LNCS 3224

Erland Jonsson
Alfonso Valdes
Magnus Almgren (Eds.)

Recent Advances in Intrusion Detection

7th International Symposium, RAID 2004
Sophia Antipolis, France, September 2004
Proceedings

Springer

Erland Jonsson Alfonso Valdes
Magnus Almgren (Eds.)

Recent Advances in Intrusion Detection

7th International Symposium, RAID 2004
Sophia Antipolis, France, September 15-17, 2004
Proceedings

Springer

Volume Editors

Erland Jonsson
Magnus Almgren
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
E-mail: {erland.jonsson, magnus.almgren}@ce.chalmers.se

Alfonso Valdes
SRI International
333 Ravenswood Ave., Menlo Park, CA 94025, USA
E-mail: alfonso.valdes@sri.com

Library of Congress Control Number: 2004111363

CR Subject Classification (1998): K.6.5, K.4, E.3, C.2, D.4.6

ISSN 0302-9743
ISBN 3-540-23123-4 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11322924 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3224

Lecture Notes in Computer Science

For information about Vols. 1–3104

please contact your bookseller or Springer

Vol. 3232: R. Heery, L. Lyon (Eds.), Research and Advanced Technology for Digital Libraries. XV, 528 pages. 2004.

Vol. 3224: E. Jonsson, A. Valdes, M. Almgren (Eds.), Recent Advances in Intrusion Detection. XII, 315 pages. 2004.

Vol. 3223: K. Slind, A. Bunker, G. Gopalakrishnan (Eds.), Theorem Proving in Higher Order Logic. VIII, 337 pages. 2004.

Vol. 3221: S. Albers, T. Radzik (Eds.), Algorithms – ESA 2004. XVIII, 836 pages. 2004.

Vol. 3220: J.C. Lester, R.M. Vicari, F. Paraguaçu (Eds.), Intelligent Tutoring Systems. XXI, 920 pages. 2004.

Vol. 3208: H.J. Ohlbach, S. Schaffert (Eds.), Principles and Practice of Semantic Web Reasoning. VII, 165 pages. 2004.

Vol. 3207: L.T. Jang, M. Guo, G.R. Gao, N.K. Jha, Embedded and Ubiquitous Computing. XX, 1116 pages. 2004.

Vol. 3206: P. Sojka, I. Kopecek, K. Pala (Eds.), Text, Speech and Dialogue. XIII, 667 pages. 2004. (Subseries LNAI).

Vol. 3205: N. Davies, E. Mynatt, I. Siio (Eds.), UbiComp 2004: Ubiquitous Computing. XVI, 452 pages. 2004.

Vol. 3203: J. Becker, M. Platzner, S. Vernalde (Eds.), Field Programmable Logic and Application. XXX, 1198 pages. 2004.

Vol. 3199: H. Schepers (Ed.), Software and Compilers for Embedded Systems. X, 259 pages. 2004.

Vol. 3198: G.-J. de Vreede, L.A. Guerrero, G. Marín Raventós (Eds.), Groupware: Design, Implementation and Use. XI, 378 pages. 2004.

Vol. 3194: R. Camacho, R. King, A. Srinivasan (Eds.), Inductive Logic Programming. XI, 361 pages. 2004. (Subseries LNAI).

Vol. 3193: P. Samarati, P. Ryan, D. Gollmann, R. Molva (Eds.), Computer Security – ESORICS 2004. X, 457 pages. 2004.

Vol. 3192: C. Bussler, D. Fensel (Eds.), Artificial Intelligence: Methodology, Systems, and Applications. XIII, 522 pages. 2004. (Subseries LNAI).

Vol. 3189: P.-C. Yew, J. Xue (Eds.), Advances in Computer Systems Architecture. XVII, 598 pages. 2004.

Vol. 3186: Z. Bellahsène, T. Milo, M. Rys, D. Suciu, R. Unland (Eds.), Database and XML Technologies. X, 235 pages. 2004.

Vol. 3185: M. Bernardo, F. Corradini (Eds.), Formal Methods for the Design of Real-Time Systems. VII, 295 pages. 2004.

Vol. 3184: S. Katsikas, J. Lopez, G. Pernul (Eds.), Trust and Privacy in Digital Business. XI, 299 pages. 2004.

