Trust in Cyberspace

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA

Introduction

As we expand on the train of thought from (ISC)²®'s whitepaper *Code (In)Security*, which closed with the admonition that "insecure code means checkmate," we must realize that the state of affairs when it comes to software security is even more than a chess game; it's nothing less than a battle – a battle between the attackers and the defenders; between those who are trying to break your software and those who are trying to defend it at all costs; between the black hats and the white hats. We are at war, and a war in which the enemy is not only merely subtle, but in most cases invisible. The theme for the Randolph Air Force base conference, "Cyber Security, the Invisible Man"[a] only accentuates this point. While the best of military efforts can secure air, land, and sea, it can all be thwarted by an infiltration in cyberspace. Malicious threat agents can exploit vulnerabilities in applications (software) and use those exploited applications as launch pads to compromise host systems and/or networks entirely.[b]

Software Security a.k.a. Trust in Cyberspace

Software security is all about trust – about assurance and confidence that the software will, first, function as it is expected to and, second, be robust enough to handle any threats that can thwart its expected operations. Some prefer to use the term software assurance interchangeably with software security. There are many Software Assurance definitions as depicted in Table 1.

Table 1. Definitions of Software Assurance

Organization	Definition
National Institute of Standards and Technology (NIST)	The planned and systematic set of activities that ensures that software processes and products conform to requirements, standards, and procedures to help achieve trustworthiness (no exploitable vulnerabilities exist, either of malicious or unintentional origin) and predictable execution (justifiable confidence that software, when executed, functions as intended).[c]
National Aeronautics and Space Administration (NASA)	The application of planned and systematic set of activities such as quality assurance, quality engineering, verification and validation, nonconformance reporting and corrective action, safety assurance, and security assurance during a software life cycle, to ensure that software processes and products conform to requirements, standards, and procedures.[d]
National Information Assurance Glossary	The level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the software functions in the intended manner.[e]
SAFECode	Confidence that software, hardware and services are free from intentional and unintentional vulnerabilities and that the software functions as intended.[f]

Organization	Definition
Object Management Group (OMG)	This justifiable trustworthiness in meeting established business and security objectives.[g]
Software Security Assurance State-of-the-Art (SOAR)	The property of software which will consistently demonstrate that the software is of quality, reliable, correct, dependable, usable, interoperable, safe, fault-tolerant and secure and the basis for gaining justifiable confidence or trust.[h]

Irrespective of the various ways in which software assurance can be defined, one common thread that is evident in all of the tabulated definitions is that software assurance/security is about confidence and trust. When software cannot consistently guarantee this confidence, attacks on the network and hosts that spur from vulnerabilities in the software are inevitable. And shifting the blame to the network layer for software-related weaknesses is akin to blaming the postal service for delivering a letter bomb. Since there is no way to prevent someone from sending the letter bomb, what's really needed is protection against the threats posed by it. The software within our organizations and homes must be reliable and resilient to attack. What's really needed, in other words, is trusted software.

Trusted Software – What is it?

While there are several aspects to trusted software within the context of software security, in this whitepaper we will primarily focus on the three qualities that distinguish trusted software from that which is not trusted: reliability, resiliency, and recoverability.

Technical discussion on the level of trust that is set for the code to execute, such as partial trust or full trust, is beyond the scope of this whitepaper.

