Bernhard M. Hämmerli
Robin Sommer (Eds.)

LNCS 4579

Detection of Intrusions and Malware, and Vulnerability Assessment

4th International Conference, DIMVA 2007
Lucerne, Switzerland, July 2007
Proceedings

DIMVA 2007

Springer

Bernhard M. Hämmerli Robin Sommer (Eds.)

Detection of Intrusions and Malware, and Vulnerability Assessment

4th International Conference, DIMVA 2007
Lucerne, Switzerland, July 12-13, 2007
Proceedings

Springer

Volume Editors

Bernhard M. Hämmerli
Acris GmbH und HTA Lucerne
Bodenhofstraße 29, 6005 Luzern, Switzerland
E-mail: bmhaemmerli@acris.ch

Robin Sommer
International Computer Science Institute
1947 Center St. Suite 600
Berkeley, CA 94704, USA
E-mail: robin@icsi.berkeley.edu

Library of Congress Control Number: 2007930209

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

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-73613-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73613-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12089918 06/3180 5 4 3 2 1 0

Lecture Notes in Computer Science 4579

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Lecture Notes in Computer Science

For information about Vols. 1–4511

please contact your bookseller or Springer

Vol. 4611: J. Indulska, J. Ma, L.T. Yang, T. Ungerer, J. Cao (Eds.), Ubiquitous Intelligence and Computing. XXIII, 1257 pages. 2007.

Vol. 4610: B. Xiao, L.T. Yang, J. Ma, C. Muller-Schloer, Y. Hua (Eds.), Autonomic and Trusted Computing. XVIII, 571 pages. 2007.

Vol. 4608: H.W. Schmidt, I. Crnkovic, G.T. Heineman, J.A. Stafford (Eds.), Component-Based Software Engineering. XII, 283 pages. 2007.

Vol. 4605: D. Papadias, D. Zhang, G. Kollios (Eds.), Advances in Spatial and Temporal Databases. X, 479 pages. 2007.

Vol. 4602: S. Barker, G.-J. Ahn (Eds.), Data and Applications Security XXI. X, 291 pages. 2007.

Vol. 4600: H. Comon-Lundh, C. Kirchner, H. Kirchner, Rewriting, Computation and Proof. XVI, 273 pages. 2007.

Vol. 4598: G. Lin (Ed.), Computing and Combinatorics. XII, 570 pages. 2007.

Vol. 4597: P. Perner (Ed.), Advances in Data Mining. XI, 353 pages. 2007. (Sublibrary LNAI).

Vol. 4596: L. Arge, C. Cachin, T. Jurdziński, A. Tarlecki (Eds.), Automata, Languages and Programming. XVII, 953 pages. 2007.

Vol. 4595: D. Bošnački, S. Edelkamp (Eds.), Model Checking Software. X, 285 pages. 2007.

Vol. 4594: R. Bellazzi, A. Abu-Hanna, J. Hunter (Eds.), Artificial Intelligence in Medicine. XVI, 509 pages. 2007. (Sublibrary LNAI).

Vol. 4592: Z. Kedad, N. Lammari, E. Métais, F. Meziane, Y. Rezgui (Eds.), Natural Language Processing and Information Systems. XIV, 442 pages. 2007.

Vol. 4591: J. Davies, J. Gibbons (Eds.), Integrated Formal Methods. IX, 660 pages. 2007.

Vol. 4590: W. Damm, H. Hermanns (Eds.), Computer Aided Verification. XV, 562 pages. 2007.

Vol. 4589: J. Münch, P. Abrahamsson (Eds.), Product-Focused Software Process Improvement. XII, 414 pages. 2007.

Vol. 4588: T. Harju, J. Karhumäki, A. Lepistö (Eds.), Developments in Language Theory. XI, 423 pages. 2007.

Vol. 4587: R. Cooper, J. Kennedy (Eds.), Data Management. XIII, 259 pages. 2007.

Vol. 4586: J. Pieprzyk, H. Ghodosi, E. Dawson (Eds.), Information Security and Privacy. XIV, 476 pages. 2007.

Vol. 4585: M. Kryszkiewicz, J.F. Peters, H. Rybinski, A. Skowron (Eds.), Rough Sets and Intelligent Systems Paradigms. XIX, 836 pages. 2007. (Sublibrary LNAI).

