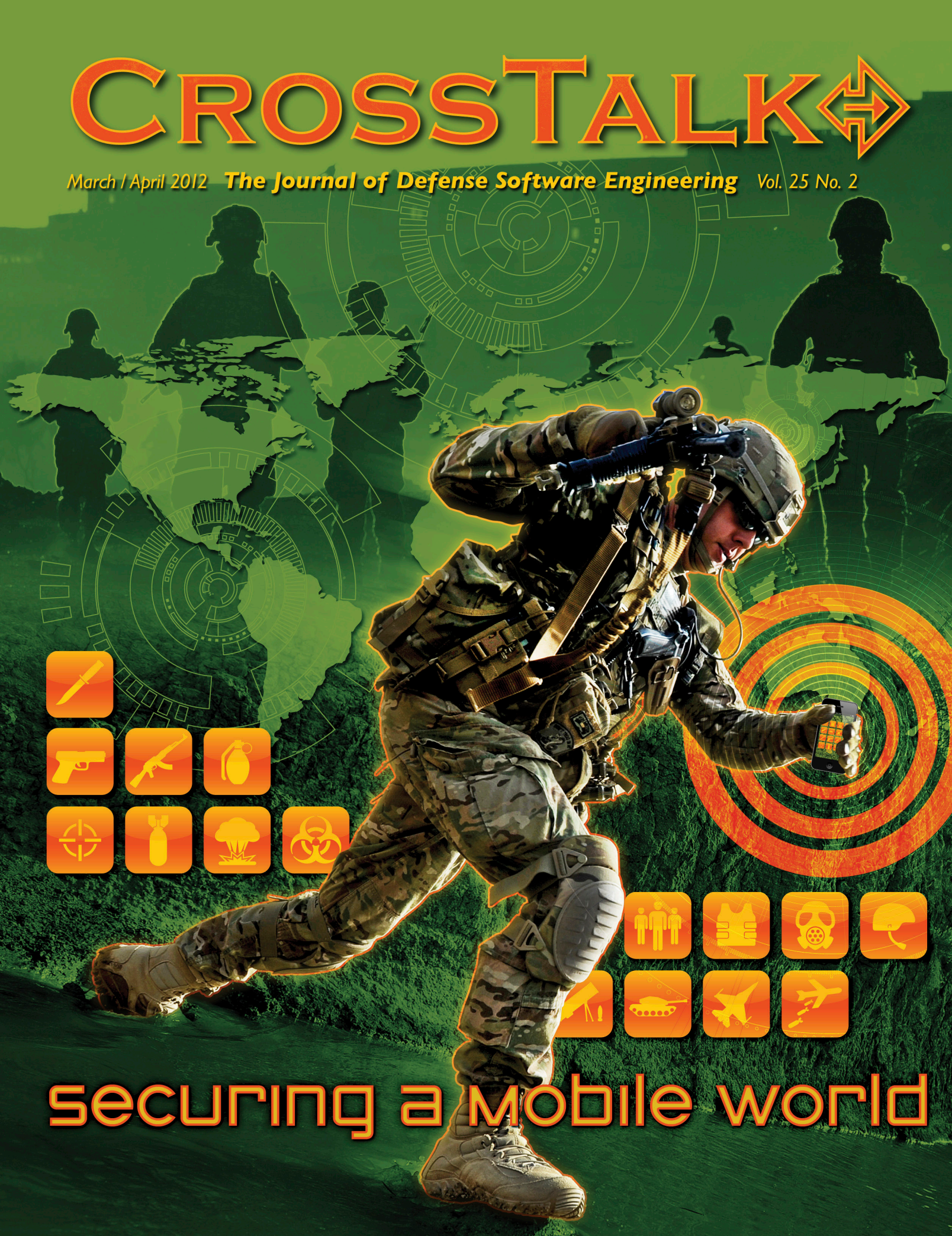
CrossTalk

March / April 2012 **The Journal of Defense Software Engineering** Vol. 25 No. 2



securing a mobile world

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
APR 2012		**00-03-2012 to 00-04-2012**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk, The Journal of Defense Software Engineering. Volume 25, Number 2. March/April 2012	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave Bldg 1238,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

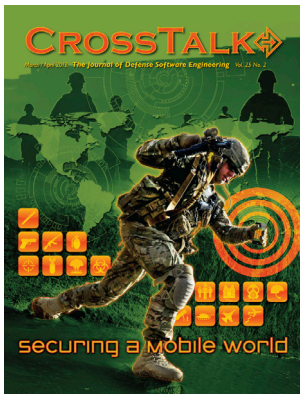
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**40**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

SECURING A MOBILE WORLD

Departments



Cover Design by Kent Bingham

Securing a Mobile World

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Tracy Stauder
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of **CrossTalk** providing both editorial oversight and technical review of the journal. **CrossTalk's** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CrossTalk** editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. **CrossTalk** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CrossTalk.**

Trademarks and Endorsements: **CrossTalk** is an authorized publication for members of the DoD. Contents of **CrossTalk** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the **CrossTalk** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

Security While On the Move

CrossTalk would like to thank DHS for sponsoring this issue.

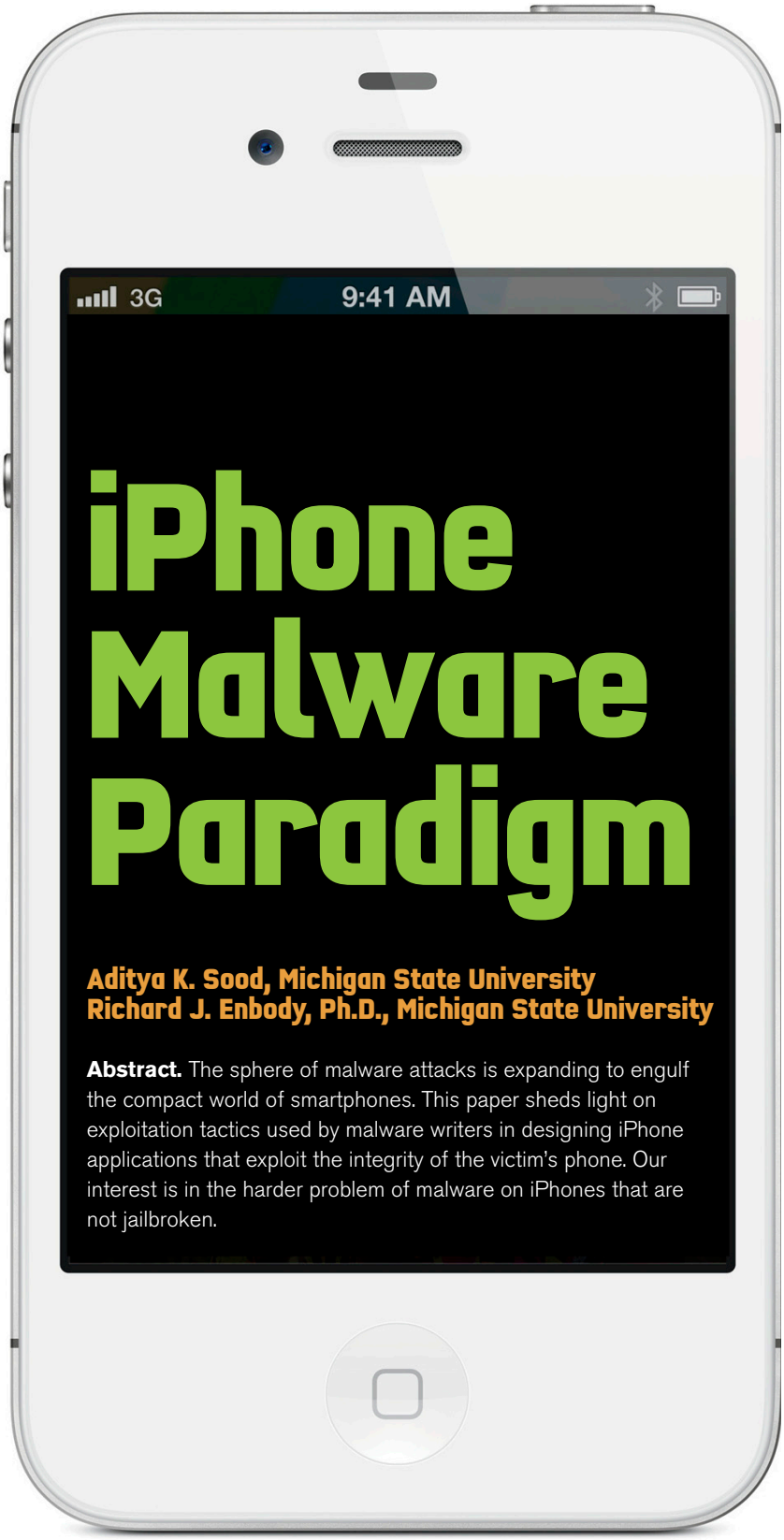
The increasing convenience and ubiquity of mobile computing and smart personal communication devices presents an irresistible target for malicious actors. The rush to provide applications means few are tested to detect, analyze, and remediate weaknesses. Public Wi-Fi networks can also provide a vulnerable entry point to our mobile device information systems. As a result, hackers are able to quickly exploit software on smartphones.

The challenge of securing the mobile world is complex and therefore requires multi-disciplinary solutions. The security models currently provided by major mobile providers are not sufficient to meet the information protection needs of civilian and defense agencies. Application developers, network administrators, and incident responders need to collaborate to address mobile computing risk. To be effective, this collaboration requires rapid sharing of standardized threat and vulnerability information so public and private stakeholders can act quickly to mitigate risks to their operations and activities.

Fortunately, much of what we already know and do in cybersecurity applies to the mobile world and the challenge of securing it. Consistency in the identification and interpretation of software weaknesses, attack patterns, and malware data is essential for quick and efficient information sharing. DHS sponsors programs that help standardize such data, thus allowing companies and organizations to collect, store, and define it in compatible formats. By promoting common data taxonomies and methodologies for storing, indexing, and interpreting malware samples, DHS is driving towards seamless diagnosis and remediation of exploitable software across the various mobile platforms. These are necessary conditions for near real-time situational awareness of vulnerabilities and malware. However, collaboration should not end with remediation of malware. DHS envisions an environment that grants public and private sector owners and operators of information technology systems access to an entire range of security automation tools and capabilities, including software assurance education materials and security-content authoring services.

The public and private sectors occupy equally important—and equally informed—roles within their particular area of cybersecurity expertise. Rapid, bidirectional information sharing ensures that both sectors are able to bridge the critical information gap between what they know and do not know. Cutting through these knowledge gaps ultimately facilitates the real-time situational awareness necessary to defend cyberspace. Together, we can develop a trustworthy, sustainable, and flexible information-sharing environment that effectively secures our Nation's cyberspace—including the ever-growing mobile domain.

Roberta "Bobbie" Stempfley
Deputy Assistant Secretary
Office of Cybersecurity and Communications
Department of Homeland Security



iPhone Malware Paradigm

Aditya K. Sood, Michigan State University
Richard J. Enbody, Ph.D., Michigan State University

Abstract. The sphere of malware attacks is expanding to engulf the compact world of smartphones. This paper sheds light on exploitation tactics used by malware writers in designing iPhone applications that exploit the integrity of the victim's phone. Our interest is in the harder problem of malware on iPhones that are not jailbroken.

Introduction

Malware has begun infecting the mobile world. Several studies [1, 2] have been conducted showing how mobile malware is exploiting the online world. Android malware infections are exploding as compared to iPhone. The primary reason is that Android is an open source platform where as iPhone's iOS is closed. Our target is to discuss the potential possibilities of malware occurrence in iPhone devices. In spite of the iPhone's strong security platform, malware is making inroads. However, successful iPhone exploitation depends on several factors. As we know, Apple has implemented several security barricades in order to secure the iPhone environment aided by tight control of their app market. Apple considers iPhones marginalized by the jailbreaking process as unsecure since all the inherent protection mechanisms have been circumvented by the attacker.

Is it possible to write a malicious application that may not exploit security vulnerability, but can still perform some spyware activity? The answer is yes. This is possible in certain scenarios where a malicious application can be designed to bypass Apple's application review process to execute illegitimate operations on an user's iPhone. In this paper, we discuss practical scenarios and effective techniques that can be used to host malicious applications on non-jailbroken Apple iPhones.

Understanding Apple's iPhone Applied Security Model

Apple enforces strict security features in order to protect the integrity of iOS. Its security model has the following features:

• With the advent of iOS 4, Apple introduced a new data protection procedure in which stored data is secured using hardware encryption. The device stores the user passcode key on an internal chip using 256-bit encryption. The Unique ID (UID) of the devices is used as a key to encrypt a file on iPhone.

• The iOS environment is divided into two main partitions. Similar to UNIX, the root partition manages the kernel and base OS. The user partition contains third-party applications and data. All applications run in a user mode with a standard set of access rights and built-in restrictions. The iOS system-level binaries are related to OS X and Darwin. In order to preserve the integrity of applications, Apple implements a code signing process [8]. The code signature consists of three parts. First, the signature consists of a UID that is present in the info.plist files under CFBundleIdentifier structure. Second, it requires a seal that is built from hashes and checksums of various files and other components of the application bundle. Third, it requires a digital signature. All the signatures are stored in the MACH-O header format. Code signing code verification is implemented in a kernel level using the execv () command.

• Third-party applications running on iOS are sandboxed [9]. This concept is implemented to force privilege separation among different components in iOS. It means that third-party applications are not able to run code at kernel level—a secure practice to avoid exploitation of privileges. The application sandbox is implemented using three techniques. First, entitlements which decide the functionality of the application. Second, containers that provide an application directory for supplying read/write operations. Third, powerbox which provides a secure way to open and handle dialog boxes. Together these three methods collaboratively form the application sandbox. Of greatest interest to malware writers, third-party applications are not allowed to interact with kernel-level extensions.

Anatomy of Jailbreaking

For completeness, let us take a brief look at jailbreaking. This attack exploits vulnerabilities in browser, plugin, and iOS components to take control of a victim's iOS device. As a result, jailbreaking [3, 4] culminates in a complete compromise of the iOS device. It primarily uses security vulnerabilities that provide root control of the device. Once the vulnerability is exploited, the attacker is able to run his native code and turn the victim's iOS device into a weapon. Jailbreaking also deploys code signing bypass mechanisms [5] in order to install open source packages such as Cydia [6]. It is also possible to spread malware after jailbreaking. In 2009, a default SSH password vulnerability was exploited on jailbroken iPhones to propagate the iKee [7] worm and its variants.

iPhone Malware–Exploitation Model

A malware infection in an iPhone can be categorized into three distinct classes:

• The first class of malware results from exploitation of security vulnerabilities to get root-level access. Jailbreaking falls into this category. Once rooted, attackers can start services on the iPhone to turn into a malicious entity for spreading malware. In this case, the attacker has to target a specific set of victims. It is difficult because it becomes an action by choice whether the user wants to jailbreak his or her iPhone or not. As a result, attackers force the user to visit a malicious domain using social media tricks to download the malicious code. In a real-time environment, it is hard to spread this class of malware on a large scale as there is a trust layer that Apple provides its users by having applications hosted on Apple's online store. The malware exploits the root privileges as the kernel is already compromised after the exploitation of the security vulnerability. iPhone rootkits [10] are also classified into this class. For example, the Dutch iPhone ransomware [11] belongs to this category of malware.

• The second class of malware exploits the default security model of Apple. This is basically exploited by spyware applications that look legitimate and bypass Apple's App Store verification process. Once in the App Store, infection is easier as the malicious application can be easily disseminated to a number of iOS users. The malicious application might not be able to compromise the kernel as it runs in the sandbox, but it can definitely steal users' sensitive information, history, address book contacts, and so on. This class of malware is a classic example of iPhone spyware that exploits the trust boundary between the user and App Store. For example, SpyPhone [12, 13] falls into this category of malware.

• The third type of malware is a hybrid of both classes of malware discussed above. Hybrid malware is triggered through a generic application that is hosted on the App Store. When a user downloads it, at first it looks legitimate but behind the scenes it starts sending texts to the phone numbers listed in the contacts directory of the victim's iPhone. The text itself carries a link to a malicious website that serves a jailbreaking code. Drive-by download attacks are used extensively for spreading this class of malware. For example, iSAM [14] is a hybrid class of iPhone malware.

The lifecycle of mobile malware is presented in Figure 1.

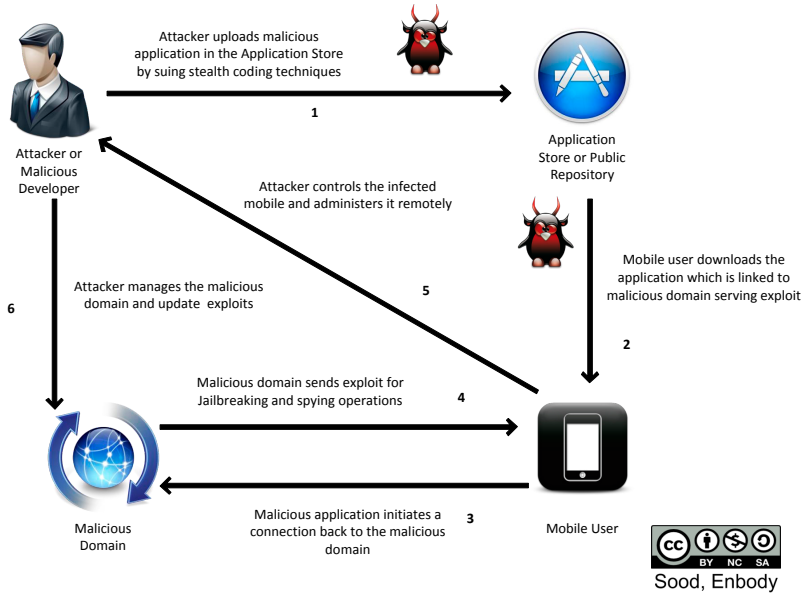


Figure 1: Lifecycle of Mobile Malware

Inside the Apple Kill Switch–Remotely Deactivating Applications

iOS has the built-in protection of a kill switch [15, 16] that enables Apple to kill a malicious application that does not comply with its policies. Applications installed on the iPhone regularly correspond back to the App Store to provide updates about the state of the device. Apple uses blacklisting with a list of applications that are malicious and should be turned off remotely. It is kept in the "unauthorzeapps" file on an Apple server. We performed a quick check on a required URL in order to see which applications are blacklisted. Figure 2 shows that currently there are no applications marked as unauthorized.

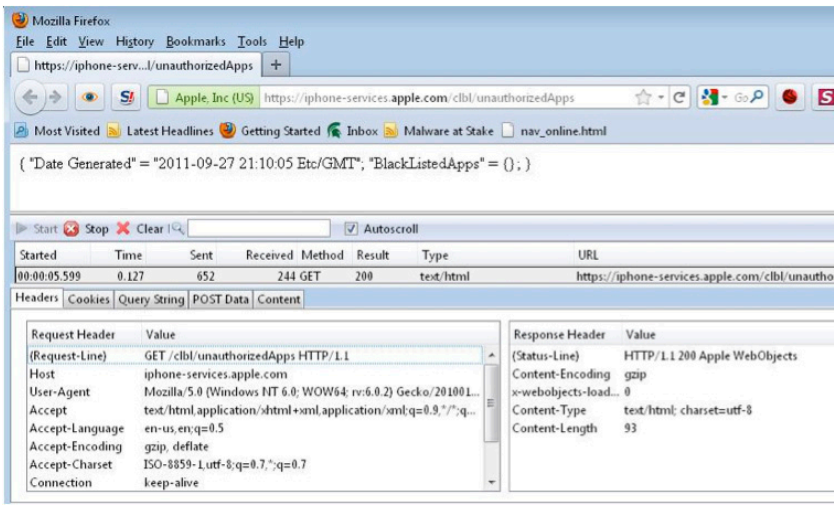


Figure 2: Blacklisted–Unauthorized Apps Check

This functionality is distinct from removing the applications from the App Store because this procedure is designed to deactivate rogue applications remotely. It seems like Apple usually removes the application directly from the App Store. However, the remote deactivation process exists as a proactive defense.

