Author(s): Paolo Falcarin, Christian Collberg, Mikhail Atallah and Mariusz Jakubowski
Title: Software protection
Year of publication: 2011
Citation: Falcarin, P., Collberg, C., Atallah, M. and Jakubowski, M. (2011) 'Software protection'. IEEE Software, 28(2), pp. 24-27

Link to published version:
http://www.computer.org/portal/web/csdl/abs/html/mags/so/2011/02/mso2011020024.htm

IEEE Software - March/April 2011 (Vol. 28, No. 2) pp. 24-27

Software Protection

Paolo Falcarin , University of East London
Christian Collberg , University of Arizona
Mikhail Atallah , Purdue University
Mariusz Jakubowski , Microsoft Research

A computer system's security can be compromised in many ways—a denial-of-service attack can make a server inoperable, a worm can destroy a user's private data, or an eavesdropper can reap financial rewards by inserting himself in the communication link between a customer and her bank through a man-in-the-middle (MITM) attack. What all these scenarios have in common is that the adversary is an untrusted entity that attacks a system from the outside—we assume that the computers under attack are operated by benign and trusted users. But if we remove this assumption, if we allow anyone operating a computer system—from system administrators down to ordinary users—to compromise that system's security, we find ourselves in a scenario that has received comparatively little attention.

Attacks by a trusted user on a computer system, called a man-at-the-end (MATE) scenario, can take many forms. In a tampering attack, an adversary violates the integrity of a piece of software under his control, perhaps by modifying it in ways the software vendor didn't intend. In a malicious reverse-engineering attack, he violates the vendor's confidentiality rights by extracting intellectual property from the software, such as algorithms, designs, or implementations. Finally, in a cloning attack, he violates copyright laws by making and distributing illegal copies of the software.

Methods for protecting against MATE attacks are variously known as anti-tamper techniques, digital asset protection, or, more commonly, software protection. [1] To protect against tampering attacks, tamper-proofing techniques look for modifications to a program, and when they find them, take offensive or defensive measures. For example, a tamper-proof program can repair itself, refuse to run, run but produce erroneous results, destroy objects in its environment, or "phone home" to alert the vendor that an integrity violation has taken place. To protect against reverse-engineering attacks, obfuscation techniques modify a program (once at compile-time or continuously at runtime) to make it harder for the adversary to analyze or comprehend. To protect against cloning attacks, software watermarking techniques modify a program to tie it uniquely to its legitimate user. Some software protection techniques are software-only, whereas others augment software-based protection techniques with various forms of tamper-resistant hardware, such as smart cards, trusted platform modules (TPMs), or crypto-processors.

Software protection is increasingly becoming an important requirement for industrial software development, especially when building systems for military defense, national infrastructure, and medical informatics. Every software vendor should be aware of the potential for MATE-style attacks against its products and the techniques available to mitigate these attacks. Employing software protection techniques can mean the difference between business survival and failure. We're therefore pleased to present this special issue on new tools and techniques for software protection.

Applications of Software Protection

At its core, a MATE attack occurs when an adversary gains physical access to a device and compromises it by inspecting or tampering with the hardware itself or the software it contains. Historically, software protection first appeared as (often feeble) attempts at adding license-checking code to computer games, followed by algorithms for white-box cryptography [2] used for digital-media piracy protection. Today, however, we see an increasing number of application areas susceptible to MATE attacks, along with a need for comprehensive software protection techniques that provide a nontrivial level of security against concerted attacks by accomplished adversaries.

More and more, our national infrastructure relies on distributed software systems. An example is the Advanced Metering Infrastructure (AMI) that lets utility companies collect electricity usage information from users' so-called smart meters. It also allows them, among other things, to shut down or limit services to a group of customers in response to power shortages. A MATE attack against a smart meter could not only help a user avoid paying electricity bills, but may also have more far-reaching consequences—for example, a terrorist who can compromise a smart meter could conceivably cause nationwide blackouts. [3] Likewise, with wireless sensor networks increasingly in use by the military to monitor combat zones, an adversary who can compromise a sensor node can insert false data into the system or refuse to forward data received from other sensors.

Computer and video games now form a significant part of the US and world economies: in 2009, US retail sales of such games added $4.9 billion to the US gross domestic product, [4] and in 2011, the US virtual-goods market will reach $2.1 billion (www.insidevirtualgoods.com/us-virtual-goods). Thus, in the past, cheating in massively multiplayer online games was only a problem for game vendors concerned with losing revenue, but rampant counterfeiting of virtual goods could now devalue online currencies and by extension impact the global economy.

