

Introduction To Application Security

March 2010

Table of Contents

1. Overview	3
2. Introduction	4
3. The network is the computer	5
4. The evolution of data security	5
5. Designed in: the new trend in application security	9
6. Intrinsic application security	11
7. Software engineering and the trend toward diversity	14
8. Conclusion	17

1. Overview

The current state of Application Security reflects the fact that security has been an afterthought. Protection of data in transit and storage was the primary concern, and cryptography successfully addressed this problem. However, the threats to applications have evolved beyond those addressable by protocols and cryptography to the software itself. This lack of security foresight has cost billions in lost revenue and now threatens the information technology infrastructure upon which the worldwide economic engine relies.

Application Security, the protection of an application against security threats, is a difficult task. Application Security must now extend beyond traditional network and data security to incorporate the need for Software Protection. The approach to Application Security must also be driven by a clear and thorough understanding of the potential threats at each point in the system or network.

2. Introduction

Application Security is the protection of an application against security threats. This is a difficult task, as the application designer or corporate security manager must incorporate defenses against every imaginable attack, whereas an attacker only has to find one vulnerability or point of attack to succeed. Past techniques of protecting applications have certainly been limited, but new technology has been developed to solve this challenging problem.

Application Security is comprised of Network Security, Data Security and Software Protection:

- Network Security addresses external attacks generally against resources inside the firewall providing a service across a network. Network security has traditionally been addressed using firewalls, intrusion detection systems and virus scanners
- Data Security is the protection of data used locally by an application or transmitted between users and servers. Cryptography is the main solution here as it is highly effective at protecting data during transmission and storage by ensuring its integrity and confidentiality
- Software Protection is the protection of the software, or services rendered by the software, from attacks, thereby preventing theft of intellectual property and licensed content and ensuring that the software continues to function as intended. Typically these attacks include reverse engineering, tampering, copying, and automated forms of these attacks that can be launched across the network or on a desktop by relatively unsophisticated attackers

3. The network is the computer

At its simplest, an application is software that runs on hardware (e.g. a computer or network device such as a router) and manipulates data.

Applications are, in fact, significantly more complicated than this, so they have been abstracted into many layers. For the purposes of this paper, an application is not just the application layer as defined in the OSI network model. For example, applications in the Java™ programming language run on the Java Virtual Machine (JVM) platform, which runs on the operating system (OS) software, which communicates with network software (such as TCP/IP), which sends data to a router, whose software redirects the data to another server, and so on.

From an Application Security perspective, every layer must be considered an application, thereby necessitating the need for security in each of the layers. This is the focus of Software Protection as a critical element of Application Security.

4. The evolution of data security

Early networks, such as the postal system, relied on the manual delivery of messages to the intended recipient. Message integrity and confidentiality were ensured using sealed envelopes and, in some cases, primitive forms of cryptography. As electronic networks evolved, so did cryptography. New algorithms and protocols were developed, eventually to the point where direct attacks on encrypted messages are extremely rare.

The problem is that not everything can be encrypted, nor can data stay encrypted throughout the system. There are a multitude of services and middleware that can only receive and process requests if they can read the data. This means that there are various points in the system at which the data must be in clear text such that it can be read and processed. Attackers have a knack for finding the weakest link—in espionage it is usually the people. In today's digital age, it is the software that processes the data.

A QUESTION OF THREAT MODEL

Whether a particular piece of software represents a risk, and to what degree, depends on the threat model. Who is the bad guy, what avenues of attack exist, and what tools are available to launch an attack? These are important, fundamental questions that help define the threat model to address, as shown in Figure 1.

NETWORK THREAT MODEL

Practitioners of network security have traditionally viewed the hardware and operating system as trusted. This is a Network Threat Model, where the attacker is external and remote. An attack on the application comes via network ports; thus firewalls, which filter external packets from the untrusted world, were the first and most prevalent form of perimeter defense. Downloaded code also posed a threat, so code signing was invented to ensure the integrity of this code. Viruses and worms were other forms of attacks, so reactive defenses such as virus scanners and intrusion detection systems were implemented. However, the vulnerabilities that exist in application software that allow attacks such as viruses and worms remain a top concern.