Vol. 4584: N. Karssemeijer, B. Lelieveldt (Eds.), Information Processing in Medical Imaging. XX, 777 pages. 2007.

Vol. 4583: S.R. Della Rocca (Ed.), Typed Lambda Calculi and Applications. X, 397 pages. 2007.

Vol. 4582: J. Lopez, P. Samarati, J.L. Ferrer (Eds.), Public Key Infrastructure. XI, 375 pages. 2007.

Vol. 4581: A. Petrenko, M. Veanes, J. Tretmans, W. Grieskamp (Eds.), Testing of Software and Communicating Systems. XII, 379 pages. 2007.

Vol. 4580: B. Ma, K. Zhang (Eds.), Combinatorial Pattern Matching. XII, 366 pages. 2007.

Vol. 4579: B. M. Hämmerli, R. Sommer (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment. X, 251 pages. 2007.

Vol. 4578: F. Masulli, S. Mitra, G. Pasi (Eds.), Applications of Fuzzy Sets Theory. XVIII, 693 pages. 2007. (Sublibrary LNAI).

Vol. 4577: N. Sebe, Y. Liu, Y. Zhuang (Eds.), Multimedia Content Analysis and Mining. XIII, 513 pages. 2007.

Vol. 4576: D. Leivant, R. de Queiroz (Eds.), Logic, Language, Information and Computation. X, 363 pages. 2007.

Vol. 4575: T. Takagi, T. Okamoto, E. Okamoto, T. Okamoto (Eds.), Pairing-Based Cryptography – Pairing 2007. XI, 408 pages. 2007.

Vol. 4574: J. Derrick, J. Vain (Eds.), Formal Techniques for Networked and Distributed Systems – FORTE 2007. XI, 375 pages. 2007.

Vol. 4573: M. Kauers, M. Kerber, R. Miner, W. Windsteiger (Eds.), Towards Mechanized Mathematical Assistants. XIII, 407 pages. 2007. (Sublibrary LNAI).

Vol. 4572: F. Stajano, C. Meadows, S. Capkun, T. Moore (Eds.), Security and Privacy in Ad-hoc and Sensor Networks. X, 247 pages. 2007.

Vol. 4570: H.G. Okuno, M. Ali (Eds.), New Trends in Applied Artificial Intelligence. XXI, 1194 pages. 2007. (Sublibrary LNAI).

Vol. 4569: A. Butz, B. Fisher, A. Krüger, P. Olivier, S. Owada (Eds.), Smart Graphics. IX, 237 pages. 2007.

Vol. 4566: M.J. Dainoff (Ed.), Ergonomics and Health Aspects of Work with Computers. XVIII, 390 pages. 2007.

Vol. 4565: D.D. Schmorrow, L.M. Reeves (Eds.), Foundations of Augmented Cognition. XIX, 450 pages. 2007. (Sublibrary LNAI).

Vol. 4564: D. Schuler (Ed.), Online Communities and Social Computing. XVII, 520 pages. 2007.

Vol. 4563: R. Shumaker (Ed.), Virtual Reality. XXII, 762 pages. 2007.

Vol. 4562: D. Harris (Ed.), Engineering Psychology and Cognitive Ergonomics. XXIII, 879 pages. 2007. (Sublibrary LNAI).

Vol. 4561: V.G. Duffy (Ed.), Digital Human Modeling. XXIII, 1068 pages. 2007.

Vol. 4560: N. Aykin (Ed.), Usability and Internationalization, Part II. XVIII, 576 pages. 2007.

Vol. 4559: N. Aykin (Ed.), Usability and Internationalization, Part I. XVIII, 661 pages. 2007.

Vol. 4558: M.J. Smith, G. Salvendy (Eds.), Human Interface and the Management of Information, Part II. XXIII, 1162 pages. 2007.

Vol. 4557: M.J. Smith, G. Salvendy (Eds.), Human Interface and the Management of Information, Part I. XXII, 1030 pages. 2007.

Vol. 4554: C. Stephanidis (Ed.), Universal Acess in Human Computer Interaction, Part I. XXII, 1054 pages. 2007.

Vol. 4553: J.A. Jacko (Ed.), Human-Computer Interaction, Part IV. XXIV, 1225 pages. 2007.

