It Is Not Too Late for Software Assurance!

Robert A. Voitle, Jr.,
Application Software Assurance Center of Excellence
Arthur J. Boote,
Application Software Assurance Center of Excellence
James "Woody" Woodworth,
Application Software Assurance Center of Excellence

Abstract. Software Assurance is the practice of designing secure software that can safely and reliably operate in a hostile environment and resist attacks when all other network defenses have failed. Real-time IA focuses on mitigating attacks while within that hostile environment but can be greatly aided by Software Assurance practices regardless of where an application is in its lifecycle. In 2012, Yahoo suffered from an attack on an application initially developed by another company and had Yahoo executed Software Assurance techniques on the mature application, they could have prevented a compromise that resulted in the release of more than 400,000 user names and passwords.

Introduction

In May of 2009, President Obama said, "Our technological advantage is a key to America's military dominance. In today's world, acts of terror could come not only from a few extremists in suicide vests but from a few key strokes on the computer—a weapon of mass disruption!"

So, how secure is your application? This is a question we ask each program office before and after Software Assurance Risk Assessments. At the start, programs are very confident with the security of their "baby"; but, once we finish, that confidence is usually reduced. As they discover, the good news is all can be fixed; and, with the proper training, tools, techniques, methods and processes, they can ensure continuous application security from within to couple with the real-time Information Assurance efforts deployed by their hosts.

Software Assurance is the discipline of defensively coding software applications/systems in order to harden them from compromise/hacking. While traditional software engineering and coding practices emphasize user functionality and follow-on sustainability: deferring security to systems-level design), Software Assurance, by comparison, bakes security into the software itself. It does not come at the expense of user functionality or follow-on sustainability—it adds security up-front, in a deliberate effort to make the systems as least-hackable as possible, especially those operating in contested environments.

In addition to being a sound systems engineering practice, Software Assurance is now a legislative requirement. More specifically, Section 933 of Public Law 112-239 [1] requires the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics and DoD Chief Information Office to develop and implement a baseline Software Assurance policy for the entire lifecycle of systems.

To fully comply with P.L. 112-239 and improve Mission Assurance for the ever-increasing majority of software-enabled systems, enforcement mechanisms must be put in place to require assessments prior to system fielding as well as throughout their lifecycle. Current policy, guidance, enforcement mechanisms and force structures are inadequate for full compliance.

In today's ever-changing environments, Software Assurance is Mission Assurance! Government and industry leaders in the software security community have advocated for years for widespread Software Assurance strategy implementation. They have struggled with insufficient official guidance, the absence of dedicated funding sources and lopsided emphasis on hardware and network security instead of software, where industry analysts believe the majority of attacks occur. Acquiring and building software without accounting for security is no longer an acceptable risk.

The "way ahead" is simple…the DoD must proactively find, identify, and assess weaknesses that may be in software developed and/or used by the warfighter before, during and after fielding. By building security in early and often throughout the software's lifecycle, we tie security into the overall quality and functionality of systems, which not only prevents malicious entities from hacking our systems proactively, but saves millions of dollars per year in cost avoidance caused by flaws in the code, work stoppage from system failure, re-engineering, patching and re-fielding. Software Assurance is not just a "Just-In-Time" process utilized for fielding new technologies but allows us to apply techniques to identify and fix problems due to bloat, years of various coding techniques, and vulnerabilities inherent in Legacy Software, which houses the vast majority of our critical information; thereby hardening the current infrastructure.

Case Study: Yahoo Voices

In July of 2012, a hacking group posted usernames and passwords that had been pulled from a Yahoo sub-domain. The group itself revealed that their attack, a SQL injection that the attackers used to pass information retrieval commands to Yahoo's database servers, had allowed them to retrieve almost half a million unencrypted email addresses and passwords [2]. As the details of the attack were released to the public, Yahoo's image suffered from both the mistakes that allowed the attack to happen and from its poor damage control actions.

The problems date back before May of 2010, when Yahoo purchased Associated Content for $100 million to rebrand as its Yahoo Voices service. Associated Content provided a site for writers and subject matter experts to have an official platform for their articles and videos [3]. Unfortunately, Associated Content had not built their platform securely and Yahoo had not fixed any of the vulnerabilities after taking control of the application [4].

Over time, Yahoo moved the authentication scheme for Yahoo Voices over to its own logon system but did not perform many security fixes beyond that. Unfortunately, as was revealed in the attack, they did not remove the unencrypted tables nor did they scan for SQL injections in the application. These two issues, combined with

the amount of time the vulnerabilities were exposed to the Internet at large meant that the attackers were able to discover information about the databases used by Yahoo and the data stored within [5].