UNTRUSTED HOST THREAT MODEL

Software Protection is at the other end of the threat model spectrum. In this case, the data and software must be protected against a legitimate but potentially hostile user, who has complete control over the computing platform and hence can use a wide range of tools—such as disassemblers, debuggers, and emulators—to discover vulnerabilities and implement an attack against the application. This is called the Untrusted Host Threat Model and is the realm of PC games copy protection and content protection techniques.

Cryptography was the basis of the first perimeter-type defenses created to protect data and software under the Untrusted Host Threat Model. Dynamic memory tracing is an attack technique that resulted in the implementation of reactive defenses such as anti-debug and self-modifying code. The severity of the threat model means that new techniques are required to prevent attacks against such systems.

INSIDER THREAT MODEL

In between lies the Insider Threat Model. The user may have limited privileges on the system, but is local to the target application under attack. Buffer overflows – typically network security type attacks – can be leveraged to boost privileges. Copying of applications and tampering or reverse engineering them off-site are also threats to intellectual property. Interestingly, it is possible for the threat model to change very quickly. For example, if a successful network intrusion attack is launched, a worm or a Trojan horse may gain complete control over a computing platform and the software running on it. In this case, the threat model changes very quickly from a Network Threat to an Untrusted Host Threat Model. Corporations often deal with this by wiping the hard drive on the machine that has been compromised. Ultimately, for high security applications, you have to assume that the software and hardware are untrusted.

Although the techniques to address different threat models are unique, there are many commonalities. The majority of attacks these days are dynamic attacks against the software. They are performed when the application is running and data is decrypted and in the clear.



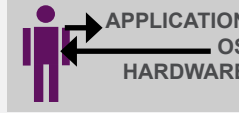
	NETWORK THREAT MODEL	INSIDER THREAT MODEL	UNTRUSTED HOST THREAT MODEL
THREATS: <ul style="list-style-type: none"> • TRUSTED • UNTRUSTED 			
ATTACKER PRIVILEGES	None	Some	Full
ATTACKER LOCATION	External, remote	Local network or same host	Same host
ATTACKS	<ul style="list-style-type: none"> • Buffer overflows • viruses • Worms • Boosting privileges • trojan horse 	<ul style="list-style-type: none"> • Reverse engineering • copying • Information stealing • tampering • Boosting privileges 	<ul style="list-style-type: none"> • Reverse engineering • Copying • Tampering • Information and content stealing
NETWORK SECURITY PRODUCTS PROTECT AGAINST THREATS TO NETWORKS AND THE AVAILABILITY OF THEIR SERVICES	<ul style="list-style-type: none"> • Firewalls • Virus Scanners • Authentication • Intrusion Detection Systems (IDS) • Intrusion Prevention Systems (IPS) 	<ul style="list-style-type: none"> • Authentication • Intrusion detection systems (IDS) • Intrusion prevention systems (IPS) 	<ul style="list-style-type: none"> • Not applicable
DATA SECURITY PRODUCTS PROTECT AGAINST THREATS TO DATA IN TRANSIT OR STORAGE ACROSS NETWORKS AND ON NETWORK AND CLIENT DEVICES	<ul style="list-style-type: none"> • VPN • Authorization • Smart cards • Hardware security modules (HSM) 	<ul style="list-style-type: none"> • Authorization • Digital rights management (DRM) • Smart cards • Hardware security modules (HSM) • Database encryption 	<ul style="list-style-type: none"> • Digital rights management (DRM) • Conditional access systems (CAS) • Smart cards
SOFTWARE PROTECTION PRODUCTS PROTECT AGAINST THREATS TO THE SOFTWARE ITSELF THAT PROVIDES SERVICES OR FUNCTIONALITY ON NETWORK AND CLIENT DEVICES	<ul style="list-style-type: none"> • Secure coding practices • Code signing • Code transformations 	<ul style="list-style-type: none"> • Secure coding practices • Digital rights management (DRM) • Code transformations • Code signing • License management 	<ul style="list-style-type: none"> • Digital rights management (DRM) • Conditional access systems (CAS) • Code transformations • Smart cards and dongles • Copy protection • Wrappers and packers

Figure 1. Threat models for application security

5. Designed in: the new trend in application security

Past approaches to Application Security have largely focused on building perimeter defenses around applications. This is also known as the eggshell

approach; once the perimeter is breached – or the shell cracked – the attacker has access to everything.