Finally, our most private information, including our medical and financial records, is stored in electronic form. On the surface, it might seem that standard practices, such as encrypting transmitted documents, would be enough to guarantee the confidentiality and integrity of such records. However, this ignores MATE-style attacks that target the endpoints of data transmission. For example, in a medical record scenario, an adversary could bypass the two-factor authentication used to protect documents downloaded to a PC in a doctor's office (say, smart card plus password) by compromising the smart-card reader or the password-checking software. This problem is exacerbated when documents are stored on mobile devices, such as laptops and PDAs, outside a physically secured environment. Similar issues arise for classified military documents or documents containing confidential intellectual property.

Techniques for Software Protection

The software protection problem is fundamentally harder (and, under very general circumstances, impossible to solve [5]) than other, more commonly studied security problems. The reason is the very liberal attack model that software protection researchers and practitioners must contend with: we must assume an all-powerful adversary who has full access to our software and hardware, and can examine, probe, and modify it at will. For this reason, no piece of software, however well protected, is expected to survive unscathed in the wild for an extended period of time. Thus, we can look forward to a situation in which we must continuously monitor the advances our adversaries make and invent new software protection techniques to counter these advances. Designing such renewable protection techniques is at the forefront of software protection research.

Software protection algorithms fall into four basic categories: code obfuscation to make programs harder to reverse engineer, tamper-proofing to make programs harder to modify, watermarking to allow programs to be tracked, and birthmarking to detect code that has been lifted from one program into another. Hardware-supported protection techniques can be used to augment software-based methods.

Fred Cohen first discussed code obfuscation as a technique for automatically creating multiple versions of the same program, thereby making each version a more difficult target for malware to analyze and modify. [6] Ironically, today's viruses use such code diversity to deter analysis by virus scanners. Typical code obfuscation techniques include splitting code into smaller pieces, merging pieces of unrelated code, randomizing code placement, randomizing instruction selection, breaking abstraction boundaries, mapping original data structures to cloaked ones, and flattening or introducing bogus control flow. [7] Obfuscating code transformations can be either static (done at compile time) or dynamic (performed continuously during runtime). [8]

Tamper-proofing techniques fall into three basic categories. Introspection techniques check the integrity of executable code, often by computing a checksum over the code and comparing to an expected value. Higher levels of protection can be achieved by including a large number of redundant and overlapping checkers. [9] Oblivious techniques indirectly verify code integrity by checking that the data values the code computes, along with paths taken through the code, are correct. [10] Finally, environment-checking techniques verify that the program is running on an operating system and hardware that are unadulterated and won't violate code integrity. Tamper-proofing is often combined with obfuscation. This not only makes the program hard to understand (and hence modify!) but also helps hide any inserted tamper-proofing code from the adversary.

Other works have introduced a trusted server to provide remote attestation of client integrity, by extending these tamper-proofing techniques for distributed systems [11] or by detecting anomalous execution times. [12] Watermarking techniques embed an identifier, known as the fingerprint, into every distributed copy of a program. This fingerprint uniquely identifies who originally bought a particular copy of the program, allowing a pirated copy to be traced back to the original customer. Popular watermarking algorithms use a few simple program structures in which to encode the fingerprint: parts of the program can be reordered, nonfunctional code can be inserted, particular instruction sequences can be selected, and bogus data structures encoding the fingerprint can be built at runtime. Watermarking is often combined with tamper-proofing to make the fingerprint difficult to remove, as well as with obfuscation to make it difficult to locate.

In This Issue

The first article, "CodeBender: A Tool for Remote Software Protection Using Orthogonal Replacement" by Mariano Ceccato and Paolo Tonella, presents a tool that makes it possible for a remote trusted server to protect from tampering the client code running on a potentially malicious host. The approach consists of frequent replacement of the client code with code that's orthogonal in the sense that any analysis of the previous one would be of no benefit in attacking the new one (even though they have the same functionality). The adversary is prevented from circumventing such replacements by strongly coupling the client version with the trusted server, thereby making the replacement effectively mandatory.

The second article, "The Trusted Platform Agent" by Giovanni Cabiddu, Emanuele Cesena, Roberto Sassu, Davide Vernizzi, Gianluca Ramunno, and Antonio Lioy, presents an open source library for writing applications that make effective use of tamper-resistant Trusted Platform Module chips, which are increasingly available on computers.

