Software Vulnerability Remediation Management Process

Agile7

Document Version: v1.1

Date: October 23, 2024

Author: Max Meinhardt

Agile7

maxm@agile7.com

www.agile7.com

Introduction

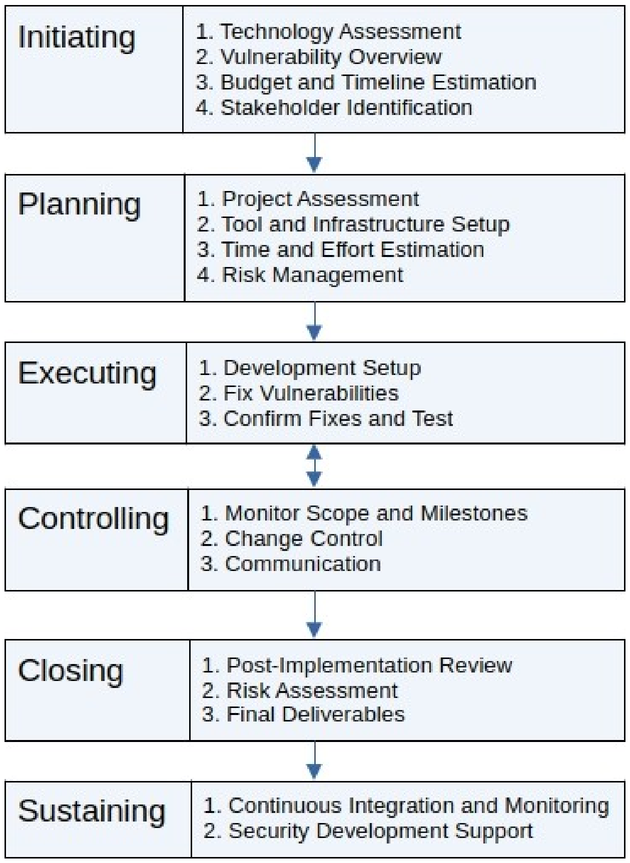
The goal of an effective software vulnerability remediation management process is to ensure that risks are identified, communicated, and mitigated while maintaining a clear understanding of the project's time, scope, and cost. A thorough understanding of each remediation phase helps keep the project on track and ensures that vulnerabilities are effectively addressed.

This document outlines a baseline project management process for remediating software security vulnerabilities, mitigating risks, and safeguarding your software. It can serve as a starting point or supplementary material when developing a project plan tailored to your specific needs.

Overview

The process of remediating software vulnerabilities has been divided into the five phases of the project management lifecycle, with an additional phase called "Sustaining," as shown in Figure 1.

Each of these stages involves tasks that guide a project from inception through to ongoing maintenance, helping to ensure an organized and thorough approach to vulnerability remediation.


Initiating
1. Technology Assessment
2. Vulnerability Overview
3. Budget and Timeline Estimation
4. Stakeholder Identification
Planning
1. Project Assessment
2. Tool and Infrastructure Setup
3. Time and Effort Estimation
4. Risk Management
Executing
1. Development Setup
2. Fix Vulnerabilities
3. Confirm Fixes and Test
Controlling
1. Monitor Scope and Milestones
2. Change Control
3. Communication
Closing
1. Post-Implementation Review
2. Risk Assessment
3. Final Deliverables
Sustaining
1. Continuous Integration and Monitoring
2. Security Development Support
Figure 1: Software Vulnerability Remediation Management Process

The Process

In the initial conversation we will communicate with you to determine the project's purpose and overall scope in order to ensure that we both agree that our consulting services align with your needs and you are ready to proceed to the Initiating stage below.

Phase 1 - Initiating

In this consultation, we give a candid and objective assessment of your remediation project. After this phase is completed, we should have the information that is required for a project charter. This should include a high-level understanding of the known constraints and assumptions, preliminary project milestones, a budget, and possibly a cost estimate range based on a top-down understanding of the project.

