How-to-Guide for Software Security
Vulnerability Remediation
by Dan Cornell
Executive Summary
The security industry often pays a tremendous amount of attention to finding security vulnerabilities. This is done via code review, penetration testing and other assessment methods. Unfortunately, finding vulnerabilities is only the first step toward actually addressing the associated risks, and addressing these risks is arguably the most critical step in the vulnerability management process. Complicating matters is the fact that most application security vulnerabilities cannot be fixed by members of the security team but require code-level changes in order to successfully address the underlying issue. Therefore, security vulnerabilities need to be communicated and transferred to software development teams and then prioritized and added to their workloads. This paper examines steps required to remediate software-level vulnerabilities properly, and recommends best practices organizations can use to be successful in their remediation efforts.

For more resources, visit www.denimgroup.com/remediation.
# Table of Contents

Best Practices Overview ................................................................. 5

I  Inception ...................................................................................... 5

II Planning ....................................................................................... 6

  Calculate Risk ............................................................................... 6
  Agree on Fix and Confirmation Methods .................................... 8
    Use Standards ............................................................................ 8
    Fix Vulnerabilities Completely ................................................ 9
    Use Manual and Automated Testing Appropriately .................. 9
  Determine Level of Effort ............................................................. 10
  Schedule (for Waterfall Projects) ................................................ 12
  Schedule (for Agile Projects) ....................................................... 12

III Execution ..................................................................................... 12

  Fix Vulnerabilities ....................................................................... 13
    Development Environment ...................................................... 13
    Fixing Technical Vulnerabilities ............................................. 13
    Fixing Logical Vulnerabilities ............................................... 14
  Confirm Fixes and General Quality Assurance ............................ 14
  Deploy the Updated Application to Production ......................... 15

Remediation Metrics ...................................................................... 15
Conclusions ..................................................................................... 16
Acknowledgements ......................................................................... 16
Best Practices Overview
The goal of any software remediation project should be to reduce the amount of risk to which an organization is exposed by fixing application-level vulnerabilities in an economically responsible manner. This involves developing an understanding of the current vulnerabilities in applications and the risks they present, and then fixing selected vulnerabilities in either the application code or the deployed configuration. Forward-looking organizations who value continual improvement will use the lessons learned from the remediation effort to focus the strategy for managing application security assurance going forward to avoid re-introducing similar vulnerabilities.

In addition, it is important for organizations to track metrics for their remediation efforts. Collecting this data demonstrates the true costs of fixing security problems after they have been introduced. Based on this information, organizations can make informed decisions on where it is appropriate to integrate security into the application development lifecycle in order to reduce remediation costs later on. In addition, tracking the root causes of the vulnerabilities in their application portfolios can help organizations make process and policy changes to prevent similar vulnerabilities from being introduced in future development efforts.

Remediation projects consist of three major phases:
1. Inception – Sets high-level goals for the remediation project
2. Planning – Defines the specific actions to be taken
3. Execution – Fixes the vulnerabilities and pushes the newly-secured application into production

Every organization develops software in different ways. In fact, most large organizations have multiple groups developing software, often with significant differences among these groups’ practices. Therefore, methodology outlined in this paper must be adapted to the practices of the organization and the team that is responsible for the vulnerable software. Security remediation efforts should integrate into the normal flow of the development and maintenance of the application.

I Inception
Although it is often approached informally, organizations should treat the beginning of remediation projects as part of a formal, structured process. This Inception phase should set clear expectations going forward for all parties involved. The Inception phase is the initial opportunity for all stakeholders in the remediation project to agree on overall strategy and goals. The Inception phase should set entry and exit criteria for subsequent phases, and allow for a candid and objective assessment of the Remediation project. The first part of Inception is to identify all of the stakeholders for the remediation effort. These will likely include representatives from security teams, development teams, quality assurance teams, business owners and, potentially, internal auditors. The initial “meeting of the minds” for these groups should be used to communicate each stakeholder’s specific concerns, capabilities and areas of responsibility.