Vol. 4552: J.A. Jacko (Ed.), Human-Computer Interaction, Part III. XXI, 1038 pages. 2007.

Vol. 4551: J.A. Jacko (Ed.), Human-Computer Interaction, Part II. XXIII, 1253 pages. 2007.

Vol. 4550: J.A. Jacko (Ed.), Human-Computer Interaction, Part I. XXIII, 1240 pages. 2007.

Vol. 4549: J. Aspnes, C. Scheideler, A. Arora, S. Madden (Eds.), Distributed Computing in Sensor Systems. XIII, 417 pages. 2007.

Vol. 4548: N. Olivetti (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. X, 245 pages. 2007. (Sublibrary LNAI).

Vol. 4547: C. Carlet, B. Sunar (Eds.), Arithmetic of Finite Fields. XI, 355 pages. 2007.

Vol. 4546: J. Kleijn, A. Yakovlev (Eds.), Petri Nets and Other Models of Concurrency – ICATPN 2007. XI, 515 pages. 2007.

Vol. 4545: H. Anai, K. Horimoto, T. Kutsia (Eds.), Algebraic Biology. XIII, 379 pages. 2007.

Vol. 4544: S. Cohen-Boulakia, V. Tannen (Eds.), Data Integration in the Life Sciences. XI, 282 pages. 2007. (Sublibrary LNBI).

Vol. 4543: A.K. Bandara, M. Burgess (Eds.), Inter-Domain Management. XII, 237 pages. 2007.

Vol. 4542: P. Sawyer, B. Paech, P. Heymans (Eds.), Requirements Engineering: Foundation for Software Quality. IX, 384 pages. 2007.

Vol. 4541: T. Okadome, T. Yamazaki, M. Makhtari (Eds.), Pervasive Computing for Quality of Life Enhancement. IX, 248 pages. 2007.

Vol. 4539: N.H. Bshouty, C. Gentile (Eds.), Learning Theory. XII, 634 pages. 2007. (Sublibrary LNAI).

Vol. 4538: F. Escolano, M. Vento (Eds.), Graph-Based Representations in Pattern Recognition. XII, 416 pages. 2007.

Vol. 4537: K.C.-C. Chang, W. Wang, L. Chen, C.A. Ellis, C.-H. Hsu, A.C. Tsoi, H. Wang (Eds.), Advances in Web and Network Technologies, and Information Management. XXIII, 707 pages. 2007.

Vol. 4536: G. Concas, E. Damiani, M. Scotto, G. Succi (Eds.), Agile Processes in Software Engineering and Extreme Programming. XV, 276 pages. 2007.

Vol. 4534: I. Tomkos, F. Neri, J. Solé Pareta, X. Masip Bruin, S. Sánchez Lopez (Eds.), Optical Network Design and Modeling. XI, 460 pages. 2007.

Vol. 4533: F. Baader (Ed.), Term Rewriting and Applications. XII, 419 pages. 2007.

Vol. 4531: J. Indulska, K. Raymond (Eds.), Distributed Applications and Interoperable Systems. XI, 337 pages. 2007.

Vol. 4530: D.H. Akehurst, R. Vogel, R.F. Paige (Eds.), Model Driven Architecture- Foundations and Applications. X, 219 pages. 2007.

Vol. 4529: P. Melin, O. Castillo, L.T. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), Foundations of Fuzzy Logic and Soft Computing. XIX, 830 pages. 2007. (Sublibrary LNAI).

Vol. 4528: J. Mira, J.R. Álvarez (Eds.), Nature Inspired Problem-Solving Methods in Knowledge Engineering, Part II. XXII, 650 pages. 2007.

Vol. 4527: J. Mira, J.R. Álvarez (Eds.), Bio-inspired Modeling of Cognitive Tasks, Part I. XXII, 630 pages. 2007.

Vol. 4526: M. Malek, M. Reitenspieß, A. van Moorsel (Eds.), Service Availability. X, 155 pages. 2007.

Vol. 4525: C. Demetrescu (Ed.), Experimental Algorithms. XIII, 448 pages. 2007.

Vol. 4524: M. Marchiori, J.Z. Pan, C.d.S. Marie (Eds.), Web Reasoning and Rule Systems. XI, 382 pages. 2007.