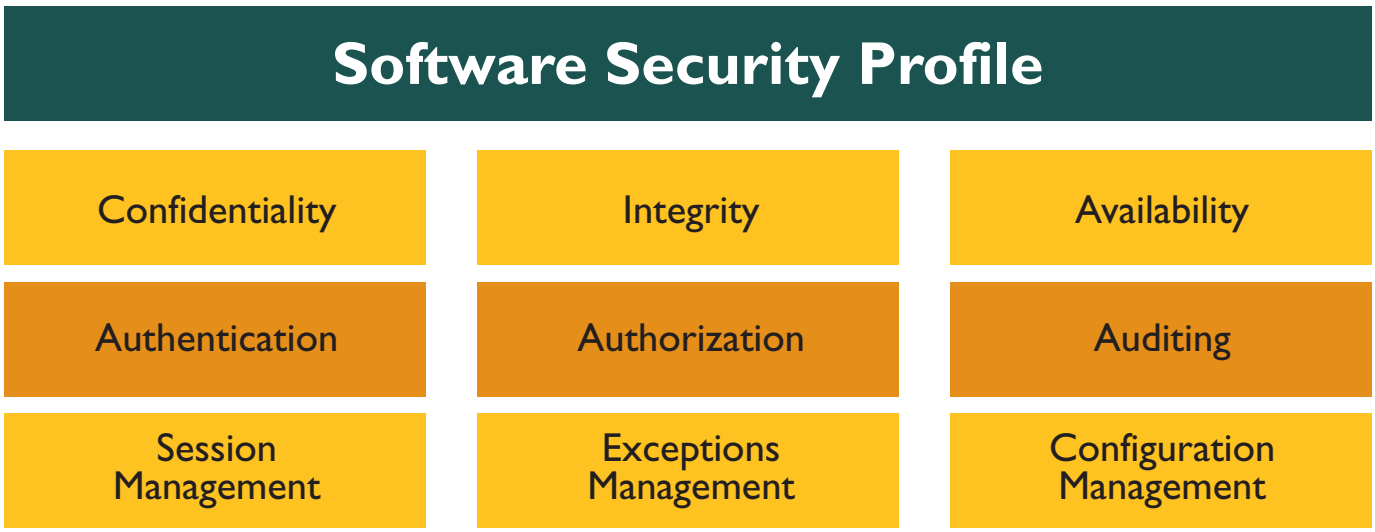
The reliability quality of trusted software means that the software will perform as it is expected to, each time, every time. The resiliency quality of trusted software means that the software will perform without breaking any component of the security profile and when broken, the recoverability quality of software will ensure that the software is robust enough to restore itself promptly, thus limiting any exposure or damage caused by the security breach.

The security profile for trusted software in the context of software assurance includes the following:

- Protection against confidentiality, integrity, and availability threats
- Assurance that authentication cannot be circumvented
- Validation of authorization credentials before access to resources are granted
- Effective implementation of auditing functionality for business-critical and administrative transactions
- Management of Sessions, Exceptions, and Configuration parameters.

Components of the software security profile are illustrated in Figure 1.

