National Defense-ISAC

Remediation Workflow Automation

March 27, 2021

Authors:
Paul Keim
Waldemar Pabon
Joe Vega
Andrew Zuehlke

Reviewers:
Mike Heim
Will Jimenez
Renee Stegman

About the Authors

Dr. Waldemar Pabon, ND-ISAC Lead & Senior Contributor

Dr. Waldemar Pabon is a Cyber Security Architect with over 25 years of experience in software engineering. Dr. Pabon leads the Application Security Working Group, Software Security Automation and the COTS Software Assessments Subgroups at the ND-ISAC. Under his leadership, the Software Security Automation Working Group has published three white papers. The papers provide ND-ISAC members and the industry with a roadmap on how to adopt application security best practices while leveraging automation as a catalyst to achieve efficiencies. Dr. Pabon has a Doctor of Science in Cybersecurity degree from Capitol Technology University.

Paul Keim, ND-ISAC Senior Contributor

Paul Keim is a Senior Security Architect. With ten years of experience in the information security field, with half of that leading the Application Security team, Paul's focus is constantly on how to automate his processes to reduce the workload for his team. A member of ND-ISAC for three years, Paul started in the Application Security subgroup looking to learn more about what other companies are doing to automate their workloads, and now looks to provide his expertise on what's worked and hasn't worked for him over the years.

Joe Vega, ND-ISAC Senior Contributor

Joe Vega is a Cyber Security Engineer for the last four years. Receiving Bachelor’s degree in both Cybersecurity (Information Assurance) and Information Systems from the University of Texas at San Antonio (UTSA), while also obtaining his CISSP. Joe has over nine years of experience within the cybersecurity industry, with five years specific to the Application Security field. In his current role, Joe is focusing on Application Security and DevOps workflows. Joe has been a member of ND-ISAC for two years, contributing to two papers in the AppSec space. Joe continues to actively participate in the ND-ISAC community offering his expertise to his peers in the field.

Andrew Zuehlke, ND-ISAC Contributor

Andrew Zuehlke graduated from Appalachian State University in 2017 with a Bachelor of Science Degree in Computer Science and Computational Mathematics. Andrew has served as the lead administrator of SIEM and EDR solutions. In early 2020, Andrew moved to his current role as Cybersecurity Liaison, primarily supporting Research & Development and Information Technology. Since joining the ND-ISAC in December 2020, Andrew has become a member of the Application Security working group as well as the Cloud Security and Architecture Working Group. As his first major contribution, Andrew coauthored the Application Security Working Group's third white paper, Remediation Workflow Automation. Andrew continues to contribute in the ND-ISAC community via working groups and peer collaboration.

Executive Summary

The implementation of security controls in the Software Development Lifecycle (SDLC) provides a mechanism for organizations to proactively detect security vulnerabilities before they impact the business. This mechanism allows teams to effectively plan for remediation, ultimately supporting a significant improvement to the process required when dealing with security vulnerabilities. Security controls report security vulnerabilities and their respective severity levels. Provided with reports from these controls, development teams must decide how to handle the findings. With hundreds–sometimes thousands–of findings, determining which threat vectors should be tackled first, how to validate the associated fix and the analysis of the root cause become key aspects of the overall effort. Having a standard remediation process supported by automation can:

- help development teams understand how to handle multiple findings identified by security controls
- provide direction on the prioritization of remediation efforts
- provide guidance on how to prevent security findings from repeating themselves in future development efforts

To address these concerns, organizations need to adopt a remediation workflow supported by automation. This paper provides a three-phase process to enforce a comprehensive remediation. The three phases needed are:

- Identify and Confirm Vulnerabilities on an Ongoing Basis

- Assess, Prioritize, Remediate and Validate
- Analyze Vulnerabilities to Identify Root Causes

These three phases provide key elements: identification of security vulnerabilities, an understanding on how to manage them, the use of a framework to help facilitate the prioritization, the embedding of remediation plans, and help prevent those same vulnerabilities from affecting future projects. Each of the stages presented in the paper follow a technology-agnostic approach. A standard set of stages, tools, and automation actions will help teams minimize the risk associated with vulnerabilities and improve remediation efficiency throughout the SDLC.

An important takeaway presented in the paper is the need to use context in conjunction with the severity score provided by the security controls when determining the priority assigned to a specific security finding. Because security controls can identify a significant number of vulnerabilities, and teams often face human resource constraints, knowing more about the context of the vulnerabilities enables developers to prioritize and address the important findings first. A prioritization framework is provided in Appendix A of the paper to help organizations rank security vulnerabilities identified by automated scans and determine the order of execution in the remediation workflow. The authors recognize that organizations do not have the same priorities; however, using a prioritization framework allows teams to assign a higher weight to those context concerns that are considered more important.

Establishing a process for remediation will help organizations create efficiencies in the security vulnerability management process while spending their resources in a more meaningful way. It becomes important to understand how to manage the significant number of security findings detected in the different stages of the SDLC. The standard set of stages, tools, automation activities and prioritization strategies presented in the white paper can be used by organizations lacking a standard remediation workflow as well as those looking to mature their implementation.

