

# Software Security Starts

## with Knowing What Is In the Code

The security of almost any system can be compromised, given the time and resources of a motivated party.

All of the components that make up a security solution – software applications, routers, networks, firewalls, encryption methods, intrusion detection mechanisms, warning systems, etc. – all operate on complex algorithms and mechanisms driven by software. In today's world of security and vulnerability, software is everywhere.

The ability to assess and verify secure operations, test for vulnerabilities, detect weaknesses and cover deficiencies relies on having a good understanding of the software involved in the secure systems. Unfortunately, an accurate knowledge of the building blocks of the software-intensive secure solutions is not always available. This software-driven security world has a visibly dynamic nature, based on availability of new solutions in the market-place and updates or upgrades to existing solutions. It is not uncommon for a security solution vendor to release updates and upgrades to a product many times in a year.

The software industry, in all its complexity and variety, has a simple flow structure. Every player in

this food chain brings in content from other players, contributes its value-added content, and passes it on to the next player in the chain. Examples of software food-chain players are contractors, outsourcers, commercial module suppliers, open source packages, or software that is reused from the very same component.

### Open Source Software and Secure Applications

It has been about 13 years since Netscape's 1998 release of the Navigator source code. Slowly but surely, open source software has found its way into every application in every industry that creates or consumes software. Gartner released a study in January 2011 which found that the top 500 IT leaders around the world all used open source in one way or another in their organization. Open source software use in the enterprise is mainstream and continuing to grow. Code re-use makes sense, so developers bring in mature open source code to accelerate their time to market and reduce their costs. Using open source code, such as Linux, Apache, PHP, or Android is common in all industries and verticals.

Fittingly, some of the earliest industrial applications of open source software were in the security domain. Given the availability of the source code, coupled with peer-review of the content by a global body of well informed developers, open source led itself to the development of secure applications. *Snort*, an open source network intrusion prevention and detection system was one of the earliest manifestation of open source in the security domain. *OpenSSH* is a free version of the SSH connectivity tools that today's secure internet applications rely on. A complete suite of open source connectivity and communication protocols

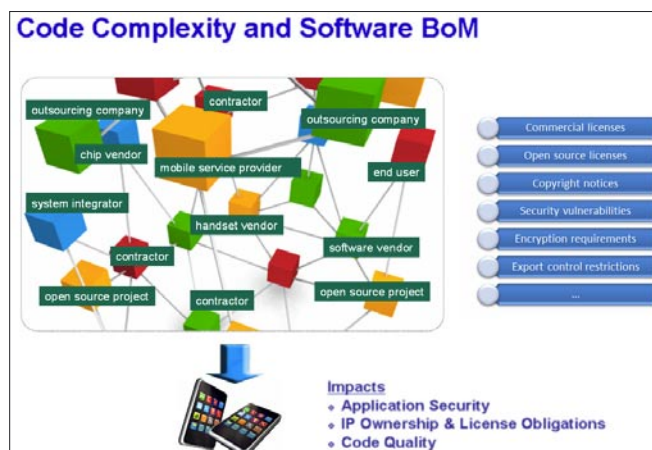


Figure 1. Code Complexity and Software BoM

is based on OpenSSH. Crypt and OpenCrypt provide solid, verifiable, encryption modules. For almost any application, and here we are only concentrating on security, you can find use of an open source software component in a project.

### Is Open Source Secure?

With software, be it open source or proprietary code, security vulnerabilities are a major concern. Certainly, open source software is not any more vulnerable than commercial software and there is an ongoing debate on whether it increases software security. With open source, more people can inspect the code to find and fix possible vulnerabilities. The end user can modify the code to implement any extra security features that they want, even down to the kernel level.

Vulnerabilities do not mean that you should avoid using open source software; in fact the majority of projects provide a patch, for issues that are identified by the user community, within hours. For example, a number of vulnerabilities in the earlier versions of OpenSSH were reported and quickly fixed in the subsequent versions. A simple Google search on security vulnerabilities of OpenSSH provides an insightful list.