Figure 1. Software Security Profile

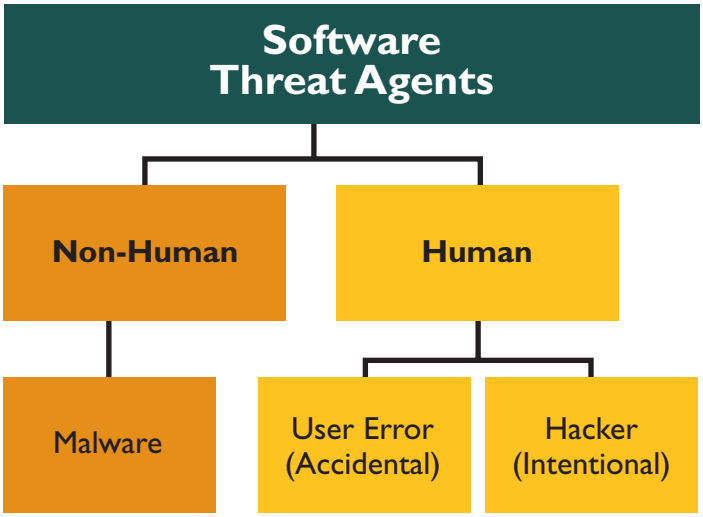


Threats that Impact Trust

There are several threats to software that can impact one's level of confidence or trust in it. These threat agents take advantage of vulnerabilities in software and may be human or non-human. Human threat agents can be broadly classified based on their underlying motivation to exploit a weakness and materialize a threat. They range from unintentional and non-malicious user errors to intentional malicious threats, which include those from hackers and crackers. Hackers and crackers also vary in their skill, ranging from minimal, limited skills where they don't necessarily understand the consequences of their actions (script kiddies), to highly-skilled organized criminal hackers. Non-human software threat agents generally include software itself that is malicious in nature. In fact the term malware has its roots in two other words: malicious and software. The maliciousness of software is only limited by the creativity and greed of the malware creator. These threats may be developed externally, including malware such as viruses and worms, spyware and adware, and Trojans; or they may be embedded in code by an insider.

Figure 2 is a depiction of some of the most common categories of software threat agents.

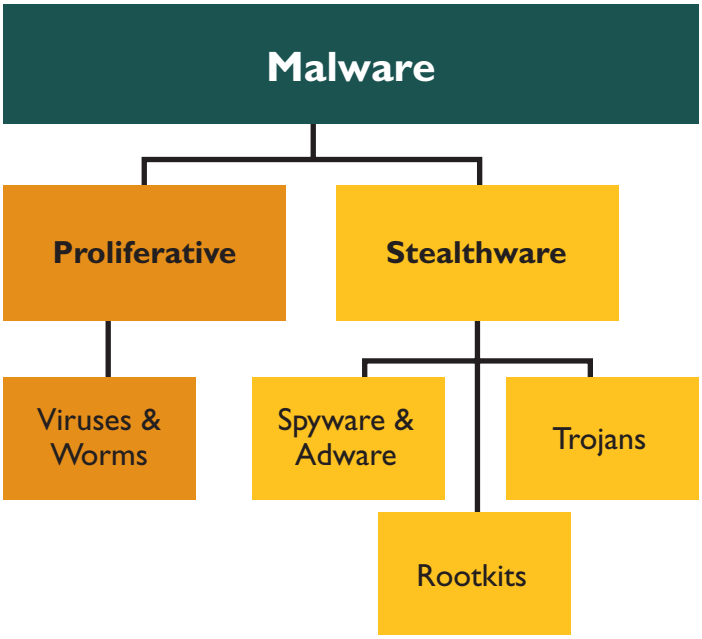
Figure 2. Software Threat Agents Categorization



With the prevalence of malware and embedded code issues in this day and age (and statistics indicate that the rate of release of malicious software supersedes even legitimate software releases[i]) it is important to be aware of the different forms of malware putting a dent in confidence and trust. The boom in broadband access and interconnectivity, combined with a movement from hacking for fun to hacking for profit, helps explain the rise in malware creation and distribution. What began primarily for fun as pranks (defacement, hard drive corruption, etc.) is now a colossal and profitable business undertaking, as evidenced by the recent case of the largest credit card theft and fraud recorded in history to date. This theft was masterminded by Albert Gonzalez who, with his accomplices, stole more than 130 million credit card numbers by exploiting injection vulnerabilities and using packet sniffing malware that allowed the hackers to create backdoors and steal sensitive data.[j]

For purposes of logical organization, the most prevalent malware threats can be categorized into proliferative (malware that spreads) and stealthware (malware that remains hidden) as illustrated in Figure 3. An exhaustive and all-inclusive description of the various types of malware in existence today is beyond the scope of this whitepaper, but the most common ones are introduced in this section.

Figure 3. Types of Malware



Proliferative Malware

Proliferative malware includes malicious software programs that, upon exploiting weaknesses in networks, hosts, and software applications, aim at propagating their malicious operations to other networks, hosts, and software applications connected to the victim. Viruses and worms are the most common form of proliferative malware.