Table of Content

About the Authors **2**

Executive Summary **4**

Introduction **8**

Objective 8

Audience 9

Structure of the paper 10

Remediation Workflow Automation **11**

Remediation Phases 11

Identify and Confirm Vulnerabilities on an Ongoing Basis 12

Assess, Prioritize, Remediate and Validate 14

Analyze Vulnerabilities to Identify Their Root Causes 26

Conclusion **28**

Appendix A - Priority Ranking Framework **30**

Introduction

Objective

The natural result of adding security controls in the SDLC is the identification of threat vectors in the software supply chain. The first step is identifying vulnerabilities. To spend time wisely and efficiently when processing security vulnerabilities identified by security controls, organizations need to establish a formal remediation workflow. Such an approach requires the implementation of tasks to help ensure vulnerabilities are identified and remediated as quickly as possible, reducing the window of opportunity for attackers while minimizing security and compliance risks.

Recognizing the challenges associated with human capital and budget constraints, it becomes imperative for organizations to leverage automation activities during the identification, prioritization, and remediation of security vulnerabilities. Leveraging automation will help organizations accelerate their ability to be efficient in their pursuit of secure software. Without the use of a standard remediation process, teams (often unintentionally) implement processes lacking the necessary checks and balances. This approach can increase the risk of development teams overlooking important steps required to maintain security as a key component of their delivery process.

Handling a remediation effort requires three basic concerns: identify the vulnerabilities, assess to prioritize the findings and remediation needed, and analyze root causes to avoid the same issues in future deployments. The development of any standard process to handle the remediation must comply with those three basic principles. Using this information, the remediation workflow proposed in this white paper includes three basic steps to handle those concerns. Figure 1 provides a visual representation of the three steps.

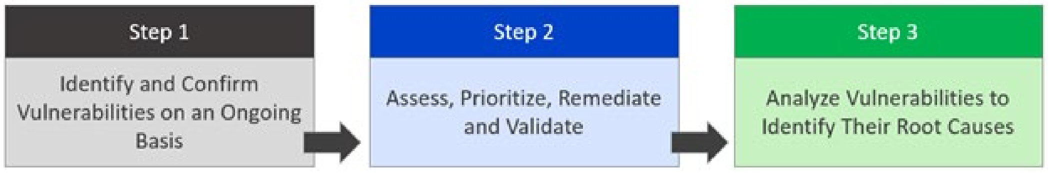
Step 1
Identify and Confirm Vulnerabilities on an Ongoing Basis
Step 2
Assess, Prioritize, Remediate and Validate
Step 3
Analyze Vulnerabilities to Identify Their Root Causes

Figure 1 Standard Remediation Workflow

The standard process defined in the white paper provides a management tool with the corresponding remediation steps used by teams when addressing security vulnerabilities in the software delivery process. Having a standard remediation process helps avoid hampering the ability of organizations to proactively minimize risk and increase productivity. Each phase in the remediation workflow is explained while stressing the different automation efforts available when applicable. The importance of root cause analysis is also presented, which can provide additional context and help organizations avoid repeating the same mistakes in future development efforts.

Audience

The audience for this white paper includes security engineers, software engineers and software architects responsible for establishing the rules governing remediation efforts whenever security vulnerabilities are detected by security controls in the SDLC.

Structure of the paper

This paper introduces the three stages of a standard remediation workflow. Each of the three stages in the workflow are explained, along with automation strategies teams can implement to provision agility to their remediation efforts. Finally, a prioritization ranking framework is presented in Appendix A of the paper to help organizations with the application of context against the security vulnerabilities identified. The use of context helps facilitate the decision-making process when prioritization of resources to complete a remediation is a concern.

Remediation Workflow Automation

Remediation Phases

The standard remediation process can be broken down into three stages, each with targeted focus areas. By organizing and aligning remediation efforts within each step, teams can more effectively and efficiently address issues during the SDLC, minimizing the time a vulnerability may be exploitable. Each stage in the remediation process builds on the previous phase, and all three stages must be addressed as part of a standard process.

The initial step in the remediation process emphasizes the identification and confirmation of vulnerabilities on an ongoing basis; during this phase, teams should focus on properly configuring systems to effectively identify vulnerabilities and limit false positives, ultimately reducing wasted time in the follow-on phases. The second step shifts focus to assessing, prioritizing, remediating, and validating each vulnerability identified during the first phase; this ensures teams properly identify a response and plan for a remediation. The final phase emphasizes an analysis of vulnerabilities in order to identify their root causes; this phase is key to preventing similar vulnerabilities from reoccurring in the future. We will explore each stage in more detail in the sections below.

Identify and Confirm Vulnerabilities on an Ongoing Basis

A remediation workflow should begin with an emphasis on identifying and confirming vulnerabilities on an ongoing basis. The ultimate goal of this phase is to identify vulnerabilities quickly, resulting in faster remediation times. To identify vulnerabilities most effectively, teams must have a strong understanding of what vulnerabilities might exist in their software, including any third-party components or libraries used. As detailed in ND-ISAC's previous papers, "Software Security Automation: A Roadmap toward Efficiency and Security" and "Software Security Automation: Security Controls Evaluation Criteria", numerous technologies exist that can help teams scan and analyze software at various stages of the SDLC as well as trigger alerts on potential vulnerabilities.