Vol. 4523: Y.-H. Lee, H.-N. Kim, J. Kim, Y. Park, L.T. Yang, S.W. Kim (Eds.), Embedded Software and Systems. XIX, 829 pages. 2007.

Vol. 4522: B.K. Ersbøll, K.S. Pedersen (Eds.), Image Analysis. XVIII, 989 pages. 2007.

Vol. 4521: J. Katz, M. Yung (Eds.), Applied Cryptography and Network Security. XIII, 498 pages. 2007.

Vol. 4519: E. Franconi, M. Kifer, W. May (Eds.), The Semantic Web: Research and Applications. XVIII, 830 pages. 2007.

Vol. 4517: F. Boavida, E. Monteiro, S. Mascolo, Y. Koucheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 382 pages. 2007.

Vol. 4516: L. Mason, T. Drwiega, J. Yan (Eds.), Managing Traffic Performance in Converged Networks. XXIII, 1191 pages. 2007.

Vol. 4515: M. Naor (Ed.), Advances in Cryptology - EUROCRYPT 2007. XIII, 591 pages. 2007.

Vol. 4514: S.N. Artemov, A. Nerode (Eds.), Logical Foundations of Computer Science. XI, 513 pages. 2007.

Vol. 4513: M. Fischetti, D.P. Williamson (Eds.), Integer Programming and Combinatorial Optimization. IX, 500 pages. 2007.

Preface

On behalf of the Program Committee, it is our pleasure to present to you the proceedings of the 4th GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA). Each year DIMVA brings together international experts from academia, industry and government to present and discuss novel security research. DIMVA is organized by the special interest group Security—Intrusion Detection and Response of the German Informatics Society (GI).

The DIMVA 2007 Program Committee received 57 submissions from 20 different countries. All submissions were carefully reviewed by Program Committee members and external experts according to the criteria of scientific novelty, importance to the field and technical quality. The final selection took place at a Program Committee meeting held on March 31, 2007, at Università Campus Bio-Medico di Roma, Italy. Twelve full papers and two extended abstracts were selected for presentation at the conference and publication in the conference proceedings. The conference took place during July 12–13, 2007, at the University of Applied Sciences and Arts Lucerne (HTA Lucerne) in Switzerland. The program featured both theoretical and practical research results grouped into five sessions. The keynote speech was given by Vern Paxson, International Computer Science Institute and Lawrence Berkeley National Laboratory. Another invited talk was presented by Marcelo Masera, Institute for the Protection and Security of the Citizen. Peter Trachsel, Deputy Head of the Federal Strategic Unit for IT in Switzerland, gave a speech during the conference dinner. The conference program further included a rump session organized by Sven Dietrich of Carnegie Mellon University; and it was complemented by the third instance of the European capture-the-flag contest CIPHER, organized by Lexi Pimenidis of RWTH Aachen.

We sincerely thank all those who submitted papers as well as the Program Committee members and our external reviewers for their valuable contributions to a great conference program.

For further details about DIMVA 2007, please refer to the conference Web site at `http://www.dimva.org/dimva2007`.

July 2007

Bernhard Hämmerli
Robin Sommer

Organization

Organizing Committee

General Chair	Bernhard Hämmerli (HTA Luzern)
Program Chair	Robin Sommer (LBNL/ICSI)
Sponsor Chair	Dirk Schadt

Program Committee

Roland Büschkes	RWE, Germany
Weidong Cui	Microsoft Research, USA
Marc Dacier	Eurécom, France
Hervé Debar	France Télécom, France
Sven Dietrich	Carnegie Mellon University, USA
Toralv Dirro	McAfee, Germany
Holger Dreger	Siemens CERT, Germany
Mohamed Eltoweissy	Virginia Tech, USA
Ulrich Flegel	University of Dortmund, Germany
Felix C. Freiling	University of Mannheim, Germany
Dirk Häger	BSI, Germany
Bernhard Hämmerli	HTA Lucerne, Switzerland
Marc Heuse	n.runs, Germany
Ming-Yuh Huang	Boeing, USA
Erland Jonsson	Chalmers University, Sweden
Klaus Julisch	IBM Research, USA
Angelos Keromytis	Columbia University, USA
Hartmut König	BTU Cottbus, Germany
Christian Kreibich	ICSI, USA
Christopher Kruegel	TU Vienna, Austria
Pavel Laskov	Fraunhofer FIRST, Germany
Wenke Lee	Georgia Tech, USA
Jun Li	Tsinghua University, China
Javier Lopez	University of Malaga, Spain
John McHugh	Dalhousie University, Canada
Michael Meier	University of Dortmund, Germany
R. Sekar	Stony Brook University, USA
Roberto Setola	Univ. CAMPUS Bio-Medico Rome, Italy
Doug Tygar	UC Berkeley, USA
Giovanni Vigna	UC Santa Barbara, USA