Vol. 3183: R. Traunmüller (Ed.), Electronic Government. XIX, 583 pages. 2004.

Vol. 3182: K. Bauknecht, M. Bichler, B. Pröll (Eds.), E-Commerce and Web Technologies. XI, 370 pages. 2004.

Vol. 3181: Y. Kambayashi, M. Mohania, W. Wöß (Eds.), Data Warehousing and Knowledge Discovery. XIV, 412 pages. 2004.

Vol. 3180: F. Galindo, M. Takizawa, R. Traunmüller (Eds.), Database and Expert Systems Applications. XXI, 972 pages. 2004.

Vol. 3179: F.J. Perales, B.A. Draper (Eds.), Articulated Motion and Deformable Objects. XI, 270 pages. 2004.

Vol. 3178: W. Jonker, M. Petkovic (Eds.), Secure Data Management. VIII, 219 pages. 2004.

Vol. 3177: Z.R. Yang, H. Yin, R. Everson (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2004. XVIII, 852 pages. 2004.

Vol. 3176: O. Bousquet, U. von Luxburg, G. Rätsch (Eds.), Advanced Lectures on Machine Learning. VIII, 241 pages. 2004. (Subseries LNAI).

Vol. 3175: C.E. Rasmussen, H.H. Bülthoff, B. Schölkopf, M.A. Giese (Eds.), Pattern Recognition. XVIII, 581 pages. 2004.

Vol. 3174: F. Yin, J. Wang, C. Guo (Eds.), Advances in Neural Networks - ISNN 2004. XXXV, 1021 pages. 2004.

Vol. 3172: M. Dorigo, M. Birattari, C. Blum, L. M.Gambardella, F. Mondada, T. Stützle (Eds.), Ant Colony, Optimization and Swarm Intelligence. XII, 434 pages. 2004.

Vol. 3170: P. Gardner, N. Yoshida (Eds.), CONCUR 2004 - Concurrency Theory. XIII, 529 pages. 2004.

Vol. 3166: M. Rauterberg (Ed.), Entertainment Computing – ICEC 2004. XXIII, 617 pages. 2004.

Vol. 3163: S. Marinai, A. Dengel (Eds.), Document Analysis Systems VI. XII, 564 pages. 2004.

Vol. 3162: R. Downey, M. Fellows, F. Dehne (Eds.), Parameterized and Exact Computation. X, 293 pages. 2004.

Vol. 3160: S. Brewster, M. Dunlop (Eds.), Mobile Human-Computer Interaction – MobileHCI 2004. XVIII, 541 pages. 2004.

Vol. 3159: U. Visser, Intelligent Information Integration for the Semantic Web. XIV, 150 pages. 2004. (Subseries LNAI).

Vol. 3158: I. Nikolaidis, M. Barbeau, E. Kranakis (Eds.), Ad-Hoc, Mobile, and Wireless Networks. IX, 344 pages. 2004.

Vol. 3157: C. Zhang, H. W. Guesgen, W.K. Yeap (Eds.), PRICAI 2004: Trends in Artificial Intelligence. XX, 1023 pages. 2004. (Subseries LNAI).

Vol. 3156: M. Joye, J.-J. Quisquater (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2004. XIII, 455 pages. 2004.

Vol. 3155: P. Funk, P.A. González Calero (Eds.), Advances in Case-Based Reasoning. XIII, 822 pages. 2004. (Subseries LNAI).

Vol. 3154: R.L. Nord (Ed.), Software Product Lines. XIV, 334 pages. 2004.

Vol. 3153: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), Mathematical Foundations of Computer Science 2004. XIV, 902 pages. 2004.

Vol. 3152: M. Franklin (Ed.), Advances in Cryptology – CRYPTO 2004. XI, 579 pages. 2004.

Vol. 3150: G.-Z. Yang, T. Jiang (Eds.), Medical Imaging and Augmented Reality. XII, 378 pages. 2004.

Vol. 3149: M. Danelutto, M. Vanneschi, D. Laforenza (Eds.), Euro-Par 2004 Parallel Processing. XXXIV, 1081 pages. 2004.

Vol. 3148: R. Giacobazzi (Ed.), Static Analysis. XI, 393 pages. 2004.

