My name is Jesus Abelarde and I am Lead Systems Security Engineer for the MITRE Corporation. I currently work on multiple engineering domains that includes Cyber, Network, Software, Testing and Integration on various government programs. Prior to joining MITRE, I led multiple research and development efforts for the Department of Defense. Since I cannot go into specifics, this two minute video about MITRE will hopefully fill in the blank (https://www.youtube.com/watch?v=7skH7SxBRM) of the type of projects and problems that I am trying to solve for MITRE. For folks not familiar with Systems Security Engineering, it is integrating security into system engineering.
This presentation will provide an approach in evaluating and integrating commercial and open source software during the System Development Life Cycle (SDLC). The software includes libraries and executables except firmware and drivers. The goal is to equip staff with suggested tools and methodology to conduct a technical evaluation with acceptable level of rigor. The evaluation process will provide an answer that goes beyond the traditional approach such as relying on the community or an approved product list. The intended audience of the presentation are for Managers, Security Officers, Entry Level, Non Cyber Engineers & Scientists, Technicians, Companies with limited cyber resources to be able to conduct a technical evaluation.
By analyzing the NIST data regarding Common Weakness Enumeration (CWE) in software that was exploited since 2008, the trend for many exploited software CWE is going exponentially higher instead of leveling. According to Verizon, software accounts for the highest exploited vector of attack year after year. An additional metric in 2017 from Synopsys shows that of the more than 1,100 commercial codebases analyzed by the company last year, 96% contained open source components, the same percentage as the previous year. However, many applications now contain more open source than proprietary code. The study shows that 78% of the examined codebases were plagued by at least one open source vulnerability, compared to 67% in the previous year. Synopsys noted that more than 4,800 vulnerabilities were found in open source software in 2017. More than half of the vulnerabilities identified during Synopsys’ analysis have been classified as “high risk” and 17% of them have been highly publicized (e.g. Heartbleed, Logjam, FREAK, DROWN, POODLE).
This seems like software projects have not really incorporated better business practice on applying secure system engineering in the system development life cycle.

As an example, many of these CWE looks like it can potentially be minimized by conducting a thorough evaluation on the Commercial-Off-The-Shelf (COTS) and Free Open Source Software (FOSS) being used in the system during system development. This will reduce unknown vulnerabilities in the system.
There are two current SDLC processes heavily used in both private and public domain: Waterfall and Agile. The Agile process is newer and is the current heavily used buzzword thrown around in many development programs. Management tends to apply it to everything, but if you study a lot of the complex software programs that experience overrun, one of the root cause is that they use agile process in a project where it was not designed to be used. The Agile development process was designed for simple software systems with fluid requirements, guidelines, processes, minimal documentation and quick acquisition lifecycles such as websites and prototypes. On the other hand, the Waterfall process is designed for complex software systems with demanding requirements, strict processes and standards, thorough documentation, and long acquisition lifecycle such as most military systems. A recommended approach is to do a hybrid where the projects utilize what works in Agile and Waterfall based on the complexity of the requirements, engineers’ skillsets and experiences, available resources and the ability to modify corporate development process policy.
Requirement Phase:
According to MIL-STD-498, operational/user requirements are divided into systems specification and then further divided into domain specific requirements as shown in the diagram. There are no recommended specific technical evaluation for this phase. But, this phase influences the level of effort to evaluate COTS/FOSS software. Therefore, if the specifications are not derived correctly, then there is potential for changes that must be done later in the lifecycle which will cost the project more. The recommendation here is to ensure that security requirements are completely specified to ensure that appropriate resources will be allocated by management such as COTS/FOSS evaluation.