In general, the topics to discuss during this meeting are as follows:

1. The application's technologies, their age, build environment, and any known issues such as missing source code and libraries, deprecated technologies, software architecture problems, etc.
2. Overview of must-fix and other vulnerabilities
3. Project approach (eg. agile or waterfall)
4. Estimated project budget
5. Desired project end date
6. Identify stakeholders, including crisis-handling contacts
7. Top-down preliminary assessment (optional). This is the first iteration of the task of determining which known vulnerabilities we should fix first and whether those fixes should be grouped, tested, and deployed in specific phases. This strategy usually evolves as a deeper understanding of the vulnerabilities are developed and your business goals change.

Phase 2 - Planning

At this next stage, we will need to obtain more information to build on the findings from the Initiating consultation by estimating a more accurate time, scope, and cost. This will require identifying vulnerability risks and their associated business risks (if applicable) as well as project risks/difficulties and mitigation of all of these risk types.

During the Planning phase, we will address the following:

1. **Scope and Logistics**
   1. **Bottom-up project assessment -** In complex projects where we create a Scope Statement and Work Breakdown structure, we will need to analyze your project from the bottom-up by assessing the level of effort needed to remediate each of your application's vulnerabilities by reviewing your code and application's system environment. While doing so, it is good to do a preliminary code review to look for additional vulnerabilities (architectural issues, unsafe 3rd-party dependencies, etc).
   2. **Tools, Infrastructure, and Logistics**
      1. Preferred project management, code management, bug tracking system, and vulnerability management tool (eg. Fortify On Demand, Burp Suite Enterprise, Defect Dojo, BlackDuck(CodeDx), etc) if applicable.
      2. VPN, security and access details
      3. Hosting/staging server information
      4. Existing system learning curve
2. **Schedule**

1. **Time and Effort -** In the process of identifying a schedule, we need to understand the amount of time and effort that will be involved to resolve as well as test the vulnerabilities. We may find that some issues can be grouped and are easy to fix yet others may sometimes require system modifications or significant underlying architectural changes. We will work with you to closely identify their risks in order to determine the best strategy to fully resolve or even mitigate them with less effort.

   We will also need to address work hours, testing hours, time-zone overlap, local holidays, and leave.

2. **Identifying false positives** - If a list of vulnerabilities was taken from a code scanner/static analysis tool, there may be thousands of reported issues with some being false positives and others being bugs which are not security-related but should be resolved anyway. The standard process is to go through each of these issues and look at the associated code in order to determine their level of effort or identify them as false positives. Documenting (via a vulnerability management tool or within a Fortify .fpr file for example) the reason for labeling a false positive as well as details for complex fixes is also a standard process.
3. **Complex vulnerabilities** - Vulnerabilities in business logic may require assistance within your organization to determine a level of effort. Software framework updates, authentication and session management architecture, and interaction with other systems can sometimes be time-consuming for various reasons. However, there may be ways to mitigate these risks and this would be discussed with you.
4. **Finding the fixes** - The more vulnerability details that we have, the easier it will be for us to find where the problems reside in the code. For example, outputs from code scanning tools or manual code reviews should give information about the specific location in the code that they occur. This may not be the exact location where the cause or fix for the problems occur, but its close relationship to root-cause will help identify and fix them.

3. **Cost**
   1. **Bottom-up approach** - With the Work Breakdown Structure (WBS), time, and risk breakdowns that we defined earlier, we will work with you to determine which activities, if any, should be removed or delayed in order to fall within your budget and time constraints. For example, not fixing low-priority vulnerabilities that were identified from a static analysis or via manual testing.
   2. **Complex vulnerabilities** - Issues that are complex to fix, such as large software framework updates or those utilizing interaction with other systems or authentication and session management architecture, may have a high time and risk quotient to justify a

dedicated full-lifecycle development effort. If these are encountered, we will work with you for a course of action.

