SOSEC Assignment 1
Group 06

2020-09-08

Ranjana Ghimire
Justas Narusas
Efstathios Psyllakis
Oskar Elfving Söderström
Faith Onwumere

Time Division

Task	Hour allocation/person
Zoom discussion meeting for team charter and finalizing the team charter on 09-03-2020 at 10:30 AM	1 hr (Completed)
Going over the necessity of software models	45min(Completed)
Basic understanding of SAMM 2.0	30 min(Completed)
Studying the allocated work for SAMM 2.0	1.5 hr(Completed)
Basic understanding of the BSIMM	1hr(Completed)
Studying step similar in alternative method to allocated step of SAMM 2.0	1 hr(Completed)
Group discussion of the completed sections on 09-07-2020 at 10:30 AM	30 min(Completed)
Team meeting with Alan presenting all the findings on 09-07-2020 at 15:00	2 hr(Completed)
Allocating time for step 4	2 hr(Completed)
Documentation of the overall work on 09-08-2020 at 3 PM	1 hr 30min(Completed)
Finalising the documentation on 09-08-2020 at 6 PM	30 min(Completed)
Total Time Spent:	**12hr**

Why is software insecure?

There are various reasons behind the claim of not having a secure software, some being:

- Market competition makes it difficult to follow the build process of the code as everyone is in a rush to deliver the product and earn money rather than making their own product.
- External library testing cycle can go back many steps creating difficulty in testing the libraries and so on, creating a loophole for securities within such libraries
- Planning for all the design steps is difficult and can not be thought during the beginning phase
- Issue with aligning security policies and ideas with other teams within the company
- Zero day has huge impact in this level as resolution of defects depends upon prioritization and resource
- Full logs might not be present to resolve the security issues in the software inturn increase the zero day timing
- Issues with freelancing companies to align with SDLC

SAMM Model

The SAMM structure consists of five business critical functions. Let's look into each phase one by one:

Governance

Overall software development activities and their management are addressed in the Governance section of SAMM.

Strategy & Metrics **-** Practice focuses on developing an efficient plan in addressing the key software security objectives within an organization and creating a unified strategy for application security as software assurance can be a complex process (OWASP, 2020). As a result, a software security program should be developed, maintained, and disseminated alongside a metrics-driven approach to monitor the security posture and possible improvements (OWASP, 2020).

Policy & Compliance **-** Practice should focus on addressing internal security standards and at the same time, ensuring compliance with external legal and regulatory requirements while meeting the business purpose of the organization (OWASP, 2020. The overall goal of the practice is twofold, meeting the internal and external standards while ensuring the operational evaluation of all projects (OWASP, 2020).

Education & Guidance **-** Practice focuses on providing resources and knowledge to the personnel participating in the software development process to ensure secure software (OWASP, 2020). As a result, security risks could be identified and mitigated proactively (OWASP, 2020).

Design

The design process in SAMM 2.0 deals with how the organization defines the security requirements on what and how the software should be developed.

Threat assessment **-** The threat assessment practice covers assessing risks and modeling potential threats and defining requirements, both on the software being developed and suppliers involved in the project (OWASP, 2020). As a result of these activities, developers could have a better understanding of what to prioritize in regards to security when developing the software.

Security architecture **-** The Security architecture practice focuses on the security of components and technology used in the architectural design of the software. It is about the selection and evaluating the security of particular features when designing the architecture, and deciding on acceptable frameworks and tools to use (OWASP, 2020). This way, security problems caused by code-reuse and external parties could be mitigated.

Security requirements **-** The Security requirements practice focuses on setting the right security requirements on the software. This includes making sure to have structured requirements available to the developers, but also to have structured frameworks for setting the requirements which can be reused for different projects as well as for external parties involved (OWASP, 2020).

Implementation

The security practices include building secure software, securing the deployment, and managing the defects (GitHub, 2000).

Secure Build **-** With ensuring that the built software is predictable, as per the source code and external libraries are with adequate security strength, one can assure that the initial security loopholes are minimized during the software build process. If the code functions only as per the source code depicts, then the whole area of bug and exploiting the bugs is out of hand, ensuring a risk-free software. Many such cyber-attacks occur through the misuse of external libraries as well, with software dependencies stream, this risk is also reduced (OWASP, 2020).