Architecture Design Phase:
This phase is the development of an integrated system architecture framework and design based on the specification derived in the requirement phase. The system architecture framework and design are usually composed of multiple layers starting from high level usually block diagrams to low level usually parts list. The recommended technical evaluation process in this phase is to conduct Threat Risk Modeling and Market Security Research.
Threat Risk Modeling identifies potential security issues and determines the necessary controls to mitigate those issues. It is recommended that other engineering domains conduct their own applicable threat risk modeling with the support from security personnel as they are the main designer of that part of the system which most security personnel may not have that level of insight. Microsoft released two softwares, one to help people understand Threat Modeling and one to provide guidance on creating and analyzing threat models. But, Microsoft tools is limited only to the application layer threat modeling and is missing the other layers such as networking, personnel, etc.

Getting Started on Threat Modeling: Microsoft Elevation of Privilege (EoP) card game (FREE)

Clarify details of threat modeling and examines possible threats to software and computer systems


Threat Modeling Tool:

Allows software architects to identify and mitigate potential security early
Designed for all developers and not security experts by providing guidance on creating and analyzing threat models

Market Security Research

- Conduct once COTS/FOSS software is identified
- Gather information to determine risks and mitigation
- Divided into three phases:

Market Security Research is basically “Googling” for information publicly available. This should be done as soon as the COTS/FOSS software is identified. The goal is to gather as much information to help determine risk and mitigations of the use of that particular software in the system. As you can see in the diagram, the Market Research can be broken into three phases. The first phase is to determine the pre-implementation evaluation criteria for each software item which will be key when conducting product and company research. Based on the data found, the second phase is to determine the Software risk per item which dictates the risk mitigation or acceptance activity. Then, the third phase is to apply other variables such as project resources and time limitation to determine if the original risk determination is still valid.
Development Phase

- Implement the system Architecture and Design based on specification that meets the requirements [6]
- Use secure coding and integration standards [7]
  - Project and organization defines set of coding/integration process, rules and guidelines
  - Carnegie Mellon Software Engineering Institute site has tons of information to help develop tailored standards
- Execute better business practice on software quality processes (validate quality through well defined metrics) [8]
- Perform secure code reviews

The development phase is the implementation of the system architecture and design based on specification. The recommended technical evaluations process are the following: use secure coding and integration standards, execute better business practice (BBP) on software quality and perform secure code reviews. The definition of secure coding and integration standards, process, rules and guidelines should be defined by each project and the organization. The Carnegie Mellon Software Engineering Institute site has tons of information that can be leveraged to help develop a tailored standards. The execution of software quality BBP starts by defining the desirable software characteristics derived from stakeholder needs. A typical quality characteristics are suitability, reliability, operability, performance efficiency, satisfaction, safety and usability. In order to monitor and validate these defined software characteristics, metrics need to be defined. The more metrics defined for each characteristics, the more fidelity to validate and monitor these quality characteristics throughout the SDLC. A strongly recommended metric is the cyclomatic complexity of the code. This metric alone can provide insight on the following: Is the code structured and well written? Is it testable? Will it incur a high cost and effort to maintain the software? Will it have high risk of vulnerable code? Many static code analysis tools such as https://pmd.github.io/ already have built-in function to capture some of these software characteristics.
Secure code reviews is the analysis of the software code. There are two approaches of conducting secure code review. These two ways supplement each other. The first approach is Manual Code Review which is a software inspection process used to identify hard-to-find vulnerabilities. This inspection process varies such as ad hoc review, pass around, pair programming, walkthrough, team review and inspection. It is recommend that manual review be done for each COTS/FOSS used as it can potentially find code issues not found in automated review such as logic vulnerabilities. The OWASP organization has secure code review processes that can be leverage by the project. The second approach is the static code automated review which employs source code analysis tool to find flaws efficiently. This tends to produce a high number of false positives and only find a small percentage that are actually exploitable. The triaging process is usually done when a high number of false positives is experienced due to time/resource constraints. There are many tools available such as

Opensource: PMD, Find Sec Bugs (https://find-sec-bugs.github.io/bugs.htm)
Commercial: Fortify, Coverity, Klocwork