The third article, "Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations" by Akito Monden, Satoshi Okahara, Yuki Manabe, and Kenichi Matsumoto, explores various metrics for detecting whether commercial software contains open source software (thereby violating the open source license). This not only protects open source developers but also the commercial companies that are often unaware that their products contain such unauthorized use of software (outsourced production of software might be implicated). The metrics quantify the amount of cloning—the duplication of code fragments—that exists between two given software products.

The fourth article, "Managing Copyrights and Moral Rights of Service-Based Software" by G.R. Gangadharan and Vincenzo D'Andrea, deals with copyright issues arising in the software-as-a-service framework, particularly the facet involving the expression of usage and access rights via a service license based on ODRL-S, an extension of the ODRL open standard that's gaining wide acceptance.

Finally, the Point/Counterpoint features Jean-Daniel Aussel and Reiner Sailer discussing the efficacy of hardware-assisted protections, and Yuan Xiang Gu, Bart Preneel, and Brecht Wyseur sharing their vision of the future of software-based protections.

Conclusions

Like many indispensable technologies, software protection can be a double-edged sword, particularly with malware increasingly resistant against reverse engineering and eradication. However, this only underscores the importance of software protection research—we must understand both black- and white-hat techniques for analysis of threats and protection of assets. We hope this issue educates and inspires both researchers and practitioners to push software protection forward, creating a more secure, stable foundation for the computing systems of today and tomorrow.

References

1. C. Collberg and J Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection,* Addison-Wesley, 2009.
2. B. Wyseur, "White-Box Cryptography," PhD thesis, Computer Security and Industrial Cryptography (COSIC), Dept. of Electrical Engineering, Katholieke Universiteit Leuven, 2009.
3. T. Claburn, "Smart Grid Lacks Smart Security," *InformationWeek,*24 Mar. 2009; www.informationweek.com/news/infrastructure/managementshowArticle.jhtml?articleID=216200240 .
4. S.E. Siwek, "Video Games in the 21st Century: The 2010 Report," Entertainment Software Assoc., 2010; www.theesa.com/facts/pdfsVideoGames21stCentury_2010.pdf .
5. B. Barak et al., "On the (Im)possibility of Obfuscating Programs," *Advances in Cryptology* (CRYPTO 01), LNCS 2139, Springer Verlag, 2001, pp. 1–18.
6. F.B. Cohen, "Operating System Protection through Program Evolution," *Computer Security,* vol. 12, no. 6, 1993, pp. 565–584.
7. C. Wang, "A Security Architecture for Survivability Mechanisms," PhD thesis, Dept. of Computer Science, Univ. Virginia, 2000.

8. P. Falcarin et al., "Exploiting Code Mobility for Dynamic Binary Obfuscation," to be published in *Proc. IEEE World Conf. Internet Security* (WorldCIS), Feb. 2011.
9. H. Chang and M.J. Atallah, "Protecting Software Code by Guards," *ACM Workshop Security and Privacy in Digital Rights Management,* ACM Press, 2001, pp. 160–175.
10. M. Jacob, M.H. Jakubowski, and R. Venkatesan, "Towards Integral Binary Execution: Implementing Oblivious Hashing Using Overlapped Instruction Encodings," *Proc. 9th ACM Workshop on Multimedia & Security,* ACM Press, 2007, pp. 129–140.
11. R. Scandariato et al., "Application-Oriented Trust in Distributed Computing," *Proc. IEEE Int'l Conf. Availability, Reliability and Security* (ARES), IEEE Press, 2008, pp. 434–439. 12. R. Kennel and L.H. Jamieson, "Establishing the Genuinity of Remote Computer Systems," *Proc. 12th Usenix Security Symp.,* Usenix Assoc., 2003, pp. 295–310.

PAOLO FALCARIN is a senior lecturer of software engineering at the University of East London, UK. His research interests are in software modeling and protection for distributed systems. Falcarin has a PhD in computer engineering from Politecnico di Torino (Italy). Contact him at falcarin@uel.ac.uk.

CHRISTIAN COLLBERG is an associate professor of computer science at the University of Arizona in Tucson. He is the author of the first comprehensive textbook on software protection, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection (Addison-Wesley, 2009). Contact him at collberg@cs.arizona.edu

MIKHAIL ATALLAH is a distinguished professor of computer science at Purdue University. He is a Fellow of both the ACM and IEEE, and was a speaker nine times in the Distinguished Lecture Series of top computer science departments. Contact him at mja@cs.purdue.edu.

MARIUSZ JAKUBOWSKI is a senior researcher in the eXtreme Computing Group at Microsoft Research. Since 1997, he has worked on applied and theoretical software protection, security, and cryptography. Contact him at mariuszj@microsoft.com.