This remediation methodology assumes that an organization has already assessed one or more of their applications and identified a body of vulnerabilities. Some organizations are using an emerging approach of using short-term protection measures such as using Web Application Firewall (WAF) or web-aware Intrusion Detection or Prevention System (IDS/IPS) rules to apply “virtual patches” to applications. From this starting point, organizations can then work to address the underlying software issues responsible for the vulnerabilities.
DENIM GROUP TIP

Software security remediation projects bring together a variety of stakeholders with competing concerns and responsibilities. From the outset encourage a balanced, win-win approach to discussions.

In addition, important factors to determine during the Inception phase are:
- Estimated project budget
- Desired project end date
- Specific compliance or audit issues that must be addressed
- Overall project approach (for example: waterfall or agile)
- Deployment strategy
- Initial success criteria (for example: “remediate all vulnerabilities that are medium and above”)

These factors may be modified in subsequent phases as the team develops a deeper understanding of the vulnerabilities in the application as well as the organization’s business goals surrounding the remediation effort, but it is important to have initial expectations defined up front.

II Planning

Once the high-level view of the project has been established and stakeholders have had their expectations aligned, the planning phase is used to establish a structured approach for addressing vulnerabilities as well as to create a consensus as to which vulnerabilities are going to be addressed, when they will be addressed and how they will be addressed. Planning phases consist of three required steps:

1. Calculate Risk
2. Agree on Fix and Confirmation Methods
3. Determine Level of Effort

* Schedule

The Planning stage is often iterative, with analysts attempting to group vulnerabilities in different ways and conceptualize ways to approach fixes in an attempt to optimize the amount of risk that can be addressed with a minimal level of effort. This iterative approach is fine as long as the remediation team ultimately reaches a common understanding of the risk, level of effort and fix methods that will be utilized going forward.

Calculate Risk

In order to best prioritize, the first step in planning a remediation effort is to determine the level of risk associated with the identified vulnerabilities. This risk is the first factor used to determine what vulnerabilities will be remediated and when. This is a challenging step because it requires stakeholders to come to agreement regarding the true business risk represented by vulnerabilities whose impact may be challenging to concretely define and clearly communicate. However, it is a critical step because decisions made at this point will drive prioritization for the remainder of the project.
In an ideal world, all vulnerabilities would be immediately remediated with the most effective fixes and countermeasures. However, in the real world, the resources needed for security fixes must often be drawn from the same pool as those allocated other development efforts – resulting in competition for these scarce resources. For applications that have reached end-of-life status, organizations may choose to leave vulnerabilities as-is rather than addressing them because of the shorter time horizon during which those vulnerabilities will expose the organization to risk. Other circumstances might cause organizations to make other tough decisions about what vulnerabilities will be addressed and which will be allowed to remain. The Calculate Risk step results in an enumeration of the business risks associated with an application’s vulnerabilities so that stakeholders can later make informed decisions about what to address, when to address, how thorough the remediation steps applied will be and what risks will be accepted by the organization or dealt with through other means such as insurance.

Severity ratings provided by automated tools can be a starting point for determination of business risk, but should not be relied on exclusively. Automated tools tend to have little or no information about the business context of the application. Additionally, they tend to focus on web applications so results for non-web applications may have their severity distorted. This lack of context leads to inaccurate characterizations of the true business risk associated with a vulnerability. For example – a SQL injection vulnerability in a web application managing credit card data would typically be considered a significant risk, whereas a SQL injection vulnerability in a thick-client application managing company picnic registrations against a locally-hosted database may be considered less of a risk. Most automated tools would classify these vulnerabilities the same, so the discretion of an analyst is required to reflect true business risk more accurately. In addition, there are classes of vulnerabilities that are not typically identified by automated tools, so it is very common for the aggregate list of vulnerabilities to contain results from one or more static or dynamic analysis tools as well as inputs from both manual code review and manual application testing. Creating a consolidated assessment of the risk from these aggregated vulnerabilities is required in order to move forward.