A suggested integration approach to code review during the SDLC is to ensure that during planning phase, there is sufficient resource to conduct this task. If applicable, the use of SANS Top 25 Most Dangerous Software Error is recommended as a CWE checkers/filters for the tool.
Secure Code Tool Recommendation

- Opensource: PMD, Find Sec Bugs
- Commercial: Fortify, Coverity, Klockwork
- If only one, use Coverity Scan website for FOSS!
  - Strong community and perfect for Market Research
  
https://scan.coverity.com/

There are many secure code tools available such as PMD (FOSS), Find Sec Bugs (FOSS), Fortify (commercial), Coverity (commercial) and Klockwork (commercial). For FOSS evaluation, the Coverity Scan website by Coverity is strongly recommended. It is a free online scanning tool that will test every line of code for Java, C/C++, C# and Javascript. It has a strong community which can be free resource to help you analyze or fix the bugs found. This tool is also great to leverage during Market security research to see if somebody has uploaded that FOSS in the website. It is not recommended to upload your non FOSS software. Remember to check terms and agreement prior to uploading.
Behavioral Analysis

- Supplements code review and perfect for executables
- Typical setup uses 2 VMs (1 VM to inject resources and the other is instrumented)
- Example: Lukas Stefanko analyzed the Android Trojan Spy Proxy APK as it cannot be detected by most AV programs [9]
  - Using network instrumentation tools like Wireshark capture Trigger Events, Malware Intent, Malware Behavior.
  - Only mitigation is Google Alert notice not to install

Behavioral Analysis is a blackbox testing method focused on external visible behavior of the software. This is a supplemental code review and is perfect for executables. A typical lab environment set up is to use virtual machines that are network to each other, but not to the host system. One VM will be set up with tools that will supply the application under test with the resources it needs to try to invoke a behavior or communication, and the other VM will contain the application under test and the necessary instrumentations such as Wireshark, Ollydbg and Process Explorer.

At the time of writing the paper, the Android Trojan Spy Proxy APK was not detected by many Anti-virus program as malicious. But, when Lukas Stefanko performed simple behavioral analysis using Wireshark and logs on the Android Trojan Spy Proxy, he was able to find the following Malware characteristics:

- The Malware Trigger Events was invoked by receiving a text message or when the phone changed connectivity or when the user unlocks the phone.
- The Malware was designed to gather personal data such as messages, call log, location, received SMS, WiFi Info, cellular data status, IMEI, syslog and user account. Then, storing these data via primary external storage to be sent to a remote server via HTTP every 30 minutes. If phone WiFi or cellular data is disabled, malware will enable cellular data as soon as the phone screen is turned off to send log file to a remote server. Then, the malware disables cellular data as soon as the log file is sent. The only mitigation is Google alert notice not to install.
If time and resource is available, an analyst can then perform reverse engineering to the application under test. On this APK example, an analyst can review the AndroidManifest XML file which is included in every APK application. This XML file will provide information about the application and the permission it will ask to the user to grant to be able to install in the phone. The diagram on the left is a snippet of the permission contained in the XML file. Does a proxy software really need those permissions to do its functions? Then, the analyst can decompile the APK application via decompiler. The diagram on the right shows a code snippet of the toggleMobileData function, which performs the enabling and disabling of the user cellular network. Finally, an analyst can write a IDS rule set from the compiled code static properties.

Alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg: “Potential Android Trojan Proxy Virus Server DNS Query”; flow:established, from_server; content: “proxylog.dyndns.org”; nocase; content; “Standard query”; nocase; sid: 18758;)
Alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg: “Potential Android Trojan Proxy Virus”; flow:established, from_client; content: “ proxy/log.php?id=”; nocase; content; “POST”; nocase; sid: 18757;)
The verification and validation phase is a system level test to ensure that it meets the specified user requirements. It is recommended that the testing be conducted in a traceable manner, which means domain specific requirements testing will support the system level specification testing which will then support the end user requirements testing of the system. The recommended technical evaluation are security audit, penetration testing and vulnerability assessment. A system security audit is a systematic evaluation of the security posture by measuring how well it follows to establish policies such as HIPAA. The audit run parallel with penetration testing and vulnerability assessment as each one have different objectives.