Viruses and Worms

This is probably the most well known type of malware. Although computer viruses and worms are often frequently clubbed together, they are distinct in their traits based on their ability to propagate. A computer virus is a piece of malicious software that infects a computer program or executable. Just as a biological virus requires the host to survive, a computer virus depends on the victimized program or executable, and its spread is contained within the victimized program. A computer worm, on the other hand, is a type of malware that can actively propagate itself over the network, infecting other computers on the network. For propagation, worms require a network but they are not limited

to exploiting vulnerabilities of hosts on the network alone. The Samy worm is an example of a worm that went beyond just infecting networks and hosts as most worms do. This Web worm exploited scripting vulnerabilities in the MySpace Web application and propagated itself to other unsuspecting users. Although the Samy worm itself did little damage (merely adding "Samy is my hero" to other MySpace user profiles without their permission) it could have been designed to do much more nefarious activities such as deface Websites, cause denial of service to all affected users (which would have reached the millions), or steal private and sensitive information. The Samy worm is a prime example of a worm that exploits weaknesses in software (Web application) and propagates itself. The Samy worm took advantage of an insecure coding vulnerability known as Cross-Site Scripting (XSS).

The use of proper anti-virus software with updated virus signatures, network segmentation, and patched and hardened hosts, are all mitigating control measures against viruses and worms, but it must also be recognized that proper coding that addresses software security vulnerabilities is equally important and needs to be layered on top of network and host protection measures.

Stealthware

Stealthware includes malicious software programs such as spyware and adware, Trojans, and rootkits that remain hidden and operate often without the consent or knowledge of the victimized system or user.

Spyware and Adware

Spyware and adware are examples of stealthware that operate by invading the privacy of an individual. Spyware is used clandestinely to harvest information about a system or user. Adware includes malware that redirects users to marketing devices, displaying annoying and unsolicited information and advertisements. Unlike proliferative viruses and worms, spyware and adware don't aim at self-replication and propagating themselves, but instead try to gain control of the system that they infect by exploiting software and operating system vulnerabilities. Spyware and adware are extremely potent malware because they can compromise all of the core tenets of information security which include confidentiality, integrity, and availability. They can pose threat to confidentiality by installing keyloggers that record key strokes and by stealing personal information, browser activity history, and cookies. They can impact the integrity of the computing system by installing software without user authentication and change computer systems and modify registry keys and values. And they can deface Websites by redirecting Web browser location references and spin off memory intensive computer processes, thereby impacting availability by causing resource exhaustion, slowness, and denial of service.

Spyware and adware doesn't just enter into one's computer when one visits malicious Websites that exploit weaknesses in browser security (commonly known as drive-by-download). They can also come disguised as legitimate software. Peer-to-peer sharing networks are notorious when it comes to sharing software that seems legitimate but is instead rife with spyware and adware. Spyware can also be installed by worms that propagate in the network.

Hardening the operating system with the use of anti-spyware software, and increased browser security in conjunction with hack-resilient software development, are safeguards that can help in mitigation efforts against the potential surreptitious infestation and exploitation attempts of spyware and adware.

Trojans

Known simply as a Trojan, Trojan horses are a type of stealthware that gets it etymology from the historic account of how the Greeks infiltrated the impenetrable defenses of Troy by presenting Troy with a wooden gift horse that was accepted and taken within its fortified walls. The Trojan horse harbored Greek soldiers who crept out of the horse at night and opened the gates of Troy from within, allowing the Greek army to penetrate and eventually take over the city. In the software security world, Trojans are primarily a threat against access control checks. Much like the wooden gift horse, Trojan horses appear as innocuous programs with desirable functionality, while they truly aim at circumventing access controls. Trojans are usually designed to have functionality that will allow the hacker to be able to connect to the victim's computer on a continual basis. Hackers can then use this covert channel to install additional software such as keyloggers, spyware, and adware. Or they can steal data and/or information, modify computer and user settings, or misuse computer resources.

A Trojan can be installed on the user's system by bypassing browser security protection mechanisms or by exploiting software on the victim's system. The most prevalent means, however, is by tricking a user into installing the Trojan. Trojans are usually spread as e-mail attachments or as seemingly benign software in peer-to-peer file-sharing networks that allow downloading of software.