When calculating risk, organizations should use a structured approach and use it in a consistent manner. This allows for a greater degree of “apples to apples” calculations and facilitates the creation of clear policies, making this decision-making process more consistent. A number of structured approaches have emerged such as DREAD (http://msdn.microsoft.com/en-us/library/aa302419.aspx) from Microsoft’s Threat Modeling approach, or the OWASP Risk Rating Methodology (http://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology) from the OWASP Testing Guide.

For applications with large numbers of vulnerabilities, it may make sense to group vulnerabilities to be classified concurrently. There are several reasons for this:

- Applications often use common coding and design idioms causing certain types of vulnerabilities to cluster.
- Identifying clusters at this point can help to save time, which is important if
the selected risk-ranking methodology requires nontrivial input in order to come up with a ranking. For example – users of the DREAD methodology must make determinations about five separate factors for each vulnerability (Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability) and the OWASP Risk Rating Methodology has even more factors for consideration.

Intelligently grouping vulnerabilities can help make the Calculate Risk step much more efficient. These groupings will be finalized later on, but starting at this point in the process can help to save time and lead to greater insight earlier in the process. Groupings can be created that will help developers later in the process.

**DENIM GROUP TIP**

This helps to reduce pushback from development teams. For example, a grouping could be “all cross-site scripting vulnerabilities in the ‘user profile’ page” or “lack of authorization checks in the ‘order’ directory.” These groupings may also form the basis for creating software defects that can be assigned to developers, as is discussed later in this paper.

**Agree on Fix and Confirmation Methods**

The outcome of the risk-ranking process is a set of vulnerabilities that have been grouped and risk-ranked. Before deciding what vulnerabilities will be remediated, levels of effort must be determined for the fixes of the vulnerabilities, and before that can happen, the team needs to understand specifically how these vulnerabilities will be fixed and how testing will demonstrate that the fixes were applied successfully. In the absence of this, remediation projects tend to produce mixed results where effective protections are used along with ineffective protections and vulnerabilities often resurface because the attempted fixes did not meet the level of effectiveness required by security teams. To prevent this, it is critical that remediation projects have an agreed-upon set of methods for making fixes as well as agreed-upon tests for considering applied fixes to be effective. This involves inputs from developers, quality assurance testers and security analysts.

**Use Standards**

If vulnerabilities were introduced into the application because developers did not follow published standards for how code was to be written, then the code must first be brought into line with accepted organizational standards. If no such standards exist, then those standards should be developed. When building such standards, a number of factors must be taken into account. These can include:

- Organizational standards
- Industry standards
- Compliance requirements

The use of industry-accepted security libraries can be a great help in setting guidelines for acceptable fixes. For example, the use of libraries such as the OWASP Enterprise Security API (ESAPI) ([http://www.owasp.org/index.php/](http://www.owasp.org/index.php/))...
Category:OWASP_Enterprise_Security_API or Microsoft’s Web Protection Library (http://wpl.codeplex.com/) can be a way to give developers the tools they need quickly to fix vulnerabilities. Reliance on these industry-accepted tools keeps development teams from having to expend resources on creating libraries themselves and also provides a more reliable base by relying on libraries that have been tested in numerous and varied production environments and reviewed by the security community at large. An added benefit of the development and enforcement of proper architecture, design and coding standards is more secure code across all of an organization’s development efforts.

**Fix Vulnerabilities Completely**
If time and effort are going to be spent remediating existing applications, then the fixes applied should last over time. When creating acceptable methods for how vulnerabilities are going to be remediated require that the methods address the underlying security issues rather than relying on band-aid changes that only address a specific attack scenario. This will most effectively reduce risk and will also save time and resources that future rework would require.

**Use Manual and Automated Testing Appropriately**
Establishing agreed-upon methods for what is to be considered an effective fix is also required so that an end point can be established for the remediation project. In cases in which automated tools identified the vulnerabilities, a common practice is to use another execution of the tool to determine whether the vulnerability is still present. There can be pitfalls with this approach – the tool must be configured properly and automated testing tool results can contain false positives so care must be taken to identify these. However, for applications with a large number of vulnerabilities this may be the only economically viable approach. If automated tools will be used for the confirmation of successful vulnerability remediation, at least a small sample of the remediated vulnerabilities should be manually code-reviewed or tested to confirm that the fixes indeed appear to be effective.