External Reviewers

Periklis Akritidis
Michael Collins
Jason Franklin
Thorsten Holz
Corrado Leita
Vern Paxson
Sebastian Schmerl
Jouni Viinikka
Thomas Biege
Gabriela Cretu
Jan Goebel
Engin Kirda
Igor Nai
Michalis Polychronakis
Yingbo Song
Nicholas Weaver
Matt Burnside
Michael E. Locasto
Van Hau Pham
Ulf Larson
Peng Ning
Maurizio Sajeva
Olivier Thonnard

Steering Committee

Chairs
Ulrich Flegel (University of Dortmund)
Michael Meier (University of Dortmund)

Members
Roland Büschkes (RWE)
Christopher Kruegel (TU Vienna)
Marc Heuse (n.runs)
Pavel Laskov (Fraunhofer FIRST)
Klaus Julisch (IBM Research)

DIMVA 2007 was organized by the Special Interest Group Security – Intrusion Detection and Response (SIDAR) of the German Informatics Society (GI), in cooperation with the IEEE Task Force on Information Assurance and the Information Security Society Switzerland.

Support

The main sponsor of DIMVA 2007 was Secude Headquarter, Lucerne, Switzerland. We sincerely thank them for their support.

Table of Contents

Web Security

Intrusion Detection

Traffic Analysis

Network Security

Host Security

Extensible Web Browser Security

Mike Ter Louw, Jin Soon Lim, and V.N. Venkatakrishnan

Department of Computer Science,
University of Illinois at Chicago
{mter,jlim,venkat}@cs.uic.edu

Abstract. In this paper we examine the security issues in functionality extension mechanisms supported by web browsers. Extensions (or "plug-ins") in modern web browsers enjoy unlimited power without restraint and thus are attractive vectors for malware. To solidify the claim, we take on the role of malware writers looking to assume control of a user's browser space. We have taken advantage of the lack of security mechanisms for browser extensions and have implemented a piece of malware for the popular Firefox web browser, which we call BROWSER-SPY, that requires no special privileges to be installed. Once installed, BROWSER-SPY takes complete control of a user's browser space and can observe all the activity performed through the browser while being undetectable. We then adopt the role of defenders to discuss defense strategies against such malware. Our primary contribution is a mechanism that uses code integrity checking techniques to control the extension installation and loading process. We also discuss techniques for runtime monitoring of extension behavior that provide a foundation for defending threats due to installed extensions.

1 Introduction

The Internet web browser, arguably the most commonly used application on a network connected computer, is becoming an increasingly capable and important platform for millions of today's computer users. The web browser is often a user's window to the world, providing them an interface to perform a wide range of activity including email correspondence, shopping, social networking, personal finance management, and professional business.

This usage gives the browser a unique perspective; it can observe and apply contextual meaning to sensitive information provided by the the user during very personal activities. Furthermore, the browser has access to this information *in the clear*, even when the user encrypts all incoming and outgoing communication. This high level of access to sensitive, personal data warrants efforts to ensure its complete confidentiality and integrity.

Ensuring that the entire code base of a browser addresses the security concerns of confidentiality and integrity is a daunting task. For instance, the current distribution of the Mozilla Firefox browser has a build size of 3.7 million lines of code (as measured using the `kloc` tool) written in a variety of languages that include C, C++, Java, JavaScript and XML. These challenges of size and implementation language diversity make it difficult to develop a "one-stop shop" solution for this problem. In this paper, we focus on the equally significant subproblem of ensuring confidentiality and integrity in a

B. M. Hämmerli and R. Sommer (Eds.): DIMVA 2007, LNCS 4579, pp. 1–19, 2007.

browser in the presence of browser extensions. We discuss this problem in the context of Mozilla Firefox, the widely used free (open source) software browser, used by about 70 million web users [1].