Listing 1: Sandbox profiles

```
kSBXProfileNoNetwork (= "nonet")
kSBXProfileNoInternet (= "nointernet")
kSBXProfilePureComputation (= "pure-computation")
kSBXProfileNoWriteExceptTemporary (= "write-tmp-only")
kSBXProfileNoWrite (= "nowrite")
```

Listing 2: Obfuscation using NSString Object

```
(NSString *)obfuscate_code:(NSString *)string withKey:(NSString *)
key {
  // Create data object from the string
NSData *data = [string dataUsingEncoding:NSUTF8StringEncoding];
char *code_ptr = (char *) [raw_data bytes];

// Mapping the pointer to key data
char *k_data = (char *) [[key dataUsingEncoding:NSUTF8StringEncoding] bytes];
char *key_ptr = k_data;
int key_index = 0;

// For each character in data, xor with current value in key
for (int x = 0; x < [raw_data length]; x++) {

// Apply XOR operation on every character
 *code_ptr = *code_ptr++ ^ *key_ptr++;
 if (++key_index == [key_length]) key_index = 0, key_ptr = k_data; }
  return [[[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding] autorelease];}
```

App Store Application Review–Dependencies and Reality

There are not many details available about Apple's app review procedures. However, based on a developer's view some details can be deduced. Some of the procedures implemented by the App Store are as follows:

• The App Store strictly requires a developer to be enrolled in the Apple's iPhone Developer program [19]. In order to get the approval, the developer has to submit a binary and not the source code, which in turn means that detailed source code analysis is not a part of the verification process. The App Store usually checks for user interface inconsistencies, private API calls and malware. However, malware scrutiny depends on the malware exploitation model mentioned earlier. It is hard to infer details of the Apple application review process, but dynamic and static analysis (pattern matching) are thought to be a part of the process. Given what we know about the review process, it is possible that stealthy programming techniques may be able to circumvent the detection modules.

• The Apple iPhone Licensing Agreement [20] requires a developer not to perform reverse engineering tactics on the applications hosted on the App Store and software developer kit components. Based on this fact, it seems reasonable to assume that Apple itself is following this practice and is not performing reverse engineering on submitted applications. In practice, it is not feasible to reverse engineer the thousands of applications submitted on a weekly basis.

• Most mobile malware aims to steal a user's data at the application layer. In spite of Apple's restrictive policies, default access to user data is available to any application running on an iPhone. The sandboxed environment prevents applications from interacting with each other but a malicious application can subvert the trust boundary of another application. In addition, the sandbox facilitates the process of preventing the background activities that are possible in jailbroken iPhones. Listing 1 shows the different set of sandbox profiles [22] available.

• Malicious applications have been hosted in the App Store in the past. For example, Aurora Feint [17] was considered malicious because the application uploaded users' contacts to the developer's server which is a straightforward breach of privacy. Another example is Pinch Media [18] that followed the same practices of breaching privacy.

Obfuscation–Bypassing Blacklisting

Obfuscation can be useful for legitimate developers as well as for malware writers. Obfuscation is used to prevent the exposure of API functionality. For example, best practices suggest avoiding the embedding of hard-coded credentials in the application. However, developers sometimes hide keys in the code using obfuscation or store credentials on a webserver and rewrite queries after verification. That is, developers implement obfuscation modules for security purposes. Such code needs to pass security testing. Apple requires the application to be robust in nature. As long as the iPhone application is stable and does not crash, the App Store easily accepts an application having obfuscated modules.

While obfuscation is used by legitimate developers to prevent information leakage, a malware writer can use obfuscation to bypass the App Store verification process.

Most of the static analysis tools use blacklisting in which a certain set of strings are blacklisted. When the scanner runs the application code it matches blacklist patterns using regular expressions. Knowing this, it is possible to bypass the static analysis tool using obfuscation. Let us consider an example; In iPhone applications, strings are declared as NSString [21] which are immutable and represented as an array of Unicode characters. Listing 2 shows a prototype of implementing obfuscation using NSString object.

It is possible to obfuscate the strings in an iPhone application and then deobfuscate them at run time. There are many algorithms to perform this functionality. However, the XOR operation is an effective way of obfuscating strings. Generally, the following steps can result in implementation of obfuscated code in iPhone applications:

• The first step is to create a data object from the required string.

• The second step involves the declaration of pointers to the data and encryption key to be obfuscated.

• The third step involves the implementation of counter that runs through every character in a string and embeds a key using the XOR operation.

Code Hiding in Objective-C and Symbols Stripping

Apple is very strict in its review policy about using private API functions that are not documented because these hidden methods can be used by malware. Generally, applications using private API functions are rejected by the App Store. Objective-C does not provide support for private methods, but it is still possible to write methods that hide malicious code. Below are the two most widely implemented steps:

• Objective-C has a dynamic resolution feature in which a method is bound during compile time. The attacker can define a

secret function whose signature matches Objective-C implementation. The secret function is declared in the class method. When that method gets called for the first time, the malicious code is bound to the class privately. This type of procedure is used to circumvent code detection using a tool such as Class-Dump. Listing 3 shows a code prototype that uses dynamic method resolution.

However, a skilled analyst may be able to figure out the presence of stealth code. For example, running Otool on a particular method results in the list of selectors that are used by the respective method. However, it is possible to obfuscate the method by generating selectors at run time using "*NSSelectorFromString()*" functions.

• In Objective-C, it is also possible to create functions that work similarly to instance methods. It means functions can access instance variables easily. These types of functions should be defined in the class implementation. It is not a normal way of doing things, but the desired method never appears in the Objective-C run time which hampers verification. Listing 4 shows the declaration of malicious function *hide_me* with instance variables. The function *hide_me* does not have its own selector rather it uses the selector of *stealth* instance (public) method defined in the class.

The two methods discussed provide a way to design code which can hide from tools that examine code so they can be accepted by the App Store.

Additionally, stripping is a technique used in UNIX platforms to remove unnecessary information from a binary and object files to improve performance. A malicious developer can use stripping to remove information prior to submission of an application binary to the App Store. Doing so removes clues that might indicate the malicious nature of the code.

Exploiting the Remote Server End Points

Generally, all iPhone applications communicate back with a webserver (HTTP End Point) in order to exchange data between the application and the server on a regular basis. It is possible for malware to exploit the HTTP end point mechanism. At the time of verification, Apple performs a behavioral analysis of the application and scrutinizes the communication pattern. At the time of submission, the attacker can make the HTTP end point legitimate and once approved by Apple, the same HTTP end point can be used to serve the exploit code which is downloaded into the victim's phone when the application interacts with a remote server. For example: consider the following scenario:

- Attacker writes an application that interacts with a remote server on the URL <http://www.mal-app-test.com/error.asp >. The error.asp webpage validates the resource and if that resource is not present then it raises an error.
- During the verification process, Apple finds it legitimate and the application is treated as good enough to host on the App Store.
- Once the application is hosted, it is possible to manipulate the "error.asp" webpage to deliver exploit code that is downloaded into the device and performs malicious functions.

This is a legitimate scenario that can be exploited to trigger malware infections in an iPhone.

Listing 3: Code hiding using dynamic resolution

```
// Setting a Class Interface as  Secret
@interface Secret ()
// secret function is defined in the class method
void hide_me(id self, SEL _command);
@end

// Implementing Class
@implementation Secret
@synthesize handle;

// Selecting hide_me secret function and binding into the class method
+ (BOOL)resolveInstanceMethod: (SEL)aSel {
   if (aSel == @selector(hide_me)) {
      class_addMethod(self, aSel, (IMP)hide_me, "v@:");
      return YES;
   }
   return [super resolveInstanceMethod: aSel];
}

// This is an Instance Method holding a reference to hide_me
- (void)stealth {[self hide_me];}
   void hide_me(id self, SEL _command) {
   HIDE(@"Inside hide_me: %d", (LMethod *)self)->handle);
}
@end

//Class Dump Output
@interface Secret : NSObject { int handle; }

// Tool does not provide information about hide_me after static discovery of Class Method
+ (BOOL)resolveInstanceMethod:(SEL)arg1;
@property(nonatomic) int handle; // @synthesize handle;

// Class Dump only lists the Instance Method
- (void)stealth;
@end
```

Listing 4: Code hiding function variables as instance methods

```
(void)stealth { hide_me(self, _cmd);}
```

Cautionary Steps

Users play a critical role in the success of malware. There are a number of steps that a user can follow to reduce risk. These proactive steps are applicable to every smartphone whether Android or iPhone and are discussed as follows:

• Mobile users should not install any unauthorized application from third-party resources. The installed applications must be verified and authorized from legitimate vendors.

• The users should think twice prior to clicking any URL from non-legitimate resources. For example: users should be careful while chatting on social media applications such as Facebook and Twitter. Push notification messages should be scrutinized critically prior to executing any action based on the information in a message. E-mail attachments should not be opened directly until the user is sure about legitimacy.

• It is always advised to install anti-virus software on your mobile device which scans the device for potential suspicious activities and notifying users about changes in the system.

• Usage of strong passwords and avoidance of default security policies is always preferred.

* Users should carefully analyze the behavior of their mobile phones against any types of anomalous activities such battery drainage, high Internet data usage, and slower execution of applications.

Conclusion

In this paper, we have discussed the state of iPhone malware. There is no doubt that Apple has designed a robust verification policy but it is still possible to create stealthy malware that can bypass Apple's verification process. However, doing so requires devising a malicious application in an intelligent way using stealthy techniques such as code obfuscation, stripping, and code hiding. We believe that malware poses an increasingly serious challenge to the security of our devices and we need to be proactive in our defenses to ensure the security of our data and privacy.

ABOUT THE AUTHORS

Aditya K. Sood is a senior security researcher and Ph.D. candidate at Michigan State University. He has worked in the security domain for Armorize, COSEINC and KPMG. He is also a founder of SecNiche Security Labs, an independent security research arena for cutting edge computer security research. At SecNiche, he also acts as an independent researcher and security practitioner for providing services including software security and malware analysis. He has been an active speaker at industry conferences and already spoken at RSA, Virus Bulletin, HackInTheBox, ToorCon, HackerHalted, Source, TRISC, AAVAR, EuSecwest, XCON, Troopers, OWASP AppSec USA, FOSS, CERT-IN, etc. He has written content for HITB Ezine, Hakin9, ISSA, ISACA, CrossTalk, Usenix Login, and Elsevier Journals such as NESE and CFS. He is also a co-author for Debugged magazine.

E-mail: adi.zerok@gmail.com
E-mail: soodadit@cse.msu.edu
Phone: 517-755-9911

Richard J. Enbody, Ph.D., is associate professor in the Department of Computer Science and Engineering at Michigan State University where he joined the faculty in 1987. He has served as acting and associate chair of the department and as director of the computer engineering undergraduate program. His research interests include computer security; computer architecture; web-based distance education; and parallel processing, especially the application of parallel processing to computational science problems. Enbody has two patents pending on hardware buffer-overflow protection that will prevent most computer worms and viruses.

Email: enbody@cse.msu.edu
Phone: 517-353-3389

REFERENCES

1. Malware Goes Mobile, http://www.cs.virginia.edu/~robins/Malware_Goes_Mobile.pdf
2. Mobile Malware Madness and How to Cap the Mad Hatters, https://media.blackhat.com/bh-us-11/Daswani/BH_US_11_Daswani_Mobile_Malware_Slides.pdf
3. IPhone Jailbreak: The Ultimate Guide, http://www.appleiphonereview.com/iphone-jailbreak/iphone-jailbreak/
4. IPhone Hacks Jailbreak, http://www.iphonehacks.com/jailbreak_iphone
5. Bypassing iPhone Code Signatures, http://www.saurik.com/id/8
6. How to Use Cydia: A Walkthrough, http://appadvice.com/appnn/2008/07/how-to-use-cydia-a-walkthrough
7. An Analysis of the iKee.B (Duh) iPhone Botnet, http://mtc.sri.com/iPhone/
8. Code Signing, http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html
9. Application Sandbox, http://developer.apple.com/library/mac/#documentation/Security/Conceptual/CodeSigningGuide/ApplicationSandboxing/ApplicationSandboxing.html#//apple_ref/doc/uid/TP40005929-CH6-SW2
10. iPhone Rootkits, http://www.ekoparty.org/archive/2010/ekoparty_2010-Monti-iphone_rootkit.pdf
11. Hacker Holds Dutch iPhones for Petty Ransom, http://www.wired.com/gadgetlab/2009/11/iphone-hacker/
12. SpyPhone iPhone App Can Harvest Personal Data, http://threatpost.com/en_us/blogs/spyphone-iphone-app-can-harvest-personal-data-120409
13. IPhone Privacy, http://seriot.ch/resources/talks_papers/iPhonePrivacy.pdf
14. iSAM: An iPhone Stealth Airborne Malware, http://www.icsd.aegean.gr/publication_files/conference/462488002.pdf
15. Apple iPhone 'kill switch' discovered, http://www.telegraph.co.uk/technology/3358115/Apple-iPhone-kill-switch-discovered.html
16. Apple's Jobs confirms iPhone 'kill switch', http://www.telegraph.co.uk/technology/3358134/Apples-Jobs-confirms-iPhone-kill-switch.html
17. Aurora Feint iPhone App Delisted For Lousy Security Practices, http://gizmodo.com/5028459/aurora-feint-iphone-app-delisted-for-lousy-security-practices
18. iPhone App Store Secrets - Pinch Media, http://www.slideshare.net/pinchmedia/iphone-appstore-secrets-pinch-media
19. Apple Developer Program, http://developer.apple.com/programs/start/standard/
20. iPhone Developer License Agreement, https://www.eff.org/files/20100127_iphone_dev_agr.pdf
21. NSString Class Reference, http://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSString_Class/Reference/NSString.html
22. Sandbox, http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man3/sandbox_init.3.html

A Practical Approach to Securing and Managing Smart Devices

Sajay Rai, Securely Yours LLC
Philip Chukwuma, Securely Yours LLC
Richard Cozart, Securely Yours LLC

Abstract. We have always said that the strength of an organization's security program is only as strong as its weakest link. Today in most organizations, this weakest link is the use of smart devices like iPads, iPhones, and Android phones. This article provides a practical approach to managing and securing these smart devices.

Introduction

Most organizations in the past deployed BlackBerry devices for corporate use to access e-mail and provide messaging for their employees. These organizations knew that the security on BlackBerries complied with their security policies. A few years ago, a genius named Steve Jobs changed all that. He announced devices like iPads and iPhones. And of course, Google was not going to be left behind. They made their Android OS available to phone manufacturers for free.

With the advent of these smart devices like iPads, iPhones, and Android phones, organizations are now searching for a secure solution for these devices similar to the one they have for their BlackBerry devices. Several vendors have developed Mobile Device Management (MDM) solutions to assist organizations in managing their smart devices.

Typically, IT organizations are chartered to manage these devices. Before they select an MDM solution, they must engage the key departments within the organization to understand the planned usage of these smart devices and gather requirements. Most IT organizations are surprised when they hear the marketing department say how they are planning to use the smart devices, or better yet hear how the CEO or the CFO is planning to utilize a newly acquired iPad.

We suggest a 10-step approach for organizations to plan, implement and manage an MDM program.

Step 1: During the acceptable use policy development, several questions should be asked and answered by key departments within the organization. These questions help identify the requirements and provide input into the next step of defining the IT architecture. It helps to identify the right solutions, after asking the right questions such as:

- Will all devices be deployed by the organization or will users be allowed to bring their own device?
- Is there a need to separate personal vs. corporate data on devices?
- Is personal use allowed or only corporate use? (Can users play Angry Birds?)
- Will employees agree to abide with corporate security policies (e.g. remote wipe, or recording of their phone calls).
- Will confidential data be allowed on smart devices and how it will be monitored and controlled?
- What type of smart devices will be allowed? Apple only? Android only?
- How are you going to manage backing up devices?
- Do devices need to connect to a corporate network?
- Which apps would you like to deploy? Corporate apps? Do you need your own marketplace?

Step 2: Once the answers to these questions have been obtained, a draft IT architecture should be designed to support the deployment of an MDM solution. For example: an answer to the question "corporate device vs. personal device" may imply whether an organization can wipe out the entire device if it is lost, or if they need a secure "container" within the device to house the corporate data.

The IT architecture may also address issues like:

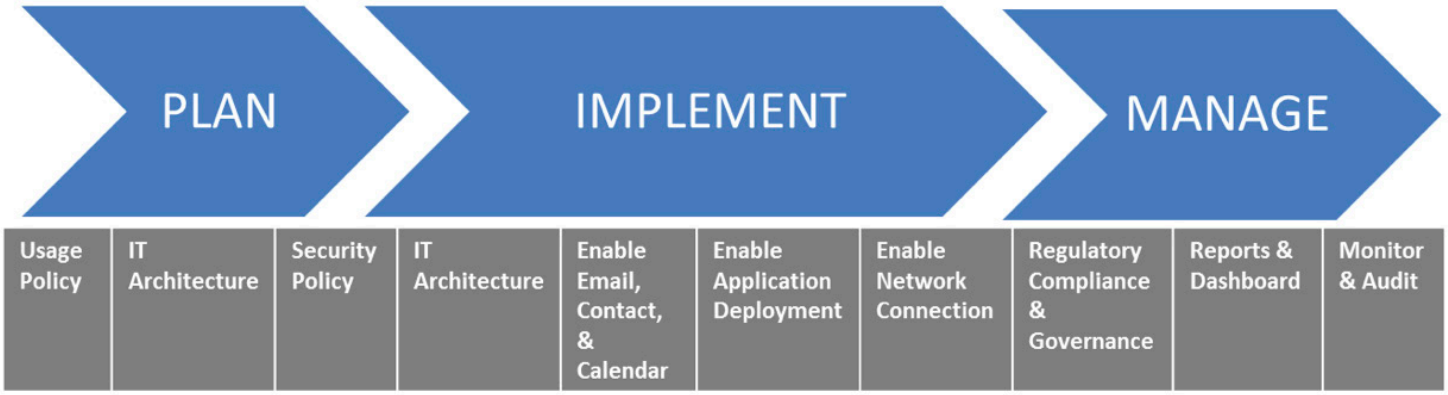
- Cloud-based solution vs. internally deployed
- Hosted vs. self-supported
- Scalability and performance issues based on number of devices
- How current IT architecture will support the mobile architecture

Step 3: Once the requirements have been defined and the supporting IT architecture has been designed, security policies to support the mobile strategy should be developed. The security policy may address some or all of these issues:

- Password policy control
- Encryption requirements
- Port control (Wi-Fi, Bluetooth, camera)
- Remote lock/unlock/wipe
- Asset tracking
- Device configuration (VPN, e-mail, Wi-Fi)
- Delivery and control of applications to the device
- Blacklisting/whitelisting
- Audit and monitoring

Step 4: Now you can use the requirements identified during the planning phase to select the right MDM solution. The implementation of the IT architecture is completed and the Proof-of-Concept (POC) or pilot program implementation is completed. Typically a select few devices are managed under the POC or pilot program. Typically the following steps are executed:

- E-mails are identified for the selected device owners
- A self-registry link is sent to the users
- Users enter the registry information and obtain credentials
- Security policies are pushed down to the device
- Device is ready for use



Step 5: Enable the e-mail, contact and calendar features according to the mobile architecture and policies defined during the planning phase. Typically, organizations combine the features available within the ActiveSync/Lotus Notes features with the features in the selected MDM solution. This step brings the same functions that are available on BlackBerries to other smart devices. At a minimum, organizations should enable the e-mail, contact, and calendar features.

Step 6: Within this step, organizations roll out custom mobile applications to the smart devices. There are several decisions you probably made during the planning phase. You probably answered these questions during the planning phase:

- Are you going to have your own marketplace, from where your employees can download applications?
- Are you going to develop applications for Apple, Android, or both?
- Are most of the applications going to be browser-based applications, or will they be native custom mobile applications?
- Will employees download these applications from the Apple App Store and/or Android Market?
- Are you going to develop these applications in-house or will a third party develop these applications for you?

During this step, you will need two major processes:

- Verify that the source code is written based on the guidelines provided by the Open Web Application Security Project. This requires appropriate source code analysis tools and the ability to perform penetration testing of the application.
- Incorporate your corporate systems development lifecycle process in the development of mobile applications.