Security vulnerabilities exist equally in commercial as well as open source applications. Major respectable vendors such as Microsoft and Adobe regularly publish reports on security vulnerabilities detected in their products and provide fixes for them. Google's open source Android client authentication vulnerabilities were reported in May 2011 and patches were made available within days. Likewise, Apple has provided several security patches for their mobile devices.

### Software Security Assessment Requires a Bill of Materials

Any security examination and management requires an accurate knowledge of the components of the software. Today's software solutions are complex with many components and sources of code (commercial, in house, contractors, open source, etc). This can make it difficult to put together an accurate picture of a Bill of Materials. In order to ensure quality and security, accurate records and an understanding of the components of the project are required (what components are used in the project, which version, etc). This information can be used to determine aspects such as known deficiencies (bugs), vulnerabilities, known security risks, code pedigree (where did it come from, when, who has worked on it, etc).

Known security vulnerabilities can be found in Vulnerability Databases. A Vulnerability Database is a collection of information about discovered vulnerabilities in various software projects. Currently,

there are many vulnerabilities databases that have been widely used to collect data from different sources on software vulnerabilities (e.g., bugs). The data essentially includes the description of the discovered vulnerability, its exploitability, its potential impact, and the workaround to be applied over the vulnerable system. Examples of web-based vulnerabilities databases are the National Vulnerability Database and the Open Source Vulnerability Database. Once the building blocks of a software application are identified, the Vulnerability Database can be consulted to identify any security issues.

To use a familiar example, Android's recent security scares draw attention to the need to detect and address security vulnerabilities in software projects. Android is especially a good case in point. Its mobility platform consists of close to 180 different software modules, knit together from many open source projects, some dating back years. Consisting of a collection of more than a hundred thousand files, Android can be customized for a specific mobile device, depending on the hardware platform and the applications delivered by the device vendor. Gaining an accurate knowledge of all the components that exists in the Android platform, exacerbated by the added functionality and customization carried out by the device and application suppliers, is very difficult if carried out manually.

### Creating a Software Bill of Materials

In order to mitigate any security risks in a software project, there must be a knowledge and understanding of the modules that make up that project. There are a number of ways to determine and keep track of a project's components (generating and maintaining a software Bill of Materials), which can be done manually or using automated methods.

In order to manually manage the building blocks of a project, records of content that is included in a project must be generated. In reality, this is almost impossible in a large project as developers prefer to develop code rather than document where each bit came from. Some organizations establish restrictive procedures throughout the development lifecycle. Only approved components, as determined by pre-set policies, are allowed to enter the code base. Also, manual methods rely heavily on guidelines, directives, and education. Developers must be educated about the policies, and some methodology for tracking the components must be utilised. In many cases information can be manually extracted from build scripts. A popular method is to carry out a manual audit of the software libraries and build components with the aim of obtaining insight into the software Bill of Materials.

However, manual processes and procedures can be very time consuming, expensive, and inaccurate. For example, a developer may unintentionally bring in 20 lines of code from an unknown or undocumented source. This could compromise the security of the entire project and may be very difficult to locate and fix.

### Automated Solutions

A number of automated solutions are beginning to address the need associated with establishing a software Bill of Materials, and are becoming part of the standard software life cycle management tools.

How do these automated software content management tools work? Basically the core function is automated code scanning and discovery of third party content, be it open source or other types of external software. The following are some common techniques:

- examining software directories
- examining folder, subfolder, and even file names in a directory
- examining text files that indicate the existence of certain packages and licenses or other use conditions (e.g. *readme.txt*, *copying.txt*, *license.txt*)
- *scrubbing* software files and detecting statements such as *copyright xxx* or *licensed under yyy*

The above techniques can detect software components if the files are used in an application in their entirety. However their effectiveness is limited in a number of scenarios, when:

- the package name, folder names, or file names are changed
- directory structures are modified
- identifying text files such as *readme.txt* are removed
- header information in the files are modified so that original copyright information or license attributions are lost

Most importantly, a limitation of the techniques described earlier is their inability to detect partial or snippet matches where a developer has incorporated lines of code from an external software file into his or her program.