Browser extensions (or "add-ons") are facilities provided to customize the browser. These extensions make use of interfaces exported by the browser and other plug-ins to alter the browser's behavior. Though the build of Firefox is platform-specific (such as one for Windows XP, Linux or Mac OS X), extensions are primarily platform-independent based on the neutral nature of JavaScript and XML, the predominant languages used to implement them.

Even though extensions plug directly into the browser, there is no provision currently in Firefox to provide protection against malicious extensions. One way to do this is to disallow extensions altogether. Firefox is able to do this when started in debugging mode, which prevents any extension code to be loaded. However, typical installation and execution in the normal mode allow extensions to be executed. Extensions offer useful functionality, as evidenced by the popularity of their download numbers [2], to several thousands of users who use them. Dismissing the security concerns about extensions by turning them off ignores the problem.

To understand the impact of running a malicious extension, we set for ourselves the goal of actually crafting one. Surprisingly, we engineered a malicious extension for the Firefox browser we call BROWSERSPY, with modest efforts and in less than three weeks. Once installed, this extension takes complete control of the browser. As further testimony, a recent attack was launched on the Firefox browser using a malware extension known as FormSpy [8], that elicited widespread media coverage and concern about naive users.

There are two main problems raised by the presence of our malware extension and the FormSpy extension:

- *Browser code base integrity* A malicious extension can compromise the integrity of the browser code base when it is installed and loaded. We demonstrate (by construction) that a malicious extension can subvert the installation process, take control of a browser, and hide its presence completely.
- *User data confidentiality and integrity* A malicious extension can read and write confidential data sent and received by the user, even over an encrypted secure connection. We demonstrate this by having our extension collect sensitive data input by a user while browsing and log it to a remote site.

In this paper we present techniques that address these problems. To address extension integrity, our solution empowers the end-user with complete control of the process by which code is selected to run as part of the browser, thereby disallowing installation integrity threats due to malware. This is done by a process of *user authorization* that detects and refuses to allow the execution of extensions that are not authorized by the end user.

To address the second challenge of data confidentiality and integrity, we augment the browser with support for policy-based monitoring of extensions by interposition mechanisms retrofitted to the Spidermonkey JavaScript engine and other means (Section 5).

A key benefit of our solution is that it is targeted to *retrofit* the browser. We consider this benefit very important, and have traded off potentially better solutions to achieve

this benefit. Other benefits of our approach are that it is convenient, user-friendly and poses very acceptable overheads. Our implementation is robust, having been tested with several Firefox extensions.

This paper is organized as follows. A discussion of related work appears in Section 2. We present the main details behind our malware extension in Section 3. We present our solution to the extension integrity problem in Section 4 and address data confidentiality in Section 5. We evaluate these approaches with several Firefox add-ons and discuss their performance in the above sections individually. In Section 6 we conclude.

2 Related Work

We examined extension support in four contemporary browsers: Firefox, Internet Explorer (IE), Safari and Opera. Among the four browsers that we studied, only the Safari browser does not support the concept of extensions. The remaining three possess extensible architecture but do not have security mechanisms addressing extension-based threats. For instance, IE primary extension mechanism is through Browser Helper Objects (BHO). The PestPatrol malware detection website lists hundreds of malware that use BHOs [5]. Furthermore, the integrity and confidentiality of the end-user's private data used in the browser is also not addressed in recent mechanisms such as "protected-browser-mode" [4] in Windows Vista.

The problem of safely running extensions in a browser is in many ways similar to the problem of executing downloaded untrusted code in an operating system. This is a well known problem, and has propelled research in ideas such as signed code, static analysis, proof-carrying code, model-carrying code and several execution monitoring approaches. Below, we discuss the applicability of these solutions to the browser extension problem highlighting several technical and practical reasons.

Signed code. The Firefox browser provides support for signed extensions; however, this is hardly used in practice. A search of extensions in the Firefox extensions repository `addons.mozilla.org` revealed several thousand unsigned extensions and only two that were signed. In addition, we note that signed extensions merely offer a first level of security. They only guarantee that they are from the browser distribution site and are unmodified in transit; no assurance is provided regarding the security implications of running the extension.