Secure Deployment - After the code is written, it is important to make sure that only correct software artifacts are deployed. The best way to make sure is to perform multiple tests in a non-production environment until the final software code is ready to be deployed. These various tests ensure that during any test cycle, unknown vulnerabilities come out and are fixed before another test cycle starts. Deployment execution must occur via secure credentials, the proper handling of such credentials make sure that the credentials don't fall in the wrong hand (OWASP, 2020).

Defect Management - Once the code is deployed, it is necessary to make sure that software developers are in track of any vulnerabilities/defects in the code or data flow. A feedback loop is created once such defects are created, pushing the software for fixes. Constant defect tracking ensures that the zero-day timing is reduced, and feeding back the metrics of defects ensures proper handling of such defects (OWASP, 2020).

Verification

This practice aims to focus on the process of how an organization assesses, checks, and tests produced artifacts in software development.

Architecture Assessment - focuses on verifying that the security and compliance requirements are met according to the standards set to mitigate identified threats. Continuously evaluates the effectiveness of the security controls implemented within the architecture, identifies weaknesses and future improvements, which then are resubmitted back to Security Architecture practice for improvement (OWASP, 2020).

By identifying if the software meets set requirements, it mitigates and eliminates some threats before they reach the testing part of the procedure to follow the SAMM module. Specialists save time and money in a way that if some requirements are not met, the software can be moved back to the implementation step before it reaches RT and ST, and more issues are found, which would require taking a few steps back again in order to reaffirm software security.

Requirements-Driven Testing (RT) - ensures that implemented security controls operate as needed and satisfy the requirements. Uniquely it focuses on both positive and negative testing, which results in the validation of correct functioning and aims to detect design flaws and implementation bugs through misuse and abuse testing (OWASP, 2020).

Security Testing (ST) – Procedure in which automated security testing is followed by manual, expert security testing routine. Uncovering technical and business-logic weaknesses to upper management is the goal of this practice irrespective of the requirements (OWASP, 2020).

Both RT and ST focus on testing the software reliability within its efficiency and security. Without testing, not only implementation requirements would suffer, but also technical weaknesses within the application would be left uncovered, which could and would create issues in the future when flaws and bugs are discovered and exposed by the users.

Operations

The Operations phase covers every effort the organization uses to ensure the Confidentiality, Integrity, and Availability is maintained throughout the lifetime of the application. It serves as an assurance that the organization can face any disruption and changes.

Incident Management*:* in the case of a security breach caused by malicious attackers or negligence, the organization has to make best effort to detect and handle the incident by using existing log data to identify the source, and respond by assigning roles and responsibilities to a trained incident response team (OWASP, 2020).

Environment Management: to ensure a secure system, the application's environment has to be secure, and new security features are being put in place regularly. These new features are created because each feature gets obsolete or non-supportive overtime and implementing newer versions helps monitor vulnerability reports and also carry out patching across all affected systems in an orderly manner (OWASP, 2020).

This stage could help the organisation patch and harden the application's configuration when needed, and provide updates for each application.

Operational Management*:* The security of the application depends on the performance of the operational functions. It involves implementing basic data log and data protection policies, and also invalidating unused services and applications (OWASP, 2020).

This stage helps the organization put in place functions that aids data backup, retrieval and archiving.

Security Models - Comparison between SAMM and BSIMM

The two security maturity models, SAMM and BSIMM, help in resolving the software security issues. Even though both models are created to reach security maturity in software, they both take different approaches in providing guidance. BSIMM being a descriptive model, was created by conducting a study on organizations in the BSIMM community, while SAMM is a prescriptive model which was put together by volunteer exports in a community maintained by Open Web Application Security Project. BSIMM delivers how different organizations have successfully practiced creating a secure software. SAMM might lack an organizational level comparison, but it very well overcomes by providing objectives for each maturity level. The main difference between the two is that BSIMM shows what organizations actually perform while SAMM provides guidelines on what must be done to reach software security maturity. With a specific goal in mind, it makes it easier to map the software development and align development objectives with security objectives (Sebastien, 2020) (Synopsys 2020).