Perimeter defenses include cryptographic wrappers, tamper-resistant hardware such as smart cards for Software Protection, and firewalls for network security. Security was an afterthought and, as such, perimeter defenses were the most feasible and least intrusive to implement. However, as tools became more sophisticated and as corporate networks and applications became more interconnected, open and distributed, just defining the perimeter was a challenge – rendering perimeter defenses insufficient.

The next logical step in Application Security was to create reactive defenses. If an intrusion or attack is detected, then steps can be taken. Virus scanners and intrusion detection systems (IDS) are examples of such techniques. Anti-debug technologies also took this direction in Software Protection. If debuggers such as SoftICE are detected as installed, then the application will not execute. A number of forensic-based systems are also reactive in nature.

However, perimeter and reactive solutions have not solved the fundamental Software Protection problem. This lack of foresight has cost billions in lost revenue and now threatens the information technology infrastructure upon which the worldwide economic engine relies.

The security community has begun to recognize the importance of making applications intrinsically secure by building Application Security in during the development process. Here are a few examples:

- The vision of the digital home is for consumers to enjoy content such as music, videos and photos, regardless of the source, across different devices and locations in the home. “We know that consumers are readily anticipating the day when PCs and consumer electronics work together easily and reliably,” said Louis Burns, vice president and co-general manager, Intel Desktop Platforms Group. “People want to be able to move their digital media content effortlessly between PCs and CE [Consumer Electronics] devices for maximum flexibility and enjoyment.”¹ A robust and interoperable content protection scheme is a prerequisite to making this vision a reality as content owners will only make premium content widely available if they can be assured that it is protected from piracy and exploitation. Content protection mechanisms must be intrinsically resistant to attack to ensure that they continue to perform as intended
- “Any real-world system is a complicated series of interconnections. Security must permeate the system: its components and connections,” says Bruce Schneier in *Secrets & Lies*. Schneier, an early advocate of the view that cryptography was sufficient to address most security requirements, has evolved his opinion. He now suggests that we need to look at the entire system, not just a single component such as cryptography, and include security in the design process²
- Security exploits of Microsoft’s applications, operating systems, and vulnerabilities in their code—due to preventable errors—continue to escalate
- With analysts recommending that companies look to other vendors for alternatives to IIS, for example,³ Microsoft shut down its entire development effort in order to teach its developers how to build security into their code

1. Intel press release *Intel’s Digital Home Vision Moves Closer To Reality With New Industry Enabling Building Blocks*, February 19, 2003: <http://www.intel.com/pressroom/archive/releases/20030219net.htm>

2. Schneier, Bruce; *Secrets & Lies*; John Wiley & Sons: 2000: Preface pp xii

3. Pescatori, John; *Nimda Worm Shows You Can’t Always Patch Fast Enough*; Gartner FirstTake FT-14-5524, September 19, 2001; <http://www.gartner.com/resources/101000/101034/101034.pdf>

during the design, implementation, and testing processes. Bill Gates has said that security was deemed to be “job one” for the recently released Vista operating system—such that security represented the largest investment in the development of the technology⁴

- PC games traditionally had copy protection applied after the game executable was created. This is not only due to the fact that games developers do not want added complexity introduced into the development process, but also because the publisher, who may be distinct from the developer, creates the media and mandates the protection. Hackers have claimed for years that until the protection is built-in, games will continue to be hacked. In recent years, combined developers/publishers have begun to heed this advice through innovative software delivery and authentication models. For example, Valve Corporation created Steam, an online system for selling, delivering, and authorizing the playing of games developed by the company