Step 7: During this step, the smart devices begin to act like a laptop and can remotely connect to the corporate network and access corporate resources like servers, LAN shared drives and other corporate data. The focus during this step is to ensure that the same rugged security features are deployed that are used for your remote laptop connections. You should look into your remote access policy to ensure that it supports the connection of smart devices to the corporate network.

VPN configuration, encryption parameters, and virtualization concepts may come into play as you deploy the right solution for this step.

Step 8: During this step, appropriate measures are taken to ensure that the implemented solution complies with regulatory requirements. If the smart device is going to contain financial data, personal health data, personally identifiable information, or credit card information (and most likely you will if you will store e-mails on your smart device), this data must be secured. In addition, the installed mobile solution must have the ability to produce appropriate reports to satisfy the audit requirements of these regulations.

Step 9: This step is to provide adequate support to monitor and report on the managed devices. Examples of type of reports include:

- Number of devices supported and inventory of the devices
- The current location of each device
- Number of remote wipes performed in a month/quarter/year
- Number of stolen/lost devices

Step 10: This step provides the necessary support to internal/external auditors when they perform their audits. More and more auditors are targeting smart devices as they are beginning to agree that the smart devices are becoming the "weakest link" of their security program.

Other considerations: Some of the other considerations related to smart devices may include:

- Evaluate your current e-Discovery process to see if smart devices need to be included in this process.
- Litigation Hold: during the litigation process, it may become important to include smart devices during litigation hold.
- Export control laws: if your organization deals with certain technologies which have export control requirements, you may want to track smart devices to ensure that the device is not in the countries where export control laws may be violated.

In summary, an MDM software solution plays a key role in helping organizations manage and secure smart devices, but preliminary planning is the key to success when deploying your smart device strategy.

ABOUT THE AUTHORS

Sajay Rai is the President and CEO of Securely Yours LLC. Securely Yours LLC provides cost-effective innovative solutions in the area of information security, privacy, disaster recovery, business continuity and IT audit. Prior to founding Securely Yours LLC, Sajay was a Partner with Ernst & Young's Security and Risk Advisory practice for 10 years. Prior to Ernst & Young, he was with IBM for 13 years where, among other responsibilities, was instrumental in starting their information security practice, and led the business continuity consulting practice.

E-mail: sajayrai@securelyyoursllc.com
Phone: 866-531-8620

Philip Chukwuma is the CTO of Securely Your LLC. Prior to joining Securely Yours, Philip was a member the Security and Risk Advisory practice at Ernst & Young for 8 years. Prior to joining Ernst & Young, Philip was a member of the Security and Technology services at KPMG, where he delivered security and technology solutions to clients.

E-mail: philipchukwuma@securelyyoursllc.com
Phone: 214-683-8588

Richard Cozart is a senior security consultant with Securely Yours LLC. He specializes in developing and evaluating secure solutions for mobile and web technologies. Prior to joining Securely Yours, Richard was a software engineer for Accenture and co-founder of the web solutions firm, A-Z computers.

E-mail: richardcozart@securelyyoursllc.com
Phone: 313-460-1885

Mobile Applications Security

Safeguarding Data in a Mobile Device World

Sean C. Mitchem, Southwest Research Institute
Sandra G. Dykes, Ph.D., Southwest Research Institute
Stephen W. Cook, Southwest Research Institute
John G. Whipple, Southwest Research Institute

Abstract. With the proliferation of mobile devices in today's information-rich environment, the security of data at rest on the device and in transit will determine the ultimate usability of mobile devices in the defense environment. Relying on the security models provided by the major OS providers such as Apple's iOS or Google's Android is not enough to meet the information protection needs of the defense environment. Researchers at Southwest Research Institute® (SwRI®) are investigating the security models available for application development on the iOS and Android platforms, the threats involved, methodologies for application-level data protection, the intersection between data security and user experience, and best practices for ensuring data security within mobile applications.

Introduction

Today's smartphones and tablets are more than communication devices. They are hip-mounted personal computers, with more memory and processing power than your laptop of just a few years ago. They are an integrated part of our lives… personal and professional. The information they provide is so vital that the Army is piloting their use as standard field issue to every soldier, complete with combat-focused applications [1]. However, smartphones and tablets raise new security issues. They are more likely to be lost or stolen, exposing sensitive data. Malware risks are increased because they connect to the Internet directly rather than from behind corporate firewalls and intrusion-protection systems.

Security of mobile devices focuses on controlling access through the use of device locks and hardware data encryption. While this may be sufficient for individual users, it is insufficient for defense needs. Many documented examples exist of hacking of the device lock, as well as defeats of the hardware-level encryption. Once the device is unlocked, there is generally unfettered access to all apps and their associated data. Military applications require additional application-level access controls to provide data security. Unfortunately, there are gaps in the application-level security model of the two predominant mobile operating systems: iOS from Apple and Google Android. Our ongoing research[1] looks to address these gaps by developing innovative approaches for fine-grained data protection and access control, taking into account mobile device usage patterns, device characteristics, and usability.

Threat Vectors

Many threat vectors for infecting personal computers arise from social-engineering attacks that bypass anti-virus defenses. Similar techniques are used in the smartphone and tablet world by deceiving users into installing malicious apps. Examples include apps that gather personal information, track location, and charge accounts by sending text messages to premium-rate numbers. Using a mobile device to access corporate email or other resources extends the threat to the organization, including the theft of sensitive data [2]. With the acknowledged role of mobile devices and social networks in the revolutions in Egypt, Libya, and Syria, malware and viruses targeted at intelligence gathering and device-usage denial will increase significantly in the future [3].

While viruses and malware targeting mobile devices would share many of the same goals as on the PC, the enhanced capabilities of these devices present expanded attack surfaces through sensors such as GPS, accelerometer, camera, microphone, and gyroscope. Recently, Kaspersky Lab discovered a new threat involving the photo-scanning of Quick Response (QR) codes [4]. QR codes are 2-D matrix barcodes increasingly used in advertising and merchandising to direct mobile-phone users to a website for further information on the tagged item. In this case, users downloaded what they thought was a legitimate app, but instead was malware that sent Simple Message System

(SMS) messages to a premium-rate number that charged for each message [5]. This app could have easily been reconfigured to send covert copies of emails and text messages to an intelligence gatherer instead.

In another example, using the unique capabilities of a mobile device, Georgia Tech researchers were able to use the phone's accelerometer to detect PC keyboard vibrations and decipher complete sentences with up to 80% accuracy. This was done by placing the phone within three inches of the keyboard of a PC, allowing the researchers to pick up the keyboard vibrations and decipher words of up to three to four characters fairly accurately.

The key to understanding the threat vectors of mobile devices is realizing that the devices have more input sources than the conventional PC, and have an extended range outside the typical home or office.

Application Security Models: iOS vs. Android

According to Nielsen, Google's Android is the most-used mobile OS, followed by Apple's iOS [6]. The threat level varies between the iOS and the Android environments, due to their app-distribution models. Because iOS apps are distributed only through the Apple App Store, the Apple review process substantially reduces the threat of downloading a malicious app. This protection, however, is lost if a user "jailbreaks" the device and installs apps from an alternative site or obtains illegal apps from elsewhere.

The Android environment is more wide open. Android apps, although primarily distributed through the Google Android Market, are legally distributed by other means. There is no review or testing of apps, although apps require a digital signature by the developer. Android apps execute in a sandbox on the device and must ask the user for permission to access critical device resources, such as GPS, SMS, and the phone dialer. Unfortunately, it is often difficult for a user to determine whether the requested permission is necessary for that app. Permissions are permanently attached to the app; once the permission is granted, the user cannot revoke it.

When we look at the security models of iOS vs. Android, they can best be summed up as "trust us" vs. "trust them."

Apple iOS

Apple's "trust us" model controls security from malicious apps by providing only one outlet for app distribution and by tightly controlling the iOS Software Development Kit (SDK). Developers submitting apps for distribution must register with Apple to obtain certificates to build and deploy apps. All apps must be signed with the certificate assigned by Apple. Apps must be built using Xcode, Apple's own development tool, and apps may use only the official iOS SDK—no third-party software APIs. Apple's development program requires a yearly fee (currently $99), which must be kept up-to-date. Apple reserves the right to revoke the developer's certificate at any time, which will take any apps developed off the App Store and prevent the developer from distributing any further apps until restoring the certificate. All apps submitted to Apple for distribution are reviewed to ensure proper use of the SDK, adherence to the Program License Agreement, and adherence to a long list of app functionality, subject matter, and content requirements that include ensuring the app is not malware.

Security within iOS is fairly strong, straight-out-of-the-box, but the SDK does not provide additional support to make apps more secure. The device can be controlled through the setting of a 4-digit pin or a password. While this security is not forced upon the user, organizations that use Mobile Device Management (MDM) software for their mobile-device fleet can force the use of pins or passwords, as well as the strength of those access codes. Alphanumeric passwords offer better protection than digit-only pins, as a Russian group showed in cracking the iOS 4-pin device lock [7]. Additionally, iOS features an encryption capability for data stored by applications. By default, all "data at rest" stored in the user partition is automatically encrypted through hardware-based encryption. While this would appear to be sufficient protection for direct attacks against the disk, booting the device with an alternate OS can provide unencrypted access to the disk [8]. Applications with data files marked "protected" will be software-encrypted when stored on-disk. Decryption keys are accessible only when the device is unlocked. The decryption keys are managed by the iOS Keychain, which is always encrypted and, unlike Keychain in the Mac OS X, is not user accessible.

An additional security measure is sandboxing applications and their data stores. Sandboxing provides an app with its own process space and prevents the app from accessing other process spaces. Apple sandboxing does not prevent malicious attacks against an app, but it does limit the damage done by the hacked app to other parts of the device. iOS apps are not allowed to start or execute other apps. Additionally, interprocess communication is allowed only through custom URL handlers, similar in functionality to the http:// and ftp:// URL schemes of Internet browsers.

It is easy to see how Apple's iOS security can be summed up as "trust us," given its complete control over app development for their platform, from the APIs and tools available for app development, to the distribution process, to the device itself. SwRI researchers have found that these mechanisms can be broken and the app accessed, exposing all the data stored within. The iOS SDK does not appear to provide support for specific application-level authentication and authorization [9]. Using an enterprise mobile-device management system can force the use of the device lock to registered devices (such as using an alphanumeric password of sufficient anticracking strength), but apps cannot force the use of device locking as a requirement for installation and execution.

Google Android

Google took a different approach with the Android OS. Whereas Apple controls everything related to the app development and distribution process, Google developed Android as an open source model. Android developers are free to add to the API, use third-party APIs, and distribute apps through

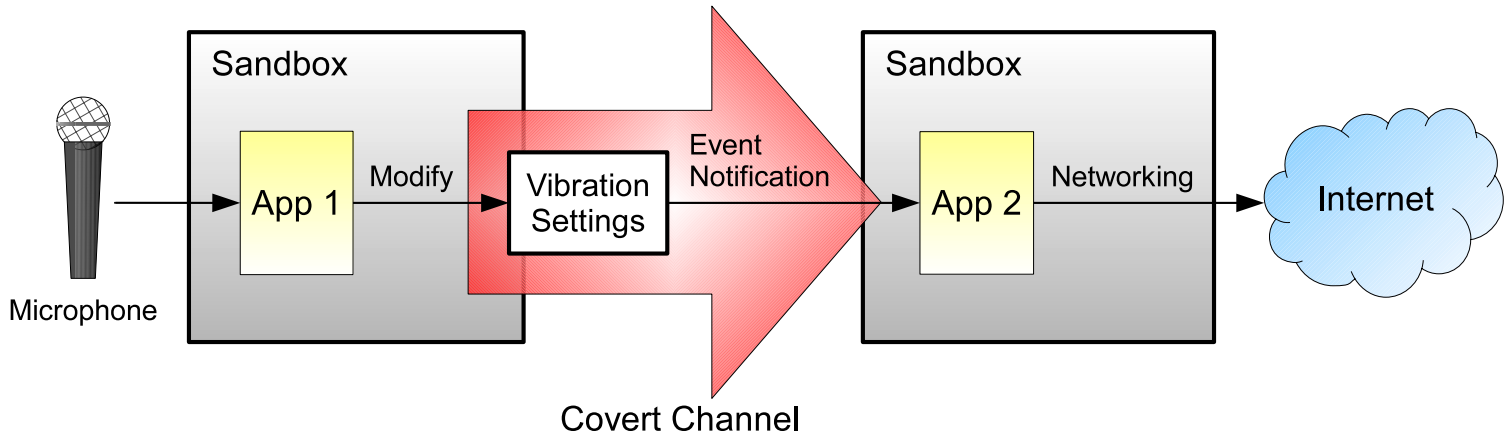


Figure 1 Using covert channels to subvert the Android sandbox-permission model

any means they see fit. While all Android apps must be signed with a certificate, developers can create their own certificates without using a certified certificate authority. Android provides the capability for greater application security than iOS, but the security model is definitely "trust them," as in, "do you trust the developer of the app is providing you a legitimate app that will provide its stated service in the manner described by the app developer and not try to steal information from you or try to damage your mobile device?" Still, Android is not without some basic security measures.

Android security is based on the Linux kernel model, which silos applications into process sandboxes that can reach out of the sandbox via user-granted permissions. All apps are assigned a Unique User ID (UID) when they are installed. However, unlike Linux, this UID is truly unique to each app rather than to each user on a Linux system. Any data the app stores on the device is tagged with this UID, and an app can access only its UID-tagged data unless granted extra-sandbox permissions to other data sources. Unlike iOS, Android applications can share resources and data through the declaration of permissions.

Android grants permissions to resources on a per-application basis during the installation of the application. The user is given a one-time option to install/not install the application after reviewing the resources requested by the application, thereby granting all the permissions or not installing the application at all. Applications requiring dangerous combinations should not be installed. For example, it may be legitimate for an application designed to provide current weather conditions to request access to GPS and networking so the user does not have to continuously input a location; but if the application also requests access to telephony (i.e., dialing phone numbers), a red flag should be raised. Unfortunately, the stock Android OS does not currently support selectively granting permissions at install time; however, third-party add-ons have begun implementing this feature [10]. Permissions cannot change once the app is installed. This does help somewhat. Once installed, an app cannot grant itself additional permissions.

A savvy user aware of the dangerous combinations of resource access can significantly reduce the security threat to their mobile device. However, one research effort [11] demonstrated how to subvert Android's sandboxing-and-permission model through two colluding Trojan-packaged applications using a covert channel. The first application requires permission to the microphone and is enticing to install. The second application requires permission to networking, and its installation is launched by the first. Access to microphone and networking is a dangerous combination if requested by a single application. The Trojan in the first application pulls out sensitive data, such as credit card and PIN numbers, using sophisticated tone-and-speech-recognition algorithms. After extracting the sensitive data, the first application changes the vibration settings (covert channel) on the phone, which then triggers notifications to the colluding second application, as shown in Figure 1.

Accessing vibration settings does not currently require any permissions and does not leave any traces. Through its networking privileges, the second application then transmits the sensitive data to the interested party. This approach is attractive, because high-value information is extracted locally on the phone, significantly reducing the amount of data needed to be transmitted to and processed by the malware master. To mediate this vulnerability, the authors of this research suggest Android restrict covert communication through event notification.

Since Android is an open API designed to run on a wide range of hardware, whole-device encryption is not provided by default unless it is an added feature of the phone maker or carrier. With Android 3.0, full-device encryption is now available and being implemented by some MDM providers.

REGISTER TODAY!

LEARN, DISCOVER, CONNECT!

Join your colleagues in beautiful Salt Lake City, Utah, and benefit from these many opportunities.

70+ Technical Presentations | Tutorials
Sponsored Tracks by Department of Homeland Security, IEEE, & INCOSE
Training/Certification Opportunities | Exhibits

23-26 APRIL 2012

24th Annual

STC *Systems & Software Technology Conference*

MARRIOTT DOWNTOWN HOTEL
SALT LAKE CITY, UTAH

GUEST SPEAKERS

Roberta (Bobbie) Stempfley *(Invited)*
Acting Assistant Secretary, Cyber Security & Communications
Department of Homeland Security, National Protection & Programs Directorate(NPPD)

Mr. Alan Paller
Director of Research
SANS Institute

Other guest speakers are being confirmed. Watch website for further updates.

Follow Us On Facebook
http://www.facebook.com/TheSSTC

ACQUISITION
Sustainment Cost Estimation
Cyber Acquisition
Measuring Enterprise Performance
Reuse of Complex Systems
Mapping & Modeling
CMMI, ISO, Baldridge
Life Cycle Cost Estimation

AGILE DEVELOPMENT
Agile Systems Engineering
Agile & Architecture
Extreme Agile
Surveillance Points
Agile Modeling & Simulation
Agile & Product Quality

ARCHITECTURE
Net-Centricity
SOAs in Embedded Systems
Mobile Access to Enterprise Data
Enterprise Architectures

CLOUD COMPUTING
Migration Strategies
Cost Considerations
Mobile options
Security Considerations

CYBER SECURITY
Trustworthy Software
Information Assurance
Cyber Hardening
Web & Mobile Security
Securing Android Technology

WAR FIGHTING TECHNOLOGIES

ENHANCE ADVANCE MODERNIZE

FOR CONFERENCE & TRADE SHOW INFORMATION, VISIT WWW.SSTC-ONLINE.ORG

Application-level Data Protection

As organizations utilize mobile devices as enhancements or replacements for computers, many will soon work to develop custom applications designed for their own needs. These applications may contain confidential, proprietary information that will need additional protections than are offered at the device or hardware level.

The typical method to protect application data is to protect access via a login specifically for the application. We do this on our PCs with applications that need an additional level of protection over and above the OS-level screen lock; sometimes to protect specific information, sometimes to log who is currently using the application, many times for both. The scenarios required for application locking on PCs also exist on mobile devices.

For iOS-based devices, it appears the only solutions are custom security codes for each app or the use of a third-party solution, which requires jailbreaking the phone. There does not appear to be any support in the iOS SDK for application-level authentication and authorization. A search of the web reveals LockDown Pro and Locktopus, designed to specify application-level password protection to apps on the iPhone; however, both require the Cydia client and a jailbroken phone. This is not to say application-level security cannot be done within iOS.

Android is different. The API includes java.security and javax.crypto packages, which provide security mechanisms that can be included into any app. Additionally, since the Android SDK is open-source, you can develop and roll-your-own application-level security programmatically. The web shows many apps already available for locking existing applications on an Android device, and they do not require rooting the device to enable app locking.

In both Android and iOS, the assumption is "one device, one user." It does not take much imagination to conceive scenarios where tablets and smartphones become like radios in a shop, lined up, charged, and ready for the next user. A flight-line maintenance shop might have several tablets available for maintenance crews, who grab one on the way to turn around an airplane for another mission. Using apps on the tablet to record maintenance done, order parts, configure the plane to mission parameters, etc., it is easy to see where specific application authentication and authorization would be vitally important, not only to protect the apps and data should the crew misplace the tablet somewhere outside the shop, but also to correctly log who accessed the apps and what they did within them.

Data Security vs. User Experience

Computer security is a balance between usability and protection, or more specifically between usability and cryptographic strength. If security controls are too demanding, ample evidence suggests users will circumvent or disable the controls. For example, most users select insecure passwords easy to remember instead of strong passwords they are prone to forget. Smartphones have an added problem in small screens and keyboards that make typing passwords more difficult and add delay. Passwords for email and other accounts are therefore entered once in the device settings and stored, even when the device is powered off. This reduces security to device locking; after a device is unlocked, the user has immediate access to password-protected data and apps. With device locking, it is difficult to guarantee the locking implementation is secure. The Internet contains numerous methods to circumvent device locks.

Smartphones offer the potential for developing new methods of authentication using sensors such as the touch screen, GPS, camera, and accelerometers. Swipe patterns on smartphones are one example; others include picture passwords, tap patterns, and arm motions. Before adopting new authentication methods, there should be a formal analysis of the cryptographic strength to determine the number of pictures, taps, or motions required. The critical question is whether new methods can improve usability for the same level of protection as traditional methods. Consequently, our research includes a study to determine user-perceived usability as a function of cryptographic strength.

