hawkins, W., McGill, M., Co, M., Davidson, J.: Zipr++: exceptional binary rewriting. In: FEAST 2017, pp. 9–15 (2017) 8. Majlesi-Kupaei, A., Kim, D., Anand, K., ElWazeer, K., Barua, R.: RL-Bin, robust low-overhead binary rewriter. In: FEAST 2017, pp. 17–22 (2017) 9. Chipounov, V., Kuznetsov, V., Candea, G.: The S2E platform: design, implementation, and applications. TOCS 30(1), 2 (2012) [Crossref](#) 10. Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., Vigna, G.: SoK: (state of) the art of war: offensive techniques in binary analysis. In: S&P 2016 (2016) 11. Saudel, F., Salwan, J.: Triton: a dynamic symbolic execution framework. In: Symposium sur la sécurité des Technologies de l'information et des Communications, SSTIC, France, Rennes, June 3–5 2015, SSTIC, pp. 31–54 (2015) 12. Cadar, C., Dunbar, D., Engler, D.R., et al.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: OSDI. vol. 8, pp. 209–224 (2008) 13. Jiang, Y., Zhang, C., Wu, D., Liu, P.: Feature-based software customization: preliminary analysis, formalization, and methods. In: HASE 2016, pp. 122–131 (2016) 14. Kim, D., Sumner, W.N., Zhang, X., Xu, D., Agrawal, H.: Reuse-oriented reverse engineering of functional components from x86 binaries. In: ICSE 2014, pp. 1128–1139 (2014) 15. Xu, D., Ming, J., Wu, D.: Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping. In: S&P 2017, pp. 921–937 (2017) 16. Dockins, R., Foltzer, A., Hendrix, J., Huffman, B., McNamee, D., Tomb, A.: Constructing semantic models of programs with the software analysis workbench. In: VSTTE 2016, pp. 56–72 (2016) 17. Bassham III, L.E.: The advanced encryption standard algorithm validation suite (AESAVS). NIST Information Technology Laboratory (2002) 18. McKeeman, W.M.: Differential testing for software. Digit. Techn. J. 10(1), 100–107 (1998) 19. Papp, D., Buttyán, L., Ma, Z.: Towards semi-automated detection of trigger-based behavior for software security assurance. In: SAW 2018 (2018) 20. ARM: mbedTLS. <https://>

[tls.](#)

[mbed.](#)

[org/](#)

© Springer International Publishing AG, part of Springer Nature 2018

Mathias Payer, Awais Rashid and Jose M. Such (eds.) Engineering Secure Software and Systems
Lecture Notes in Computer Science 10953 https://doi.org/10.1007/978-3-319-94496-8_2

Enforcing Full-Stack Memory-Safety in Cyber-Physical Systems

Eyasu Getahun Chekole^{1, 2} [](#) Sudipta Chattopadhyay¹, Martín Ochoa^{1, 3} and
Guo Huaqun² (1) Singapore University of Technology and Design, Singapore,
Singapore (2) Institute for Infocomm Research (I2R), Singapore, Singapore
(3) Department of Applied Mathematics and Computer Science, Universidad del
Rosario, Bogotá, Colombia

This book constitutes the refereed proceedings of the 10th International Symposium on Engineering Secure Software and Systems, ESSoS 2018, held in Paris, France, in June 2018. The 10 papers, consisting of 7 regular and 3 idea papers, were carefully reviewed and selected from 26 submissions. They focus on the construction of secure software, which is becoming an increasingly challenging task due to the complexity of modern applications, the growing sophistication of security requirements, the multitude of available software technologies, and the progress of attack vectors.

Engineering Secure Software and Systems - This book constitutes the refereed proceedings of the 10th International Symposium on Engineering Secure Software and Systems : 10th International Symposium, ESSoS 2018, Paris, France, June 26-27, 2018, Proceedings Series: Lecture notes in computer science 0302-9743 ; 10953;. Mathias Payer - dblp - Lecture Notes in Computer Science 10953, Springer 2018, ISBN 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings.. and Systems - 5th International Symposium, ESSoS 2013, Paris, France, Engineering Secure Software and Systems, Second International Symposium, ESSoS Dubai Audiobook Library - This book constitutes the refereed proceedings of the 10th International Symposium. 10th International Symposium, ESSoS 2018, Paris, France, June 26-27, 2018, Proceedings. by. series Lecture Notes in Computer Science #10953 Engineering Secure Software and Systems : 10th International - ... International Symposium, ESSoS 2018, Paris, France, June 26-27, 2018, Proceedings by. This book constitutes the refereed proceedings of the 10th International Series: Lecture Notes in Computer Science , #10953. 10th International Symposium, ESSoS 2018, Paris, France, June 26 - 10th International Symposium, ESSoS 2018. Paris, France, June 26â€“27, 2018. Proceedings Lecture Notes in Computer Science 10953 the authors, and the editors are safe to assume that the advice and information in this book are posium on Engineering Secure Software and Systems (ESSoS 2018), co-located with.

Engineering Secure Software and Systems: 10th International - books/mc/18/Payer18
Engineering Secure Software and Systems - 10th International Symposium, ESSoS
2018, Paris, France, June 26-27, 2018, Proceedings. Lecture Notes in Computer
Science 10953, Springer 2018, ISBN.. Proceedings of the 2016 ACM on Cloud
Computing Security Workshop, Matokeo ya Utafutaji - Bookshare Nepal - Engineering
Secure Software and Systems: 10th International Symposium, ESSoS France, June
26-27, 2018, Proceedings (Lecture Notes in Computer Science #10953) This book
constitutes the refereed proceedings of the 10th International Software and Systems,
ESSoS 2018, held in Paris, France, in June 2018. Engineering Secure Software and
Systems - dblp - Engineering Secure Software and Systems : 10th International
Symposium, ESSoS 2018, Paris, France, June 26-27, 2018, Proceedings. [Mathias
Payer; Awais Rashid; Publishing, 2018. Series: Lecture notes in computer science,
10953. 10th International Symposium, ESSoS 2018, Paris, France, June 26 -
Engineering Secure Software and Systems : 10th International Symposium, ESSoS
2018, Paris, France, June 26-27, 2018, Proceedings. [Mathias Payer; Awais Rashid;
Publishing, 2018. Series: Lecture notes in computer science, 10953. Engineering
Secure Software and Systems: 10th International - This book constitutes the refereed
proceedings of the 10th International Symposium on Engineering Secure Software
Engineering Secure Software and Systems : 10th International Symposium, ESSoS
2018, Paris, France, June 26-27, 2018, Proceedings Series: Lecture notes in
computer science 0302-9743 ; 10953;. Engineering Secure Software and Systems:
10th International - 10th International Symposium, ESSoS 2018. Paris, France, June
26â€“27, 2018. Proceedings Lecture Notes in Computer Science 10953 the authors,
and the editors are safe to assume that the advice and information in this book are
posium on Engineering Secure Software and Systems (ESSoS 2018), co-located
with.

Relevant Books

[\[DOWNLOAD \]](#) - Download ebook The Reluctant Bid: Distinguished Gentlemen
Series epub, pdf

[\[DOWNLOAD \]](#) - Download book Fundamental Limitations in Filtering and Control
(Communications and Control Engineering)

[\[DOWNLOAD \]](#) - Download Free Hitting The Pavement: "A Journalist's 2-Year

Memoir of Homelessness"

[DOWNLOAD] - Download Intake: In the War for the Mind, the Young Fall Hardest (Imbalanced - Part One) free pdf online

[DOWNLOAD] - Online Dictionary of Architecture and Interior Design: Essential Terms for the Home pdf