Once the information was in the wild, Yahoo lost the opportunity to become the primary source of information about the attack. It was unable to control what was reported and what details were released when the attackers themselves and security experts began putting the pieces together and reporting it themselves. The statements released by Yahoo were limited to an acknowledgement of the posting and that there was an investigation and fix action under way [6].

Real-Time Information Assurance

Modern network defense involves reconfiguring automated defenses to prevent attacks and close holes uncovered in previous attacks. Whenever an attack occurs, trained incident responders comb the logs and network for any details about the attack. They identify the exploit, possibly gather details about the attacker, and refine the network devices to prevent that attack from happening again. Whether it involves updating a blacklist, providing recommendations for an emergency patch or evaluating if an application is so insecure that a portion of it needs to be disabled or hidden to protect the network as a whole, all of these techniques are used reactively.

In order to protect their infrastructure and security processes, Yahoo released only the details that were required by law and did not confirm or deny any additional details released by third parties. Based on the responses and news releases following the attack, we cannot say whether they followed industry best practices when closing security holes identified in this attack, but we can explain those best practices that Yahoo could have followed in response to a compromise of their application.

After an attack, a forensic investigation will be launched with the goals of gathering as much information about the attack to secure the network and application as well as gathering information about the attacker's origin and motives. During the preservation and acquisition portions of the response network hardware, servers, and other items which contain logs or information about the attack will need to be protected from further alteration. This may include disconnecting network systems causing disruptions and outages of services. If all systems are left connected to the network until all the evidence of the attack is preserved, the network operators will need to ensure that the live environment does not overwrite or alter critical information. Identifying and maintaining data sources will require cooperation between the application maintainer, the network support technicians, and any other experts that may be able to provide inputs into locations of potentially relevant data.

From the logs and application itself, the forensic expert or persons responsible will have to compile all potentially relevant data and sift through the log files to reconstruct a sequence of events. There may be hundreds of thousands or millions of log entries captured as part of the data collection process and the analyst will have to identify which entries are normal operations and which entries are potentially relevant. From those relevant entries, the analyst will build a timeline of events in order to determine the extent and execution of the attack. The sequence of events will also point to the weaknesses used by the attacker to gain access to the compromised system and identify any other compromised systems or planted malware.

Once the results of the forensic investigation are complete, the network administrators can begin to patch holes at the network layer by disabling IP addresses at the firewalls and limiting traffic to the impacted system. One new protective measure that has seen widespread press is the Web Application Firewall (WAF). Just as a firewall sets up whitelists and blacklists which allow or deny blocks of IP addresses and allowed ports, a WAF is a device that learns expected HTTP commands and blacklists HTTP commands that may contain malicious logic.

It is possible to configure a WAF by hand with the results of a forensic investigation, and the WAF will forever prevent that same attack from occurring again, but WAFs share the same limitations as blacklists in that unknown vulnerabilities cannot be prevented. To address this, existing vulnerability scan results can be converted into a WAF rule set. A dynamic vulnerability scan can test every field in an application for susceptibility to known attacks such as SQL injection, cross-site scripting, and buffer overflows. When integrated into a WAF, the results can then provide a basic level of protection against known types of attacks.

If a dynamic scan is not run, most WAFs have a learning mode that can identify expected behavior over a period of time. Unfortunately, a deployed application may undergo constant attacks and certain exploits might be allowed in rules created during that learning period. Once the rules are created, they will be monitored and the operator can view any requests that may violate them to determine if a rule will cause issues. All of the rules and the results of their monitoring phase will need to be evaluated.

In order to get the maximum utility out of a WAF, the operators must be knowledgeable in several areas that include the specific application behavior and application security in order to ensure that rules will close security vulnerabilities while maintaining functionality of the application. Just as network firewall rules implemented to improve security can prevent legitimate traffic from passing, WAF rules can disable application functionality by preventing legitimate data from passing. However, sometimes functionality may need to be disabled in the name of security.

Once an attacker has identified a critical security vulnerability that can bring down a system or reveal sensitive information, the assumption is that they and other attackers will use that vulnerability again. Sometimes the hole that was used cannot be fixed without rewriting source code or re-designing the application and the only way to prevent the loss of further data is to disable the vulnerable functionality. This is reserved for cases where a self denial of service is preferable to further compromise.

Unfortunately the users are left to fill the gap in functionality with workarounds or manual processes that can cost an organization more man hours than any investigation or hotfix action. After taking into account the massive costs endured by an organization following a successful attack, the increased development costs, schedule time, and additional coding requirements of a rigorous software assurance process begins to make sense for both software developers and their customers.

In Yahoo's case, the vulnerability identified in their investigation was fixed and the unencrypted username and password data-

base was removed. However, had Yahoo executed a Software Assurance Assessment prior to integration of the Yahoo Voices application into their environment, the injection vulnerability and the insecure database would have been identified before it was exploited. By not proactively eliminating vulnerabilities, Yahoo's customers and public image suffered.