Another novel approach to authentication is to apply application-specific constraints. Imagine a field-deployable app for a small combat unit, which provides and disseminates mission information across the unit. The information is highly sensitive and needs protection from loss of the device in the battlefield, but soldiers in battle cannot afford to repeatedly authenticate. An application-specific constraint could be realized using GPS where:

a. **the application registers as a member of the unit's device group;**
b. **the application does not operate if it is located more than 1,000 feet from another registered device; and**
c. **the application automatically uninstalls and erases all data if accessed outside the required separation distance.**

A soldier could use the application without explicit login while protecting data if the device was found or captured. Although hypothetical, this example shows how device sensors can be used to develop new application-level security controls.

Ongoing SwRI research is focusing on the use of mobile-device capabilities to investigate new paradigms for application security and their impact on user experience. Using custom mobile applications, formal cryptographic analysis, and a varied user base, this research is shedding light on how applications and data can be secured on mobile devices with minimal impact on usability.

Summary

Apple's iOS model provides greater security out-of-the-box given Apple's total control over the device, the app-development environment, and the app-distribution model. Google's Android provides greater potential for application-level security due to the extensive and open nature of the SDK. Neither OS model currently provides any significant focus for application-level security. To truly allow mobile devices to replace PCs and laptops, further research and development will be necessary to enable true application security within the mobile-device environment.

ABOUT THE AUTHORS

Sean Mitchem is a Principal Analyst at SwRI, a private non-profit applied research and development laboratory located in San Antonio, Texas. Sean is an Air Force veteran with over 20 years of software development and enterprise architecture experience in areas such as strategic command and control, air traffic control, human resources, and medical systems. His current research interests include mobile device security, smart grid security, energy market economics, and electrochemical energy storage systems.

E-mail: Sean.Mitchem@swri.org

Sandra Dykes, Ph.D. is a Principal Scientist at SwRI, specializing in protocol design, network monitoring, statistical modeling, and malware detection. Her research includes usable security, infrastructure protection, insider threat, and statistical anomaly detection. Dr. Dykes received her B.S. in Chemistry from the University of Texas at Austin and her Ph.D. in Computer Science from The University of Texas at San Antonio.

E-mail: Sdykes@swri.org

Stephen Cook is a Senior Research Analyst at SwRI. His background and expertise are in software security, smart grid security, parallel and distributed computing, compilers, and object-oriented and generic programming. He has an M.S. in Computer Science from Texas A&M University and a B.S. in Geophysical Engineering from the Colorado School of Mines.

E-mail: Scook@swri.org

John Whipple is a Research Analyst at SwRI. John has designed and developed software for medical, space science, intelligent transportation and commercial data system applications. He is currently researching mobile device security as well as exposing novel uses of smartphone sensors through data mining.

E-mail: Jwhipple@swri.org

Communications and Embedded Systems Department
Southwest Research Institute
6220 Culebra Road
San Antonio, TX 78228
Phone: 210-522-2698

REFERENCES

1. Horn, Leslie. "Army might give troops smartphones soon." PCmag.com, 18 July, 2011. <http://www.pcmag.com/article2/0,2817,2388629,00.asp#fbid=g-34uH0qMXP>
2. Rothman, Wilson. "Smart phone malware: The six worst offenders." MSNBC.com, 16 Feb., 2011. <http://technolog.msnbc.msn.com/_news/2011/02/16/6063185-smartphone-malware-the-six-worst-offenders>
3. Riberio, John. "Mobile Phone Tapping Allegation Disrupts Indian Parliament." PCWorld, 26 Apr., 2011. <http://pcworld.com/article/194961/mobile_phone_tapping_allegation_disrupts_indian_parliament.html>
4. Shanklin, Will. "QR codes are being used to spread malware." geek.com, 21 Oct., 2011. <http://www.geek.com/articles/mobile/qr-codes-are-being-used-to-spread-malware-20111021/>
5. Wasserman, Todd. "New Security Threat: Infected QR Codes." Mashable Tech, 20 Oct., 2011. <http://mashable.com/2011/10/20/qr-code-security-threat/>
6. Nielsen. "In U.S. Smartphone Market, Android is Top Operating System, Apple is Top Manufacturer." <http://blog.nielsen.com/nielsenwire/?p=28516>
7. Byrne, Ciara. "Russians crack Apple's iOS encryption." VentureBeat, 25 May 2011. <http://venturebeat.com/2011/05/25/russians-crack-apples-ios-encryption/>
8. McClune, Rory. "Apple iOS Devices and Encryption." Blog Posting, 7Elements, 16 Dec., 2010. <http://blog.7elements.co.uk/2010/12/apple-ios-devices-and-encryption.html>
9. Security Overview: Authentication, Identification, and Authorization. <https://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Concepts/Concepts.html#//apple_ref/doc/uid/TP30000976-CH203-TPXREF101>
10. Whisper Systems. "Device and data protection for Android", <http://whispersys.com/whispercore.html>
11. R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia and X. Wang, "Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones." In Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS), 2011.

NOTES

1. The authors gratefully acknowledge the significant ongoing contribution to the research by Allison Bertrand, SwRI. This work is supported under Southwest Research Institute Internal Research Grant 10-R8244.

ADDITIONAL READING

1. Redman, Phillip; Girard, John; Wallin, Leif-Olof. "Magic Quadrant for Mobile Device Management Software", Gartner Research Note G00211101, Gartner, Inc., 13 April 2011, <http://www.gartner.com> (available for free through several major MDM vendors)
2. Redman, Phillip; Basso, Monica. "Critical Capabilities for Mobile Device Management", Gartner Research Note G00213877, Gartner, Inc., 29 July, 2011, <http://www.gartner.com/technology/>
3. Chen, Brian X. Always On – how the iPhone unlocked the anything-anytime-anywhere future – and locked us in. Cambridge, MA: Da Capo Press, 2011.

Engaging the Community

Strategies for Software Assurance Curricula Outreach

Dr. Carol A. Sledge, Software Engineering Institute

Abstract. How to better achieve secure and correctly functioning software systems, regardless of their origins, application domain, or operational environments? Engaging a knowledgeable team of educators to develop curricula, courses, and other materials for the discipline of software assurance is but the start. If we build it, will they come? In this paper, I explore strategies this team of educators used to encourage the community of computing educators to adopt software assurance curricula.

Background

Our lives and our world depend on software. Highly complex, interdependent software systems are critical to virtually every aspect and domain of society today. However ubiquitous software has become, security advances have not been commensurate with the vital role software now plays. As a consequence, our exposure to risk is ever increasing.

The complexity of software and software-intensive systems has inherent risk: it obscures the essential intent of the software, masks potentially harmful uses, precludes exhaustive testing, and also introduces additional problems with respect to the operation and maintenance of the software. The interdependence of these systems means attackers can focus on the most vulnerable component to damage the larger system(s), while today's interconnectivity makes the proliferation of malware easy, but the identification of its source difficult [1]. Threats are large and diverse, from unsophisticated opportunists to technically savvy entities backed by organized crime [2], nation states, and similar organizations with malicious intent.

Software Assurance Curriculum Project

Understanding the importance of the software assurance discipline for protecting national infrastructures and systems, the DHS National Cyber Security Division has recognized the growing need for skilled practitioners in this area. At the direction of DHS, researchers in SEI[1] at Carnegie Mellon University developed the Software Assurance Curriculum Project (SwACP). The SwACP development team is composed of knowledgeable educators from a number of institutions of higher education,[2] who collectively have substantial background in software assurance research, software engineering research and practice, and software engineering education [3], and who participate in related professional society curricula development.

What is software assurance? The definition used by the SwACP team is, "Software assurance (SwA) is the application of technologies and processes to achieve a required level of confidence that software systems and services function in the intended manner, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures [4]."[3] This is a slight extension of the Committee on National Security Systems' definition [5] used by our DHS sponsor.

Many colleges and universities have degree programs in areas such as software engineering and information security, but programs and tracks in software assurance are lacking. The work of the SwACP addresses this gap.

The focus of the SwACP is to:

- Identify a core body of knowledge that educational institutions can use to develop Master of Software Assurance (MSwA) degree programs
- Mentor universities in developing standalone MSwA degree programs and tracks within existing software engineering and computer science master's degree programs
- Promote an undergraduate curriculum specialization for software assurance
- Address community college needs

To date the SwACP team has produced four volumes[4]:

- Master of Software Assurance Reference Curriculum[5] [4]
- Undergraduate Course Outlines[6] [6]
- Master of Software Assurance Course Syllabi [7]
- Community College Education[7] [8]

In addition to these reports, the team also developed papers [1, 3, 9,10, 11], presentations [12,13], and workshops [14].[8]

Both the Association for Computing Machinery (ACM) and the IEEE Computer Society (IEEE-CS) have recognized the MSwA Reference Curriculum as appropriate for a master's program in software assurance. This formal recognition signifies to the educational community that the MSwA Reference Curriculum is suitable for creating graduate programs or tracks in software assurance.[9]

Outreach

Defining transition strategies for future implementation of the software assurance curricula is one of the goals of the SwACP. Many SwACP team members had been previously involved in curriculum work and understood the need to have a comprehensive plan for promoting the transition and adoption of the various curricula. In the academic world, transition is a lengthy process, with a number of potential barriers to adoption. While introducing one new elective course may be relatively easy, introducing a new track takes significant effort, and adding a new degree program is a real challenge. Many barriers exist: insufficient interested students in the surrounding geographic area,

lack of qualified faculty, lack of administrative support, funding, etc. For the SwACP to succeed, a comprehensive outreach and promotional plan was needed.

For the first volume produced, the MSwA Reference Curriculum, planned promotional activities targeting educators included [3]:

- Publicity–SwACP team members disseminated announcements, press releases, and flyers regarding the team's work via email, websites, educational publications, and professional societies; they also distributed promotional materials to colleagues when they attended conferences.
- Software assurance education discussion group–We established a LinkedIn discussion group in which faculty interested in implementing all or portions of the curriculum could interact with the team and other colleagues who are using the curriculum.
- Awareness–Team members conducted and videotaped[10] an awareness-raising faculty workshop at the Conference on Software Engineering Education and Training (CSEET) 2010 [14]. This workshop was among the various presentations given at faculty and curriculum development venues. Additionally an overview podcast was produced, including a discussion of what students and employers can expect.[11]
- Mentoring–The SwACP team is mentoring universities and faculty members who wish to offer a course, track, or MSwA degree program. This support includes review of implementation plans and course outlines and advice on references and other materials.
- Publication–SwACP team members have written papers and given talks on the curriculum.
- Professional society recognition–As mentioned previously, both ACM and IEEE-CS officially recognize the MSwA curriculum.

For transition and promotion of the MSwA Reference Curriculum, early adoption is important. The Stevens Institute of Technology, home of one of the SwACP team members, was the first school to adopt elements of the curriculum: it has developed two tracks in software assurance within its Master of Science in Software Engineering program. One track is for students who anticipate a career in secure software development, while the other is for students interested in acquisition and management of trusted software systems. For those students who already have an advanced degree or who are not ready to commit to a full graduate program, graduate certificates are available [3].[12] Consideration and plans for adoption of courses and tracks are underway at the universities of the team members, as well as other schools.

Outreach: Leverage and Trust

For the MSwA curriculum transition and promotion goal, all planned activities were successfully completed and continue to be pursued. Long term, a key point of leverage is the continued participation by SwACP team members in reviewing and updating professional society curriculum guidelines. For example, SwACP team member Mark Ardis is the chair of the Software Engineering 2004 Review Task Force, a joint effort of the ACM and the IEEE-CS. This task force has collected comments from the software engineering community about the need to update Software Engineering 2004, the recommended guidelines for undergraduate software engineering education. Ardis noted that several reviewers had commented on the need for more material on software security and assurance. SwACP team member Elizabeth Hawthorne is chair of the ACM Committee for Computing Education in Community Colleges and is also a member of the ACM delegation to the Steering Committee of the joint ACM and IEEE-CS Computing Curriculum: Computer Science 2013,[13] an effort in its planning stages focused on international curricular guidelines for undergraduate programs in computing. She reported that one new knowledge area under consideration is dedicated to "computer security" (called Information Assurance and Security).[14] Through these relationships, the SwACP team can stay updated and engaged with current curricula development efforts and seek ways to leverage the curricula the team developed in graduate, undergraduate, and community college programs.

In the short term, the need for quick educational community feedback on draft SwACP documents and for broader awareness and involvement suggested a focused leveraging of trusted, personal relationships, in addition to the promotion and transition mechanisms already cited. Specifically, I was tasked with extending the SwACP team's ongoing efforts to faculty and entities whom I knew to be involved in course, resource, and curriculum development for software engineering, information systems, information assurance, computer science, information security, etc. at the master's, undergraduate, and community college levels. By no means was this complete coverage, but the trusted relationships increased the likelihood that faculty would engage (and redistribute the information). Utilizing relationships with other colleagues, appropriate faculty at, for example, the U.S. Service Academies, were specifically targeted via a trusted intermediary.

Targeted faculty included[15]:

- Past participants in the National Science Foundation (NSF)-funded Information Assurance Capacity Building Program at Carnegie Mellon University
- Principal investigators of the 15 NSF-funded Advanced Technological Education (ATE) Centers and through the NSF ATE program manager to other NSF program managers
- Those at 17 NSA/DHS Centers of Academic Excellence in IA Education (CAE/IA) and CAE-Research (CAE-R) programs[16]
- California State University Discipline Council (department heads of computer science, information science/information systems, and software engineering at the 23 schools that make up the council)
- Participants in the educational outreach and curriculum development activities and members of the NSF Science and Technology Center Team for Research in Ubiquitous Secure Technologies[17]

• Members of the Association of Computer/Information Sciences and Engineering Departments at Minority Institutions[18]

• Members of various faculty email lists, including personal lists of faculty in related disciplines interested in course and curriculum development, and those working on articulation agreements with community colleges

• U.S. service academies and postgraduate schools[19]

Over the years, faculty from these entities formed collaborative relationships to create, adapt, adopt, and share new materials as appropriate for their departments and prospective students, as well as for others. Given their interest in related disciplines, these communities of interest were prime targets for our outreach effort.

In addition to faculty and academic institutions, it was important to leverage related government and practitioner efforts. Collaborating with organizations in the DoD and NIST, the DHS National Cyber Security Division Software Assurance (SwA) Program cosponsors the Software Assurance Community. In this community, members of government, industry, and academia come together to discuss, develop, and implement software security practices, methodologies, and technologies in forums and working groups.[20] Because of SwACP team member participation in this community, the15th semi-annual SwA Forum in September 2011 examined the implications of trends and emerging factors in training and education for software assurance workers. The NIST National Initiative for Cybersecurity Education (NICE) has a goal to "bolster formal cybersecurity education programs encompassing kindergarten through 12th grade, higher education and vocational programs."[21] At the December 2011 DHS Working Group meeting, co-chaired by the SwACP team lead, Nancy Mead, the alignment with NICE was discussed.

Outreach Outcomes

From the beginning, the SwACP recognized the importance of transition strategies for the implementation of the software assurance curricula, including the ongoing promotion of the curriculum work and outreach to the various communities of interest to encourage them to participate. Given the time constraints, the various educational levels addressed, and potential constituencies involved, multiple people and entities employed multiple outreach mechanisms, coordinating where possible with related efforts.

Challenges to our outreach effort include the usual potential barriers to adoption of courses, tracks, and curricula, including the time and resources needed, especially in light of sometimes-severe funding cuts in departments. Another challenge was the alignment and timing regarding revision cycles of both departmental and the related professional curriculum development efforts.

Outreach mechanisms that are proving effective include:

• The Build Security In website, sponsored by DHS, and the SEI MSwA website

• Ongoing SwACP team member participation (previously and currently) with professional curricula development activities

• Papers and presentations at appropriate educator conferences and workshops

• Leveraging trusted relationships with educators in related disciplines to increase the likelihood of engagement and dissemination (to other interested faculty) of information related to SwA curricula and content.

One example of successfully leveraging trusted relationships with educators is the Department of Computer Science at the U.S. Air Force Academy. They recently undertook a curriculum review that defined multiple cross-curricular initiatives to support program outcomes, including "secure programming" (security and software assurance) [15]. Among the resources used was the Undergraduate Course Outlines [6]. They are also considering the development of some undergraduate course exercises and projects that focus on secure coding and software assurance, to be incorporated into existing undergraduate courses as a means to integrate these topics as "natural and normal practices inherent to software development."[22]

Faculty and educators have contacted the SwACP team lead for information about how to build a BS or MS program with an SwA concentration.[23] One department at the University of Houston has adopted significant portions of the software assurance curriculum in their program by incorporating elements in several courses, where appropriate, with the majority in focus courses (two each in the undergraduate and graduate programs).[24]

Other outreach mechanisms are early in their respective cycles or require more of a critical mass to be effective. For example, the Software Assurance Education discussion group on LinkedIn provides a forum for faculty to share problems and experiences in teaching software assurance courses. As more educators incorporate software assurance topics, modules, and courses into their departmental programs, we hope they will utilize this forum. Ongoing participation in the related government and practitioner efforts will help with the alignment and leveraging of these activities, with the common goal to increase awareness, participation, and adoption of appropriate software assurance practices.

Summary

The SwACP team feels that software assurance education at all levels is essential to ensure that software and software-intensive systems are developed with assurance in mind [11]. While software assurance supports and complements the educational objectives of a software engineering program, it also supports and complements the educational objectives of related disciplines such as computer science and information systems. Engaging knowledgeable educators experienced in related curriculum development to produce software assurance curricula and related materials is but one part of this DHS-funded effort. Multiple mechanisms must be continually utilized to reach the various educator communities to increase awareness, encourage participation, and ultimately adopt software assurance topics, courses, tracks, and curricula. Certain outreach strategies have proved to be successful in the relatively short time the SwACP has been in existence. Leveraging professional curricula development entities, as well as alignment with related government efforts, while longer term, should provide the foundation for sustainment.

ABOUT THE AUTHOR

Carol A. Sledge, Ph.D., is a senior technical staff member at SEI. She is also a Carnegie Mellon adjunct faculty member. Her research interests include software assurance and SoS interoperability. Previously at CERT, Sledge led development of a reference curriculum in survivability and information assurance. She is a senior member of the IEEE, ACM, and AIAA. Sledge received her master's and doctorate in computer science, and her bachelor's degree in mathematics from University of Pittsburgh.

4500 Fifth Avenue
Pittsburgh, PA 15213-2612
Phone: 412-268-7708
E-mail: cas@sei.cmu.edu

REFERENCES

1. Mead, Nancy R., Julia H. Allen, Thomas B. Hilburn, Andrew J. Kornecki, Rick Linger, and James McDonald. "Development of a Master of Software Assurance Reference Curriculum." International Journal of Secure Software Engineering 1.4 (2010): 18-34. Print.
2. Anderson, R. J. Security Engineering: A Guide to Building Dependable Distributed Systems. 2nd ed. New York: John Wiley, 2008. Print.
3. Ardis, Mark, and Nancy Mead. "The Development of a Graduate Curriculum for Software Assurance." AMCIS 2011 Proceedings - All Submissions. 17th Americas Conference on Information Systems (AMCIS), Detroit. Web. 28 Oct. 2011. <http://aisel.aisnet.org/amcis2011_submissions/34/>.
4. Mead, Nancy R., Julia H. Allen, Mark Ardis, Thomas B. Hilburn, Andrew J. Kornecki, Rick Linger, and James McDonald. Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum. Rep. no. CMU/SEI-2010-TR-005. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2010. Web. <http://www.sei.cmu.edu/library/abstracts/reports/10tr005.cfm>.
5. United States. Instruction No. 4009, National Information Assurance Glossary. By Committee on National Security Systems. Revised June 2009. Print.
6. Mead, Nancy R., Thomas B. Hilburn, and Richard C. Linger. Software Assurance Curriculum Project Volume II: Undergraduate Course Outlines. Rep. no. CMU/SEI-2010-TR-019. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2010. Web. <http://www.sei.cmu.edu/library/abstracts/reports/10tr019.cfm>
7. Mead, Nancy R., Julia H. Allen, Mark Ardis, Thomas B. Hilburn, Andrew J. Kornecki, and Rick Linger. Software Assurance Curriculum Project Volume III: Master of Software Assurance Course Syllabi. Rep. no. CMU/SEI-2011-TR-013. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2011. Web. <http://www.sei.cmu.edu/library/abstracts/reports/11tr013.cfm>.
8. Mead, Nancy R., Elizabeth K. Hawthorne, and Mark Ardis. Software Assurance Curriculum Project Volume IV: Community College Education. Rep. no. CMU/SEI-2011-TR-17. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2011. Web. <http://www.sei.cmu.edu/library/abstracts/reports/11tr017.cfm>.
9. Ardis, Mark, and Peter Henderson. "Software Engineering Education (SEEd): Educating Our Students to Build Security In." ACM SIGSOFT Software Engineering Notes 35.6 (2010). Print.
10. Mead, Nancy R., Linda M. Laird, and Dan Shoemaker. "Getting Secure Software Assurance Knowledge into Conventional Practice: Three Educational Initiatives." COMPSAC. Proc. of 2011 IEEE 35th Annual Computer Software and Applications Conference. 193-98. Print.
11. Mead, Nancy R. and Dan Shoemaker. "Two Initiatives for Disseminating Software Assurance Knowledge." CrossTalk (September-October 2010): 25-29. Web. <http://www.sstc.hill.af.mil>.
12. Mead, Nancy and Joe Jarzombek. "Educating the Next Generation of Software Engineering Professionals (Keynote)." Colloquium for Information Systems Security Education, June 2011, Fairborn, OH.
13. Sledge, Carol A. "Master of Software Assurance Curriculum: A Briefing for Faculty." 2010 Workshop on Curriculum Development in Security and Information Assurance (CDSIA). May 21, 2010, San Jose, CA.
14. Mead, Nancy; Jeff Ingalsbe, and Mark Ardis. "Faculty Development Workshop: How to Get Started in Software Assurance Education." Conference on Software Engineering Education and Training, March 2010, Pittsburgh, PA.
15. Hadfield, S., D. Schweitzer, D. Gibson, B. Fagin, M. Carlisle, J. Boleng, and D. Bibighaus. "Defining, Integrating, and Assessing a Purposeful Progression of Cross-Curricular Initiatives into a Computer Science Program." Proc. of the 41st ASEE/IEEE Frontiers in Education Conference, October 2011. Print.