Deep-scanning techniques detect code similarity more accurately. The method involves comparing a software file to a large reference database of tens of millions of known software files, and looking for similarity between the software file under examination and the reference software files in the database. Again, this could be as primitive as looking for exact matches (to source or binary files) in the database, or as sophisticated as

trying to match the code structure of a source file, in whole or partially, to the billions of lines of code in the reference database.

The upside of a large database is that you can find a match to your software. The downside of a large database is that unless you throw some intelligence at it, the results become useless. Firstly, without intelligent searching techniques, it may take significant amount of time to detect all matches.

Secondly, without intelligent sorting techniques, you find a lot of matches to your file. Hence the term *false positive* which is really the inability of the matching algorithm to sift intelligently through all the matching database records and position the relevant match at the top of all that was found.

Fortunately there are algorithmic methods available that can reduce the problems associated with large reference database.

Ideally, these tools should be integrated into the existing processes and development tools deployed within organization's *Application Lifecycle Management* (ALM) portfolio. Automated tools can seamlessly, transparently, and unobtrusively record and maintain a list of all components used in a project in all stages of development.

### Open Source Software Adoption Process (OSSAP)

As usual, any tool is effective only if used properly. Automated software component management tools or open source management tools are no exception. A process must be defined and put in place that employs automated solutions to advantage.

A survey of best practices in the industry has revealed the following 8 steps for a quality Software Adoption Process:

- Policy definition – Where acceptable vendors, packages, or licenses are specified. Notification and remedial actions are defined in case of a violation of the policy. Automated tools help with the definition and digital capture of the policy.
- Package pre approval – Where the process for requesting, examining, scanning, and approving or rejecting an external software module is established.
- Baseline scan – Where the existing code portfolio is scanned and identified using automated tools.
- Incoming software examination – Where software delivered by contractor, outsourcers, or software commercially obtained is automatically scanned and identified.
- Regular project scans – The entire project code under development is regularly (for example every weekend), automatically scanned and the results

- are compared to policies, highlighting violations.
- Library Audits – Includes automated, real-time, analysis of the code as it is checked into the company’s software library, and raising flags if violations of the policies are detected.
- Developer Assistants – Where automated solutions installed on developer workstations actually advise developers as they put the code together. The value of an automated Developer Assistant is the ability to detect and remedy policy violations at the earliest stage of development, leading to significant savings in effort and cost later on.
- Build Analysis – Where the final software artefact is analysed, results compared to the established policies, and an unknown or violating software component actually stops the build process until it is remedied.

## Conclusion

In the industrialized world of the twenty first century, security solutions invariably involve components driven by large and complex software applications or modules. These solutions are only as secure as the individual links that make up the application security chain. The ability to assess, identify and correct security vulnerabilities relies heavily on an accurate understanding of the composition of the secure application. The building blocks of the secure solutions: software packages, modules and applications, originate from many sources that include in-house code and third party software such as those supplied

by contractors, outsourcers, commercial vendors, or obtained from open source software repositories. While manual methods of establishing and maintaining a software Bill of Materials may be used in simple projects, complex solutions of today require automated code scanning and software pedigree determination. Third party and Open Source License Management tools, can quickly provide a Software Bill of Materials, reporting all third party code in an organization or a product. Once contributing software components have been identified, attributes such as authorship, copyright and licensing obligations, encryption properties, security vulnerabilities, and latest updates can be automatically highlighted.

Automated code scanning tools are effective only if used properly within a total third party adoption process. Practices such as the *Open Source Software Adoption Process* (OSSAP) can effectively institutionalize software component analysis and leverage automated tools in all stages of a quality software development process to advantage.



Figure 2. OSSAP Diagram

## MAHSHAD KOOHGOLI

*Mahshad Koohgoli is CEO of Protecode, Inc. (<http://www.protecode.com>), an innovative provider of open source license management solutions, based in Canada. He has more than 25 years of experience in the telecommunications industry, specializing in technology start-up businesses, and holds several patents in the computer and communications field.*

## ANDY SPARKS

*Andy Sparks is responsible for business development activities at Protecode. Prior to joining Protecode, he held positions with Deloitte, GlitchSoft, Magmic Games, NewHeights Software, March Networks, and xwave. Andy has a Bachelor of Commerce degree with a concentration in Marketing from Carleton University.*