Software Assurance

Real-time information assurance tools such as a WAF offer program offices a measure of security. They provide very specific protections to an application, but if an attack is not defined by their threat identification systems and a security hole still exists within the source code, that application is still vulnerable. So how does a program office solve this problem? The integration of a Software Assurance policy that includes both dynamic and static analysis of the application source code is essential to preventing exploitation of those vulnerabilities not covered by WAF rules and filters.

There are many ways Software Assurance can be integrated into the development lifecycle of an application, but they all come down to two core implementation methods: the deep dive approach and the triage approach. Both have their merits and drawbacks. It is up to program managers and technical leaders to discuss both approaches and determine which is best for a given application.

The Deep-Dive Approach

The core idea behind the deep-dive approach is to identify as many security issues as possible through a combination of static and dynamic analysis and manual penetration testing. Every method of every class in every file must be scrutinized and declared vulnerability free. Even the system architecture, risk management procedures, and systems engineering methodologies are evaluated for security vulnerabilities. No attack surface is left unexamined. As a result, this approach is often quite time consuming; taking anywhere from 8 to 12 weeks to complete.

A deep-dive is often conducted toward the end of a lifecycle when active development has finished, but before QA testing begins. Once done, the application can be expected to pass rigorous security testing and is ready for deployment. This approach works best when used with large-platform, MAC I –type systems utilizing more classical development lifecycles such as the Waterfall lifecycle.

Because the costs in both true dollar amounts and man-hours can be extensive, it is necessary to such programs to plan for deep dive approaches in the planning and requirements phases of their development lifecycles. Even with proper planning, the costs and time needed to perform a deep dive security analysis makes this approach prohibitive for most standard applications in use in the DoD today.

The Triage Approach

In today's fast-paced development environments a Software Assurance approach that takes weeks to complete just is not feasible. Program offices utilizing RAD or Agile development cycles simply cannot allocate more than a week to Software Assurance practices. In such cases, a triage approach may be best.

Time is not the only resource saved by the triage approach to Software Assurance. The material costs are also dramatically lower. Triage is an overall cheaper alternative to the deep dive security analysis of a system. As a result, it lends itself to legacy systems (those systems that have entered into the maintenance and sustainment portions of their lifecycles) where funding for extensive testing, evaluation and repair is often not available.

A triage approach to Software Assurance emphasizes identifying and remediating low-hanging fruit through static and/or dynamic analysis. In this approach, it is not so important to identify and remediate every single security threat, only the easiest or those deemed to be the highest risk should be considered. As each iteration of the lifecycle completes, more and more issues will be identified and remediated.

While this issue takes significantly less time than the deep-dive approach to Software Assurance, there is one glaring flaw: if a high-risk security issue is not properly identified, the application could remain vulnerable through several development iterations. This is where real-time IA measures, like WAFs, can help mitigate the risk to an application.

Consider the Yahoo case study presented earlier: Associated Content was not a large mega-corporation capable of spending millions of dollars to implement a deep-dive Software Assurance program. At the time of their acquisition by Yahoo in May of 2010, the triage approach to Software Assurance was still in its infancy, but it could have saved Yahoo from both the financial and reputation losses it undoubtedly suffered.

A triage of the Associated Content software would have revealed the SQL Injection vulnerability that allowed attackers to retrieve the unencrypted passwords and e-mail address of the systems users. Real-time Information Assurance measures could have been put into place that would have prevented the exploitation while Yahoo developers took a closer, more analytical, look at the vulnerability that would have revealed the lack of proper encryption of the data stored in the legacy database. Unfortunately, many of the developers, managers, and Information Assurance personnel at Yahoo may not have known that a Software Assurance plan was a necessity because they lacked the proper training to identify such risks.

Education is Key

Regardless of which approach is used to identify, catalog, and remediate security vulnerabilities, it may all be wasted effort without proper education and training of program personnel. While security is becoming a hot topic to teach as part of a software and computer engineering program, the majority of programmers have not yet had much exposure to software security and secure development training. Ideas such as data validation for security and whitelists are foreign concepts to a large percentage of the software development workforce. As a result, it becomes necessary to ensure that everyone involved in the development lifecycle receives a degree of training concerning Software Assurance. It is imperative that developers and testers understand how to identify security risks to the application and how to remediate those risks.

In the end, it is equally important that IA personnel and program management learn the language of Software Assurance so they can communicate openly with the developers and tes-

ters about the sort of risks associated with different vulnerabilities. This can only be achieved through repeated exposure to education and training materials, and an open dialog about the overall security profile of an application.