These technologies can be broken down into five categories based on the type of scanning utilized as illustrated in Table 1; from "Software Security Automation: A Roadmap toward Efficiency and Security". A third column was added to this Table to indicate the type of vulnerabilities detected. Developers and Security Teams can leverage automation in coordination with these tools to improve efficiencies of the SDLC. It is critical, however, that each tool be properly configured for the specific environment to identify vulnerabilities and return actionable results most effectively. In addition to these tools, public data sources such as the National Vulnerability Database and other vulnerability databases can be used to gather additional information on potential vulnerabilities in software.

Table 1

Application Security Technologies

Technology	Description	Type of Vulnerabilities Detected
Static Application Security Testing (SAST)	Conducts white box testing, performing analysis of source code for security vulnerabilities early in the software development process as part of the Integrated Development Environment (IDE), the commit or build process in a Development Operations (DevOps) methodology.	Vulnerabilities in the static source code
Software Composition Analysis (SCA)	Provides detection capabilities for security vulnerabilities in third-party components.	Vulnerabilities in any third-party libraries imported
Dynamic Application Security Testing (DAST)	Conducts black box testing to detect vulnerabilities associated with the application behavior by evaluating content from user/attacker perspective while application is running.	Vulnerabilities and runtime problems in a running application
Runtime Application Self Protection (RASP)	Provides detection and protection capabilities during runtime through instrumentation by monitoring an application's behavior and context of the behavior.	Vulnerabilities that may not have been identified in the previous scanning solutions
Interactive Application Security Testing (IAST)	Analyzes code by scanning for security vulnerabilities while the application is being tested (automated or manual test). This security control runs outside of the continuous integration continuous delivery (CI/CD) pipeline.	A wide variety of vulnerabilities from anywhere in the development process.

Assess, Prioritize, Remediate and Validate

Once issues have been identified, the teams responsible for the remediation must assess each finding using context. This evaluation will help teams assign a priority to each vulnerability identified and determine where to focus resources. As priorities are set, development teams can work on remediation efforts for the most pressing issues. Finally, once remediation has been implemented, teams need to execute proper validation to ensure the security findings were remediated. This section provides a guideline on how to assess, prioritize, remediate, and validate the remediation of security findings identified by the security controls implemented in the SDLC toolchain.

Assess

One of the most important steps in the remediation effort is the assessment of a threat. Assessing vulnerabilities allows teams to properly identify a response and plan for a remediation effort. A successful remediation requires teams to analyze each identified threat vector, using as much information as needed. The need for efficiencies during the assessment step requires the implementation of a process leveraging different automation tasks as well as specific criteria to use during the evaluation. Assessing threats will require a good understanding of the risk, the context under which the vulnerability can be exploited, the effort associated with a remediation, and the need to determine whether we are dealing with a false positive or not. All of these steps play a vital role in the assessment stage. To better support this process, it becomes important to capture all the needed information from the security vulnerabilities. Here, issue trackers become a great asset to the assessment stage.

One of the first steps needed to support remediation efforts involves the use of an issue tracking software or service to centralize the collection of security vulnerabilities. There are different options available in the industry to help integrate with the security controls in the SDLC. Organizations should pursue the use of issue tracking solutions that will best integrate with their specific security controls. After each successful scan, the issue tracker should create an issue for each security vulnerability identified.

Each issue should reflect the security control used to better understand where to address any potential remediation need. For example, a security finding in a SAST solution may require a very different remediation approach than those findings provided by a SCA solution. In one case, development teams will have to provide a fix at the code level while in the other a library upgrade may be required. This exemplifies the need to have specific metadata captured at the issue tracking system to help during the assessment stage. Table 2 provides a summary of minimal important metadata needed for each vulnerability recorded as part of each issue:

Table 2

Important Security Vulnerability Metadata to Include in the Issue Tracker

Metadata	Description
Scan Source	This field should reflect the source of a finding (SAST, SCA, DAST, RASP, or IAST).
Severity	This is a standard severity level/score provided by security controls.
Application	This field helps associate all the findings with a specific development effort.
Description	Provides in-depth understanding of the security vulnerability and its impact to the business.
CVE	Some security controls include a reference to a Common Vulnerabilities and Exposure (CVE) number. CVEs provide details regarding a specific vulnerability.
Priority Score	Allows teams to store the adjusted score provided by the priority framework included in the white paper.

Generating an issue for each finding and associating them with each development effort will enable teams to better plan remediation efforts at specific build cycles. In agile environments, issue tracking software is an integral part of development efforts; the issue tracker provides useful information about the multiple issues to be included in a specific build. Whether teams are using integration between the security controls and the issue tracker or handling the issue creation manually, the goal is to ensure teams have all the security findings included for better planning. Having the security vulnerabilities included in tandem with the development efforts for new features and fixes will help provide the visibility needed by a team when deciding which features or tasks should be part of the next sprint or build.