Vol. 3146: P. Érdi, A. Esposito, M. Marinaro, S. Scarpetta (Eds.), Computational Neuroscience: Cortical Dynamics. XI, 161 pages. 2004.

Vol. 3144: M. Papatriantafilou, P. Hunel (Eds.), Principles of Distributed Systems. XI, 246 pages. 2004.

Vol. 3143: W. Liu, Y. Shi, Q. Li (Eds.), Advances in Web-Based Learning – ICWL 2004. XIV, 459 pages. 2004.

Vol. 3142: J. Diaz, J. Karhumäki, A. Lepistö, D. Sannella (Eds.), Automata, Languages and Programming. XIX, 1253 pages. 2004.

Vol. 3140: N. Koch, P. Fraternali, M. Wirsing (Eds.), Web Engineering. XXI, 623 pages. 2004.

Vol. 3139: F. Iida, R. Pfeifer, L. Steels, Y. Kuniyoshi (Eds.), Embodied Artificial Intelligence. IX, 331 pages. 2004. (Subseries LNAI).

Vol. 3138: A. Fred, T. Caelli, R.P.W. Duin, A. Campilho, D.d. Ridder (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. XXII, 1168 pages. 2004.

Vol. 3137: P. De Bra, W. Nejdl (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. XIV, 442 pages. 2004.

Vol. 3136: F. Meziane, E. Métais (Eds.), Natural Language Processing and Information Systems. XII, 436 pages. 2004.

Vol. 3134: C. Zannier, H. Erdogmus, L. Lindstrom (Eds.), Extreme Programming and Agile Methods - XP/Agile Universe 2004. XIV, 233 pages. 2004.

Vol. 3133: A.D. Pimentel, S. Vassiliadis (Eds.), Computer Systems: Architectures, Modeling, and Simulation. XIII, 562 pages. 2004.

Vol. 3132: B. Demoen, V. Lifschitz (Eds.), Logic Programming. XII, 480 pages. 2004.

Vol. 3131: V. Torra, Y. Narukawa (Eds.), Modeling Decisions for Artificial Intelligence. XI, 327 pages. 2004. (Subseries LNAI).

Vol. 3130: A. Syropoulos, K. Berry, Y. Haralambous, B. Hughes, S. Peter, J. Plaice (Eds.), TeX, XML, and Digital Typography. VIII, 265 pages. 2004.

Vol. 3129: Q. Li, G. Wang, L. Feng (Eds.), Advances in Web-Age Information Management. XVII, 753 pages. 2004.

Vol. 3128: D. Asonov (Ed.), Querying Databases Privately. IX, 115 pages. 2004.

Vol. 3127: K.E. Wolff, H.D. Pfeiffer, H.S. Delugach (Eds.), Conceptual Structures at Work. XI, 403 pages. 2004. (Subseries LNAI).

Vol. 3126: P. Dini, P. Lorenz, J.N.d. Souza (Eds.), Service Assurance with Partial and Intermittent Resources. XI, 312 pages. 2004.

Vol. 3125: D. Kozen (Ed.), Mathematics of Program Construction. X, 401 pages. 2004.

Vol. 3124: J.N. de Souza, P. Dini, P. Lorenz (Eds.), Telecommunications and Networking - ICT 2004. XXVI, 1390 pages. 2004.

Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), Natural Language Generation. X, 219 pages. 2004. (Subseries LNAI).

Vol. 3122: K. Jansen, S. Khanna, J.D.P. Rolim, D. Ron (Eds.), Approximation, Randomization, and Combinatorial Optimization. IX, 428 pages. 2004.

Vol. 3121: S. Nikoletseas, J.D.P. Rolim (Eds.), Algorithmic Aspects of Wireless Sensor Networks. X, 201 pages. 2004.

Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), Learning Theory. X, 648 pages. 2004. (Subseries LNAI).

Vol. 3118: K. Miesenberger, J. Klaus, W. Zagler, D. Burger (Eds.), Computer Helping People with Special Needs. XXIII, 1191 pages. 2004.

Vol. 3116: C. Rattray, S. Maharaj, C. Shankland (Eds.), Algebraic Methodology and Software Technology. XI, 569 pages. 2004.

Vol. 3115: P. Enser, Y. Kompatsiaris, N.E. O'Connor, A.F. Smeaton, A.W.M. Smeulders (Eds.), Image and Video Retrieval. XVII, 679 pages. 2004.

