Agile Security Patching

Michael Hoehl
Agile Security Patching

GIAC (GCIH) Gold Certification

Author: Michael Hoehl, mmhoehl@gmail.com
Advisor: Rob Vandenbrink

Accepted: April 18, 2018

Abstract

Security Patch Management is one of the biggest security and compliance challenges for organizations to sustain. History reveals that many of the large data breaches were successful because of a missing critical security update. Further, the frequency and scope of patching continue to grow. This paper presents a new approach to security patching following Agile and NIST methodology.
“A goal without a method is cruel.”
William Edwards Deming

1. Introduction

Software Patch and Vulnerability Management continue to be a major challenge for many organizations. As shown in the figure below from US-CERT, the total number of patches continues to grow year over year with a spike in 2017 (USCERT, 2017).

![Figure 1: CVSS Severity Distribution Over Time (USCERT, 2017)]

There is no single software product or vendor source of these vulnerabilities. Organizations must consider patching at all levels of software. Only applying Microsoft Patch Tuesday updates to protect systems and data from cyber-attack is not sufficient. Organizations that were diligent with Microsoft patches avoided WannaCry related ransomware. However, flaws with Apache Struts and Intel Processors left organizations vulnerable to cyber-attack (e.g., Spectre and Meltdown). According to Flexera, “On average, over a five-year period, the share of non-Microsoft vulnerabilities has hovered around 78%, peaking at 88.5% in 2012. This high-level percentage plateau is significant and makes it evident why end users and organizations cannot manage security by focusing on patching their Microsoft applications and operating systems alone. If they do that, they are only protecting their computers and IT infrastructures from 22.5% - less than a quarter – of the total risk posed by vulnerabilities” (FLEXERA, 2016).

Author: Michael Hoehl, mmhoehl@gmail.com
Several software vendors have elected to stop providing individual patches each release period. Instead, separate and distinct patches are bundled in a roll-up model. The reason for this change is to prevent patch fragmentation that led to problems like dependency errors, lengthy scans, and testing complexity (MICROSOFT, 2016). This software vendor practice has created an all or nothing condition for customers in which selecting individual patches are no longer available. Further, software vendors are building these patch bundles in a monthly rollup manner. These patch bundles not only contain all the recently announced patches, but also the previously shipped patches. This cumulative update model is intended to improve security, quality, and reliability. However, with this model in practice comes the requirement for customers to perform extensive application program compatibility testing in a short period of time—especially when functionality and non-functionality (i.e., security) code changes are mixed in the update. The days of cherry-picking patches are over.

These trends are having a direct impact on IT by driving up the level of effort and number of resources necessary to implement security updates and patches. Further, the number of exploits continue to rise. Organizations of many sizes and industries are making front page news after a cyber-criminal has exploited a missing patch (e.g., Apache Struts vulnerability at Equifax). Lastly, the cyber-criminals seem to exploit these vulnerabilities faster and in more creative ways (e.g., EternalRocks, WannaCry, Petya, etc.). IT organizations are faced with the need to deploy more patches and in less time just to keep up with compliance, sustain security, and prevent service interruption.

Orchestrating patching is complex and costly. Patching has many dependencies including asset management, vendor notification tracking, risk assessment, patch preparation, QA, release management, communications, and auditing. As with the installation of any software update, many teams must collaborate to ensure success and avoid unintended interruption of service. If any of these teams are not resourced and prepared for this demand, then patches are not properly tested and announced prior to deployment creating availability and integrity risks. If patch deployment is delayed to perform necessary QA and communication, vulnerabilities linger longer for cyber-criminals to discover and exploit. Traditional operations and project management

Author: Michael Hoehl, mmhoehl@gmail.com
methods of patching are not nearly rapid enough. A new, rapid method of patch management is needed.

The Agile methodology is an iterative approach to software development and delivery. Agile is popular for software engineering and decision-making that demands fast results. Agile isn’t just for traditional software development anymore. Today, Agile is used for many purposes and in many ways. Agile has spread to other IT teams including Service Delivery and Operations. Even business process teams have embraced Agile across several industries. For example, “National Public Radio employs Agile methods to create new programming. John Deere uses them to develop new machines, and Saab to produce new fighter jets. Intronis, a leader in cloud backup services, uses them in marketing. C.H. Robinson, a global third-party logistics provider, applies them in human resources. Mission Bell Winery uses them for everything from wine production to warehousing to running its senior leadership group” (HBR, 2016).

Zero-day cyber-attacks, ransomware, data breaches, denial-of-service, and compliance are causing organizations to rethink their approach to patch management. Agile has a great reputation where time-to-value is essential. Therefore, Agile is ideal for IT and Security teams managing the implementation of critical security patches where rapid decision-making, preparation, and deployment are necessary. This paper does not define how developers code new patches using Agile. Instead, this paper explores how Agile can be used by organizations to rapidly deploy security patches for common commercial software (e.g., Microsoft Windows, Adobe Acrobat, Chrome, etc.) and open-source software (e.g., Apache, Tomcat, OpenSSL, etc.).

2. What is Agile?

For readers that are not familiar with the Agile framework, this section provides a brief introduction to Agile history and Scrum key components. The Agile concept was born in 2000. The software development paradigms of the time had the reputation of not meeting the demand for speed and customer satisfaction. A team of thought leaders met to discuss lightweight development methods. Together they published the Manifesto for Agile Software Development (Agile, 2001). This Manifesto proposed a new approach to development based on four value statements.
The Agile Manifesto inspired a new approach to software engineering and project management that focused on reducing time-to-value.