While anti-virus programs can mitigate Trojan-based access control breaches by detecting and quarantining (or deleting) the Trojan, trusted computing safeguards, awareness training, and education of end-users are effective safeguards against Trojan-based threats.

Rootkits

Rootkits have earned a malicious reputation as highly dangerous programs that can cause complete compromise. In the renowned book Rootkits, authors Hoglund and Butler define a rootkit as "a set (kit) of programs and code that allows an attacker to maintain a permanent or consistent undetectable access to "root",

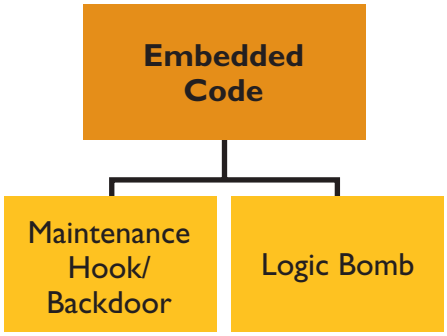
the most powerful user on a computer.[k]" However, it must be recognized that rootkits can be used for legitimate non-malicious purposes such as remote control and software eavesdropping when required for espionage, monitoring user behavior, and consented law enforcement reconnaissance situations. Malicious rootkits attempt to compromise system integrity by modifying the operating system, masquerading as legitimate programs (as loadable program modules or device drivers) and taking the OS under siege. When rootkits are used for malicious purposes they act as the proverbial wolf in sheep's clothing.

Rootkits operate at high (root) privileges and because they usually modify the operating system, they often go undetected. This means that malicious usage of rootkits can potentially have dire consequences and serious effects on trusted computing. Some of the more prevalent uses of rootkits for malicious purposes include the installation of keyloggers, the alteration of log files, and the establishment of covert channels, all the while evading detection and removal. Spyware and hackers that exploit unhardened operating systems and vulnerabilities in software are primary sources for the installation of rootkits. This warrants not only the need to ensure that host systems are patched appropriately, but also that the software that is built or bought is reviewed for weaknesses that are discoverable and exploitable.

"Embedded code issues such as insider backdoors and logic bombs also pose a threat to trusted computing."

In addition to externally-developed malware, embedded code issues such as insider backdoors and logic bombs, as depicted in Figure 4, also pose a threat to trusted computing.

Figure 4. Types of Embedded Code



Backdoors

Backdoors are code constructs embedded in code to allow programmers to bypass security mechanisms. They are often designed to bypass authentication to gain remote access to the system and are usually non-maliciously embedded in code for troubleshooting or maintenance purposes. When backdoors are implanted in code for troubleshooting purposes, they are also referred to as maintenance hooks. Sometimes programming errors or business logic flaws in design can also result in intentional or accidental creation of backdoors. Although maintenance hooks are usually designed without any malicious intent, they are a threat to trusted computing and can potentially compromise authentication controls. Attackers and malware can take advantage of backdoors to gain remote access to systems. The infamous Nimda worm is purported to have taken advantage of a backdoor created by the Code Red II worm which took advantage of unpatched Microsoft Internet Information Server.

Hardening operating systems, and developer education in conjunction with proper configuration management processes, alleviate the threats posed by backdoors. Static code reviews are effective in identifying backdoors planted by insiders. Maintenance hooks may be allowed in non-production environments, but prior to deployment into production these need to be removed entirely to avoid any potential threat.

Logic bombs

Like backdoors, logic bombs are also embedded code constructs that remain dormant in code and are executed when specific events and/or time conditions are met. Although the expiration notice of a demo or trial piece of software can be deemed to be a logic bomb, it is really not since the intent is not malicious. However, in situations when the embedded code is triggered by specific events or time to undertake a malicious activity, such as deletion of media contents, denial of service, etc., then such code constructs are referred to as logic bombs. Logic bombs are also referred to as "slag code" because what remains after the detonation (execution) of the implanted code is usually computer slag.