Vol. 3114: R. Alur, D.A. Peled (Eds.), Computer Aided Verification. XII, 536 pages. 2004.

Vol. 3113: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever. X, 283 pages. 2004.

Vol. 3112: H. Williams, L. MacKinnon (Eds.), Key Technologies for Data Management. XII, 265 pages. 2004.

Vol. 3111: T. Hagerup, J. Katajainen (Eds.), Algorithm Theory - SWAT 2004. XI, 506 pages. 2004.

Vol. 3110: A. Juels (Ed.), Financial Cryptography. XI, 281 pages. 2004.

Vol. 3109: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), Combinatorial Pattern Matching. XII, 486 pages. 2004.

Vol. 3108: H. Wang, J. Pieprzyk, V. Varadharajan (Eds.), Information Security and Privacy. XII, 494 pages. 2004.

Vol. 3107: J. Bosch, C. Krueger (Eds.), Software Reuse: Methods, Techniques and Tools. XI, 339 pages. 2004.

Vol. 3106: K.-Y. Chwa, J.I. Munro (Eds.), Computing and Combinatorics. XIII, 474 pages. 2004.

Vol. 3105: S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. XVI, 304 pages. 2004.

Preface

On behalf of the Program Committee, it is our pleasure to present to you the proceedings of the 7th Symposium on Recent Advances in Intrusion Detection (RAID 2004), which took place in Sophia-Antipolis, French Riviera, France, September 15–17, 2004.

The symposium brought together leading researchers and practitioners from academia, government and industry to discuss intrusion detection from research as well as commercial perspectives. We also encouraged discussions that addressed issues that arise when studying intrusion detection, including information gathering and monitoring, from a wider perspective. Thus, we had sessions on detection of worms and viruses, attack analysis, and practical experience reports.

The RAID 2004 Program Committee received 118 paper submissions from all over the world. All submissions were carefully reviewed by several members of the Program Committee and selection was made on the basis of scientific novelty, importance to the field, and technical quality. Final selection took place at a meeting held May 24 in Paris, France. Fourteen papers and two practical experience reports were selected for presentation and publication in the conference proceedings. In addition, a number of papers describing work in progress were selected for presentation at the symposium. The keynote address was given by Bruce Schneier of Counterpane Systems. Håkan Kvarnström of TeliaSonera gave an invited talk on the topic "Fighting Fraud in Telecom Environments."

A successful symposium is the result of the joint effort of many people. In particular, we would like to thank all authors who submitted papers, whether accepted or not. Our thanks also go to the Program Committee members and additional reviewers for their hard work with the large number of submissions. In addition, we want to thank the General Chair, Refik Molva, for handling conference arrangements, Magnus Almgren for preparing the conference proceedings, Marc Dacier for finding support from our sponsors, Yves Roudier for maintaining the conference Web site, and Hervé Debar at France Télécom R&D for arranging the Program Committee meeting. Finally, we extend our thanks to the sponsors: SAP, France Télécom, and Conseil Régional Provence Alpes Côte d'Azur.

September 2004

Erland Jonsson
Alfonso Valdes

Organization

RAID 2004 was organized by Institut Eurécom and held in conjunction with ESORICS 2004.

Conference Chairs

General Chair	Refik Molva (Institut Eurécom, France)
Program Chairs	Erland Jonsson (Chalmers University of Technology, Sweden) Alfonso Valdes (SRI International, USA)
Publications Chair	Magnus Almgren (Chalmers University of Technology, Sweden)
Publicity Chair	Yves Roudier (Institut Eurécom, France)
Sponsor Chair	Marc Dacier (Institut Eurécom, France)