Once we have centralized the collection of the security vulnerability data, the next step is the use of context to determine the sense of urgency associated with the remediation. Assessing an identified threat requires the appropriate context to determine which priority is assigned to it. Organizations need to consider the risk posed by a threat and the viable entry points, among many others concerns, to properly decide the sense of urgency and the level of effort required to address findings. The combination of these elements will help determine how important pursuing a remediation should be. For example, an identified security vulnerability that could be exploited by an attacker requires a faster turnaround by an organization that lacks a compensatory control compared to a company where a compensatory control is provisioning protections against the same vulnerability. Organization can focus their attention on higher value targets while enabling teams to have more time to remediate threat vectors with a compensatory control providing temporary protections, thus reinforcing the need to evaluate context.

Analyzing a vulnerability requires context, which adds all the required criteria needed for teams to understand the potential damage of a threat to the organization and helps facilitate the prioritization process. There are several analytical elements providing enrichment to the assessment process shown in Figure 2. These analytical elements provide a summary of the context criteria to use when performing an assessment of a security threat as the team sets the stage for prioritization. Including these elements during the assessment will support a consequential outcome.

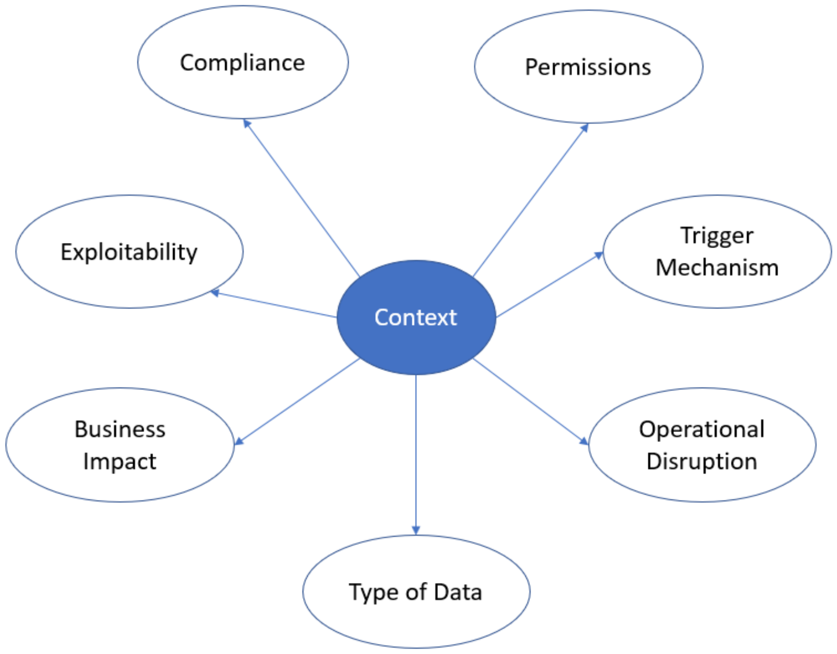

Figure 2 Context criteria needed to assess the impact of an identified threat vector

Appendix A of this paper provides a detailed process and framework to help determine priorities using context. Each of the previous criteria covers an important piece of the puzzle when assessing security vulnerability findings. Understanding each criterion helps to calibrate the risk associated with each in terms of the systems affected and the required response. Organizations need to employ the use of context to help determine the sense of urgency for any remediation effort while leveraging the severity information provided by security controls to understand the associated risk.

One potential outcome of the assessment process involves the detection of false positives. Assuming a strategy of discarding security findings without the proper vetting could put an organization at risk. Therefore, there is real value behind a false positive management strategy. Properly identifying false positives before prioritizing remediation efforts helps minimize the cost impact associated with attempting to remediate a false positive. Nevertheless, without the right tools and process, validating multiple security vulnerabilities could become a non-manageable and cost ineffective exercise. There are several key concepts organizations need to take into consideration to minimize the impact of false positive validation.

Minimizing the false positive ratio requires a combination of tools, proper configuration, and proof of exploitation capabilities. However, having detection capabilities that only provide security scan results, without configuration management and proof of exploitability, is not sufficient. There is no value in identifying a vulnerability in the application for technology A, when the implementation uses technology B. Therefore, having security controls within SDLC with the appropriate configurations will help the testing exercise use the applicable use cases against the implementation. For example, configuring the type of web server, application server, and database used in a DAST scanning tool will force the process to select a prescriptive set of security tests designed for that specific implementation.

Second, leveraging the proof of exploitation capability of security controls in the SDLC provides a validation mechanism needed to facilitate and accelerate the confirmation of findings. Using the proof of exploitation can help organizations create efficiencies. Without such a capability, teams would be forced to manually test the security threats identified, which could hamper the viability of an efficient assessment stage within the remediation workflow. Before any prioritization can take place, minimizing the false positive ratio through tools and effective configuration management becomes an important step in the process and should serve as a gate to allow the security finding to be included in any prioritization effort.

Prioritize

Deciding on priority assignment when addressing the remediation efforts requires the evaluation of multiple components as depicted in Figure 2: potential impact to the organization, effort needed to remediate, resources needed to weaponize the threat vector, compliance, etc. There are instances when an organization may decide to remediate specific security vulnerabilities for compliance purposes regardless of context or risk score. In those scenarios, the remediation workflow must be open to decision points steering the direction of the priorities outside the consideration of the regular decision-making criteria. For instances where there is no immediate mandate forcing the teams to remediate, it is important to use a thought process combining the security risk score with context.

