DIRECTORATUL NAȚIONAL
DE SECURITATE CIBERNETICĂ

ISACA

Practical approaches to building secure applications

Author: Sabina-Daniela Axinte

Companies that ignore security issues that their products and processes are exposed to are, in fact, prone to failing at providing their customers with quality services, or suffering significant financial losses in case of data breaches. Adopting a proactive approach when building software products is as compelling as developing the customers' desired functionalities. Even if the effort to implement and maintain the proper processes might be substantial, the return on investment will prove its value in time.

1. Organizational processes that ensure product safety

Similar to regular bugs, security vulnerabilities discovered at a more advanced stage of the software development lifecycle are up to 100 times more expensive [1]. Therefore, it is of utmost importance to find a **software security strategy** (SSI) that harmonizes with your organization and your employees. In order to achieve this, the following steps must be followed: build, measure, verify, improve, manage. The executive board should take under consideration that investing solely in application security testing is insufficient. Setting proper and tangible objectives, collecting metrics and measurements, implementing a framework containing proper procedures and controls, and documenting development lifecycle enhancements and checkpoints are also important aspects.

Complementary to the above, the organization has to make sure that, during the implementation stage, all **employees** are **involved** by keeping them informed about the benefits, the mandatory changes, and by asking for their anonymous feedback. Your employees are playing one of, if not the most important roles in achieving the desired security level

(Security = Commitment * (People+Tools+ +Processes^2) [2]). After you establish the necessary procedures that your employees have to follow based on their current position, ensure you are also offering them an updated, well-organized **security training,** so they can fully understand how to integrate data and asset protection into their daily routine.

Identify the necessary tools to **automate** all **security-related routine tasks**, even if is testing or monitoring servers' configuration (firewall changes, device configurations etc.). **Systematic software updates** should also be automated since many attacks are based on vulnerabilities specific to an application version. If you are using open-source solutions that are particularly harder to keep track of, you can use a Software Composition Analysis (SCA) tool. Moreover, you can subscribe to CVE alerts for your list of vendors or products and be notified as soon as a vulnerability is published, to start preparing for the update to the new version.

Create an **incident response plan** for the most undesirable scenarios. It will assist you in minimizing the impact in case of an attack. In addition, you can also **define and monitor key metrics** that are relevant to your organization and will point out any abnormalities that might lead to an attack.

2. Risk management for software products

Regardless if your software projects are personal experiments, internal or available for a large number of customers, targeting a specific market segment, the general public or governmental, a broad range of serious potential problems can surface, especially from the security perspective. The ISO 31000 standard provides proper guidelines, principles and processes in order to protect your organization by managing risks, making the right informed decisions, setting and achieving objectives. You can decide to simply implement the recommendations stated in it, or even to continue with obtaining the certification after the implementation is successful.

2.1 Risk identification

For security risks identification related to software products, OWASP Top 10 [3] is the recommended guideline. OWASP (Open Web Application Security Project) is a nonprofit foundation that aims to help developers, product owners and quality engineers alike, in order to build robust software solutions in a continuously evolving digital world.