Step – 4 Experienced Scenario

I was working in a company that requested to create a patch management system. So basically, the requirement from the senior operations manager and the communication was as follows: *create a software that verifies the patches applied in the software*. No other details were provided, and no proper tools were defined as what to use. The deadline was limited, so there was no option of full testing of the software. The validation of software provided some false positive, and those false positive were to be done manually. Since it was an inhouse project, no such importance was given to version updates and development. However, the point to be noted was it was the only portal where other releases would be tested, thus making it a very important internal software.

Best Practises for the above mentioned Scenario using SAMM

As the software was used as a validating point for releases of all other client-oriented products/software, it could have been improved by implementing security metrics from each of the SAMM and pushing the patch management system to a refined maturity level:

Practice	Metric	Addressing scenario
Governance	Strategy & Metrics	● Define metrics which will measure the program's effectiveness and efficiency.
	Compliance management	● Identification of 3rd party compliance and organizational requirements and standards.
	Training	● An awareness training could be used prior to the design in order to ensure developers' knowledge with regard to the software security trends.
Design	Threat assessment	● Potential risks could be known at an early stage, helping in prioritising what to address and focus on in the development.
	Security architecture	● Tested and approved components, tools, and frameworks would be defined in advance. This could make the developers more confident in what would be a good way to solve different challenges in a secure way.
	Security requirements	● Clear requirements would have provided a better starting point for the developers knowing what is expected regarding the security of the application.
Implementation	Secure Build	● A proper build process could be defined to carry out coding section ● Using the external database libraries could be tested to confirm the integrity of the plugins
	Secure Deployment	● No manual work need to be done for false positive if a secure build was completed thus creating an fully automated verification process ● As the software accesses all the other software the company produces, it acts as an integral to perform auditing and access control on this software
	Security Architecture	● Even with the tracted false positive defects, the software did not continue to be developed, the software had to go through feedback process aiding in defect management
Verification	Architecture Assessment	● Allows policies and security requirements to comply with the ones set at the start of the project eliminating flaws in the security controls within the architecture.
	Requirements - Driven Testing	● Verifying correct implementations of security requirements would eliminate some of the flaws and bugs. ● Denial of Service attack is a good RT example
	Security Testing	● Automated and manual security testing performed would complete in-depth application testing and reveal discoverable weaknesses
Operations	Incident Management	● In the case of any security breach, An Automated Log Evaluation system can help detect breaches and help identify the source or cause of the incidents.
	Environment Management	● Updates should be given more importance since it's a patching system. Regular patching and hardening of its components should be considered.
	Operational Management	● A data log and data protection policy should be created, reviewed, and practiced.

Conclusion

Security models provide a good framework that can ensure security issues are addressed, or even avoided as early as possible. However, they can be quite extensive and might require a lot of work to be implemented. After a group discussion, we landed the following points:

- Security models need to be implemented in a way suitable for the specific organization or project in order to be efficient.
- BSIMM is using real-life cases as a starting point, perhaps making it more easily applicable, but as no organizations are the same, there might be issues when applying these methods to new organizations.
- In such cases, SAMM seems to be "winning" as it provides guidelines from the security experts which are tailored for each organization specifically.
- As the models require a holistic approach due to the software development process, an organization should ensure a high-security awareness level across the employees involved in the project.
- A limit needs to be defined as how back the dependencies check must go in order to reduce the extensiveness.
- Market competition seems to play a huge role in aligning with secure software practices

As SAMM and BSIMM models emphasize, it is not necessary that we must follow all the security practices, but it is good if we can keep up with just one. These models act as guidelines rather than strict rules, which aid in creating friendly policies for each maturity level for any software field. It would be good to see organizations try as much as they can to align with these policies and create secure software for all of us.

References

GitHub (2020), *OWASP/SAMM,* Retrieved 2020-09-06, from https://github.com/OWASP/samm

McGraw, G., Migues, S. and West, J. (2018). *Building Security in Maturity Model (BSIMM) Version 9*. USA; Retrieved 2020-09-06, from https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf

OWASP (2020). *SAMM Model.* Retrieved 2020-09-04, from https://owaspsamm.org/model/

Sebastien, D. (2020). *OWASP SAMM v2.0 Released.* Retrieved 2020-09-06, from https://owasp.org/2020/02/11/SAMM-v2.html

Synopsys (2020). *What is BSIMM and How Does It Work?* , Retrieved 2020-09-06 from https://www.synopsys.com/glossary/what-is-bsimm.html