Since organization have limited resources, careful consideration of the components needed to address a remediation effort will help spend the time and human capital wisely. Context element awareness is an important step when assigning a priority in the remediation workflow. The industry, as well as security controls in the SDLC, provide standard scoring mechanisms to help visualize the risk associated with security findings. Even though the information provided by those mechanisms is useful to understand the level of risk associated with a threat vector, organizations still need to use context to make constructive decisions when assigning priorities. For example, one common scoring system used in the industry is the Common Vulnerability Scoring System (CVSS). However, reacting and enforcing a remediation based solely on a high score from the CVSS may be misguided, as a high score does not necessarily reflect a need for a pressing mitigation.

Imagine a vulnerability with a high score in an application deployed in an isolated environment with no internet access, handling non-sensitive data, and having only two users impacted. If no context is applied to the remediation prioritization decision and only the security control severity/score is used, the organization may be tempted to simply act upon a score that does not necessarily reflect the sense of urgency needed for a remediation. Therefore, it becomes important to include context in the evaluation process when deciding priorities in addition to the use of the score/severity provided by the security controls (SAST, SCA, DAST, etc.) in the SDLC.

With hundreds of security vulnerabilities, development teams need a mechanism to help them decide which findings should be addressed first. To help determine the priority of the security findings, this paper provides a Priority Ranking Framework in Appendix A. Keep in mind that the security controls will provide a severity score/level based on risk elements. The ranking criteria presented in Appendix A represent context elements included in the discussion to help guide decision making in terms of priority and do not replace the severity of a specific finding. Once priorities have been identified, teams need to establish a remediation plan for each vulnerability. The next section provides guidance on how to manage the remediation of the security findings.

Remediate/Mitigate

After the vulnerability identification and discovery process, development teams need to be able to quickly assess and prioritize findings, giving priority to legitimate findings and reducing human intervention and review. It is important to mention that the process described previously cannot be 100% automated and will typically still require human intervention and analysis; but what can be automated is testing and guidance related to remediation. Automated remediation or remediation guidance can come in many forms, but we recommend tools that provide context aware finding and mitigation/remediation details directly into a developer's pipeline and/or IDE; in the form of remediation plans.

This allows a developer to receive results directly within the pipeline or promotion/development process and offer either things for the developer to review (links, documentation, etc.) or preferably it will offer solutions that the developer can quickly act on.

There are multiple remediation factors and recommendations to consider when planning to inject efficiency in the remediation process:

- Determining whether a finding is a false positive or legitimate
- Assessing possible impact and exploitability
- Awareness of security tools and capabilities that can be leveraged as mitigating factors
- Ability to log / track common findings and solutions
- While looking for opportunities to automate/implement these solutions earlier into the development process (i.e., prevent common / repeated mistakes)
- Working remediations to common vulnerabilities into coding standards and policies
- Mature secure coding guidelines and standards
- Automated and manual testing and assessment of software
- Implementation of principles like code reuse / code reuse repositories or code repositories of known or quality code
- Implementation/adoption of open-source library assessments/scanning
- Searching for legal risk, vulnerable libraries, etc.
- Implementation of SAST/SCA and DAST scanning; at a minimum
- Implementation/adoption of Secure Code training
- Implement / work towards Clean Build Concept

Diving deeper into the previously mentioned remediation factors and with the embedded remediation plans in mind, developers can build automated responses and actions for commonly identified findings. These efforts will take the form of automated remediation actions where these deviations can be detected during a code promotion run and corrected during the run, leading to a successful deployment without the need for rework. This is not always possible, so developers should look to identify common failures or findings and work these into their development and coding standards.

This allows developers to avoid these common mistakes, while also improving code quality. In addition to remediation plans, developers should look to implement tools that also provide mitigation suggestions or details for findings that are not easily remediated programmatically or are non-issues that can be addressed with other tools or capabilities within the environment. An example of this would be SAST and DAST tools that automate the building or suggestion of Web Application Firewall (WAF) rules, these can be fully automated in the form of allowing the tool to implement necessary protections as part of a run / deployment or as stated previously this can take the form of an embedded mitigation plan that the developer can then work with the proper security teams to implement the suggested WAF rules to properly address and mitigate these findings.

Embedded mitigations are harder to automate as they rely on other security tools or capabilities that exist within the developer's environment. This is why it is crucial for developers to understand their organizations capabilities, tools, and process. Organizations should plan to actively work with the security / DevOps security teams to properly integrate or feed these mitigations either directly into security tools, to the appropriate security team members or even leveraging Security Orchestration, Automation and Response (SOAR) tools (if applicable in their environment). After specifying criteria questions for a given category, weight each criterion within each individual category, applying a higher weight to more important criteria. The criteria weights assigned within each category need to add up to one.

Validate

After a root cause remediation or mitigating control has been implemented, the Security team needs to validate that the vulnerability is no longer present or is sufficiently protected against. The process of validation largely depends on how the original vulnerability was identified. Vulnerabilities identified via automated scans (SAST, DAST, SCA, etc.) should have the scans re-run on the new artifact or deployment to confirm the vulnerability has been appropriately addressed. Manually identified vulnerabilities (penetration testing, bug bounty reports, etc.) should have the Security team execute their retest process and confirm that the vulnerability has been remediated. If a vulnerability was identified by both automated and manual processes, we recommend executing the automated scans first, confirming that they show as remediated, and then executing the manual retest process.