Combining Software Assurance and Real-Time Information Assurance measures helps to ensure that security gaps in an application's source code are covered, at least temporarily, by the rules governing the WAF. But, once a WAF is in place, why bother repairing those gaps? As mentioned earlier, WAFs and other Real-Time Information Assurance measures essentially amount to security blacklists: defining what harmful information should look like and blocking it at the server and/or application layers. As attackers refine their techniques and new, more creative methods to defeat these blacklists are created, the rules governing what tainted or harmful data looks like will need to change.

If security flaws are not fixed at the source, engineers will find themselves in an arms race trying to keep Real-Time Information Assurance rules updated to defend against current attack vectors. Repairing the security holes at the source, when done correctly using whitelists and proper data validation, puts an end to the arms race by eliminating the security threat. But may all be for naught if the developers, testers, program management, and Information Assurance staff are not properly educated regarding Software Assurance and the risk that not integrating such principles into the development lifecycle can bring.

Software Assurance is not just an option to secure our critical software applications, it is an absolute necessity. The DoD must be proactive when it comes to identifying and assessing the weaknesses present in software in use by the warfighter and our national infrastructure. Building Security in to the software lifecycle is critical to this task, especially if it is built in early and often. Doing so will prevent cyber attacks to critical systems and save millions of dollars each year. Not just a solution for new software being developed, triage approaches to Software Assurance allow legacy systems to incorporate these techniques into their sustainment plans improving the preparedness of the nation as a whole to all cyber threats.

Disclaimer: The views expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the US Air Force, Department of Defense or the U.S. Government.

ABOUT THE AUTHORS

Robert A. Voitle, Jr. has been a contractor working as a Senior Software Assurance Analyst for the Air Force at the Application Software Assurance Center of Excellence since 2008, and is regarded as an expert on Software Assurance and application security. He has worked as a consultant for Fortify Software, Aspect Security and, currently, Cigital Inc. He has authored several works that range in scope from informal news articles to white papers. Robert is a graduate of Auburn University where he studied Journalism and Technical and Professional Communication.

E-mail: rvoitle@cigital.com

First Lieutenant Arthur J. Boote is currently assigned to the Business & Enterprise Services PEO, Gunter Annex in Montgomery, Alabama serving as the Chief Technology Officer for the Application Software Assurance Center of Excellence. 1Lt Boote has previous experience as a Cyberspace Crew Commander for the 561 NOS and Digital Forensics technician prior to joining the Air Force. 1Lt Boote was instrumental in the publication of the Secure Programming Best Practices Guide for Department of Defense programmers. 1Lt Boote has a degree in Computer Engineering from Florida State University.

E-mail: arthur.boote@gunter.af.mil

James "Woody" Woodworth is a Department of the Air Force Civilian currently assigned to the Business & Enterprise Services PEO, Gunter Annex in Montgomery, Alabama serving as the Chief of the Application Software Assurance Center of Excellence (ASACoE). He has served in the Air Force for 33 years, the first 20 years on active duty in the Air Force JAG Corps and the remaining 13 as a civilian in a variety of software development positions. He has been assigned to the ASACoE since its stand-up in 2007, first as the Operations Chief and then as Chief upon his return from deployment to Iraq. He is considered one of the leading Software Assurance Subject Matter Experts in the Department of Defense.

E-mail: james.woodworth@gunter.af.mil

REFERENCES

1. National Defense Authorization Act (NDAA) for Fiscal Year 2013. Pub. L. 112-239 §933. 2 Jan 2013.
2. Schwartz, Mathew. "Yahoo Hack Leaks 453,000 Voice Passwords." InformationWeek. UBM Tech, 12 Jul 2012. Web. 29 May 2013. <http://www.informationweek.com/security/attacks/yahoo-hack-leaks-453000-voice-passwords/240003587>.
3. Heist, Matt. "What Yahoo needs to do with Associated Content." The Digital Content Blog. Guardian News and Media Limited, 08 Jun 2010. Web. 29 May 2013. <http://www.guardian.co.uk/media/pda/2010/jun/08/yahoo-associated-content>.
4. Arthur, Charles. "Yahoo Voice hack leaks 450,000 passwords." thegaurdian. Guardian News and Media Limited, 12 Jul 2012. Web. 29 May 2013. <http://www.guardian.co.uk/technology/2012/jul/12/yahoo-voice-hack-attack-passwords-stolen>.
5. "Yahoo! Voices Website Breached 400,000 Compromised." TrustedSec. TrustedSec, 11 Jul 2012. Web. 29 May 2013. <https://www.trustedsec.com/july-2012/yahoo-voice-website-breached-400000-compromised/>.
6. Prince, Brian. "Yahoo Confirms 400,000 Passwords Stolen in Hack." eWeek. Quinstreet Enterprise, 07 Jul 2012. Web. 29 May 2013. <http://www.eweek.com/c/a/Security/Yahoo-Confirms-400000-Passwords-Stolen-in-Hack-112338/>.