Logic bombs are usually associated with disgruntled or angry employees who have access to the organization's code. The famous case of the disgruntled UBS PaineWebber employee, Roger Duronio, who caused the company more than three million dollars in recovery costs and almost half a decade to recover, is a testament to the potency of attack that can be caused by a logic bomb. Duronio was charged for having implanted a logic bomb in code that was triggered on a specific date to delete important and sensitive files from hard drives, which, when the effects of the bomb were disclosed, also caused a drop in the price of the company stock.[l]

Such computer sabotage and disruptions of operations can be avoided when code is reviewed for the presence of logic bombs. It must be recognized, however, that mere automated static code reviews may not necessarily detect logic bombs as the bombs are perfectly correct code that will compile. Human inspection of code becomes critical since the reasoning behind how a logic bomb code construct will execute cannot, for the most part, be

detected by automated code review scanners. This is particularly important in situations when the code is being developed in an environment in which the organization has little or no control, as is the case with outsourcing.

Security in the SDLC is Trust Assured

Irrespective of whether the source of threat is human or non-human, the motivation of the threat agent is intentional or unintentional, and the orchestration of a threat to materialize is organized or not, it is absolutely necessary for a confluence of people, processes, and technology to assure confidence in the software that is built or bought. Derek Slater, Editor in Chief of *Chief Security Officer* magazine, has rightfully expressed that it is high time to get organized in addressing security threats that are prevalent today.[m]

It is critical to ensure that the software can be trusted, whether you build it in-house or acquire it from a third-party software publisher. Throughout the software development life cycle (SDLC), activities that verify and validate that the software is reliable and resilient are necessary. A breakdown in any one phase of the SDLC is all that is necessary to completely nullify any efforts that the software development team or organization has undertaken to assure justifiable confidence to its end-users.

> *"Security in the SDLC from requirements to retirement is necessary to assure trust and confidence."*

Security in the SDLC from requirements to retirement such as security requirements gathering, threat modeling, attack surface analysis and reduction efforts, writing code that addresses components of the software security profile, code reviews, security testing, secure installation and deployment, security operations and secure disposal, are all necessary to assure trust and confidence. Defensive coding and anti-tampering techniques such as code obfuscation and code signing can help deter security attacks and provide heightened degrees of assurance when used to provide authenticity of the source of the code. All these must be done in conjunction with hardening the operating systems to be ironclad and establishing appropriate change and configuration management processes which should complement and not contradict efforts taken to ensure that the software that is built or bought can be trusted. Verification and validation (V&V) as part of a certification and accreditation (C&A) process, and independent third party assessments can also be used to determine levels of trust in the software.

Conclusion

Ensuring trust in cyberspace is imperative and this means that the software that we build or buy must be trustworthy. Tabulated below are some characteristics of trusted software and computing.

Table 1. Trusted Software Characteristics

Trusted Software Characteristics
Functions as expected (reliable)
Ensures security policy (resilient)
Is fault-tolerant and robust (recoverable)
Maintains confidentiality, integrity, and availability of software and the data it handles
Prevents circumvention of authentication and access control checks
Handles sessions, configurations, and exceptions securely
Is deployed on host systems that are adequately hardened
Ensures protection against proliferative malware (viruses and worms)
Defends against malware that causes disclosure and destruction (spyware and adware)
Ensures protection against harmful malware that is purported as benign (Trojans)
Does not allow privilege escalation from user land to kernel land (rootkits)
Is deployed/released without any maintenance hooks (backdoors)
Ensures that there are no embedded code security threats that can be conditionally triggered (logic bombs)
Anti-tampering (obfuscation) and authenticity (signed code) controls are present
Tested, validated, and verified for software security by the organization or by an independent third party.