4. **Risk**
    1. **Build environment risks -** We will work on determining if there will be any problems creating a working application from your source code. If your application is complex and has not been maintained recently, we will need to quickly identify and assess any serious problems such as missing dependencies, etc.
       Ideally, major risks for this scenario should be identified as soon as possible and preferably in the Initiating phase of this project. If you have developers currently maintaining the application properly, these risks should be minimal or absent.
    2. **Vulnerability risks** - We will work with you to align our knowledge of each vulnerability risk to your knowledge of the associated business risks if applicable. You may have already identified some, but we may identify others from our past experience with application security and from our preliminary source code review.
5. **Communication** - We will need to finalize the modes of status reporting (email, phone, Skype, GoToMeeting, etc), frequency of status reporting (weekly/on-need), feedback cycle, ongoing collaboration tools (Trello, etc), timesheet maintenance option, and feedback response time.
6. **Quality**
    1. **Testing method** - After identifying the risks of the vulnerabilities (they may be grouped together in many cases), we will identify the best approach to fix and test them. For example, if some vulnerabilities were identified from a static analysis tool, we would need to re-run it to validate fixes in this context. If others were found in a manual architectural or code review, fixes would of course need to be reviewed and tested as well. In all situations, testing via various tools, techniques (static, dynamic) and resources (integration testing by us and/or your QA team) will be used depending on what is currently in place and possibly including using other tools that we will discuss.
    2. **Security compliance** - If you have a security team, we can work closely with them in order to help fulfill your security compliance requirement (eg ATO Authority To Operate) for this project. If not, then we can fill this gap with regards to application vulnerability mitigation.
7. **Software Development Process**
    1. **Waterfall method** - We recommend this method for small projects and will need a specific date to remediate as many vulnerabilities as possible from the scope, risk, and time assessment that was discussed with you earlier. Your most critical vulnerabilities that are easiest to fix will be remediated first.
    2. **Agile method**
        1. Essentially, the same tasks that are used in the Waterfall method above should apply for every sprint. This will allow you to more easily re-prioritize if needed.

2. We recommend placing the vulnerabilities and associated tasks in a defect tracking system where their progress can be updated, monitored, and assigned as needed for each sprint.
3. If you have incorporated the running of application security testing tools into your continuous integration (CI) process and want to offload or improve the remediation of the found issues, we can either show your developers how to mitigate them as needed or let us do it for you by working in parallel. We can talk with you about how to do this while minimizing disruption and maintaining or improving process flow.

Phase 3 - Executing

After the project's scope, schedule, and cost has been determined, we can go through the following steps to finalize the process of fixing, testing, and deploying the vulnerability fixes to your production environment.

1. **Setup development environment and build working application**
    1. **Source control** - We will need access to your source code. If it is in a file-system directory, then it will need to be moved into a source code repository. If the project is complex, then we will need access to it during the Planning stage of this project in order to bring the code into an IDE and more easily navigate through it in order to do a preliminary code review and bottom-up assessment.
    2. **Database** - We will need a copy of your database and there are various ways to get this.
    3. **Issue tracking system** - If you use an issue tracking software (eg. Jira) or vulnerability management tool that interfaces with one, we will need access to both so that we can monitor and maintain the vulnerability backlog.
    4. **Syncing from production** - In some cases, database schemas, certain system configurations, and/or software patches between production and test/staging environments may be different because of improper synchronization between the two. These environments should be aligned before starting the development process.
2. **Fix vulnerabilities**
    1. **Technical vulnerabilities** - If the vulnerabilities were discovered from a code review or static analysis tool such as Fortify, then their approximate locations in the source code should be quickly identified in some cases and require further investigation in others. Identification of most false positives and issue groupings should have been identified in this project's Planning phase.
    2. **Business Logic vulnerabilities** - These vulnerabilities may have been found from a top-down (testing as a user, dynamic/penetration testing) or bottom-up approach (code-scanning/static analysis, architectural, or code-review). Some of their fixes may be

simple and others may require significant changes to the underlying architecture or need coordination of various stakeholders in your business, such as for authentication and other issues that would modify the user's experience.