If vulnerabilities were identified by manual application assessments or code reviews, then other means are required to determine whether or not fixes have been successful. A common approach for these is to have the original analyst re-test or re-review the code to make a determination if the security defect has been successfully fixed. Another third party could also be used to perform spot inspections. The focus of these re-reviews would typically be limited to the specific vulnerabilities in question in order to control costs. Similarly, vulnerabilities identified by

---

**DENIM GROUP TIP**

It is far preferable to apply positive checks rather than negative checks. Positive checks require a value to have specific characteristics such as type, size and conformance to business rules. These tend to be more aggressive in denying inputs and encoding outputs and require more stringent authorization. Positive checks are superior to validation methods that look for “known bad” payloads, encoding methods that only encode a subset of special characters.

At this point, these standards are being created reactively in order to respond to identified vulnerabilities in applications. It is also a best practice to roll out these standard protective measures proactively for team members who are creating new applications and new features for existing applications. Adopting a proactive approach to prevent the introduction of new vulnerabilities is a vital process improvement step for organizations that are serious about addressing software risk.
architecture or design reviews typically require in-depth inspection and will likely require manual testing or code review of the area in which design or architecture was previously in question. Sometimes, automated functional tests can be created that demonstrate a vulnerability in the original version of an application that will then fail to exercise the vulnerability in the remediated code. If this approach is used, care must be taken to ensure that the tests accurately reflect that a “successful” remediation has actually addressed the underlying vulnerability and not only specific attack payloads.

**Determine Level of Effort**
Risk is a critical factor to consider when determining which vulnerabilities should be fixed and when, but it is not the only factor. Unlike the world of infrastructure where security fixes are often fairly straightforward and the issues can be addressed by changing a configuration setting, applying a given patch, or turning off a feature, application fixes typically involve making updates to code and potentially changing the way features behave. Each fix, in effect, its own software development project and the variability of the required level of effort for different fixes can be significant. Therefore, a critical step in Planning for the remediation of software-level security defects is determining the level of effort required to implement the fix. The fix methods agreed upon by the team will be used as the basis for determining this level of effort.

Successful remediation projects estimate this effort “bottom up” by enumerating all of the tasks involved in a project at a fairly granular level, estimating time for each of these tasks and then calculating the sum of efforts for all tasks to determine the actual effort for the project. In addition, successful projects account for all of the different necessary resources. Fixes likely require time from developers in order to make code changes as well as time from quality assurance (QA) staff to confirm the application still performs as desired. Time from security analysts may also be needed to determine if the security fixes have been performed successfully. Incorporating all labor required for a successful fix allows for greater visibility into the actual timeline for the remediation project, and a greater chance for project success.

Using a simple Work Breakdown Structure (WBS) to calculate the level of effort is a time-tested approach to task estimation that tends to be a good fit for remediation projects. (For more information on Work Breakdown Structures, see http://en.wikipedia.org/wiki/Work_breakdown_structure)

The estimation process in itself can be daunting and time consuming until an organization has developed a body of historical data. The process can be streamlined using the following approach:

1. Separate logical and technical vulnerabilities – Fixes for technical vulnerabilities usually require rote code-level fixes that are straightforward and repetitive, making them relatively easy to estimate. Logical vulnerabilities

**DENIM GROUP TIP**
Based on Denim Group observations, a granularity of 4-16 hours for most tasks in a WBS has proven to be appropriate for remediation projects. This prevents getting too deep into discussions of minor tasks that can be grouped and also prevents a degeneration into top-down macro estimates such as 40-80 hours which may indicate that insufficient attention has been paid to the specific tasks required for the fixes.
tend to deal with the specific business context and business logic of an application and can require developers to redesign parts of the application. As a result, the level of effort required to remediate logical application vulnerabilities can be far more varied.