Potential outcomes boil down to three possibilities:

- The vulnerability was remediated
- The vulnerability was not fixed, or only partially fixed
- The remediation revealed another vulnerability as a side effect

In the first instance, the team should follow process to close out the documentation and proceed to a root cause analysis if needed. For the second, the team should relay the issue back to the vulnerability owner and update the documentation if any new payloads or steps were needed to re-exploit the vulnerability. For the last result, the team should close out the documentation for the first vulnerability, then document and restart the process with the new vulnerability's information. As a final note, the validation process also needs to handle potential false positives. It is important to present the vulnerability owner the option to mark the finding as a false positive in the event a validation fails. From there, the false positive evaluation process should result in the Security team either agreeing or returning to evaluate the vulnerability.

Analyze Vulnerabilities to Identify Their Root Causes

Root Cause Analysis (RCA) exercises are meant to identify and document the root causes issues faced by the team. These exercises are executed for a variety of purposes, but the most common ones are:

- Critical vulnerability identified in Production
- Addressing persistent false positives
- Periodic review of common vulnerabilities caught by security tooling

The RCA for an identified vulnerability will primarily seek to answer the following:

- If the vulnerability was found in production, were our controls bypassed?

- Why did our existing controls not catch this vulnerability?
- Are our existing controls able to be updated to address future instances of this vulnerability in the future?
- What new controls (if any) need to be implemented to address identifying and protecting against future instances?

The first is the most important as a bypass of controls indicates either a risk acceptance process (which should have appropriate documentation), or a process/policy weakness to be addressed. The remaining questions are looking to understand if/how the technology failed, and if it did, how to attempt to prevent the issue in the future. In the case of persistent false positives, the team should investigate the tool flagging the false positives and determine if there are any persistent mitigations that can be applied, or if the development team can rearchitect the code to avoid triggering the false positive. If no long-term solution is possible to address the false positives, the team should document the issues and maintain that as a reference for future triage steps.

Finally, periodic review of common vulnerabilities is recommended to adjust training programs and developer resources. Even though the vulnerabilities may be caught by the security tooling, understanding the root causes of the common issues faced by the development teams will help assist in identifying better training and development opportunities. Output of these RCA exercises will depend on the cause of the exercise, but each root cause should be documented and made available to teams once any confidentiality or sensitivity around the root cause has been addressed. By sharing the root causes and giving developers the ability to review and search, the development teams are better able to avoid issues in the future.

Conclusion

This paper highlighted our three-stage approach to the remediation of identified findings through the use of an automation centric workflow. Throughout the paper, a thoughtful approach was taken to ensure that we stayed technology-agnostic, while also leveraging known tool capabilities or use cases that can be applied to real world scenarios and workflow. We challenge organizations to embrace this approach, while tailoring it to their specific environment (context) and understanding that resource challenges exist in all organization today. The three stages are intended to help remediations effort by speeding up the remediation workflow, while preventing unnecessary rework.

To summarize, here are some of the most important takeaways of this paper (provided as a recap):

- An understanding of current capabilities within the application security field
- The discovery/identification of security vulnerabilities will be performed by security controls in the SDLC (SAST, DAST, SCA, RASP, IAST)
- The use of an issue tracking service or software is crucial

- Prioritization must factor in false positive detection and it must take organizational context into consideration
- Identify a false positive management strategy that is backed by a trusted vetting source/method
- Relying only on security tools without vetting or verification can put an organization at risk
- Recommend a multi-tool approach and automating where it makes sense, not all phases of this process can be fully automated
- Optimization of security tools will reduce false positives and save time
- Based on technology in use, this allows tools to only perform pertinent security checks/tests reducing noise
- Keep the workflow flexible, organizational priorities can change at any time and findings that are the focus of remediations are not always dictated by its risk score
- Customized risk scoring framework should be developed and implemented
- Use organizational context aware scoring as an overlay to identify findings and for comparison to raw scores
- A priority scoring framework is provided as a starting point in this paper
- Organizational compliance requirements must be factored into any custom scoring or decision-making framework
- When performing analysis of findings, you must understand exploitability and its relationship to permissions and business impacts

- Understanding your data, data classifications and how findings might affect the security of the data is key
- Consider security tools and capabilities that can be leveraged in your environment as mitigating factors
- Mature secure coding guidelines and standards are a necessity and foundational for any AppSec program
- Implementation/adoption of Secure Code training program
- Root Cause Analysis should be performed to determine patterns and similarities in findings, this leads to more impactful and detailed validation of findings and remediations.