NOTES

1. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.
2. In addition to educators in the SEI, collaborators include educators from Embry-Riddle Aeronautical University, Monmouth University, Stevens Institute of Technology, University of Detroit Mercy, Union County College, and University of Arkansas, Little Rock.
3. Note that computing capabilities may be acquired through services as well as new development. Recovery is an important capability for organizational continuity and survival.
4. These volumes, plus related information and faculty resources, can be found at <www.cert.org/mswa/> and the DHS Build Security In (BSI) website <https://buildsecurityin.us-cert.gov/bsi/>.
5. The Reference Curriculum addresses topics such as assurance across life cycles, risk management, assurance assessment, assurance management, system security assurance, assured software analytics, and system operational assurance. This can be implemented as a standalone program, or as a track within an existing master's program, such as a Master of Software Engineering program.
6. This set of outlines includes seven course descriptions that could be included in a software assurance specialization track of a traditional computer science degree program. To provide an emphasis on software assurance topics in the first year of a curriculum, descriptions of alternative forms of Computer Science I and II are included.
7. The ACM Committee for Computing Education in Community Colleges (<www.acmccecc.org>), led by Elizabeth Hawthorne, partnered with the SwACP to produce this volume that includes discussion of existing curricula related to software security that are suitable for community colleges. The target audiences are students planning to transfer to a four-year program and students with prior undergraduate technical degrees who wish to become more specialized in software assurance. The report includes course outlines and identification of resources.
8. These are just a few of the papers and presentations. Reference [1] provides the best overview of the SwACP, while reference [11] provides a much briefer synopsis, including those artifacts, foundational materials, and recent curriculum guidelines referenced in the development of the SwACP curricula.
9. The ACM and its partner, the IEEE-CS, have developed several computing curricula and are community leaders in curricula development.
10. The CSEET 2010 three hour workshop is available at <https://www.vte.cert.org/vteweb/RequestAccess/ClassPreview.aspx?Classid=120>
11. Podcast is available at <http://www.cert.org/podcast/show/20101026mead.html>
12. The environment that allowed SIT to quickly create its software assurance program, as well as potential adaptations of the MSwA curriculum for information systems curricula are also described in [3].
13. <www.cs2013.org>
14. A recent addition to the SwACP, Remzi Seker, University of Arkansas, Little Rock, is a member of the IEEE-CS delegation.
15. In addition to those targeted by other team members and means, this outreach effort included faculty and educational institutions granting master's, bachelor's, and associates' degrees in 21 states and the District of Columbia.
16. <http://www.nsa.gov/ia/academic_outreach/nat_cae/index.shtml>
17. <http://www.truststc.org/>
18. <http://www.admiusa.org/>
19. U.S. Air Force Academy, U.S. Military Academy, U.S. Naval Academy, Air Force Institute of Technology, Naval Postgraduate School, U.S. Naval War College, and U.S. Coast Guard Academy
20. Information about SwA Community activities including forums and working groups can be found at <https://buildsecurityin.us-cert.gov/swa/forums.html>; see especially the Workforce Education and Training Working Group.
21. <http://csrc.nist.gov/nice/aboutUs.htm>
22. Communication to Carol Sledge by Dr. Steve Hadfield, Associate Professor and Curriculum Chair, Department of Computer Science, U.S. Air Force Academy
23. These include Hampton University, TRUST, Gunter Air Force Base, Southeast Missouri State University, Cleveland State University, and University of Detroit Mercy.
24. Communication to Nancy Mead by Wm. Arthur Conklin of the Department of Information and Logistics Technology

The PC Evolution and Diaspora

James A. Senn, Ph.D., California Polytechnic State University

Abstract. This paper examines the development, evolution, and diaspora of the personal computer. An overview of the Innovation Diffusion Technology (IDT) model is presented. Using this as a framework, the personal computer is categorized from multiple perspectives. The direction, durability, and mutations (the diaspora) of the personal computer are presented using Ansoff's model of diversification.

Introduction

In August of 2011 Hewlett Packard, the world's largest seller of PCs, confirmed it was looking to sell off its personal computing business—possibly getting out of the hardware game altogether and dropping its tablet and smartphone operations as well. This event along with IBM's decision in 2004 to sell its PC business line to Lenovo, a China-based firm, is a harbinger that the low-margin PC business may not be worth pursuing. Concurrently the rise of alternatives to traditional PCs, the tablet(s), continues unabated—with forecasts through 2011 at 60 million tablets and in 2012 to be 90 million units [1]. There still will be over 100 million PCs sold worldwide for several years because people need them for certain tasks [2]. Many of the habits we associate with personal computers can be carried out with touchscreen and an Internet connection—done anywhere, and quickly. The iPhone has demonstrated what could be done with a relatively small device that could single task very well. With Android and Apple netbooks being circulated, this idea of a small, relatively inexpensive device connected to back-end services is the leading edge of a paradigm-shifting platform—along with the application layer in the private cloud [3].

According to Brodkin [4], "since the personal computer debuted in 1971, a Darwin-esque evolution process has lifted the PC from modest beginnings to its prevailing role as an indispensable part of life in the 21st century" … "evolving from clunky commercial flops to slick, high-powered machines that play a vital role in our daily lives, both for work and play." Personal computers have been the technology engine drivers from Intel to Microsoft to Dell to HP to Google to Facebook. But the rise of mobile computing is upending the technology business and is simultaneously redefining what is a personal computer and how we use it [1, 5].

This paper is a sector case study that seeks to examine the development, evolution, and diaspora of the personal computer. The remainder of this paper proceeds as follows. First, we present an overview of the IDT model and discuss innovation characteristics. Then we categorize the personal computer from multiple perspectives using this as a framework. Finally, we comment on the direction, durability, and mutations (the diaspora) of the personal computer using Ansoff's model of diversification.

Overview of Diffusion of Innovation

Diffusion of innovation theory [6] describes the process through which new ideas, practices, or technologies are spread into a social system. According to Murray [7], diffusion of innovation theory holds that innovation diffusion is a general process, not bound by the type of innovation studied, by who the adopters are, or by place or culture, such that the process through which an innovation becomes diffused has universal applications to all fields that develop innovations. Diffusion is defined as the process in which an innovation is communicated through certain channels over time among the members of a social system. Innovation is an idea, practice, or object that is perceived as new by an individual or other unit of adoption. In addition, innovation also does not necessarily mean better or that the new idea is more beneficial to an individual. Whereas innovation can refer to something abstract, like an idea, it can also be concrete, like a new piece of technology. This article focuses specifically on the personal computer as a particular type of innovation of interest.

Rogers [6, 8] suggested that there were/are four main elements in the diffusion process:

- The innovation
- The communication channels through which the innovation is diffused
- Time
- The social system

The end results [9] of diffusion are adoption, implementation, and institutionalization. Diffusion researchers across many academic disciplines have identified a consistent process through which innovations are diffused into social systems. There is generally a period of slow growth, followed by more rapid expansion, followed ultimately by a plateau or another slow growth period. Different characteristics of the innovation, communication channels, and social system are likely to have varying influences at different times throughout the diffusion process [10].

Individuals vary in their willingness to accept new ideas and change [11]. Rogers [6] classified adopters into the following five categories on the basis of their rates of adoption:

- Innovators, who are among the first 2.5% in the population to adopt the innovation and demonstrate an adventurous, cosmopolite nature.
- Early adopters, who fall into the next 13.5% of adopters and who are integrated closely into the social network and are often opinion leaders.
- The early majority, who are the next 34% of adopters and are described as deliberate followers.
- The late majority, the next 34% who are often skeptical of the innovation at first but eventually succumb to peer pressure.
- Laggards, who are the final 16% and who tend to be more traditional and isolated compared with earlier adopters. Individuals who are among the last to adopt an innovation often exhibit the longest decision-making processes prior to deciding.

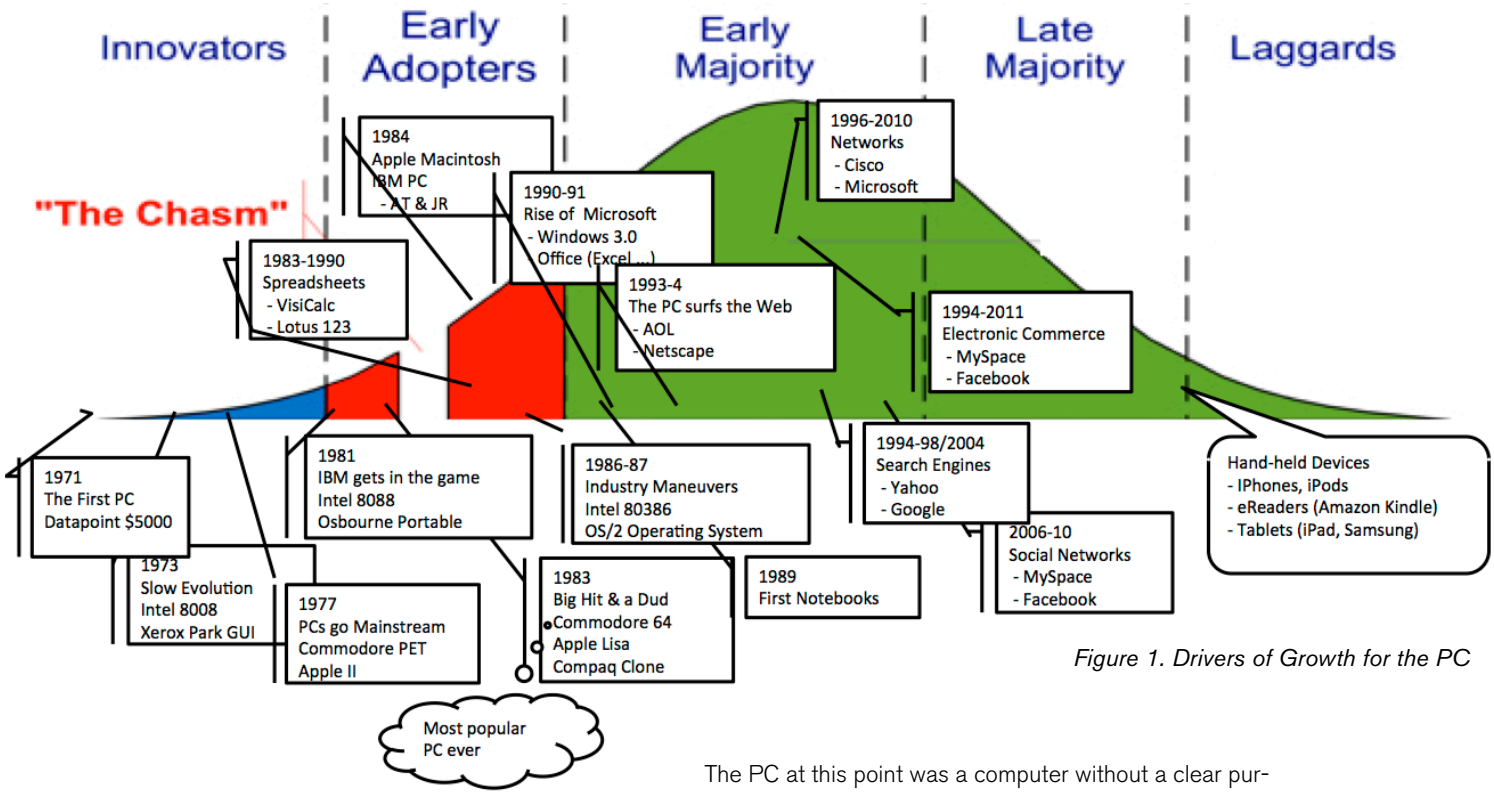


Figure 1. Drivers of Growth for the PC

Marketing of the PC

The PC did not magically appear in its current wide-screen, multi-core, viewing form overnight. It took many years to evolve from the IBM PC of 1981 to the high-powered tech gadgets. The original idea of the PC was sound: using off the shelf parts combined with a relatively open, but curated set of standards to avoid reinventing from one version of the PC to the next. There are a number of milestones passed along the way, from the introduction of the IBM PC in August 1981, and moving on to the appearance of the first PC clones in 1982, leading to the "post-PC" tablets of 2010-2011. Microsoft, AMD, and Intel have outplayed and outlasted their rivals. Plus, many of the hundreds of "PC clone makers" have either been left by the wayside, or have been absorbed into larger conglomerates. Apple has been friend, rival, and self-appointed nemesis during this period, and without that competition, it is unlikely that we would see the technology move in the directions it has. There are an abundance of time-lines and papers addressing the evolution and eras of the PC. We have constructed a time-line fitted to the IDT curve—products, and way stations in the journey through PC technology are depicted. Our intention was not to be exhaustive but to cite major events as well as game-changing/legitimizing turning points. The time-line is shown in Figure 1.

The first key event in the PC era was the introduction of the IBM PC. IBM dominated the computer industry during the pre-computer era (machine-accounting); championed the many generations of mainframes (e.g. the 360 – 370); operating systems and most other software applications; extended and expanded the minicomputer industry (System 32, 34, 36, 38…) and then entered into the PC arena on a major scale—setting a standard for operating systems and controlling the overall market in its initial stages. These were the "early adopters"—primarily computer professionals that transitioned from the mainframe to the minicomputer. Many users experienced the computer as a stand-alone, special-purpose desktop (e.g the graphics machines created by HP).

The PC at this point was a computer without a clear purpose—the accompanying event was the spreadsheet—VisiCalc—followed closely by Lotus 123. Abruptly businesses and the general computer population had a tool that legitimized the PC. Simultaneously word processors and database managers were introduced and followed by graphic/presentation software. The speed, storage capacity and communication channels still were lacking. General business users and professionals began to use the PC for individual and departmental applications and analysis —they formed the nucleus of the "early adopters." At this juncture in time problems arose in many businesses and government agencies – the IT administration did not want to relinquish control to the end users.

As hardware enhancements and network connections were introduced the capabilities of the PC made possible the use of the graphical user interface—obsoleting the use of the PC as a terminal to the mainframe. This marked the rise of Microsoft not only as a provider of operating systems but also the visual aspects of Windows 3 and the business acumen of Microsoft Office—the suite of products for the business at the individual and department level. As hardware technology enabled more elaborate software and system use the PC became a standard within most businesses—the "early majority" embraced the PC as their primary desktop tool for basic tasks along all levels of business activity.

Closely following these enhancements was the introduction of the World Wide Web [WWW] browser as an overlay over the Internet. Prior to this introduction the Internet existed for an extended period but did not have wide-spread use except at the business level for file transfers and email-type commerce. Netscape was the killer app that started the paradigm shift—followed by Microsoft's Internet Explorer. In effect, the PC became the vehicle for every man to communicate – no longer just a business-level system. This marked the peak of the PC era and the diffusion of PC use to the general public—"the majority". Not only was the PC the inherent tool in the office but also the home and school.

New Markets/Mission	New Products: Related Technology	New Products: Unrelated Technology
	Horizontal Diversification	
Same Type	Barnes & Noble -- Amazon -- Sale of books for eReaders Apple iPad & iPod -- and Android -- sale/use of Books, pdfs, music, video as Aps -- Aps for Business (~ office tools) -- Aps for Games, Personal Use -- Aps for alternative Media (Newspapers,etc)	Google - Samsung - HP Slate --- Intel Ultrabook -- Enter tablet market Barnes & Noble --- Amazon Kindle and Tablet --- Software for PCs & other devices Google - Amazon - HP Cloud Computing -- offer services to existing customers Microsoft Cloud Computing -- Office 365 -- Windows 8
	Vertical Integration	
Firm its own customer	Amazon is able to use its web-based ordering and electronic distribution system without incurring additional costs Microsoft, Google use products internally as development tools for new/revised products	Google's - Commitment to digitize most books in the public domain provides a ready audience for ereading devices and other medium - Use of network infrastructure created for search engine as competitive alternative for -- Online office products -- Social networks (Google +) Publishers can partner with Amazon and Barnes & Noble distribution and other supply chain capabilities

Table 1. Horizontal and Vertical Integration

The culminating event was the rise of the networks—the local area networks in businesses and later in homes—but also the wide area networks for businesses and ultimately the utilization of the complex already in place, the Internet. The operating system and router/switch defined by Cisco became the vehicle for communication worldwide. That combined with the browser enabled access for business and the general public ("the late majority"). Of note the browser became the basic user interface for many businesses and government agencies—the U.S. Navy mandated that Internet Explorer be the standard interface for most contractor software development. Following this accessibility the introduction of the search engine (Yahoo and then Google) made the web a pervasive tool. And lastly the social networks (MySpace and Facebook) involved an extensive array of the population as participants.

At the beginning of the 1990s the stability of the personal computer structure and industry changed. IBM dominance of the PC industry and its role as standard-bearer started to erode in the late 1980s. By the early 1990s the market structure was one in which a number of firms possessed the capability to supply interoperable components. Throughout the 1990s and beyond, thousands of manufacturers built PCs around hardware and software components mainly supplied by Microsoft and Intel. There are two distinct types of supplier to the PC industry. The first type supplies components such as disk drives, RAM, peripherals etc. Products in this category are available from a wide variety of sources at highly competitive prices. The other type of supplier provides products—most notably CPUs and operating systems—available from just a few sources .i.e. Microsoft and Intel.

Most firms outsourced the production of components manufacturing contractors carrying out simple manufacturing operations at high-volume plants in low-cost locations. Eventually, these contractors took on more complex tasks, such as design and testing. By the beginning of this century, large contract manufacturers began to build entire PCs for brand-name companies, designing and assembling basic computers in Asia and shipping them to geographic hubs for production to be completed. These full-line distributors dominated the industry with the broadest customer and product base. Many PC manufacturers aimed to streamline their operations by moving from a build-to-stock to a build-to-order model. Reduced inventory led to reduced costs.

The four main types of PC buyer have remained the same since the early 1990s. Namely: business, home, government, and education. However, the distribution channels have changed. Business buyers now buy direct from vendors or distributors as opposed to full-service dealers. Consumer markets are serviced by web-based retailers that can service all types of demand often at steep discounts.