Program Committee

Tatsuya Baba	NTT Data Corporation, Japan
Lee Badger	DARPA, USA
Sungdeok Cha	Korea Advanced Institute of Science and Technology, Korea
Steven Cheung	SRI International, USA
Hervé Debar	France Télécom R&D, France
Simone Fischer-Hübner	Karlstad University, Sweden
Steven Furnell	University of Plymouth, UK
Dogan Kesdogan	RWTH Aachen, Germany
Christopher Kruegel	Technical University Vienna, Austria
Håkan Kvarnström	TeliaSonera AB, Sweden
Wenke Lee	Georgia Institute of Technology, USA
Roy A. Maxion	Carnegie Mellon University, USA
John McHugh	CMU/SEI CERT, USA
Ludovic Mé	Supélec, France
George Mohay	Queensland University of Technology, Australia
Vern Paxson	International Computer Science Institute and Lawrence Berkeley National Laboratory, USA
Giovanni Vigna	UCSB, USA
Andreas Wespi	IBM Zurich Research Laboratory, Switzerland
S. Felix Wu	UC Davis, USA
Diego Zamboni	IBM Zurich Research Laboratory, Switzerland

Steering Committee

Marc Dacier (Chair)	Institut Eurécom, France
Hervé Debar	France Télécom R&D, France
Deborah Frincke	Pacific Northwest National Laboratory, USA
Ming-Yuh Huang	The Boeing Company, USA
Wenke Lee	Georgia Institute of Technology, USA
Ludovic Mé	Supélec, France
Giovanni Vigna	UCSB, USA
Andreas Wespi	IBM Zurich Research Laboratory, Switzerland
S. Felix Wu	UC Davis, USA

Additional Reviewers

Magnus Almgren	Chalmers University of Technology, Sweden
Christophe Bidan	Supélec, France
Phil Brooke	University of Plymouth, UK
Sanghyun Cho	Korea Advanced Institute of Science and Technology, Korea
Andrew Clark	Queensland University of Technology, Australia
Chris Clark	Georgia Institute of Technology, USA
Marc Dacier	Institut Eurécom, France
Drew Dean	SRI International, USA
Maximillian Dornseif	RWTH Aachen, Germany
Bruno Dutertre	SRI International, USA
Ulrich Flegel	Dortmund University, Germany
Deborah Frincke	Pacific Northwest National Laboratory, USA
Anup Ghosh	DARPA, USA
Satoshi Hakomori	NTT Data Corporation, Japan
Jeffery P. Hansen	Carnegie Mellon University, USA
Anders Hansmats	TeliaSonera AB, Sweden
Hans Hedbom	Karlstad University, Sweden
Thorsten Holz	RWTH Aachen, Germany
Gary Hong	UC Davis, USA
Ming-Yuh Huang	The Boeing Company, USA
Klaus Julisch	IBM Zurich Research Laboratory, Switzerland
Jaeyeon Jung	Massachusetts Institute of Technology, USA
Kevin S. Killourhy	Carnegie Mellon University, USA
Hansung Kim	Korea Advanced Institute of Science and Technology, Korea
Oleg Kolesnikov	Georgia Institute of Technology, USA
Tobias Kölsch	RWTH Aachen, Germany
Takayoshi Kusaka	NTT Data Corporation, Japan

Additional Reviewers (continued)

Byunghee Lee	Korea Advanced Institute of Science and Technology, Korea
Stefan Lindskog	Karlstad University, Sweden
Emilie Lundin Barse	Chalmers University of Technology, Sweden
Shigeyuki Matsuda	NTT Data Corporation, Japan
Michael Meier	Brandenburg University of Technology Cottbus, Germany
Benjamin Morin	France Télécom R&D, France
Darren Mutz	UCSB, USA
Tadeusz Pietraszek	IBM Zurich Research Laboratory, Switzerland
Alexis Pimenidis	RWTH Aachen, Germany
Xinzhou Qin	Georgia Institute of Technology, USA
Rob Reeder	Carnegie Mellon University, USA
Will Robertson	UCSB, USA
Tim Seipold	RWTH Aachen, Germany
Jeongseok Seo	Korea Advanced Institute of Science and Technology, Korea
Hervé Sibert	France Télécom R&D, France
Kymie M.C. Tan	Carnegie Mellon University, USA
Axel Tanner	IBM Zurich Research Laboratory, Switzerland
Elvis Tombini	France Télécom R&D, France
Eric Totel	Supélec, France
Fredrik Valeur	UCSB, USA
Chris Vanden Berghe	IBM Zurich Research Laboratory, Switzerland
Jouni Viinikka	France Télécom R&D, France
Nicholas Weaver	International Computer Science Institute, USA
Ralf Wienzek	RWTH Aachen, Germany
Jacob Zimmerman	Supélec, France
Albin Zuccato	Karlstad University, Sweden