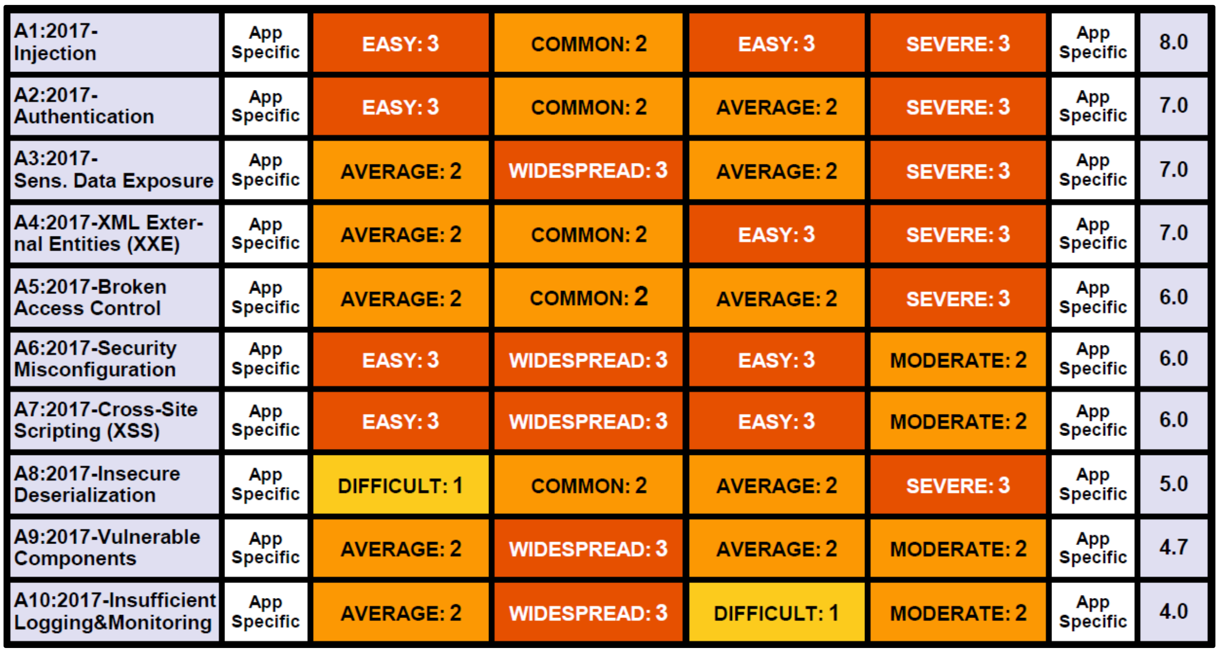
The documentation describes most common vulnerabilities (Figure 1) with the highest impact if exploited. Additionally, it provides exploitation examples and practical methods of minimizing the exposure of your products and elevating the confidence level for both your production team and your customers.

It is of the utmost importance that all development team members to know the aforementioned vulnerabilities, their impact, how to mitigate their associated risks and how to validate that the slated changes will not lead to the emergence of such an exposure. Based on the nature of your product (web-based, APIs, mobile), you should be able to identify the risks associated with the changes (features, technical debt, or bug fixes) when you are deciding what should be included in the next iteration.

2.2 Risk analysis

Ideally, the assigned developer, together with the team/technical leader, should identify all the risks associated with the change under discussion, propose mitigation solutions and reduce the exposure down to a low level. This process will be further detailed in the following sections. All mitigation tasks have to be linked with the originating issue and estimated according with the effort involved.

Furthermore, the developer has to set the impact and the probability for each identified risk. The impact refers to an estimation of the potential losses in case the stated vulnerability will be exploited [4]. This has to include financial costs, reputation loss, downtime and information disclosure. To make sure this component has the correct value, a combination between the risk level stated in OWASP Top 10 (Figure 1) and the importance of the component where the originating change has to take place is the recommended approach.



A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML Exter-nal Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

Figure 1. OWASP Top 10 vulnerabilities and their associated risk levels [3]

The probability of occurrence represents the chance for a risk to arise in the production environment. These two risk parameters are either set on a simple scale, with values corresponding to low, medium, high, or a more complex one with intermediate values. Having this information, we can proceed with computing the exposure rate, based on the risk matrix (Figure 2).

$$Risk = Probability\ (of\ the\ event) * Impact$$

		Impact				
		Negligible 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Probability of occurrence	5 Almost certain	Moderate 5	High 10	Extreme 15	Extreme 20	Extreme 25
	4 Likely	Moderate 4	High 8	High 12	Extreme 16	Extreme 20
	3 Possible	Low 3	Moderate 6	High 9	High 12	Extreme 15
	2 Unlikely	Low 2	Moderate 4	Moderate 6	High 8	High 10
	1 Rare	Low 1	Low 2	Low 3	Moderate 4	Moderate 5

Figure 2. Risk matrix example

2.3 Risk evaluation

All the above steps are mandatory for this next stage in which the authority board (Change Advisory Board - CAB) will decide whether the changes requested by market, sales, support or R&D teams will be selected for development and included in the next iteration based on their assigned priorities. Given the exposure rate and the organizationally-set acceptable risk threshold, the authority board can decide one of the following treatments: mitigation, acceptance, avoidance or transfer.

Acceptance is recommended for risk estimated to produce a low exposure, but, in exceptional conditions, risks above this level can be deployed to production. In this case the residual risk has same value as previously set. **Avoidance** refers to the option in which it is decided that the risk is either too high, or the development effort associated with the change and the mitigation task is unacceptable. **Transfer** is only possible in those specific scenarios in which the risk under discussion originates from a different component, internal, open-source or enterprise solutions. This treatment is recommended only when we know that the other component is still maintained, and specifically if it is internal. For the rest we have no guarantee that the expected change will not impact our iteration delivery since the related changes cannot be released into production until the risk is mitigated by the component owner.