Static analysis. A very desirable approach for enforcing policies on extension code is by use of static analysis. Static analysis has been employed in several past efforts in identifying vulnerabilities or malicious intent. The primary advantages of using static analysis are the absence of execution overhead and runtime aborts, which are typical of dynamic analysis based solutions.

It is difficult to employ static analysis for JavaScript code without making conservative assumptions, however. A first example is the *eval* statement in JavaScript that allows a string to be interpreted as executable code. Without knowing the runtime values of the arguments to the *eval* statement, it is extremely difficult—if not impossible—to determine the runtime actions of the script. Another problem is tracing the flow of object references in a prototype-based object oriented language such as JavaScript. For

instance, variable assignment to or from an array element or object property (when the object is indexed as an associative array) can decisively hamper the tracking of object reference flow as references are stored or retrieved.

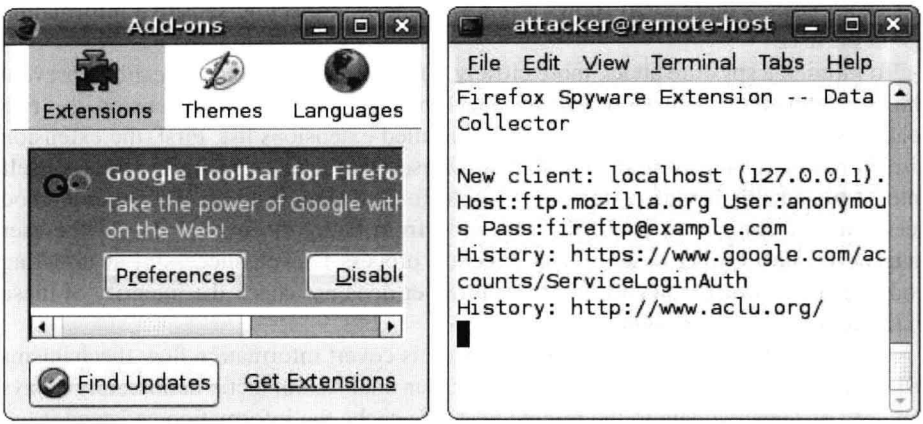
Consequently, recent efforts that trace JavaScript code [11] use runtime approaches to track references. An exception is [13] that employs static analysis for JavaScript for detecting *cross-site scripting* (XSS) attacks. In their approach, scenarios like the above are handled by a conservative form of tainting. This is suitable for their purpose of preventing XSS attacks as evidenced by their experimental results, and the fact that typical scripts from web pages are not expected to have complex *eval* constructs. This approach is unsuitable for statically analyzing extension code in JavaScript, however. Almost half (45%) of the extensions that we tested make heavy use of complex *eval* constructs, while all generously use objects as associative arrays, making static analysis very hard.

PCC and MCC. The difficulties for static analysis make frameworks such as proof-carrying code (PCC) [10] unsuitable for this problem. It will be difficult to produce proofs for extensions that make heavy use of constructs such as *eval* as part of their code. The typical approach to employ PCC in scenarios that require runtime data is to: (a) transform the original script with runtime checks that enforce the desired security property, and (b) produce a proof that the transformed program respects this property. The proof in this case is primarily used to demonstrate the correctness of the placement of runtime checks.

In the browser situation, transformation needs to be made before all *eval* statements. Policy enforcement would still be carried out by runtime checks, and therefore we did not adopt this route of using PCC. Another solution is model-carrying-code [12] which employs runtime techniques to learn the behavior of code that will be downloaded. The difficulty in using this approach is in obtaining test suites for exhaustive code coverage required for approaches based on runtime learning of models.

Execution monitoring. Several execution monitoring techniques [14,6,7] have previously looked at the problem of safely executing malicious code. The closest related project to our approach is by Hallaraker and Vigna [7]. This was the first work that looked at the security issues of executing malicious code in a large mainstream browser. Their focus is on protection against pages with malicious content rather than the ensuring the integrity of a browser's internal operations. For them it is not necessary to address the problem of browser code integrity, as scripts from web pages are sandboxed to prevent them from performing sensitive actions. In contrast we address the extension installation integrity problem, as extension code is unmonitored and can perform many sensitive operations.