In a survey conducted by eWeek, one in five U.S. adults surveyed said they planned to own a tablet by 2014. The survey included application use on tablets, including the iPad and machines/tablets based on Google's Android platform. Some 78% of respondents said they planned to use their tablets to surf the Web. Three-quarters of people said they would use their machines for e-mail. Other uses include electronic reading of books and newspapers, (53%), social networking (50%), consuming TV and other apps (43%). Tablet use is attractive for enterprises as well, with 37% of respondents planning to use their machines for business concerns. The use of the term "laggard" is probably not appropriately used in this presentation of the PC diffusion—another interpretation would be the deployment of the basic technology and ideas rooted in the PC as it evolved from a desktop to a laptop and currently the variety of devices that make use of and expand on the PC platform—the tablets, the smartphones and even the virtual PC.

Diversification–the Diaspora

Diversification is the name given to the growth strategy where a business markets new products in new markets. For a business to adopt a diversification strategy it must have a clear idea about what it expects to gain from the strategy and a clear assessment of the risks. Diversification in new markets concerns the inclusion of activities other than those directly relating to the product or associated services. There are four underlying reasons why companies diversify [12]:

- When their objectives can no longer be met within the product-market scope defined by expansion—even if attractive expansion opportunities are still available and past objectives are being met, a firm may diversify because the retained cash exceeds the total expansion needs. (The pressure may be on the firm to invest money more profitably.)
- When diversification opportunities promise greater profitability than expansion opportunities. This may occur under several conditions.

New Products		
New Markets/Mission	**Similar Types**	**New Types**
Marketing & Technology Related	Amazon eReader has capabilites for reading not offered by any other device Apple -- iPad - Apple's B2B volume purchasing agreement Gaming Industry-- companies will be pressed harder and harder to come up with new ideas, which could make for an uphill battle (Caron, 2009)	A number of new technologies for tablets are being applied and used for business applications - PC manufacturers are designing Hybrid Tablet PCs able to perform heavy duty work Cloud Services --high cost of power and space is going to force the IT world to look at cloud services, with a shift to computing as a cloud resource (Infoworld,2008) Consumers can now use smaller gadgets to do many of the same things they once did with PCs, such as surfing the Internet, storing photos and sending e-mail. (Robertson, 2011) Mobile Workers and related products - Telecommuting -- the home office - Pressure to provide tools and access to corporate system
Marketing Related	Apple -- iPad (Rawson, 2011) - Apple's retail stores - Use by Children - Deployed in Higher Education	internet and technology companies taking a different approach: -- Introducing a wide range of "smart" devices, from phones to TVs, become the access points to digital information, which resides in the "cloud" (Nutall and Waters, 2011) For aspirant writers the ereader medium now provides a channel/outlet for private label media publications (Castro, 2007) (Egol, 2009)
Technology Related	Apple -- Google -- Introduce Operating Systems for devices -- Google Android available other systems PC makers are countering the threat is with iPad-style tablets running Android	Advances in network medium will reduce delivery time and cost and provide speed of access Virtualization - Desktop Virtualization (Fogarty, 2010)

Table 2. Concentric Diversification

- When the firm's research and development organization produces outstanding diversification by-products.
- When synergy is not an important consideration and therefore the synergy advantages of expansion over diversification are not important.

Firms may continue to explore diversification when the available information is not reliable enough to permit a conclusive comparison between expansion and diversification.

Ansoff has identified different forms of diversification—these are set out in Tables 1 and 2. Table 1 depicts Horizontal Diversification—consisting of moves within the economic environment of the diversifying firms and is complementary to their existing activities, marketing synergy is strong as they continue to sell through established marketing channels. This has been the lexis for the changes in the supply line, customer relations and expectation management in the PC's competitive environment; and Vertical Integration—referring to the development of activities which involve the preceding or succeeding stages in their production processes and is often more sensitive to instabilities and offers less assurance of flexibility—increases the dependence on a particular segment of economic demand—here most of the main competitors are able to channel some of their related products and distribution channels to provide a competitive edge.

These two diversification strategies offer limited potential for objectives; they make a limited contribution to flexibility and stability and will contribute to the other objectives only if the present economic environment of the firm is healthy and growing. This inflection point is just as dramatic as when the PC came on the scene and cut the cord between the mainframes and minis and made the personal computing local. Another way to think of this is that we are moving into a phase in which people want a PC on their desktop and in their pocket [13].

Table 2 depicts Concentric Diversification—having a degree of common thread with those firms that possess marketing and/or technology capabilities. A concentric strategy is generally flexible and usually more profitable and less risky because of synergy. For the PC industry sector the major players can bring to bear resources established from their other product lines to enhance their competitive position.

PCs are being replaced at the center of computing not by another type of device—but by new ideas about the role that computing can play in progress. According to Burt [14], "it is becoming clear that innovation flourishes best, not on devices but in the social spaces between them, where people and ideas meet and interact. It is there that computing can have the most powerful impact on economy, society and people's lives." Software and technology-based companies need to understand where computing is headed and to embrace "that which is technologically inevitable"—a future of varied devices connected to the cloud. The days of the PC-centric environment, which helped fuel Microsoft's success, are declining as the use of mobile devices and cloud computing rises, implied Ozzie, a Microsoft's chief software architect [14].

At the unveiling of the iPad 2 in March, 2011, then Apple CEO Steve Jobs affirmed that the post-PC world would be dominated by such devices as smartphones and tablets. Some

other vendors view tablets as something new in the PC market, but that, "is not the right approach to this," Jobs said. "These are post-PC devices that need to be easier to use than a PC, more intuitive. The hardware and software need to intertwine more than they do on a PC."

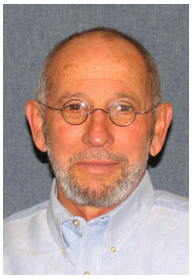
Given the diaspora it is well to note that the smartphones and tablets are hybrids– variations of not just the PC but other technologies. A smartphone is a mobile phone that combines the functions of a personal digital assistant and a mobile phone—also serving as portable media players and cameras with high-resolution touchscreens, web browsers and mobile broadband access. A tablet PC is just that—a tablet-sized computer that has the key features of a full-size personal computer. With the introduction of the iPad and later the Samsung tablet, these devices have taken on many of the features of the smartphone and iPod-like devices.

Two other related directives altering the PC are virtualization and the cloud. These concepts are somewhat intertwined. Cloud computing delivers applications via the Internet and the web browser—the business software and/or user data are stored at remote location. Virtualization is the creation of a virtual (rather than actual) version of something, such as a hardware platform, operating system, a storage device or network resources. It can be viewed as part of an overall trend in enterprise IT that in which the IT environment will be able to manage itself based on perceived activity, and utility computing, in which computer processing power is seen as a utility that clients can pay for only as needed. These innovations extend the PC by enabling any web-enabled device to serve as a conduit to an organization's applications and data. For the consumer we already see this trend with Google's Gmail and apps being stored on Google servers—these are just the tip of the data and application iceberg.

PC sales are decelerating in the U.S. because the same technological advances that fueled the PC industry's rise—faster processors and lower costs—are now benefiting the devices that are usurping it. Consumers can now use smaller gadgets to do many of the same things they once did with PCs, such as surfing the Internet, storing photos, and sending e-mail. Apple even boasts that users can edit home movies on an iPad [15].

In summary, just as with the PC evolution the diaspora is marked by several significant diversifications. The PC itself has mutated into many products ranging from laptops to mini and micro PC-laptops to Tablet-PCs. Accompanying this mutation is the software that supports this array of devices—much of it can be found on other hand-held devices such as the BlackBerry and the smartphones. The promotion and progression of the array of Apple products ranging from the iPhone to the IPod to the iPad has created and fostered not only a market niche but moreover an extension into a worldwide set of devices—engendering other software and hardware companies to follow suit. The iPad has come to be a multi-use product—serving as an access point or portal to the web, a gaming device, a communication medium, and perhaps a substitute for media devices such as the burgeoning eReader market—going directly against the Amazon Kindle. The way that firms now do business is changing. This also has fallout to the consumer that can now use a hand-held device to access a plethora of data anywhere, anytime, and anyplace. The PC is in some sense becoming a virtual machine.

ABOUT THE AUTHOR



James A. Sena, Ph.D., is a Professor of Management and Information Systems in the Orfalea College of Business at California Polytechnic State University. He currently teaches Organization Systems and Technology and Strategy and Policy Teaching specialties include Project Management, Business and IT Strategy, Computer Security, Network Systems, and, Management Information Systems. Dr. Sena has a Ph.D. from the University of Kentucky in Organization Theory and Computer Science. His current research interests include emerging electronic technologies and analyses of executive data concerning information technology.

E-mail: jsena@calpoly.edu
Phone: (805) 756-5318

REFERENCES

1. Ogg, E. (2011) The end of the PC era. GigaOM Pro Aug. 18, 2011, 2:00pm PT
2. Oswald, E (2011) IBM Declares the End of the PC Era, PCWorld Aug 11, 2011
3. Enderle, R. (2009) 3rd Rebirth of Computing: The End of PCs and Game Consoles. TechNewsWorld04/06/09
4. Brodkin, J. (2009) Evolution of the PC, NetworkWorld, May 27, 2009 11:00 pm
5. Volmer, Christopher (2009), Digital Darwinism, Strategy-Business, Spring, 2009.
6. Rogers, E. M. (2003). Diffusion of innovations (5th ed.). New York: Free Press
7. Murray, C. (2009)Diffusion of Innovation Theory: A Bridge for the Research-Practice Gap in Counseling Journal of Counseling and Development : JCD. Alexandria: Winter 2009. Vol. 87, Iss. 1; pg. 108, 9 pgs
8. Rogers, E. M. (2002). Diffusion of preventive innovations. Addictive Behaviors, 27, 989-993
9. Dusenbury, L., & Hansen, W. B. (2004). Pursuing the course from research to practice. Prevention Science, 5, 55-59
10. Moore, G. and Benbasat, I. (1991), "Development of an instrument to measure the perceptions of adopting an information technology innovation", Information Systems Research, Vol. 2 No. 3, pp. 192-222.
11. Valente, T. W. (1996). Social network thresholds in the diffusion of innovations. Social Networks, 18, 69-89
12. Ansoff, I. (1987), Corporate Strategy, rev. ed., Penguin, Harmondsworth
13. Dalrymple, J. (2011) The future of the PC industry http://www.loopinsight.com/2011/07/22/the-future-of-the-pc-industry/
14. Burt. J. (2011) IBM Exec: The End of the PC Era Is Here http://www.eweek.com/c/a/Desktops-and-Notebooks/IBM-Exec-The-End-of-the-PC-Era-is-Here-609114/
15. Robertson , J. (20110Rebooting the PC industry: Tablets force a shift. Associate Press.9:00 PM, Jul. 23, 2011

New ISO/IEC Technical Report Describes Vulnerabilities in Programming Languages

James W. Moore, The MITRE Corporation
John Benito, Blue Pilot
Larry Wagoner, National Security Agency

Abstract. A recent joint technical report from two major international standards bodies, the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), identifies classes of vulnerabilities in programming languages—those features of the languages that encourage or permit the writing of code that contains application vulnerabilities—and suggests ways to avoid or mitigate them. According to the report, programming language vulnerabilities should especially be avoided "in the development of systems where assured behavior is required for security, safety, mission critical and business critical software. [However], this guidance is applicable to the software developed, reviewed, or maintained for any application." This paper provides a brief summary of the ISO/IEC Technical Report.

Introduction

The necessity of mitigating vulnerabilities in software applications is well understood by organizations today. To identify them in existing applications, organizations can use vendor alerts along with public resources such as the Common Vulnerabilities and Exposures [1] and the Open Web Application Security Project's Top 10 Web Application Security Flaws [2] lists. Programmers can help to avoid including them in new applications or maintenance of existing applications by consulting the public Common Weakness Enumeration (CWE™) [3] and CWE/SANS Top 25 Most Dangerous Software Errors [4] lists. Other organizations (e.g., CERT [5] and MISRA [6]) have developed public or private style guides to assist programmers in avoiding application vulnerabilities.

An application vulnerability is a weakness in a software application that permits exploitation by unauthorized persons or contributes to safety hazards. The frequent patches provided by our software vendors have alerted most of us to the problem of vulnerabilities in software designs. Not as well known, however, is that the programming languages in which software applications are written, also have vulnerabilities of their own that can cause applications not to work as intended, behave in unpredictable ways, or lead to application vulnerabilities. Simply stated, deficiencies in the design of programming languages encourage programmers to code in a manner that creates application vulnerabilities. The consequences for organizations can be costly as well as dangerous.

To address this problem, ISO [7] and IEC [8] issued a Technical Report entitled ISO/IEC TR 24772:2010, Information technology—Programming languages—Guidance to avoiding vulnerabilities in programming languages through language selection and use [9], in September 2010 that lists 51 common types of vulnerabilities found in programming languages, along with suggestions for how to avoid them. The report also lists 20 application vulnerabilities that could be addressed by improved language library routines.

No one language contains all of the vulnerabilities described in the report, but most are very common. In addition, 17 of the vulnerabilities detailed in the report also appear on the 2010 CWE/SANS Top 25 Most Dangerous Software Errors list.

Reduce Risk by Mitigating Programming Language Vulnerabilities

By understanding the different ways in which their programming languages might be vulnerable, writers of language standards can eliminate or reduce those vulnerabilities in their languages and thereby make them more secure. In turn, application developers can know how secure a language is before choosing it. Developers will also be able to ensure that the potential for vulnerabilities in their applications are minimized in their software applications, and that they have chosen the most effective and comprehensive source code evaluation tools. Project managers can use the guide to make better-informed selections of programming languages and establish mitigations for the risks inherent in the chosen language.

This is of special importance to those who develop, maintain, and regulate:

- Safety-critical applications that might cause loss of life, human injury, or damage to the environment.
- Security-critical applications that must ensure properties of confidentiality, integrity, and availability.
- Mission-critical applications that must avoid loss or damage to property or finance.
- Business-critical applications where correct operation is essential to the successful operation of the business.
- Scientific, modeling, and simulation applications that require high confidence in the results of possibly complex, expensive, and extended calculation.

Reducing risk in all of these areas will, over time, yield organizations cost savings due to less work, and ultimately lead to more secure systems.

Types of Programming Language Vulnerabilities

When a programmer writes a software application, regardless of the programming language used—be it Ada, C, COBOL, Fortran, etc.—the code should execute in a manner that can be predicted by the developer. If it does not, and an attacker can then make use of the mistake to access a system or network, it is considered a vulnerability in the software code.

With programming languages, vulnerabilities arise in six main ways:

Incomplete or Evolving Programming Language Behavior

Programming language standards are continuously evolving with new releases and features, resulting in issues that might affect predictability. Such issues include the need for compatibility with previous releases, and the interaction of that language's features, separately and in any combination, under all foreseeable circumstances.

Choice of compiler can also have an effect. Compilers are used by programmers to transform their source code to a binary code (commonly called object code). However, unless the compiler comes from a trusted source and was developed according to agreed standards, it could inadvertently or maliciously insert bad code into the binary, resulting in a potential vulnerability. This is especially important to avoid because this type of vulnerability would then be inserted into every piece of software that the compiler processes.

Unspecified behavior must also be avoided. While most behavior is specified by programming languages, unspecified behaviors can result when a programming language construct is specified to have two or more possibilities of behavior. In such a case, different compilers may generate different behaviors from the same source code, resulting in a vulnerability. The problem is exacerbated if the compiler(s) are run on different computers; if the compilers use different software libraries; or if they run on different operating systems, different releases of an operating system, or different configurations of an operating system.

Another issue is implementation-defined behavior. Programming languages sometimes allow compilers to support a variety of behaviors for a single language feature, or combination of features, that may enable usage on a wider range of hardware or enable use of the language in a wider variety of circumstances. However, there is a requirement that each implementation document the behavior. Vulnerabilities can occur when the programmer does not take into account this documented behavior or ports code from one machine to another without considering changes in implementation-defined behavior.

Undefined behavior is also a threat. Programming languages sometimes specify that program behavior is undefined or simply leave some behavior undefined. Common examples include recovery from an error in the software, and use of the value of a variable that has not yet been assigned. In some cases, attackers can use expert knowledge to stimulate behavior that can lead to a vulnerability.

Human Cognitive Limitations

Programming languages are created with different purposes, some are for general use and others for specific tasks or needs, but all are created as tools to be used by software programmers to manipulate data and produce a desired result. This means the intended audiences for the languages are different. For instance, C was created for programmers implementing system software, while COBOL was created for programmers writing business applications.

Because everyone is different and each person has their own levels of understanding and areas of expertise, vulnerabilities can occur because of the abilities of the person writing the code as well as by those who maintain it. Programmers may choose syntaxes that make the most sense to them, even though the language provides another syntax that would accomplish the same task, or may have performed the function more efficiently. Also, as people, programmers have to deal with the stresses of their personal and professional lives, any of which may have an impact on the quality of code that person writes, which could in turn result in a potential vulnerability.

This can be addressed by standardizing and simplifying as much as possible, and by improving documentation and resources, including project coding standards and review processes, that directly deal with these issues.

Lack of Portability and Interoperability

In addition to potential issues resulting from how code is written and from variations in the compilers or configurations of the same compiler, other factors can result in potential vulnerabilities when the application is run, such as if the application is used with different software libraries, on different operating systems, or on different hardware.

Developers must be aware of these possibilities and plan for them, for instance, by using only semantics specifically defined by the language, and by using software libraries specifically created for the language whenever possible.

Inadequate Intrinsic Support in the Language

Although using specified software libraries for an application can reduce risk, sometimes no libraries are specified by the programming language or the libraries used are not validated to the same standard as the compiler and the applications being developed, are proprietary and inclined to change in later releases, or are discontinued and no longer supported by the vendor. Such instances can lead to potential vulnerabilities.

A programmer can reduce this risk by using stronger types or controls to perform certain operations, though this may reduce the performance and flexibility of the application. Therefore, the developer must strike a balance between the intrinsic support provided by the language to help avoid vulnerabilities and the ultimate utility of the application.

Language Features Prone to Erroneous Use

In some programming languages the syntactic constructs used by the language are simple and straightforward to use, while others in that same language are extremely complex. Vulnerabilities can result when language constructs are used improperly, when complex constructs are misused in acceptable but unintended ways, or when complex constructs that can be substituted for by a series of simpler constructs are used without an understanding of the full effects of the constructs.

Such vulnerabilities can be reduced by those creating the language by identifying such constructs, and providing standardized ways for dealing with them.

The common strand throughout all of the causes listed above is lack of knowledge. With perfect knowledge, the execution of code can be predicted, but this is seldom the case. Expert attackers can exploit superior knowledge to "trick" the code into executing function that the code's developer did not intend or foresee.

Example Vulnerabilities

An example of a vulnerability described in the Technical Report would be the following:

When subexpressions with side effects are used within an expression, the unspecified order of evaluation can result in a program producing different results on different platforms, or even at different times on the same platform. For example, consider

```
a = f(b) + g(b);
```

where f and g both modify b. If f(b) is evaluated first, then the b used as a parameter to g(b) may be a different value than if g(b) is performed first. Likewise, if g(b) is performed first, f(b) may be called with a different value of b.

Other examples of unspecified order, or even undefined behavior, can be manifested, such as

```
a = f(i) + i++;
```

or

```
a[i++] = b[i++];
```

Parentheses around expressions can assist in removing ambiguity about grouping, but the issues regarding side effects and order of evaluation are not changed by the presence of parentheses; consider

```
j = i++ * i++;
```

where even if parentheses are placed around the i++ subexpressions, undefined behavior still remains. (This example uses the syntax of C; the effects can be created in any language that allows functions with side effects in the places where the example shows the increment operations.)

In this case, the report suggests that programmers should decompose the expression into sequential statements so that the order of evaluation can be controlled.

The unpredictable nature of the calculation means that the program cannot be tested adequately to any degree of confidence. A knowledgeable attacker can take advantage of this characteristic to manipulate data values triggering execution that was not anticipated by the developer.

An example of an application vulnerability included in the report would be storing a password in vulnerable cleartext because the programming language did not provide a library function for encrypting the password. For this problem, the project should acquire a subroutine library that provides the functionality missing from the language library.

A Catalog of Language Vulnerability Types

Vulnerabilities included in the report were identified and selected using two different analyses. A bottom-up analysis surveyed application security vulnerabilities observed "in the wild" and identified language characteristics that can serve as root causes of the application vulnerabilities. A top-down analysis surveyed existing language style and usage guides for the production of safety-related software.