2. Group and estimate technical vulnerabilities – Estimating the level of remediation effort for certain technical vulnerabilities such as SQL injection or Cross-Site Scripting (XSS) can often be done via the groupings that were identified during the Calculate Risk step described previously. Rather than estimating these specific tasks on their own, a reasonably accurate and time-saving approach to these calculations can be to estimate the time required per fix and multiplying it by the number of vulnerabilities to be fixed. Variability can be expected in the time required for fixes even for the same vulnerability type, so the focus should be on a reasonable average. Historical data can be extremely valuable in supporting this practice, or it may make sense to perform fixes for a small number of the vulnerabilities as a test to determine an average to using going forward. With practice, it should be quick and simple to estimate the remediation for technical vulnerabilities.

3. Estimate logical vulnerabilities – As mentioned above, fixes for logical vulnerabilities are application-specific and therefore these changes have the potential to be more invasive. When they deal with authentication or authorization issues, the fixes may require coordination between different groups within the organization as well as customers and business partners. Access to individuals familiar with the application can help create accurate estimates by highlighting the potential impact of changes.

Serious vulnerabilities, especially those dealing with authentication or interaction with other systems, may require fixes complicated enough to justify dedicated, full-lifecycle software development efforts. These projects can quickly become very expensive.

The amount of detail provided by the results of the assessment regarding the location of vulnerabilities will impact the time required to fix. Outputs from code scanning tools or manual code reviews should have vulnerability descriptions including code-level information about the specific locations of vulnerabilities. This allows for quicker remediation than cases where vulnerability descriptions only include attack surface information such as a vulnerable web URL and parameter.

DENIM GROUP TIP

It is also important to take into account the composition of the team that will be performing fixes. Team members with little experience in secure coding may require training prior to performing security remediation tasks and may also require more supervision from security-savvy developers.
Schedule (for Waterfall Projects)
A common and effective approach is to either establish a specific budget or to “time-box” the remediation effort. Time- or budget-boxing is a project management technique whereby a specific date or budget is set for the end of a project. Remediation tasks are added to the project based on priority until the budget or calendar time has been fully allocated. Some buffer time is often added to accommodate unforeseen delays or tasks that have not been enumerated. Time-boxing can help to facilitate discussions between security teams and development teams. Both sides negotiate the priorities of vulnerabilities in the face of pre-determined constraints so that everyone feels as if their goals have been respected.

Based on the vulnerability risks enumerated and the level of effort required to fix them, the remediation team should work to identify which vulnerabilities can be fixed given available budget and other resources. Vulnerabilities representing more risk that are easier to fix are obvious choices for early inclusion in remediation schedules. Vulnerabilities that are risky but challenging to fix or that are easy to fix but less risky require the project team to engage in more discussion. The resulting schedule should lay out one or more Execution phases with specific vulnerabilities to be addressed in each phase as well as the expected team size for each phase.

Schedule (for Agile Projects)
Longer-running remediation efforts or those based on Agile software development projects could instead defer the Schedule step to be done at the start of each Execution phase. This allows the team flexibility to re-prioritize based on current resource levels and needs.

### DENIM GROUP TIP
Existing defect tracking systems can be used to support the scheduling effort. Related vulnerabilities can be grouped into single defects so that these defects can then be prioritized, scheduled and assigned to developers. This is an effective way for security personnel to transition vulnerabilities – which are their concern – to defects – which are the software development team’s concern. Communicating with development teams using their existing tools and processes helps to decrease friction between security and development organizations. If security analysts do not have direct access to defect tracking systems, this can be an opportunity to work closely with development teams transferring the data that development teams will need.

### III Execution
After Inception and Planning phases have been completed, goals and accepted methods for the remediation effort have been set. Then the remediation team performs one or more Execution phases in which they will make code changes required to address vulnerabilities, confirm the effectiveness of those code changes and then push the updated code into production. Execution phases are essentially software maintenance efforts and, as such, can leverage many of the tools and processes that an organization already has in place for these activities. The three steps involved in an Execution phase include:
1. Fix Vulnerabilities
2. Confirm the Fixes and Perform General Quality Assurance
3. Deploy the Updated Application to Production.