To effectively regulate extension behavior, a runtime monitor must be able to determine the particular extension responsible for each operation. A direct adaptation of their execution monitoring approach does not provide this ability, and is therefore not suited for runtime supervision of extensions. To fill this void we describe two new *action attribution* mechanisms making use of browser facilities and JavaScript interposition in Section 5.



(a) Extension hiding from the browser UI. (b) Data collector receiving sensitive information.

Fig. 1. Two views of the BROWSERSPY extension in operation.

3 A Malware Extension

To gain a better understanding of the threat posed by a malware extension, we set ourselves the task of actually writing one. The motivations for creating the malicious software are to: (a) help us identify the scope of threats malicious extensions pose by understanding the facilities available to an extension in a browser, (b) increase our understanding of architecture-level and implementation-level weaknesses in the browser's extension manager, (c) give us a practical estimate in understanding the ease with which malware writers may be able to craft such extensions, and (d) provide a concrete implementation of a malicious extension to serve as a benchmark for malware analysis.

Extension Capabilities. BROWSERSPY, the extension we authored, is capable of harvesting every piece of form data (e.g., passwords) submitted by the user, including those sent over encrypted connections. Furthermore, once the extension enters the system, it ensures that it remains undetectable by users (Figure 1 (a)).

Once BROWSERSPY is installed, it begins collection of personal data that will ultimately fall into the hands of an attacker. As a user navigates the Internet, BROWSERSPY harvests the URLs comprising their browsing history and stores them in a cache. Any username and password pairs that are stored in Firefox's built-in password manager are retrieved, along with the URL of the site they pertain to. Form data that the user submits finds its way into the extension as well. All of this information is stored and periodically sent over the network to a remote host.

Given enough data the spy can effectively steal the identity of the person using the browser. Intercepted form fields can give an attacker credit card numbers, street addresses, Social Security Numbers, and other highly sensitive information. The username / password pairs can readily provide access to the user's accounts on external sites. The history items can give the attacker a profile of the victim's browsing patterns, and serve as candidate sites for further break-in attempts using the retrieved set of username /

password pairs. Figure 1 (b) shows a remote window collecting sensitive information about the user.

To mimick a spyware attack more closely, BROWSERSPY employs stealth to prevent the user from knowing that anything unusual is being conducted. The extension uses two techniques to shroud itself from Firefox's installed extensions list. First, the extension simply removes itself from the list so that the user won't see it. Second, it injects itself into a (presumably benign) extension, Google Toolbar (Figure 1 (a)). The latter method serves as a technique to guard the extension from being discovered should the user inspect the files on her system. The injection process is even successful at infecting code signed browser extensions,[1] as the browser does not check the integrity of these extensions following installation.

A common technique practiced by malware is covert information flow mechanisms [9] for transmission. To mimic this behavior, our final stealth tactic deliberately delays delivery of sensitive data to the remote host. We cache the information and send it out in periodic bursts to offset the network activity from the event that triggers it, making it harder for an observant user to correlate the added traffic with security sensitive operations. Thus, the composite effect of some relatively easy measures employed by our extension is alarming.

Extension entry vectors. The typical process of extension installation requires the user to download and install the extension through a browser interface window. Though the BROWSERSPY extension can be installed this way, this is not the only route by which this malicious extension can be delivered to a browser. It can be delivered by preexisting malware on the system without involving the browser. It can also be delivered outside the browser given user account access for a short duration. These entry vectors are all too common with unpatched systems, public terminals, and naive users who do not suspect such threats.

Extension development effort. Very little effort was required to create this extension. The lack of any security in the browser's Extension Manager module assisted in its speedy creation. It only took one graduate student (who had no prior experience in developing extensions) three weeks working part time to complete this extension. We present this information merely to argue the ease with which this task can be accomplished. We note that this period of three weeks is merely an upper bound of effort for creating malicious extensions. Malware writers have more resources, experience and time to create extensions that could be more stealthy, perhaps employing increasingly sophisticated covert mechanisms for information transmission.

Our implementation techniques. We started by studying the procedure of how extensions are created, installed and executed in the system. Firefox extensions make use of the *Cross-Platform Component Object Model* (XPCOM) framework, which provides a variety of services within the browser such as file access abstraction. We carefully studied interfaces to the XPCOM framework available for use by an extension, and discerned that one could easily program event observers for various operations performed

[1] Case in point, the code in the Google Toolbar extension is signed by Google, Inc.