Appendix A - Priority Ranking Framework

The Priority Ranking Framework uses a weighted approach to determine priorities. Table 3 provides a representation of the high-level implementation when ranking vulnerabilities. The table provides three pieces of information: a context criterion, a weighted value representing the importance of the particular context criteria per the organization, and a priority score. Priority scores are represented using a high, medium, low approach with a corresponding value. To simplify the calculation, any high importance context criteria represent a value of 3, mediums represent a scale value of 2, and low represents a value of 1. Each context criteria should provide specific reference values which will match the corresponding high, medium, low scales. To calculate an adjusted score, the weight must be multiplied against the assigned priority score. This calculation will provide a total value which could be contrasted against each security vulnerability. The higher the value, the higher the sense of urgency to be assigned for the security finding during a remediation effort. Each organization has a specific and unique set of priorities for the context criteria; the weight column provides a mechanism for companies to decide the level of importance of the different context criteria scores.

Table 3 (Priority Framework Example)

Security Finding #1: Cross Site Scripting

Context Criteria	Weight	Priority Score	Adjusted Score
Compliance	0.3	2	0.6
Exploitability	0.1	3	0.9
Permissions	0.1	2	0.2
Trigger Mechanism	0.1	2	0.2
Business Impact	0.1	1	0.1
Type of Data	0.2	3	0.6
Operational Disruption	0.1	2	0.2
Totals	**1.0**	**15**	**2.8**

The end result of the calculation is to provide a total adjusted score for each security vulnerability, which, in conjunction with the severity score provided by the security controls, will help determine prioritization. The total adjusted score can be summarized and contrasted among the security vulnerability findings to determine the order when addressing remediation.

The importance behind this calculation is associated with the need to determine the priority for the remediation. As stated previously in this paper, the priority is not a substitution for the severity score level, but an additional component to help determine how organizations should spend their time and effort during remediation. Once each security threat adjusted score has been calculated, teams can use the scores to contrast the security findings against each other.

Security Finding	Adjusted Score
Security Finding #1 XSS attack vector	2.8
Security Finding #2 SQL Injection attack vector	3.1
Security finding #3 Sensitive Information Disclosure	2.2

To understand how to reach each specific value in the priority score column, organizations need some level of guidance for each context criteria. The next several sections provide information to help decide how to assign a priority score for each context criteria. The examples provided for each criterion should help map the values to be selected for the tabulation exercise in the priority framework.

Compliance

Some security vulnerabilities identified by the security controls in the SDLC may need to be considered higher priorities if they affect the compliance level of the organization. Therefore, when using compliance as a context criterion, organizations will have to deal with two specific use cases: the security finding impacts compliance, or it does not. The most pressing use case involves a security finding which affects the compliance level of the organization. For example, a security vulnerability supporting a violation of the General Data Protection Regulation (GDPR) could pose substantial economic hardship. Therefore, organizations can decide to assign a higher priority score for those entries associated with compliance violations versus those who do not. Table 4 provides a summary of the possible values to use when scoring the compliance criteria of a security finding.

Table 4

Compliance Criteria

Criteria	Score
Impacts compliance	3
Does not impact compliance	1

Exploitability

Measuring exploitability helps organizations understand how easy or hard it is for an attacker to leverage a specific threat vector. The easier it is for an attacker to exploit a vulnerability, the higher the priority that should be assigned to it. Having a clear guideline on how to classify a threat vector will help during the prioritization. Using the exploitability criteria, we can assign and map specific scores to each concern, which helps determine the priority. Table 5 is intended to provide a classification which could be used during the prioritization process.

Table 5

Exploitability Criteria

Criteria	Description	Score
Detected	Evidence exists of attackers successfully exploiting the vulnerability in the wild.	3
Likely	Attackers are able to consistently exploit the vulnerability even when no public record exists of it being exploited in the wild.	2
Less Likely	Attackers face different levels of difficulty to consistently exploit the vulnerability. Level of difficulty could encompass expertise level needed, specific timing required, or additional supporting components needed to enable the exploit.	1

Unlikely	Even when the code could be exploited, the existence of special circumstances will make the exploit unlikely (for example, a Web Application Firewall (WAF) preventing code injection or physical access required)	0

Permissions

Understanding the permissions required to exploit a security vulnerability helps understand the potential risk to the organization. Some threat vectors will require an authenticated user to be effective, while others can be exploited without any need to submit credentials. Having security vulnerabilities that could be exploited without any credential submission allows an attacker to be more effective–and more likely to be successful–than if the attack requires credentials to be exploitable. Therefore, the scoring assigned to each criterion under permissions portraits a higher score for non-authenticated attacks since the layer of protection provided by the use of credentials is non-existent. Table 6 provides a summary of the criteria as well as the assigned scores.

Table 6

Permission Criteria Scoring

Criteria	Description	Score
Non-Authenticated	No need for credentials to exploit the security vulnerability	3
Authenticated	Credentials are needed to exploit the vulnerability.	1

Trigger Mechanism

Understanding how a security vulnerability is triggered helps understand whether organizations have compensatory controls in their environment that could minimize the risk associated with the threat. When evaluating the trigger mechanism during the assessment process, organizations should decide if an attack vector requires the interaction of an end user or if the attacker can exploit the vulnerability directly. Examples of scenarios where the threat vector requires user interaction include a user visiting a specific page or an end user opening a malicious binary. From a risk perspective, attack vectors where no user interaction is required should reflect a higher risk than those requiring the end user to execute specific steps or actions. Table 7 summarizes the ranking scores for the criteria to be used for the trigger mechanism.