6. Intrinsic application security

Intrinsic security includes technologies and techniques that are incorporated during the design and development process. These include design methodologies and manual code inspection processes, as well as applications and development tools. Intrinsic defenses can be used in conjunction with other traditional defenses. Intrinsic security typically works at the source code level and may include some or all of the following techniques:

- Control flow transformations. Control flow refers to the execution path followed as programs run and control is transferred to various blocks of statements

4. Ricadela, Aaron; Gates Says Security Is Job One For Vista; Information Week, February 14, 2006; <http://informationweek.com/news/showArticle.jhtml?articleID=180201580>

Control flow transformation secures a program by randomizing the block bodies of the target source code. This results in code that is extremely difficult to trace, and thus vastly increases the cost to the attacker who is attempting to reverse engineer the flow of the application

- Branch protection. In software, instructions that potentially transfer control to another instruction are referred to as “branches”. A conditional branch is a branch whose destination is determined by its input value(s), and include IF statements, SWITCH statements, and conditional operators. Attackers typically try to jam or bypass important branches in the code in order to sidestep security checking or in an attempt to modify the original flow of the program. Branch protection prevents branch jamming by adding code that causes the program to behave incorrectly if the branch is jammed
- Routine in-lining. In this technique, separate logical sections of code within a file are merged before transforms are applied. (This approach is different from compiler in-line options, which are done after pre-processing.) The goal is to combine operations and obscure the logic of the program
- Control flow flattening. Today’s compilers implement the control flow of procedural languages using a fixed set of jump and conditional branch instructions of the target instruction-set. Generally, the flow-of-control is achieved in machine code using a systematic or rule-driven approach. Control constructs are translated into canned and predictable instruction sequences. Consequently, reverse-engineering tools such as decompilers and program slicers can often reproduce the control-flow of the original program successfully. Control Flow Flattening changes the control flow into a SWITCH statement, which prevents static control flow analysis
- White-box cryptography. White-box cryptography functions are used when there is a concern that an attacker may be able to monitor the application and extract one or more cryptographic keys embedded or generated by

the application. In traditional cryptography, black-box attacks describe the situation where the attacker tries to obtain the key by knowing the algorithm and monitoring the inputs and outputs, but without the execution being visible. White-box cryptography addresses the much more severe threat model of content protection systems where the attacker can observe everything. Encryption and decryption are still required, but without exposing the cryptographic key. With proper design, applications can keep sensitive data either encrypted or transformed – or both – such that the original data is never exposed. All data operations will occur in a transformed state

- Integrity verification. Integrity Verification is a more secure variation of code signing that ensures trust on an untrusted host. It provides a secure method of validating the integrity of an application and can also ensure the integrity of external modules interacting with that application, including components of the operating system. Integrity Verification ensures that software cannot be tampered with—either statically or dynamically— without detection. This significantly raises the bar in tamper resistance because an attacker must not only reverse-engineer a program and make modifications to the binary, but must also defeat the integrity checking as well
- Anti-debug. Any debugging or diagnostic functionality running in an application's environment aids end-users whose intent is to reverse engineer or subvert the normal functionality of the deployed application. Anti-Debug techniques permit the detection of debuggers running in the same environment as the application. If detected, the application can take action to either de-activate the debugger or stop running
- Secure packager/loader. In this technique, a mini-application called a Secure Packager/Loader intercepts user or application calls to a target file during run-time. The Secure Packager/Loader must first validate the trigger event before it unpacks and executes the called target file. Encrypting the target

executable or DLL as well makes it hard for an attacker to statically analyze the file in storage

The above techniques, when applied during the development process, create security that is intrinsic to the code and cannot be separated or removed from the application data or software. Furthermore, the techniques enable software diversity, whereby random changes are made to the techniques each time they are applied. This approach creates diverse instances of the software, which reduces the effectiveness of automated attacks and hides incremental improvements in the software to prevent differential analysis attacks.

7. Software engineering and the trend toward diversity

Software is the antithesis of diversity. Historically, the entire business model for software vendors has been based on near-zero marginal cost of goods sold. But is this the future trend?