Table of Contents

Automatic Extraction of Accurate Application-Specific Sandboxing Policy

Lap Chung Lam and Tzi-cker Chiueh

Rether Networks, Inc.
99 Mark Tree RD Suite 301, Centereach NY 11720, USA
`lclam@cs.sunysb.edu`, `chiueh@rether.com`
`http://www.rether.com`

Abstract. One of the most dangerous cybersecurity threats is *control hijacking* attacks, which hijack the control of a victim application, and execute arbitrary system calls assuming the identity of the victim program's effective user. System call monitoring has been touted as an effective defense against control hijacking attacks because it could prevent remote attackers from inflicting damage upon a victim system even if they can successfully compromise certain applications running on the system. However, the Achilles' heel of the system call monitoring approach is the construction of accurate system call behavior model that minimizes false positives and negatives. This paper describes the design, implementation, and evaluation of a Program semantics-Aware Intrusion Detection system called *Paid*, which automatically derives an application-specific system call behavior model from the application's source code, and checks the application's run-time system call pattern against this model to thwart any control hijacking attacks. The per-application behavior model is in the form of the *sites* and *ordering* of system calls made in the application, as well as its partial control flow. Experiments on a fully working *Paid* prototype show that *Paid* can indeed stop attacks that exploit non-standard security holes, such as format string attacks that modify function pointers, and that the run-time latency and throughput penalty of *Paid* are under 11.66% and 10.44%, respectively, for a set of production-mode network server applications including Apache, Sendmail, Ftp daemon, etc.

Keywords: intrusion detection, system call graph, sandboxing, mimicry attack, non-deterministic finite state automaton

1 Introduction

Many computer security vulnerabilities arise from software bugs. One particular class of bugs allows remote attackers to hijack the control of victim programs and inflict damage upon victim machines. These *control hijacking* exploits are considered among the most dangerous cybersecurity threats because remote attackers can unilaterally mount an attack without requiring any special set-up or any actions on the part of victim users (unlike email attachment or web page download). Moreover, many production-mode network applications appear to be rife with software defects that expose such vulnerabilities. For example, in the

E. Jonsson et al. (Eds.): RAID 2004, LNCS 3224, pp. 1–20, 2004.

most recent quarterly CERT Advisory summary (03/2003) [4], seven out of ten vulnerabilities can lead to control hijacking attacks. As another example, the notorious SQL Slammer worm also relies on control hijacking attacks to duplicate and propagate itself epidemically across the net.

An effective way to defeat control-hijacking attacks is application-based anomaly intrusion detection. An application-based anomaly intrusion detection system closely monitors the activities of a process. If any activity deviates from the predefined acceptable behavior model, the system terminates the process or flags the activity as intrusion. The most common way to model the acceptable behavior of an application is to use system calls made by the application. The underlying assumption of the system call-based intrusion detection is that remote attackers can damage a victim system only by making malicious system calls once they hijack a victim application. Given that system call is the only means to inflict damage, it follows logically that by closely monitoring the system calls made by a network application at run time, it is possible to detect and prevent malicious system calls that attackers issue, and thus protect a computer system from attackers even if some of its network applications have been compromised. While the mechanics of system call-based anomaly intrusion detection is well understood, successful application of this technology requires an accurate system call model that minimizes false positives and negatives.

Wagner and Dean [22] first introduced the idea of using compiler to derive a call graph that can capture the system call ordering of an application. At run time, any system call that does not follow the statically derived order is considered as an act of intrusion and thus should be prohibited. A call graph derived from a program's control flow graph (CFG) is a non-deterministic finite-state automaton (NFA) due to such control constructs as if-then-else and function call/return. The degree of non-determinism determines the extent to which mimicry attack [23] is possible, through so-called impossible paths [22]. This paper describes the design, implementation, and evaluation of a Program semantics-Aware Intrusion Detection system called *Paid*, which consists of a compiler that can derive a deterministic finite-state automaton (DFA) model which captures the system call *sites*, system call *ordering*, and *partial control flow* from an application's source code, and an in-kernel run-time verifier that compares an application's run-time system call pattern against its statically derived system call model, even in the presence of function pointers, signals, and setjmp/longjmp calls. *Paid* features several unique techniques:

- *Paid* inlines each system call site in the program with its associated system call stub so that each system call is uniquely labeled by the return address of its corresponding `int 0x80` instruction,
- *Paid* inlines each call in the application call graph to a function having multiple call sites with the function's call graph, thus eliminating the non-determinism associated with the exit point of such functions,
- *Paid* introduces a `notify` system call that its compiler component can use to inform its run-time verifier component of information that cannot be determined statically such as function pointers, signal delivery, and to eliminate whatever non-determinism that cannot be resolved through system call inlining and graph inlining, and

- *Paid* inserts random null system calls (which are also part of the system call graph) at compile time and performs run-time stack integrity check to prevent attackers from mounting mimicry attacks.

The combination of these techniques enables *Paid* to derive an accurate DFA system call graph model from the source code of application programs, which in turn minimizes the run-time checking overhead. However, the current *Paid* prototype has one drawback: it does not perform system call argument analysis. But we will include this feature in the next version of *Paid*.

2 Related Work

2.1 System Call-Based Sandboxing

Many recent anomaly detection systems [22, 8, 18, 13, 9, 24, 15, 17] defines normal behavior model using run-time application activities. Although such systems cannot stop all attacks, they can effectively detect and stop many control hijacking attacks. Among these systems, system call pattern has become the most popular choice for modeling application behavior. However, simply keeping track of system calls may not be sufficient because it cannot capture other program information such as user-level application states.

Wagner and Dean's work [22] advocated a compiler approach to derive three different system call models, callgraph model (NFA), abstract stack or push-down automaton model (PDA), and digraph model. Among all three models, the PDA model, which models the stack operations to eliminate the impossible paths, is the most precise model, but it is also the most expensive model in terms of time and space in many cases. *Paid*'s DFA model represents a significant advance over their work. First, *Paid* uses `notify` system call, system call inlining, and graph inlining to reduce the degree of non-determinism in the input programs. Second, *Paid* uses stack integrity check and random insertion of null system calls to greatly raise the barrier for mimicry attacks. Third, *Paid* is more efficient than Wagner and Dean's system in run-time checking overhead. For example, for a single transaction, their PDA model took 42 minutes for `qpopper` and more than 1 hour for `sendmail`, whereas *Paid* only takes 0.040679 seconds for `qpopper` and 0.047133 seconds for `sendmail`.

Giffin et al. [9] extended Wagner's work to application binaries for secure remote execution. They used null system call to eliminate impossible paths in their NFA model by placing a null system call after a function call. *Paid* is different from this work because it places a null system call only where non-determinism cannot be resolved through graph inlining and system call stub inlining. As a result, *Paid* can use the DFA model to implement a simple and efficient runtime verifier inside the kernel. Giffin et al. also tried graph inlining, which they called automaton inlining. They found graph inlining increases the state space dramatically, but *Paid*'s implementation on Linux only increases the state space around 100%. This discrepancy is due to the `libc` library on Solaris. For example, for a single `socket` call, it only needs a single edge or transition on Linux, while it takes more than 100 edges on Solaris. They found numerous other library functions that share the same problem. Giffin's PDA model is similar to Wagner's model, and they used a bounded-stack to solve the infinite stack problem. However, when the stack is full, the PDA model eventually becomes a less precise NFA model. Giffin et al. also proposed a Dyck model [10] to solve non-determinism problem by placing a null system call before and after a function call to simulate stack operation. To reduce performance overhead, a null system call from a function call does not actually trap to the kernel if the function call itself does not make a system call.

Behavior blocking is a variation of system call-based intrusion detection. Behavior blocking systems run applications in a sandbox. All sandboxed applications can only have the privileges specified by the sandbox. Even if an application is hijacked, it cannot use more privileges than as specified. Existing behavior blocking systems include MAPbox [1], WindBox [3], Janus [11], and Consh [2]. The key issue of behavior blocking systems is to define an accurate sandboxing policy, which is what *Paid* is designed for.