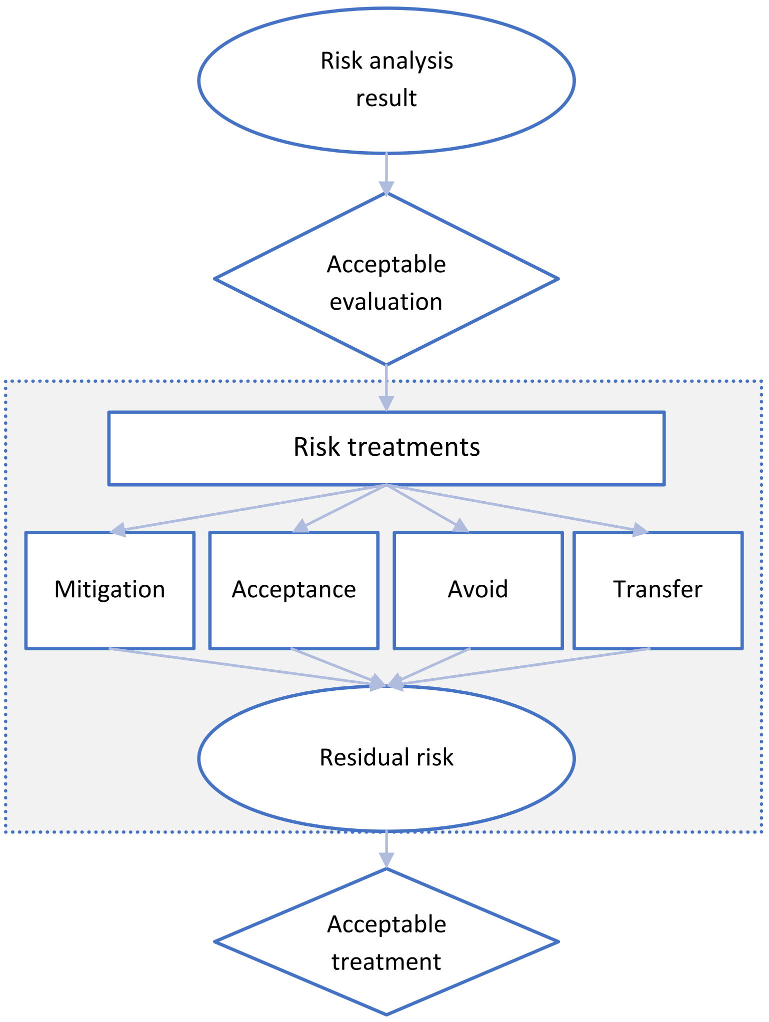


Figure 3. Risk evaluation workflow [4]

Mitigation is the scenario in which the change associated with the risk is selected for the next iteration and the development effort is acceptable. All the mitigation tasks have to be included in the same development cycle and a Quality Assurance Engineer with expertise in security testing must validate the changes. Mitigations can consist of testing tasks, code reviews, unit or integration tests. If the testing outcome is negative, meaning that the initial threat is detected, the residual probability on the risk issue should be increased, and the originating task should be reopened. If the threat cannot be reproduced, residual probability will be decreased, and the risk task will be closed. If the residual exposure is above low level, the CAB will have to approve the release.

2.4 Tools

Risk management can either be manual, partially automated or fully automated. Since it is a complex process and it involves various stakeholders, it is recommended to use automated tools for both tracking and detecting the risks.

For tracking, an extension to your preferred bug tracking solution is the best approach. If you are using Jira, **Risk Register** plugin is a useful solution that facilitates the linking between the original development task, its associated risks and the mitigation tasks with conditions like ‘is caused by’, ‘is treated by’, and allows the setting of impact and probability values. Additionally, it provides a report with all active risks for your project, their original and residual exposure and the proposed mitigation tasks.

If your strategy leans more toward the proactive attitude, then you are looking for an automated coding review platform. **Codacy** [5] is a solution that will help you with OWASP Top 10 vulnerabilities prevention, code standardization, and overview of the code coverage, while offering a tailored setup and pipeline integration. It is suitable for over 40 languages or frameworks, supports integration with GitHub, Slack, Jira, Git, GitLab, Bitbucket, and it can be either self-hosted or available in the cloud. Another viable option is **SonarQube**, an automatic code review tool that facilitates bug and vulnerability detection [6], that, if integrated in your CI/CD cycle, will continuously inspect all your project branches and pull requests. Its owners propose a package of solutions, including an IDE (Integrated Development

Environment), integration with your chosen bug tracking solution for automatically raising and assigning bugs and vulnerabilities and various webhooks for notification purposes.