All language vulnerabilities in the ISO/IEC report are described in a language-independent manner allowing readers to quickly comprehend and utilize the information.

Programming Language Vulnerability Description

Each type of programming language vulnerability is described in a uniform format to permit easy reference. Information in the description includes:

- An arbitrary three-letter identifier that can be used to identify the vulnerability.
- A brief summary of the programming language vulnerability.
- Cross-references, such as CWE identifier.
- A description of the mechanism of failure, giving the link between the programming language vulnerability and resulting application vulnerabilities.
- A list summarizing the characteristics of languages for which this vulnerability is applicable.
- A brief description of how application developers can avoid the vulnerability or mitigate its negative effects.
- Comments regarding how the maintainers of the language's specification might make improvements.

Application Vulnerability Description

The report also lists a handful of application vulnerabilities that might be mitigated if better support were provided in programming language libraries. These are described similarly to the language vulnerabilities, except that the comments to language maintainers are omitted.

An Ongoing Process

The list of vulnerabilities detailed in the ISO/IEC report is not complete. With new vulnerabilities being discovered regularly, the process will always be ongoing. The report therefore only describes those programming language vulnerability types that were determined to have sufficient probability and significance to date.

In addition, the following five subject areas were not addressed in this initial release but will be addressed in future editions of the report:

- Object-oriented language features, though certain simple issues related to inheritance are discussed.
- Concurrency.
- Numerical analysis, though certain simple items regarding the use of floating point are discussed.
- Scripting languages.
- Inter-language operability.

The second edition of the Technical Report will also add annexes describing how the vulnerabilities appear in particular programming languages. Currently, annexes are planned for Ada, C, Python, Ruby and SPARK. Future editions will add more language-specific annexes as well as describing additional vulnerabilities.

The report is available for purchase from <http://www.iso.org> and <http://www.ansi.org>. Individual users can obtain the report for free at <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>.

About ISO/IEC Standards and Technical Reports

There are three major international standards associations that bring together national bodies from participating nations, as well other international governmental and nongovernmental organizations, to focus on the development of international standards for business, government, and society—the ISO, IEC, and International Telecommunication Union [10].

While the primary work of ISO and IEC is to prepare international standards, some subjects are not appropriate for standardization but are suitable for technical reports that provide guidance and information that have been formed via consensus.

The ISO/IEC report about programming language vulnerability types discussed in this article, Technical Report 24772:2010, was published by a subcommittee working group of the ISO/IEC Joint Technical Committee for the field of information technology that is responsible for, "programming languages, their environments, and system software interfaces."

REFERENCES

1. <http://www.cve.mitre.org/>
2. <http://www.owasp.org/index.php/OWASP_Top_Ten_Project>
3. <http://www.cwe.mitre.org/>
4. <http://cwe.mitre.org/top25/index.html>
5. <http://www.cert.org/>
6. <http://www.misra-c.com/>
7. <http://www.iso.org/>
8. <http://www.iec.ch/>
9. <http://www.iso.org/iso/catalogue_detail.htm?csnumber=41542>
10. <http://www.itu.int/>

ABOUT THE AUTHORS

James W. Moore is a 40-year veteran of software engineering in IBM and, now, the MITRE Corporation. He is a leader in software and systems engineering standardization for the IEEE, serving as its liaison to ISO/IEC JTC1/SC7 and as a member of the Executive Committee of the IEEE Software and Systems Engineering Standards Committee. He serves as a member of the IEEE Computer Society's Board of Governors. He was an Executive Editor of the Society's 2004 "Guide to the Software Engineering Body of Knowledge" and a member of the Editorial Board of the revision of the "Encyclopedia of Software Engineering." The IEEE Computer Society has recognized him as a Charter Member of their Golden Core, and the IEEE has named him a Fellow of the IEEE. His work on software engineering standards has been recognized by the International Committee on Information Technology Standards (INCITS) with their International Award, by the Computer Society with the Hans Karlsson Award, and by the IEEE with the Charles Proteus Steinmetz Award. His latest book on software engineering standards was published in 2006 by John Wiley & Son. He holds two US patents and, dating to times when software was not regarded as patentable, two "defensive publications". He graduated from the University of North Carolina with a B.S. in Mathematics, and Syracuse University with an M.S. in Systems and Information Science.

E-mail: James.W.Moore@ieee.org
Phone: 301-938-0260

John Benito is an independent consultant providing software development, project management, and software testing. He is the current Convener of ISO/IEC JTC 1/SC 22/WG14 the ISO group responsible for Standard C, the Convener of ISO/IEC JTC 1/SC 22 WG 23 (was OWG Vulnerabilities), the project editor for the Technical Report 24772, and a member of the INCITS PL22.11 (ANSI C) technical committee. He previously was a member of INCITS PL22.16 (ANSI C++) and the ISO Java Study group. He has been in software development, project management, and testing for over 35 years. Mr. Benito has been participating in International Standard development for the past 22 years, and is the recipient of the INCITS Exceptional International Leadership Award.

E-mail: benito@bluepilot.com
Phone: 831-427-0528

Dr. Larry Wagoner has served in a variety of technical and/or analytic organizations within the National Security Agency for over 25 years. Before coming to the Information Assurance Directorate, he worked primarily in the Signals Intelligence Directorate and the Research Directorate. He has a Ph.D. in computer science from the University of Maryland, Baltimore County.

E-mail: l.wagone@radium.ncsc.mil

Supply Chain Risk Management

Understanding Vulnerabilities in Code You Buy, Build, or Integrate

Paul R. Croll, CSC

Abstract. This paper describes the scope of the problem regarding software vulnerabilities and the current state of the practice in static code analysis for software assurance. Recommendations are made regarding the use of static analysis methods and tools during the software life. Static code analysis touch points during lifecycle reviews and challenges to automated static code analysis are also discussed.

Section 1. Introduction

Managing software risk in the supply chain is in large part about discovering and understanding the vulnerabilities that might exist in code that you might buy as standalone applications or integrate into other systems or products. It is also about vulnerabilities you might build into code that you develop in-house. Static code analysis can be an effective means for determining the vulnerabilities in your code.

a. Scope of the Problem

Capers Jones [1] described the results of a survey of the U.S. software industry as of 2008. Based on those data, Tables 1 and 2 address the number and severity of software vulnerabilities in several classes of application projects. For military projects, as one approaches systems the size of typical large combat systems (expressed as function points), the estimated number of security vulnerabilities rises to above 3000 and the probability of serious vulnerabilities rises to 45%. The statistics are much worse for civilian and commercial systems. These systems have tended to make much more extensive use of COTS. As we move more and more into COTS and open source software for our national defense and critical infrastructure systems, one might expect that the extent of vulnerabilities in these critical systems might nearly double.

In a study by Reifer and Bryant [2], 100 packages were selected at random from 50 public open source and COTS libraries. These spanned a full range of applications and sites like SourceForge. The packages were analyzed by college students using a variety of tools.

	Commercial Projects	Civilian Government Projects	Military Projects	***Average***
Size in FP				
1	1	1	1	*1*
10	5	5	4	*5*
100	28	29	14	*24*
1,000	150	155	55	*120*
10,000	794	832	209	*600*
100,000	4,217	4,467	794	*3,031*
1,000,000	22,387	23,988	3,020	*15,412*
Average	*3,940*	*4,211*	*585*	*2,742*

Table 1. Estimated Number of Vulnerabilities in Software Applications

	Commercial Projects	Civilian Government Projects	Military Projects	***Average***
Size in FP				
1	15.00%	25.00%	5.00%	*11.29%*
10	30.00%	35.00%	15.00%	*26.00%*
100	40.00%	45.00%	20.00%	*33.57%*
1,000	60.00%	62.00%	30.00%	*54.57%*
10,000	77.00%	80.00%	35.00%	*74.00%*
100,000	85.00%	87.00%	40.00%	*80.14%*
1,000,000	96.00%	92.00%	45.00%	*86.29%*
Average	*57.57%*	*60.86%*	*27.14%*	*52.27%*

Table 2. Probability of Serious Security Vulnerabilities in Software Applications

The objectives were to:

- Determine if the packages were up-to-date with respect to vendor identified vulnerabilities and patches
- Assess if packages were free of known viruses, worms, Trojans and spyware
- Assess if the packages had weaknesses in the code and backdoors, using reverse engineering techniques
- Assess if the packages had potential dead code, malware, unwanted behaviors, or undesired functionality

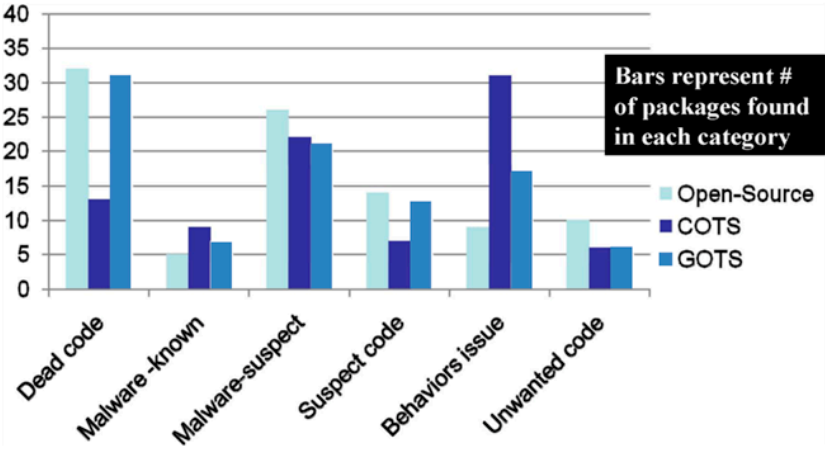


Figure 1. COTS Study Findings. Source: D. Reifer and E. Bryant, Software Assurance in COTS and open source Packages, DHS Software Assurance Forum, October 2008

Figure 1 describes the results of this small study. Over 30% of open source and Government Off the Shelf (GOTS) packages analyzed had dead code, an anathema to the software safety community, and a concern of the software security community as well. Over 20% of the open source, COTS, and GOTS packages had suspected malware, and over 30% of the COTS packages analyzed had behavioral problems.

Reifer and Bryant conclude that the potential for malicious code in applications software is large as more and more packages are used in developing a system. They have been developing a tool for analyzing software executables, often the only thing available from COTS suppliers. They have a method and tool that is available now. These focus on analyzing software executables, often the only thing available from COTS suppliers.

b. What is Static Source Code Analysis?

Static analysis is the process of evaluating a system or component based on its form, structure, content, or documentation [3]. From a software assurance perspective, static analysis addresses weaknesses in program code that might lead to vulnerabilities. Such analysis may be manual, as in code inspections, or automated through the use of one or more tools. A static analysis tool is a program written to analyze other programs for flaws [4]. Such analyzers typically check source code. There is also a smaller set of analyzers that check byte code and binary code as well. While testing requires code that is relatively complete, static analysis can be performed on modules or unfinished code. Manual analysis, or code inspection, can be very time-consuming, and inspection teams must know what security vulnerabilities look like in order to effectively examine the code. Static analysis tools are faster and do not require the tool operator to have the same level of security expertise as a code inspector [5].

Section 2. Strategies for Effective Source Code Analysis

a. What Code Do You Analyze?

How do you prioritize a code review effort when you have thousands of lines of source code, and perhaps object code to review? From a software assurance perspective, looking at attack surfaces is not a bad place to start [6]. A system's attack surface can be thought of as the set of ways in which an adversary can enter the system and potentially cause damage. The larger the attack surface, the more insecure the system [7]. Higher attack surface software requires deeper review than code in lower attack surface components. Howard [8] proposes several heuristics as an aid to determining code review priority, that is, given a large amount of code to review, what kinds of code do you emphasize for review. They are summarized below:

Legacy code: Howard points out that legacy code may have more vulnerabilities than newly developed code because security issues likely were not as well understood when the legacy code was created.

Code that runs by default: Howard suggests that attackers will often attempt to exploit code that runs by default. He also suggests that code running by default increases an application's attack surface, which is a product of all code accessible to attackers.

Code that runs in elevated context: Code that runs with elevated privileges, e.g. root privileges, for example, should also be reviewed earlier and deeper because compromise of such code can allow attackers to execute commands that are intended only for privileged users such as a site administrator.

Anonymously accessible code: Howard suggests that code that permits anonymous access should be reviewed in greater depth than code that only allows access to valid users and administrators.

Code connected to a globally accessible network interface: Howard strongly states that code that interfaces with a network, especially uncontrolled networks like the Internet, presents substantial risk. Such code increases the potential attack surface for the system.

Code written in a language whose features facilitate building in vulnerabilities: Howard suggest that code written in languages like C and C++, have features, like direct memory access, that allow programmers to inadvertently insert vulnerabilities, like buffer-overflow vulnerabilities. Howard also points out other language vulnerabilities, such as SQL-injection vulnerabilities in Java, or C# code. ISO/IEC TR 24772:2010 [9] specifies software programming language vulnerabilities to be avoided where assured behavior is required. These vulnerabilities are described in a generic manner that is applicable to a broad range of programming languages.

Code with a history of vulnerabilities: Code that has had a number of past security vulnerabilities should be suspect, unless it can be demonstrated that those vulnerabilities have been effectively removed.

Code that handles sensitive data: Code that handles sensitive data should be analyzed to ensure that weaknesses in the code not compromise such data by disclosing it to untrusted users.

Complex code: Complex code has a higher bug probability, is more difficult to understand, and may likely have more security vulnerabilities.

Code that changes frequently: Howard points out that frequently changing code often results in new bugs being introduced. Not all of these bugs will be security vulnerabilities, but compared with a stable set of code that is updated only infrequently, code that is less stable will probably have more vulnerabilities in it.

b. A Three-phase Code Analysis Process

Howard [8] also suggests a notional three-phase code analysis process that optimizes the use of static analysis tools.

1. Phase 1 – Run all available code-analysis tools

Howard suggests that multiple tools should be used to offset tool biases and minimize false positives and false negatives. This makes great sense if your organization can afford it. Strengths and weaknesses vary from tool to tool [10, 11]. Warnings from multiple tools may indicate code that needs closer scrutiny through manual inspection.

Additionally, these tools are most effective when run early in the lifecycle and run often [12]

Howard also suggests that code should be evaluated early, and re-evaluated throughout its development cycle.

2. Phase 2 – Look for common vulnerability patterns

Howard recommends that analysts make sure that code reviews cover the most common vulnerabilities and weaknesses. Sources for such common vulnerabilities and weaknesses include the Common Vulnerabilities and Exposures (CVE) and Common Weaknesses Enumeration (CWE) databases, maintained by the MITRE Corporation and accessible on the web at: <http://cve.mitre.org/cve/> and <http://cwe.mitre.org/>. MITRE, in cooperation with the SANS Institute, also maintains a list of the "Top 25 Most Dangerous Programming Errors [13]" that can lead to serious vulnerabilities. The top three classes of errors as of December 2010 were cross-site scripting, SQL injection, and buffer overflows. Static code analysis tool and manual techniques should at a minimum, address these Top 25.

3. Phase 3 – Use manual analysis for risky code

Howard also suggests that analysts should also use manual analysis (e.g. code inspection) to more thoroughly evaluate any risky code that has been identified based on the attack surface, or based on the heuristics described earlier. Manual analysis allows detailed tracing of code paths and data usage.

Section 3. The Assurance Case

An Assurance Case is a set of structured assurance claims, supported by evidence and reasoning that demonstrates how assurance needs have been satisfied [14].

- It shows compliance with assurance objectives.
- It provides an argument for the safety and security of the product or service.
- It is built, collected, and maintained throughout the lifecycle.
- It is derived from multiple sources.

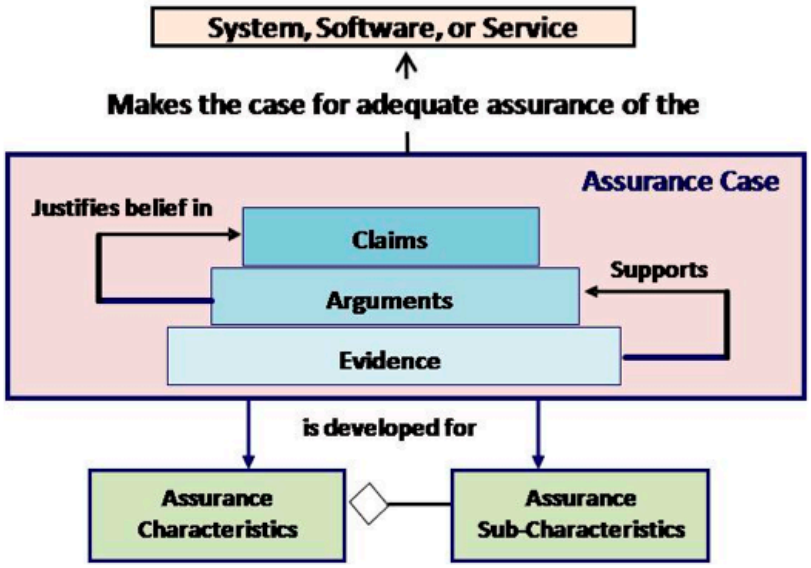


Figure 2. The Assurance Case

As shown in Figure 2, the Assurance Case should be used to document claims about the security of a software product or system. Those claims must be supported by arguments regarding the security characteristics of the software, and those arguments must be firmly supported by evidence.

The results obtained from static code analysis provide evidence regarding vulnerabilities in code, and should be documented as part of the Assurance Case.

The Sub-parts of an assurance case include:

- A high level summary
- Justification that product or service is acceptably safe, secure, or dependable
- Rationale for claiming a specified level of safety and security
- Conformance with relevant standards and regulatory requirements
- The configuration baseline
- Identified hazards and threats and residual risk of each hazard and threat
- Operational and support assumptions

An Assurance Case should be part of every acquisition in which there is concern for IT security. It should be prepared by the supplier and describe the assurance-related claims for the software being delivered, the arguments backing up those claims, and the hard evidence supporting those arguments.

The details of how static code analysis was used in the development process and the results of such static analysis should be included to support assurance arguments.

Section 4. Static Code Analysis in the Software Lifecycle

Project Managers (PMs) have a responsibility to ensure that security requirements are addressed throughout the software lifecycle. This responsibility includes conducting risk assessments; documenting system threats and vulnerabilities, including test and remediation plans on a continuing basis. Static code analysis contributes to documenting system weaknesses and vulnerabilities.

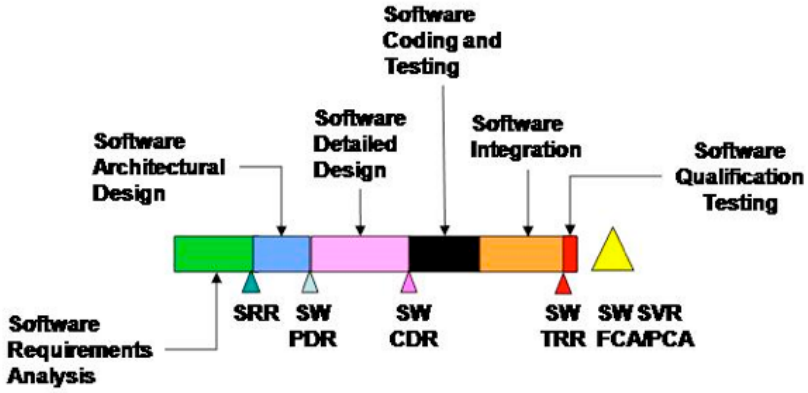


Figure 3. Reviews in the Software Lifecycle

Static code analysis should be applied at several points in the software acquisition and development lifecycle.

The reviews that are associated with software are shown in Figure 3 [15]. The following discussion addresses the objectives and expected outcomes of these reviews, describing the touch points for static code analysis in the software lifecycle review process [16].

a. System Requirements Review (SRR)

1. Objectives

The SRR helps the PM understand the scope of the software assurance landscape (assurance requirements, elements to be protected, the threat environment) in which context static code analysis should be applied.

2. Outcomes

• Establishment of the System Assurance Case

The Assurance Case both sets the context for static code analysis and provides a repository for analysis results. As discussed earlier and emphasized here, the Assurance Case should include:

-Specification of the top-level system assurance claims that address identified threats.

-Identification of the approach for developing the system assurance case.

-Identification of all critical elements to be protected.