Table 7

Permission Criteria Scoring

Criteria	Description	Score
No end user or 3rd party interaction needed	Exploit can be leveraged by the attacker without the need of user steps or actions taken.	3
User or 3rd party interaction needed	User must execute a step or action to enable the exploit.	1

Business Impact

Cyber-attacks have varying levels of impact to organizations. Some attacks may affect the reputation of an organization while others may be isolated to specific environments. Understanding the classification of the business impact becomes key during the assessment stage. To simplify the discussion, business impact can be categorized in three main scopes: significant, moderate, and limited. There are multiple scenarios which could be used to understand the demarcation and the determination of when to use each scope. Note how data classification is not considered part of the business impact in this context since we have a data classification concern which could be used during the assessment process to classify the threat.

When dealing with attacks which could cause significant business impact, there are several scenarios' organizations could use to drive their determination:

- Attack vector can result in adverse reputation impact
- Attack vector could cause a critical loss of use of the software
- Attack vector can disrupt a significant number of users, project schedules, costs, business functions, etc.
- Business disruption caused by leveraging the attack vector will force the organization to incur significant legal fees and/or regulatory fines
- Attack vectors can compromise services/products which will affect the organization's bottom line
- Attacks could cause the non-compliance of aggressive Service Level Agreements (SLAs), such as 99.9% uptime

Moderate business impact criteria can be used in scenarios where the attack vector could impact several business units, but not to the degree posed by significant events. In moderate scenarios, the loss of use is of non-critical nature. When evaluating moderate business impact, organizations should consider the following scenarios:

- Attack vector can affect several business units in the organization in a setting where the application is not externally exposed
- Threat vector could impact service availability without affecting any SLAs

Limited business impact still represents a concern to the organization, but due to the isolated nature of the attack vector, the risk posed is minimal. There are different scenarios that can be used to determine when to select the limited scope:

- Attack vector targets an internal application used by a small number of users
- Attack vector targets an isolated environment with no access to the internet or other systems
- Security vulnerability requires the exploitation of other security controls before any attempt can be made to leverage it

Table 8 provides a summary of the different business impact criteria and their corresponding scores.

Table 8

Business Impact Criteria Scoring

Criteria	Score
Significant	3
Moderate	2
Limited	1

Type of Data

Organizations manage different types of data. Some applications are created to share public information, other are created to handle extremely sensitive data. It becomes important to understand the type of data which could be affected by a security vulnerability. Depending on the type of data we are dealing with, the risk of a compromise could change the level of responsibility of the business. Imagine a scenario where national security sensitive data can be stolen. Understanding the relationship between the type of data and security vulnerabilities identified will help organizations decide how aggressive they have to be in their efforts to mitigate and remediate the threat vector.

To facilitate the assessment, data types are grouped in three main areas: sensitive, internal and public. Sensitive data covers all data needed to comply with industry/regulatory compliance. Examples of sensitive data include CUI, ITAR, EAR, PII, HIPAA. Also, any proprietary information regarding the products and services used to differentiate the business in a competitive setting should also be considered sensitive in nature.

Internal data refers to any piece of information used to support business functions but does not fall under the sensitive classification. For example, having a security vulnerability which could expose the configuration of a plugin in the Continuous Integration platform (assuming does not contain credentials or sensitive information) is to be considered internal. This type of data has the power to help an attacker understand the inner workings of an implementation. Finally, public domain data is information already publicly accessible. Table 9 provides a summary of the criteria to use when scoring the type of data.

Table 9

Type of Data Criteria Scoring

Criteria	Score
Sensitive	3
Internal	2
Public Domain	1

Operational Disruption

Some security vulnerabilities can affect the availability of services to the business. In some instances, when we have competing priorities, it becomes important to determine which security threats could pose a more significant risk of operation disruption. There are three main criteria we could use to classify operational disruptions: complete or significant disruption, partial disruption, or no disruption. Complete or significant disruption poses a higher risk to the organization as it hampers the ability to operate. A good example of a significant disruption involves the detection of a 3rd party library which could be used to cause a Denial of Service (DoS) to the entire application.

Partial disruptions are associated with those instances when the security vulnerability could be used to hamper some of the functions of an application but not all of them. For example, a security vulnerability allowing an attacker to prevent the usage of a REST API while the application is still operational represents a good example of a partial disruption. Finally, since not all security vulnerabilities would affect or disrupt the operation, the final criteria, no disruption, can be used for those security threats which have no operational impact. Table 10 provides a summary of the operational disruption criteria.

Table 10

Operational Disruption Criteria Scoring

Criteria	Score
Complete or Significant Disruption	3
Partial Disruption	2
No Disruption	1

Ideally, organizations would implement a strategy to automate the ranking process to save time and resources. Even when security controls do not provide a way to inject the context criteria in their evaluation, a custom solution could be created to map the most common vulnerabilities and define the corresponding context criteria values. The effort would require the combination of using tags in application profiles to define some of the context criteria and a reference table to link. When a security control generates a finding, the custom implementation could map the findings with the pre-defined context criteria and automatically rank the priority. Though the use of REST APIs, the result of the automated ranking process could be used to update the issue tracker with the calculated priority value. This activity could result in time savings and the removal of the manual ranking process.