Fix Vulnerabilities
Fixing vulnerabilities typically requires that developers change application source code in order to change application behavior. The level of invasiveness of these changes will vary based on the nature of the vulnerabilities being addressed.

Development Environment
In some cases, the developers performing the fixes are the same developers who created the original code. This allows them to benefit from their existing knowledge of the codebase as well as existing development installations of the application, development tools and automated test suites. If that is not the case, then the developers performing the fixes must first create a development environment, get access to source code and make sure they can successfully create new running builds of the application. Once these prerequisites are in place, actual fixes can begin. Times required to address these prerequisites can vary widely depending on the amount of active development on the application and the availability of application experts.

This can be a time-consuming and challenging part of a remediation project. Techniques such as provisioning remote access to existing development systems or using virtual machines with installed development environments can help make this easier.

Fixing Technical Vulnerabilities
Fixes for technical vulnerabilities tend to be limited to tactical code-level changes that do not alter the behavior of the application under its normal usage scenarios. The Agree on Fix and Confirmation Methods step of the Planning phase should have resulted in agreed-upon remediation techniques, so at this point, there should be little debate regarding how specific technical vulnerabilities will be fixed. Validation and encoding libraries should have been selected, and developers should only need to locate the appropriate place for the code-level fixes and make the prescribed changes.

Applications that have only recently been subjected to a program of assessment often have large numbers of technical vulnerabilities due to poor coding practices being used consistently throughout the legacy codebase. Performing many of the same type of code changes can become monotonous and error-prone. Be wary of introducing new or more defects due to “cut and paste” issues.
Finding the proper location for code-level changes can vary in difficulty. If the vulnerabilities were identified by manual code review or code scanning, then specific information should be available showing the files and line numbers for the code responsible for the vulnerability. If the vulnerabilities were identified by application assessments or application scanning, then the vulnerability information will likely have vulnerability type, URL and potentially the parameter responsible for the vulnerability. In this case, it is the responsibility of the developer performing the fix to trace through the code in order to identify the proper fix location. This can be time consuming at first, but developers should become more adept over time as they learn the patterns of coding that lead to vulnerabilities. Use of tools such as integrated development environments (IDEs) are invaluable for these tasks and code scanning tools that integrate into IDEs can make remediation even simpler by allowing developers to navigate to the locations of vulnerabilities being remediated quickly.

DENIM GROUP TIP

Addressing many instances of the same type of issue can be monotonous for developers, so it can help to mix timed “sprints” of finding and fixing defects with other work. Spending two hours or an afternoon making similar fixes and then using the remaining time to work on more involved security vulnerability remediation or other feature development work can help reduce the monotony and decrease the chances that fixes will be applied improperly.

Fixing Logical Vulnerabilities

Logical vulnerabilities are based on the specific business logic and requirements of an application. As such, fixes for logical vulnerabilities tend to require application-specific fixes that take into account the particular business rules and authorization mechanisms of the application. In addition, the level of effort required to address logical vulnerabilities is more variable than that required to fix technical vulnerabilities because the developer performing the fix might need to address design issues or even architectural or requirements issues. Some fixes for logical vulnerabilities are simple and do not require changes to the behavior of an application for normal use scenarios, but others may require application users to enter different information, re-authenticate themselves, or otherwise change the way they use the application in order to have a resulting application that has successfully addressed the security issue.

Confirm Fixes and General Quality Assurance

Once the team applies the fixes it is important to confirm:

- Fixes applied successfully address the vulnerabilities
- Fixes have not broken previously working functionality

Testing to determine whether the security fixes applied successfully addressed the security issues should be based on the previously agreed-upon methods from the Agree on Fix and Confirmation Methods step of the Planning phase. For many vulnerabilities, this may involve re-running scanning tools to note whether or not the vulnerabilities that should have been fixed still appear in the scans. Other vulnerabilities may require repeats of previous manual testing to determine if the same
attacks are still successful.