-Identification of all relevant system assurance threats and their potential impact on critical system assets.

-Identification of high-level potential weaknesses in the system.

-Determination and derivation of system assurance requirements (as a subset of the system requirements).

• Test and Evaluation Master Plan (TEMP) addressing system assurance.

The TEMP or establishes the test strategy for testing throughout the development lifecycle.

-Examine the TEMP to ensure testing processes are sufficient for system assurance. This may include planning for static code analysis.

• Support and Maintenance Concepts

Support and Maintenance concepts addresses the need to address assurance concerns beyond development, throughout the life of the system. Outcomes include:

-Documentation of the support and maintenance concepts including a description of how assurance will be maintained.

-Description of what static code analysis tools will be used post deployment and how and when they will be applied.

b. Preliminary Design Review (PDR)

1. Objectives

The PDR is a multi-disciplined technical review to ensure that the system under review can proceed into detailed design, and can meet the stated performance requirements within cost (program budget), schedule (program schedule), risk, and specific assurance requirements and constraints.

2. Outcomes

• Information security technology evaluation of all critical COTS/GOTS elements.

As discussed earlier, COTS/GOTS components might present security risks. As part of the analysis of alternatives process, candidate components should be vetted with respect to their security characteristics. The Assurance Case should also be updated based on the components selected, and any new weaknesses and vulnerabilities identified.

The outcomes from the evaluation of COTS/GOTS elements should include:

-Specification of assurance-specific static analysis and assurance-specific criteria to be examined during code reviews.

-Documentation of the results of static code analyses performed on GOTS/COTS components.

-Documentation regarding which tools were used to perform static code analysis.

-Documentation of weaknesses and vulnerabilities that were discovered.

-Documentation of code reviews performed during implementation.

• Configuration management.

The preliminary configuration management plan must support protection of each configuration item, addressing vulnerabilities that might creep in during the change process. This includes requirements, architectures, designs, and code. The outcomes associated with configuration management include:

-Discussion regarding at which stages of the configuration management process static code analysis will be applied.

-Discussion of what configuration change events will trigger code analysis.

-Description of which components will be analyzed.

-Description of how the results of the analyses will be documented.

The Assurance Case should also be updated with relevant evidence as a result of the PDR.

3. Other Considerations

Use of COTS and open source presents a supply chain assurance challenge. As part of an analysis of the supplier and its processes, the following should be determined.

- Will the supplier perform static code analysis as part of its code development and/or code integration processes?
- Which components will be analyzed? Which will not?
- What tools do they plan to use?
- What are the details of their code inspection process for manual security analysis?
- How will they mitigated any discovered vulnerabilities

or weaknesses?

COTS source code is rarely available to the acquirer for independent code review.

PMs should request COTS vendors provide Assurance Cases for their COTS products detailing both the vendor's secure coding practices and the results of internal static code analysis or third party assessment (e.g. Common Criteria certification).

In cases where such information is unavailable, and there is still a desire to use the COTS component, the PM should consider analyzing the executables using binary code analysis.

c. Critical Design Review (CDR)

1. Objectives

The CDR is a multi-disciplined technical review to ensure that the system under review can proceed into system fabrication, demonstration, and test, and can meet the stated performance requirements within cost (program budget), schedule (program schedule), risk, and specific assurance requirements and constraints.

From a software perspective, the CDR focuses on the completeness of the detailed design and how it supports functional, performance, and assurance requirements.

2. Outcomes

With respect to software security and code analysis, the CDR should document:

• Identification and use of the selected static analysis tools for source code evaluation.

• Selection of additional development tools and guidelines to counter weaknesses and vulnerabilities in the system elements and development environment(s), including:

-Definition and selection of assurance-specific static analyses and assurance-specific criteria to be examined during peer reviews performed during implementation.

-Planning for training for assurance-unique static analysis tools and peer reviews.

The Assurance Case should also be updated with relevant evidence as a result of the CDR.

d. Test Readiness Review (TRR)

1. Objectives

The TRR is a multi-disciplined technical review to ensure that the subsystem or system under review is ready to proceed into formal test. The TRR also examines lower-level test results, test plans, test objectives, test methods, and procedures to verify the traceability of planned tests to program requirements.

2. Outcomes

•Verification of static code analysis.

Verification regarding static code analysis determines if assurance-specific static analyses and peer reviews of assurance criteria have been completed. Such verification includes:

-Documentation of evidence that static analysis has been performed (both source and binary) to identify weaknesses and vulnerabilities such as cross-site scripting, SQL injection, and buffer overruns.

-Verification that another party other than the developer (such as a peer) performed static analysis and peer review.

-Documentation regarding the selection of any additional static analysis tools to identify or verify weaknesses and vulnerabilities in the system elements and development environment(s).

-For COTS/GOTS software products with no source code, identification of industry tools and test cases to be used for the testing of any binary or machine-executable files.

The Assurance Case should also be updated with relevant evidence as a result of the TRR.

Even for those with less formal lifecycle review processes, there will generally be a requirements development phase, one or more design phases, and implementation and testing phases. For some organizations there will be operations and maintenance phases as well. The objectives and outcomes of the lifecycle touch points described above for static code analysis should provide guidance and help set expectations, no matter how formal or informal the lifecycle review process.

Section 5. Challenges to Automated Static Code Analysis

There are two challenges to the effective uses of automated static code analysis.

a. Procurement and Maintenance of Tools

The better static code analysis tools are expensive. However, the best results are obtained when multiple tools are used to offset tool biases and minimize false positives and false negatives. Use of multiple tools can quickly become cost prohibitive for a single project.

In addition, maintenance agreements to ensure a tool is up to date with respect to the spectrum of threats, weaknesses, and vulnerabilities add long term costs.

The concept of "buy it once, use it often" provides the most bang for the buck. Pooled resources analysis labs that support multiple projects within organizations may make the most economic sense.

b. Training

Static code analysis is not for sissies, although it may be for CISSPs® (Certified Information System Security Professionals). This tongue-in-cheek statement belies the difficulty in using static code analysis tools to their best advantage.

Chandra, Chess, and Steven [17] point out that when static code analysis tools are employed by a trained team of code analysts, false positives are less of a concern; the analysts become skilled with the tools very quickly; and greater overall audit capacity results.

In addition, in order to determine the validity of static code analysis results, it is important for PMs to understand the level of training that code analysts have had with the tools employed for static code analysis as well as their understanding of code weaknesses and vulnerabilities. Even a good tool in the hands of a poorly trained or inexperienced code analyst can produce misleading results. A tool is just a tool. How it is used and how its results are interpreted are key to useful and valid results.

Section 6. Useful Links

a. NIST Software Assurance Metrics and Tool Evaluation (SAMATE) Static Analysis Tool Survey

The NIST SAMATE project provides tables describing current static code analysis tools for source, byte, and binary code

analysis <http://samate.nist.gov/>.

b. DHS Build Security In Web Site

This site contains a wealth of software and information assurance information, including white papers on static code analysis tools. More information on Build Security In can be found at:

<https://buildsecurityin.us-cert.gov/daisy/bsi/home.html>

c. CWE

This site provides a formal list of software weakness types created to:

- Serve as a common language for describing software security weaknesses in architecture, design, or code.
- Serve as a standard measuring stick for software security tools targeting these weaknesses.
- Provide a common baseline standard for weakness identification, mitigation, and prevention efforts. <http://cwe.mitre.org/>

d. CWE/SANS Top 25 Most Dangerous Software Errors

The 2010 CWE/SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

<http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.pdf>

Section 7. Summary

This paper has described the scope of the problem regarding vulnerabilities in the code we buy, build, or integrate. As more and more COTS and open source components are integrated into our systems, the problem becomes ever more exacerbated.

The paper has also discussed strategies for effective static code analysis as a means to understand an manage supply chain risk, and has described the expected outcomes regarding such analysis at appropriate touch points in the software lifecycle. Although the lifecycle reviews described were fairly formal, the activities associated with those reviews apply to any software development, integration, or maintenance effort. In addition, the paper has described the Assurance Case, the repository for, among other things, the empirical results of static analysis.

Lastly, the paper touched on challenges to automated static code analysis, regarding the procurement and maintenance of tools and the training required for tool users in order to facilitate accurate results. Such analysis is most effective when multiple tools are used to offset tool biases, and are employed by analysts with proper training in both tool use and in security-related code inspection.

To be sure, there are other means for assessing and managing supply chain risk with respect to software, but at the bottom line, it is all about the code and the vulnerabilities it might contain.

The Software Assurance Community Resources and Information Clearinghouse contains links to free Pocket Guides on other aspects of supply chain risk management, including:

- Software Assurance in Acquisition and Contract Language
- Software Supply Chain Risk Management and Due Diligence
- Key Practices for Mitigating the Most Egregious Exploitable Software Weaknesses
- Software Security Testing
- Secure Coding

Guides on other aspects of software assurance include:

- Requirements and Analysis for Secure Software
- Architecture and Design Considerations for Secure Software
- Software Assurance in Education, Training & Certification

All of these guides can be found at:

<https://buildsecurityin.us-cert.gov/swa/pocket_guide_series.html>

CALL FOR ARTICLES

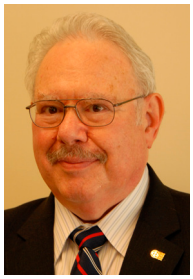
If your experience or research has produced information that could be useful to others, CrossTalk can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

Resilient Cyber Ecosystem
Sept/Oct 2012 Issue
Submission Deadline: Apr 10, 2012

Virtualization
Nov/Dec 2012 Issue
Submission Deadline: June 10, 2012

Please follow the Author Guidelines for CrossTalk, available on the Internet at <www.crosstalkonline.org/submission-guidelines>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and BackTalk. To see a list of themes for upcoming issues or to learn more about the types of articles we're looking for visit <www.crosstalkonline.org/theme-calendar>.

ABOUT THE AUTHOR



Paul Croll is a Fellow in CSC's Defense Group and Chief Scientist of the Defense & Maritime Enterprise Technology Center, where he is responsible for researching, developing and deploying systems and software engineering practices, including practices for cybersecurity.

Paul has over 35 years experience in mission-critical systems and software engineering. His experience spans the full life cycle and includes requirements specification, architecture, design, development, verification, validation, test and evaluation, and sustainment for complex systems and systems-of-systems. He has brought his skills to high profile, cutting edge technology programs in areas as diverse as surface warfare, air traffic control, computerized adaptive testing, and nuclear power generation.

Paul is also the IEEE Computer Society Vice President for Technical and Conference Activities, and has been an active Computer Society volunteer for over 25 years, working primarily to engage researchers, educators, and practitioners in advancing the state of the practice in software and systems engineering. He was most recently Chair of the Technical Council on Software Engineering and is also the current Chair of the IEEE Software and Systems Engineering Standards Committee. Paul is also the past Chair and current Vice Chair of the ISO/IEC JTC1/SC7 U.S. Technical Advisory Group (SC7 TAG).

Paul is also active in industry organizations and is the Chair of the NDIA Software Industry Experts Panel and the Industry Co-Chair for the National Defense Industrial Association (NDIA) Software and Systems Assurance Committees. In addition, Paul is Co-Chair of the DHS/DoD/NIST Software Assurance Forum Processes and Practices Working Group advancing cybersecurity awareness and practice.

E-mail: pcroll@csc.com

REFERENCES

1. Jones, Capers. Overview of the United States Software Industry Results Circa 2008, DHS Software Assurance Forum working paper, June 20, 2008.
2. Reifer, D, and Bryant, E. "Software Assurance in COTS and Open Source Packages," Proceedings of the DHS Software Assurance Forum, October 14-16, 2008.
3. ISO/IEC JTC1/SC7. ISO/IEC 24765:2009, Systems and software engineering vocabulary.
4. Black, P. Static "Analyzers in Software Engineering," CrossTalk, The Journal of Defense Software Engineering, pp. 16-17, March-April 2009.
5. McGraw, G. "Automated Code Review Tools for Security," Computer, vol. 41, no. 12, pp. 108-111, Dec. 2008.
6. Howard, M. Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users, <http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface, November, 2004>.
7. Manadhata, P., Tan, K, Maxion, R, and Wing, J. An Approach to Measuring a System's Attack Surface, CMU-CS-07-146, Carnegie Mellon University, August 2007.
8. Howard, M. "A Process for Performing Security Code Reviews," IEEE Security & Privacy, pp. 74-79, July-August 2006.
9. ISO/IEC TR 24772:2010, Guidance to avoiding vulnerabilities in programming languages through language selection and use.
10. U.S. Department of the Navy, Software Security Assessment Tools Review, March 2009, <https://buildsecurityin.us-cert.gov/swa/downloads/NAVSEA-Tools-Paper-2009-03-02.pdf>
11. Okun, V., Delaitre, A, Black, P. The Second Static Analysis Tool Exposition (SATE) 2009, NIST Special Publication 500-287, National Institute of Standards and Technology, June 2010.
12. Goertzel, K., Winograd, T., Enhancing the Development Life Cycle to Produce Secure Software, Rome, NY: DACS Data & Analysis Center for Software, 2008.
13. Cristey, S. 2010 CWE/SANS Top 25 Most Dangerous Software Errors. The MITRE Corporation, 2010, <http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.pdf>
14. ISO/IEC/IEEE 15026-2:2010, Systems and software engineering – Systems and software assurance – Part 2: Assurance case.
15. Program Executive Office (PEO) Integrated Warfare Systems (IWS) Technical Review Manual (TRM) (Draft), Department of the Navy, Naval Sea Systems Command, Program Executive Office, Integrated Warfare Systems, December 2008.
16. National Defense Industrial Association (NDIA) System Assurance Committee. Engineering for System Assurance, Version 1.0, 2008.
17. Chandra, P., Chess, B., and Steven, J. "Putting the Tools to Work: How to Succeed with Source Code Analysis, "IEEE Security & Privacy, pp. 80-83, May-June 2006.

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

INCOSE International Workshop 2012
21-24 January 2012
Jacksonville, FL
<http://www.incose.org/newsevents/events/details.aspx?id=140>

National Security Technology Expo
6-8 February 2012
San Diego, CA
<http://www.ubm.com>

28th Annual National Test and Evaluation Conference
12-15 March 2012
Hilton Head Island, SC
<http://www.ndia.org/meetings/2910/Pages/default.aspx>

IEEE International Systems Conference
19-23 March 2012
Vancouver, BC
<http://ieeesyscon.org>

SEPG North America 2012
23 March 2012
Albuquerque, NM
<http://www.sei.cmu.edu/sepg/na/2012>

28th Annual National Logistics Conference
26-29 March 2012
Miami, FL
<http://www.ndia.org/meetings/2730/Pages/default.aspx>

Software Assurance Forum - Spring 2012
26-29 March 2012
McLean, VA
<https://buildsecurityin.us-cert.gov/bsi/events.html>

2012 DoDIIS Worldwide Conference
1-4 April 2012
Denver, CO
<http://www.ncsi.com/dodiis12/index.html>

GovSec 2012
2-4 April 2012
Washington, DC
<http://govsecinfo.com/Home.aspx>

Systems and Software Technology Conference
23-26 April 2012
Salt Lake City, UT
<http://sstc-online.org>

SEPG Europe 2012
5-7 June 2012
Madrid, Spain
<http://www.sei.cmu.edu/sepg/europe/2012>

Software Assurance Working Groups - Summer 2012
26-28 June 2012
McLean, VA
<https://buildsecurityin.us-cert.gov/bsi/events.html>

International Symposium 2012
9-12 July 2012
Roma, Italy
<http://www.incose.org/newsevents/events/details.aspx?id=142>

GFIRST8
19-24 August 2012
Atlanta, GA
<http://www.us-cert.gov/GFIRST>

26th International Biometrics Conference
26-28 August 2012
Kobe, Japan
<http://www.ourglocal.com/event/?eventid=11988>

15th Annual Systems Engineering Conference
22-25 October 2012
San Diego, CA
<http://www.ndia.org/meetings/3870/Pages/default.aspx>

There is an "I" in Security

Oh sure, you have received one. A rather innocuous e-mail that typically starts out something like:

"URGENT - HELP ME DISTRIBUTE MY $15 MILLION TO CHARITY

IN SUMMARY: I have $15 million USD and I want you to assist me in distributing the money to charity organizations. I agree to reward you with 10% of the money for your assistance, kindness, and participation in this Godly project. This e-mail might come to you as a surprise and the temptation to ignore it as unserious could come into your mind, but please consider it a divine wish and accept it with a deep sense of humility."

I mean, after all, you are a savvy Internet user and you just KNOW that nobody is going to give you 10% of $15 million, right? I teach Enterprise Security here at Stephen F. Austin State University. Last month, right before class, I received the following e-mail (copied verbatim):

"How are you? I do hope that you receive this e-mail in good health. I am presently in Madrid, Spain to be with my ill cousin (Chloe). She is suffering from a critical medical condition and must undergo surgery to save her life. I am deeply sorry for not writing or calling you before leaving, the news of her illness arrived to me as an emergency and that she needs family support to keep her going. I hope you understand my plight and pardon me.

I want to transfer her back home to have the surgery implemented there because surgery is very expensive here. I am wondering if you can be of any assistance to me. I need about ($2,500) to make the necessary arrangement; I traveled with little money due to the short time I had to prepare for this trip and never expected things to be the way it is right now. I will surely pay you back once I get back home. I need to get her home ASAP because she is going through a lot of pain at the moment and the doctor have advised it necessary that the tumor is operated on soon to avoid anything from going wrong. I will reimburse you at my return.

Anticipating your reply at the earliest to my request! Thank you for all of your assistance."

For those of you who have not been spammed this way, it appeared to come from a Facebook friend. Seems my friend's Facebook page had been compromised—he had responded to the following e-mail:

"HELLO, your Facebook account has been suspended due to suspected compromise. Please reset your account password with the link below to reactivate your account."

My friend's compromised account let the spammer target all of his friends (using spoofed e-mail) trying to get some money from them. The fun part for me was sharing this with my class, and, over the space of a full week, playing this scammer along. First I offered to send the money direct to his wife (by the way, I knew he was not married). After a return e-mail explaining that ONLY a money order to Spain would suffice, I then offered to send the money as a direct deposit to his bank account. Over the next week, his cousin Chloe mysteriously progressed from a tumor to cancer, then to emergency abdominal surgery. I was expecting either malaria or the Black Plague next. I sent him on a wild goose chase towards the end, saying the money order was returned due to a wrong address. After about five e-mail exchanges, I finally told the scammer than he/she had provided entertainment and education for my security class. The spammer, in turn, had the class to say "good luck" on the final e-mail.

Seriously, how many scams do we get via e-mail and the Internet? I am not really the 1,000,000th visitor to the web site. I have not won a free iPad for participating in a survey. There is no magic sweepstakes that you are automatically entered in based on having an e-mail account. The IRS is not trying to send me a refund using an .aol address. FedEx did not send an executable file to me explaining why they did not complete my delivery. The Better Business Bureau is not trying to redirect me to a page to explain a "suspicious complaint" against me.

Yet, every day somebody "clicks before thinking." We often forget that our e-mail address and our Internet connections serve as a huge "ATTACK ME HERE" target to those with no scruples. No matter how good your firewall, spam filter, or antivirus software is, nothing in the world that will protect you from momentary stupidity on the part of the user. You need to have frequent education and training on how to keep yourself safe. There are SO many ways that security can be compromised by literally inviting malware or viruses onto your computer. Do not think that a quick, "I will take the online test, and get an 80% without studying," is enough. You have to be prepared and educated every time you sit down at a computer. You have to think. Indeed, you might be the "Weakest Link". You do not need an unsecured USB drive to compromise security. You have to think "E-mail and Internet Security" ALL the time, no matter where you are …

… he says, as he sits with his MacBook at Starbucks, sipping on a venti coffee with extra cream, answering e-mails, paying off a few bills, and sending this column off to the wonderful and humble staff at CrossTalk …

… using an open, unencrypted, unsecure Internet connection with at least 10 people nearby on their own computers, potentially monitoring and copying every keystroke and piece of data going into and coming out of my computer.

Go figure. It is easier to preach than to actually follow my own good advice.

But that is another column.

David A. Cook Ph.D.
Stephen F. Austin State University
E-mail: cookda@sfasu.edu