3. **Confirm the fixes and perform general QA** - After vulnerability remediation has been done, proper testing is necessary in order to determine that they have not broken previously working functionality.
    1. **Testing method** - Many will require a re-run of the scanning/static analysis tool in order to verify if the fix addressed the tool's finding. In some instances, these tools may still tag the same vulnerability after it has been fixed, but since the code changes are not identified by the tool's algorithm/rule-set in these cases, it will be labeled as such once testing has confirmed this.
       Others will require a repeat of the manual test that was used to originally reproduce the problem.
    2. **Full or partial regression** - How and in what environment this is done will be determined by your existing process and resources as well as the scope of the changes.
4. **Deploy the updated application to Production** - Once testing is completed in your staging environment, we will work with you to push your updated application to your production environment as needed for each sprint. The following steps should be done to increase the probability of a successful deployment.
    1. **Write an installation and back-out procedure** - This document should be included as part of the release package documentation. Deployment risk contingencies should be included within the instructions. For example, "if problem X happens, then complete this other step."
    2. **Working with DevOps team** - If you have a DevOps team with exclusive access to Production, we will need to work with them to enable your maintenance page, deploy the application, and stay present while we smoke-test your application to make sure that the deployment is successful and does not require a backout. We may need to monitor the application logs during smoke-testing.
    3. **Production deployment failures** - There are many reasons that applications can fail to deploy and run properly (or at all) when moving from a staging environment to production. For instance, there may be production-specific services or databases containing differences that exercise parts of the application's functionality which weren't able to be previously tested in the staging environment. Also, there may be different authentication servers, or perhaps the configuration of your staging environment has not been synced consistently to and from production during previous releases or system updates.
       These risks and others should be addressed during the Planning stage of this project and during testing of the application before deployment to production. Doing so is crucial in

order to ensure that the deployment is as swift and trouble-free as possible in order to minimize the amount of time that your website is down as well as efficiently utilize your devops resources during this period.

    4. **Speed and efficiency** - An effortless and problem-free deployment to Production is important to us. We have guided hundreds of releases and know how to properly mitigate risks during this process. Keeping your devops team waiting during troubleshooting or having them run the backout procedure is not time well spent, but a clean problem-free deployment is.

5. **Production post-deployment testing** - Depending on your needs and the project's scope, we can work with you to determine the breadth of this action.

Phase 4 - Controlling

As we progress through the project, we will communicate the status of our work items (as outlined in the Planning-Communication section) and collaborate with you to monitor the current scope and milestones. Deep analysis may reveal hidden complexities, additional vulnerabilities, or the need to apply risk contingencies as the project advances. Scope creep may occur in such instances, but by using your change-control process, continuously monitoring and assessing vulnerabilities and project risks, and addressing the most critical and quickly solvable issues first, we can significantly reduce software-related business risks early on and re-assess less critical tasks later.

Phase 5 - Closing

Once the project has concluded, we will work with you to initiate the administrative closure process, ensuring that all final deliverables are provided (e.g., additional information we gathered about your build environment, architecture, and application vulnerabilities). We will also conduct a post-implementation review to assess the project's success, lessons learned, and any remaining vulnerabilities. If applicable, we will then discuss alignment with our sustainment services.

Phase 6 - Sustaining

You can rely on us to maintain the security of your software, whether through application security development within your team in a continuous integration environment, periodic code reviews and static analysis to address new vulnerabilities, or other tailored approaches to application vulnerability remediation. We are committed to solving these problems consistently and reliably.

Please contact us at **info@agile7.com**.

About Agile7

Agile7 provides software development services for application vulnerability remediation and software energy efficiency. Learn more at **www.agile7.com**.

Document Revision History

Author	Version	Date	Description
Max Meinhardt	1.0	October 16, 2024	Initial version
Max Meinhardt	1.1	October 23, 2024	Change company logo and add document version and subtitle in title page. Add Version column to Document Revision History.

Author Biography

About Max Meinhardt

Max is the founder of Agile7 and he has over three decades of industry experience in software engineering and systems deployment leading the architecture and implementation of enterprise Web applications and carrier-class networking and telecommunications equipment firmware. He holds a BS in Computer Engineering Technology from Rochester Institute of Technology and an MBA from Thunderbird School of Global Management.