As in a game of chess, the defender has to always outthink the attacker's next move and strategize on how he can bring the opponent to checkmate. However it must be recognized that unlike the finality of checkmate in the game of chess, security is not a one-time thing. It has been rightfully expressed by individuals in high management echelons, such as the CEO of Microsoft, Steve Ballmer, and Sir Tim Berners-Lee, credited with the invention of the World Wide Web, that security is a never-ending battle. One primary reason that validates this position is the need, in this day and age, to conduct commerce and our dependence on software to make those business transactions possible. The network and host systems are essentially innocuous until software (network and host operating systems) that manages those systems is installed, on top of which is layered software for conducting business transactions. The breakdown starts at the software layer, and to win this perpetual battle, software security is imperative. By the fulfilling of the tabulated characteristics, trusted software, in short, ensures justifiable customer confidence and trust, which is what software security is all about.

About (ISC)²®

(ISC)² is the largest not-for-profit membership body of certified information security professionals worldwide, with over 66,000 members in more than 135 countries. Globally recognized as the Gold Standard, (ISC)² issues the Certified Information Systems Security Professional (CISSP®) and related concentrations, as well as the Certified Secure Software Lifecycle Professional (CSSLP®), Certification and Accreditation Professional (CAP®), and Systems Security Certified Practitioner (SSCP®) credentials to qualifying candidates. (ISC)²'s certifications are among the first information technology credentials to meet the stringent requirements of ANSI/ISO/IEC Standard 17024, a global benchmark for assessing and certifying personnel. (ISC)² also offers education programs and services based on its CBK®, a compendium of information security topics. More information is available at **www.isc2.org.**

About the Author

Mano Paul, CSSLP, CISSP, AMBCI, MCAD, MCSD, Network+, ECSA is CEO and President of Express Certifications and SecuRisk Solutions, companies specializing in professional training, certification, security products and security consulting. His security experience includes designing and developing software security programs from Compliance-to-Coding, application security risk management, security strategy and management, and conducting security awareness sessions, training, and other educational activities. He is currently authoring the *Official (ISC)² Guide to the CSSLP*, is a contributing author for the *Information Security Management Handbook*, writes periodically for *Certification, Software Development* and *Security* magazines and has contributed to several security topics for the Microsoft Solutions Developer Network. He has been featured in various domestic and international security conferences and is an invited speaker and panelist in the CSI (Computer Security Institute), Catalyst (Burton Group), TRISC (Texas Regional Infrastructure Security Conference), SC World Congress, and the OWASP (Open Web Application Security Project) application security conferences. He can be reached at **mano.paul@expresscertifications.com** or **mano.paul@securisksolutions.com.**

---

a Cybersecurity, the Invisible Man
http://www.randolph.af.mil/news/story.asp?id=123159917

b New Wave of SQL Injection Attacks Alarm Researchers
http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1314697,00.html#

c National Institute of Standards and Technology (NIST) Software Assurance Metrics And Tool Evaluation (SAMATE)
http://samate.nist.gov/index.php/Main_Page.html

d National Aeronautical and Space Administration (NASA) Software Assurance Guidebook and Standard
http://satc.gsfc.nasa.gov/assure/assurepage.html

e National Information Assurance Glossary (NIAG)
http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf

f SAFECode Publication – Software Assurance: An Overview of Current Industry Best Practices
http://www.safecode.org/publications/SAFECode_BestPractices0208.pdf

g Object Management Group
http://swa.omg.org/docs/softwareassurance.v3.pdf

h State-of-the-Art Report (SOAR) Software Security Assurance
http://iac.dtic.mil/iatac/download/security.pdf

i Symantec Internet Security Threat Report Volume XII, April 2008

j Computer Hacker Gonzalez to Admit Guilt, Forfeit $1.65 million
http://www.bloomberg.com/apps/news?pid=20601087&sid=aXHW5gbDC0EA

k Hoglund,G and Butler, J. Subverting the Windows Kernel Rootkit. 1st Ed.

l United States of America v. Roger Duronio
http://www.usdoj.gov/usao/nj/press/files/pdffiles/duronioindictment.pdf

m CSO Magazine, September 2009.
http://www.csoonline.com/issue/20090901