The Penetration Testing is an attempt to validate whether the potential vulnerabilities can be exploited. If exploited, then help determine the impact to the system and environment. A great sample tool is Kali which contains many pen testing tools to leverage. A System Security Assessment provides a list of potential known security vulnerabilities of the system and its components, including any COTS/FOSS used. If it is decided that vulnerabilities will not be addressed in a timely manner, then recommend pen-test team determine the exploitable of those vulnerabilities and risk management review be performed. At a minimum a system security assessment should be done when a system first is fully integrated and prior to system deployment. Some sample tools are Nessus (Commercial) and OPENVAS(FOSS).
The following slide shows a sample assessment process in four stages:

The Pre-scan preparation stage captures system under test configuration (i.e. software builds) and the assessment tools configuration (i.e. plugins, build)

The Reconnaissance stage includes conducting network mapping and fingerprinting using tools such as NMAP. The following are sample commands to leverage:

**Pre-scan preparation**: capture system under test and tools configuration (i.e. software builds)

**Reconnaissance**: Network Mapping + Fingerprint tool
  - Nmap and Amap

**Vulnerability Scan**: Nessus (Preferred) or Openvas

**Penetration Testing**: Kali OS image w/ Metasploit + Armitage

** Remember to update tools, capture configuration information of plugins, review + test dangerous plugins and invoke TCPdump on the background**
Finding host: ./nmap -n -sP IP Addresses
Finding Ports: ./nmap -n -sT IP Address -p 1-65535
Os Fingerprint: ./nmap -n -o -ST -p Ports Open IP Address
Software Version: ./nmap -n -sV -p Ports Open IP Address

The Vulnerability Scan stage uses vulnerability assessment tools such as Nessus (Preferred) or Openvas to try to find vulnerabilities in the system.

The Penetration Testing stage uses tool such Kali OS image which includes many penetration tools to try to exploit vulnerabilities found. Do not forget to do the following administrative task prior to any assessment: updating tools, capturing configuration information of plugins, review and test dangerous plugins on replicative system and invoking Tcpdump on the background.
Summary

- Almost any phase in SDLC allows you to conduct an evaluation
- Each evaluation provides data points (evidence) about the FOSS/COTS tools
- The data can be used to support a decisions on the level of risk acceptance in the development.
  - If risk is too high, then conduct more in-depth analysis or simply don’t used it

*Now that you’ve sat through this presentation, you should be able to use this consistent evaluation process and to equip your staff to properly integrate commercial and open source software during the System Development Life Cycle*

Back Up
References


Acronyms (1)

- FOSS: Free Open Source Software
- COTS: Commercial Off-The-Shelf
- SDLC: System Development Life Cycle
- CISSP: Certified Information Systems Security Professional
- GIAC: Global Information Assurance Certification
- GCIA: GIAC Certified Intrusion Analyst
- GSEC: GIAC Security Essentials
- GAWN: GIAC Assessing and Auditing Wireless Networks
- GCIH: GIAC Certified Incident Handler
- GPEN: GIAC Penetration Tester
- GWAPT: GIAC Web Application Penetration Tester
- GSNA: GIAC Systems and Network Auditor
- GMOB: GIAC Mobile Device Security Analyst
- GCCC: GIAC Critical Control Certification
- GREM: GIAC Reverse Engineering Malware
Acronyms (2)

- BLUF: Bottom Line Up Front
- CWE: Common Weakness Enumeration
- NIST: National Institute of Standards and Technology
- CVE: Common Vulnerabilities and Exposure
- CMMI: Capability Maturity Model Integration
- IDS: Intrusion Detection System
- XML: eXtensible Markup Language
- VM: Virtual Machine
- APK: Android Package Kit
- AV: Anti-Virus