Systems such as StackGuard [6], StackShield [21] and RAD [5,16] tried to protect the return addresses on the stack, which are common targets of buffer overflow attacks. Non-executable stack [19] prevents applications from executing any code on the stack. Another problem is that they cannot prevent attacks that target function pointers. IBM's GCC extension [7] reorders local variables and places all pointer variables at lower addresses than buffers. This technique offers some protection against buffer overflow attacks, but not buffer underflow attacks. Purify [12] instruments binaries to check each memory access at run time. However, the performance degradation and the increased memory usage are the key issues that prevent Purify from being used in production mode. Kiriansky [14] checks every branch instruction to ensure that no illegal code can be executed.

3 Program Semantics-Aware Intrusion Detection

3.1 Overview

Paid includes a compiler that automatically derives a system call site flow graph or SCSFG from an application's source code, and a DFA-based run-time verifier that checks the legitimacy of each system call. To be efficient, the run-time verifier of *Paid* is embedded in the kernel to avoid the context-switching overhead associated with user-level system call monitors. The in-kernel verifier has to be simple so that it itself does not introduce additional vulnerabilities. It also needs to be fast so as to reduce the performance overheard visible to applications. Finally it should not consume much of the kernel memory. The key challenge of *Paid* is to minimize the degree of non-determinism in the derived SCSFG such that the number of checks that the run-time verifier needs to perform is minimized.

Once an application's SCSFG is known, the attacker can also use this information to mount a mimicry attack, in which the attacker follows a legitimate path through the application's SCSFG until it reaches a system call she needs to deal the fatal blow. For example, assume an application has a buffer overflow vulnerability and the system call sequence following the vulnerability point is {`open, setreuid, write, close, exec`}, and an attacker needs `setreuid` and `exec` for her attack. After the attacker hijacks the application's control using a buffer overflow attack, she can mimic the legitimate system call sequence by interleaving calls to `open`, `write` and `close` with those to `setreuid` and `exec` properly, thus successfully fooling any intrusion detection systems that check only system call ordering. To address mimicry attacks, *Paid* applies two simple techniques: stack integrity check and random insertion of null system calls. In the next version of *Paid*, we will add a comprehensive checking mechanism on system call arguments as well.

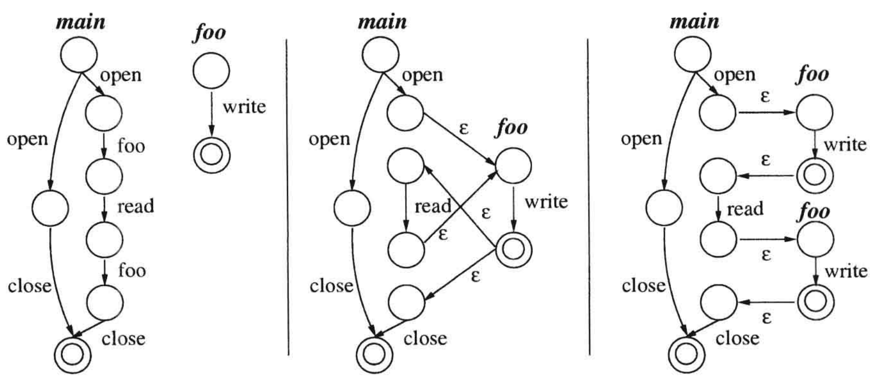

Fig. 1. Graph interlinking and graph inlining are two alternative to constructing a whole-program system call graph from the system call graphs of individual functions

3.2 From NFA to DFA

The simplest way to construct a call graph for an application is to extract a local call graph for each function from the function's CFG, and then construct the final application call graph by linking per-function local call graphs using either *graph interlinking* or *graph inlining*, which are illustrated in Figure 1. A local call graph or an application call graph is naturally an NFA because of such control constructs as if-then-else and function call/return. To remove non-determinism, we employ the following techniques: 1) system call stub inlining, 2) graph inlining, and 3) insertion of `notify` system call.

One source of non-determinism is due to functions that have many call sites. For these functions, the number of out-going edges of the final state of their local call graph is more than one, as exemplified by the function `foo` in Figure 1(B). To eliminate this type of non-determinism, we use graph inlining as illustrated in Figure 1(C). In the application call graph, each call to a function with multiple call sites points to a unique duplicate of the function's call graph, thus ensuring that the final state of each such duplicated call graph have a single out-going edge. Graph inlining can significantly increase the state space if not applied carefully. We use an ε-transition removal algorithm to remove all non-system call edges from a function's CFG before merging the per-function call graphs.