Applications with a quality assurance (QA) program including comprehensive automated tests have an advantage when it comes to confirming that security fixes have not broken existing functionality, because these tests can be run to confirm that previous functionality is still working properly. Projects without automated tests typically must be manually tested in order to determine if the fixes have had an adverse impact on application functionality. Depending on the scope of the code that was changed during remediation, this could require a full, manual regression test of the entire application. Teams also may choose to create a suite of automated acceptance tests prior to or while embarking on the remediation effort in order to speed the re-testing of the application once fixes have been applied. These automated acceptance tests can be used to demonstrate that the remediation efforts did not break existing functionality.

Deploy the Updated Application to Production
Even the most elegant and technically sound of fixes are of no use until they are put into production properly. Once code-level fixes have been confirmed and the application is considered tested and stable enough to go into production, the remediated code needs to be turned over to the application operations team to be taken live. At this point, the fixes can be confirmed in production and the vulnerabilities can be considered successfully remediated. In addition, if some of the vulnerabilities were remediated by updating the configuration of the production application these configuration changes must be applied and should also be included in deployment process and documentation so that future deployments will not re-introduce incorrect configurations.

Remediation Metrics
Tracking at least rudimentary metrics on remediation projects is important for demonstrating the success of individual projects as well as in justifying future up-front activities to proactively increase the security of software being produced. Being able to articulate costs and levels of effort associated with code-level security fixes helps communicate the impact of these vulnerabilities to management and executives beyond merely appealing to fear, uncertainty and doubt (FUD).

Two key areas for tracking software security remediation metrics are:

1. **Calendar time** – Tracking the calendar time associated with the remediation of vulnerabilities demonstrates how long an organization knew they were exposed to risks as well as how effective the organization was in addressing those risks. Important dates to capture include:

   - Date when a vulnerability was identified
   - Date when a fix was requested
   - Date when a fix was implemented
   - Date when a fix was tested
   - Date when a fix was deployed to production

   Important dates can help organizations understand how long they were exposed to risk and how quickly they were able to address vulnerabilities.
• Date a vulnerability was identified
• Date a short-term mitigation – such as a WAF virtual patch – was put in place
• Date development on a fix was started
• Date development of the fix was completed
• Date the confirmed fix was placed into production

This data and the associated durations for each vulnerability can then be aggregated by vulnerability type and vulnerability severity. This allows security teams to make statements to management such as “It takes us an average of 75 days to fix ‘High’ severity application vulnerabilities.” Resources such as the WhiteHat Website Security Statistics Report (http://www.whitehatsec.com/home/resource/stats.html) can provide some metrics organizations can use to benchmark themselves versus emerging industry norms.

2. **Level of effort** – This involves the calculation of the hard costs as well as person-hours involved in the remediation project. Tracking the level of effort required to address vulnerabilities demonstrates the cost associated with building insecure software and fixing it after the fact. If outside vendors are used to perform fixes, then it is straightforward to attach a dollar value to those parts of the remediation efforts. Internal resources used to perform fixes and manage the associated projects should be captured as well. Aggregating costs by vulnerability type allows an organization to see not just the prevalence of certain types of vulnerabilities for different development teams, but also the cost associated with different vulnerability types. If the organization has also looked at the root cause that resulted in the vulnerability being introduced into the system, they can look for cost-effective ways to avoid introducing these vulnerabilities in the future and have financial justification for taking proactive action.

**Conclusions**

The security industry pays a tremendous amount of attention to finding security vulnerabilities. This is done via code review, penetration testing and other assessment methods. Finding vulnerabilities is only the first step toward actually addressing the associated risks and addressing these risks is the critical step in the vulnerability management process. Complicating matters is the fact that most application security vulnerabilities require code-level changes in order to successfully address the underlying issue and collaborating with the development teams adds to the overhead of remediation efforts. This paper has outlined an approach to treating application-level vulnerabilities as security defects as well as best practices for planning and executing successful software security remediation projects.

**Acknowledgements**

Many thanks to colleagues who reviewed and contributed to this paper including: Michael Anderson, Lara August, Chris Brown, Bryan Beverly, Aaron Copeland, John Dickson, Cap Diebel, Greg Leeds, Allyn McClendon, and Jarret Raim.

For more resources, visit www.denimgroup.com/remediation.