If configured correctly, all the above solutions will guarantee that your shipped product is built in a robust and secure manner, gain customers' trust, minimize the abandon rate and heighten the development team's morale and efficacy.

3. Best practices for secure products

In addition to the organizational procedure implementation, the following section states the most important best practices that are considered to provide your organization and, subsequently, your products an extra layer of protection from a security perspective.

Even if all the aforementioned practices are in place, it is highly recommended to acquire an **external security audit** in order to receive unbiased and relevant feedback from security experts. Auditors are people with solid experience that know where to search for vulnerabilities in both expected and hidden areas. They usually provide information about latest exploits that are not common knowledge at that point, but may affect your products. A **penetration test** is desirable, especially for components that have the highest exposure (e.g. public APIs or internal ones that are handling user management). Their final report will include a list of encountered problems, their impact if found and exploited with malicious intent, steps and necessary tools to reproduce and solutions for addressing or eradicating them.

Regardless of your application type, you should also focus your efforts on adopting **logging** for all your solutions. Aside from its benefits when a bug arises, proper logging across all applications can surface an increased number of breaching attempts and will proactively compel you to strengthen your security prevention mechanisms. Both successful and failed calls/actions must be logged with the appropriate flag (verbose, error, warning, information). You can choose to build your own solution if you have the resources, or you can integrate an off-shelf solution (Kibana, Papertrail, Prometheus, Tableau, Grafana, Logstash, Loggly, Splunk etc.) [7].

All the effort you channeled into the application level will prove delusive if there is no **real-time security monitoring and protection** at the infrastructure level. In order to block online threats in real-time, you might want to use Runtime Applications Self-Protection (RASP) solutions. RASP continuously oversees both web and non-web applications' behavior and protects against malicious input, being a more modern and adaptable alternative to Web Application Firewall (WAF). Some examples are Fortify, Sqreen, OpenRASP, Imperva, JSDefender etc.) [8].

Particularly if you process personal data based on GDPR classification, you might want to use **encryption** for all data states: in transit, in use or at rest. In addition to exclusively using HTTPS and HSTS protocols while transferring data, you must protect the database entries with advanced encryption mechanisms in order to prevent unintentional information disclosure or data breaches.

Conclusions

Software Quality Assurance is the result of advanced knowledge related to secure software engineering practices, use of proper tools that monitor and automate daily tasks, constant heroic employee effort, checklists, tailored procedures and controls, and involved branch managers and the executive board.

Achieving a security culture is a team effort that has to be backed up by a fitting software security strategy, incident response plans for emergency scenarios, and triggers based on key metrics and logging activity that will detect the abnormalities related to an attack.

Whether a certain security level is demanded by your partners and customers, or you simply chose to provide not only functional, but high-quality services, you should accommodate risk management process in your daily activities to identify and prevent any vulnerabilities that might emerge in your projects, products and organization. In addition to the proposed solutions, the ISO 27001 standard offers additional techniques and processes at the organizational level, that will expand your systems' resilience when it comes to security management and business continuity.

References

[1] NIST, "The Economic Impact of Inadequate Infrastructure for Software Testing" [Online]. Available: www.nist.gov/system/files/documents/director/planning/report02-3.pdf. [Accessed 15 05 2021].

[2] OWASP, "How to start a software security," [Online]. Available: www.owasp.org/www-pdf-archive/Owaspday2Morana.pdf. [Accessed 15 05 2021].

[3] OWASP, "Top Ten Web Application Security Risks," www.owasp.org/www-project-top-te, 2017.

[4] "ISO 31000 - Risk management," 2018.

[5] "Codacy," [Online]. Available: www.codacy.com/product. [Accessed 21 05 2021].

[6] "SonarQube," [Online]. Available: www.sonarqube.org/. [Accessed 21 05 2021].

[7] "Best monitoring tools," [Online]. Available: www.stackshare.io/monitoring-tools. [Accessed 23 05 2021].

[8] "Best Runtime Application Self-Protection (RASP) Software," [Online]. Available: www.g2.com/categories/runtime-application-self-protection-rasp. [Accessed 23 05 2021].