The reason the Internet and cyber infrastructure is so vulnerable to massively scalable attacks such as worms is a lack of diversity in the installed base of software. There are very few installed combinations of operating systems and server software, resulting in a very homogenous software base. For example, as of February 2007, approximately 58.7% of domain names are hosted by Apache servers, followed by Microsoft at 31.1%, SunOne at 1.7% and others that make up the remaining 8.5%.⁵

5. Netcraft; February 2007 Web Server Survey; http://news.netcraft.com/archives/web_server_survey.html

Analysts are now promoting a defense strategy based on manual diversity: the use of both Linux- and Windows-based products in a computing infrastructure.⁶ This approach has already been taken to protect domain name servers—the backbone of Internet addressing. The solution is to add more diversity into an installed base of an application. Just as genetic diversity in biological systems prevents catastrophic loss of a species facing a new disease, diversity must be added to protect the IT infrastructure against catastrophic failure.

Reference designs are also a culprit, as demonstrated when a vulnerability was found in Simple Network Management Protocol (SNMP)⁷. Because so many application developers used the reference design code, the vulnerability existed in most implementations.

Similarly in the traditional Software Protection domain, object oriented languages, good software engineering practices, and CASE tools have all resulted in software that is significantly easier to attack. Simply put, the easiest code to reverse engineer is object code that has been compiled from well-structured source code. One could then credibly argue that the best defense against reverse engineering is to write code poorly—“spaghetti” code. Often legacy software applications must be rewritten because they have effectively become unmaintainable, and in the process, tamper resistant! However, the improvements made in maintainability and rapid application development that modularity and code reuse provide cannot be relinquished. This points to the need for automated tools to provide Application Security and diversity. The solution to this apparent paradox lies in being able to generate spaghetti code from well-written source code.

6. Holtzmann, David; Diversity Training; CSO Magazine, June 2003

7. CERT Coordination Center Advisory CA-2002-03, Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP), February 12, 2002; <http://www.cert.org/advisories/CA-2002-03.html>

Microsoft's experience also demonstrates the need for an automated solution to Application Security. Despite educating their developers and reviewing their code base, vulnerabilities continue to surface in their applications. Buffer overflows—theoretically preventable via good coding practices—remain the top vulnerability in terms of threat severity. That said, good design practices and implementation will remain important.

Closely related to software diversity is the important concept of software renewability. It refers to the practice of issuing software updates—either proactively or on-demand—which implement some or all of the intrinsic software security techniques described above. This approach increases software diversity, makes it harder to develop a successful attack, reduces the time available to develop an attack, and reduces the scope and impact of a successful attack.

Can diversity and renewability be deployed effectively without destroying the software business model? It is already happening today and is the future in Application Security.

Product development pressures will persist and humans will continue to be fallible, meaning software flaws will not disappear. However, automated security techniques and the use of diversity can prevent scalable automated attacks. Just as with biology—everyone is a little different with different vulnerabilities, or more importantly, different defenses—the infrastructure and its on-going viability can be assured.

8. Conclusion

Software Protection has been largely neglected to date as a fundamental and critical element of Application Security. Cryptography effectively protects data in transit and storage, but leaves the application—really any software in the communication chain—subject to attack.

Since Application Security has largely been an afterthought in the development and deployment of software applications, traditional network security and Software Protection techniques have focused on perimeter and reactive defenses. These are no longer enough. Security experts and corporations realize the need to make applications inherently secure by building in Application Security up front. While good application design remains important, trends point to automated tools and the use of software diversity and renewability as strategies to ensure Application Security and prevent scalable attacks that can threaten the cyberspace infrastructure. By combining several fundamental security building blocks, it is possible to deliver a simple solution that is intuitive, easy to deploy, robust, and scalable.

©2012 Irdeto. All Rights Reserved.

This document and the information contained herein is the subject of copyright and intellectual property rights under international convention. All rights reserved. No part of this document may be reproduced, stored in a retrieval system or transmitted in any form by any means, electronic, mechanical or optical, in whole or in part, without the prior written permission of Irdeto. All non-Irdeto company names, product names, and service names mentioned are used for identification purposes only and may be the registered trademarks, trademarks, or service marks of their respective owners. All information is without participation, authorization, or endorsement of the other party.