Since 2001, several versions of Agile have emerged including Scrum, Lean, KanBan, Extreme, Crystal, and others. The purpose of this paper is not to examine the various versions of Agile as these are already well documented. The Scrum approach is proposed for Agile implementation of patches. Scrum’s benefits of small team, frequent deployment, progress visibility, dynamic prioritization, strong collaboration, and ownership focus serve the demands of today’s patch management. A Scrum framework is visually depicted below:

![Scrum Framework](image-url)

Figure 3: Overview of Scrum Agile Development Methodology (Ghahrai, 2015)
The following table maps Agile terms to NIST SP800-40 terms used for describing patching:

<table>
<thead>
<tr>
<th>Agile Scrum terms</th>
<th>NIST SP 800-40 terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stories</td>
<td>Patches</td>
</tr>
<tr>
<td>Grooming</td>
<td>Patch Prioritization</td>
</tr>
<tr>
<td>1 Product Backlog</td>
<td>Remediation Database</td>
</tr>
<tr>
<td>2 Spring Planning Meeting</td>
<td>Selection of next patches to deploy</td>
</tr>
<tr>
<td>3 Spring Backlog</td>
<td>Patch Activity Definition and Sign-up</td>
</tr>
<tr>
<td>4 1-4 week Sprint</td>
<td>Patch Preparation and Testing</td>
</tr>
<tr>
<td>5 Sprint Review</td>
<td>Remediation Verification/Audit</td>
</tr>
<tr>
<td>6 Sprint Retrospective</td>
<td>Patch Lessons Learned</td>
</tr>
</tbody>
</table>

The Agile Scrum framework has essentially three major components: roles, artifacts, and events. Key roles include the Scrum Team, Scrum Master, and Product Owner. The Scrum Team is cross-functional (programmers, testers, QA, etc.) with 7-9 dedicated full-time members. The Scrum Master is a servant leader immersed in the activities of the team. This role is vital to keeping the team productive. This role is accountable for team alignment with Scrum values and practices. Additional Scrum Master duties include assuring dates and deliverables do not change (scope creep). The Product Owner represents the stakeholders and business requirements. The role defines feature deliverables and priorities.

Artifacts of Scrum include Stories, Backlogs and Burn Down Charts. Stories are a light-weight approach to define business requirements. Stories are intended to promote discussion. They provide a structured way for separate and discrete business needs to be conveyed so a technical team understands at a high level what work must be done. Stories focus on “what?” and “why?” of requirements—not the “how?”. Product Backlogs (identified as 1 in Figure 3 above) contain the list of prioritized work (Stories) to be performed. Sprint Backlog (identified as 3 in Figure 3 above) contains a list of tasks to complete during the Sprint that Scrum members select from. A Burn Down Chart shows progress with the work committed to by the Scrum Team within a Sprint and is updated daily.

Author: Michael Hoehl, mmhoehl@gmail.com
Major events (also known as Agile ceremonies) include Story selection, Sprint planning, Scrum daily meetings, Sprint review, and Sprint retrospective. The Scrum team selects from the Product Backlog a Story or Stories that can be completed within the next Sprint. A Sprint is a fixed amount of time typically 30 days or less (identified as 4 in Figure 3 above). The Product Owner, Scrum Master and Team all meet (identified as 2 in Figure 3 above) to plan the work necessary to fulfill the Sprint commitments (Stories). Scrum team members then collaborate to create task breakout and select tasks (identified as 3 in Figure 3 above) The Scrum Master and Team then meet daily to align status and actions. These daily standup meetings are typically at the beginning of the day and last 15-30 minutes. A review (identified as 5 in Figure 3 above) occurs toward the end of the Sprint to demonstrate the new functionality for stakeholder review and acceptance. After Sprint finished work is in place (value is realized), a retrospective meeting (identified as 6 in Figure 3 above) is held between team and stakeholders to discuss what worked well and opportunities for improvement.

3. Why Agile for Patching?

Agile is release-oriented and intended for conditions where value is required fast, frequent, and without fail. The advantages of the Agile approach for patching are:

<table>
<thead>
<tr>
<th>Iterative</th>
<th>Continuous feedback during each incremental change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>Daily standup meetings</td>
</tr>
<tr>
<td>Fast</td>
<td>Short 30 day sprints</td>
</tr>
<tr>
<td>Low Overhead</td>
<td>Low costs in terms of process and management</td>
</tr>
<tr>
<td>Visibility</td>
<td>Backlog shared with business and IT management</td>
</tr>
<tr>
<td>Adaptive</td>
<td>Retrospective meetings after each sprint</td>
</tr>
<tr>
<td>Cross-Functional</td>
<td>Operations, Infrastructure, QA, and Security teams</td>
</tr>
<tr>
<td>Flexible</td>
<td>Priority can change as new critical security patches released</td>
</tr>
</tbody>
</table>

A key principle of Agile is recognition that during a project, business requirements (including risk) change. Therefore, the Agile framework is built to maximize the ability to adapt to evolving requirements in rapid manner.

Author: Michael Hoehl, mmhoehl@gmail.com
Because the proposed Agile Scrum approach avoids “borrowing” team members each time a new patch is announced, it prevents the mad scramble to find free key resources when a critical security update is released. Scrum teams are dedicated and cross-functional, with all necessary skills represented to fulfill the Story commitments. Further, multiple Agile Scrum teams can work autonomously or be linked as part of a software “train”. If the volume of patches gets too high, an additional Agile team can be temporarily formed to reduce the backlog queue length. With security updates and enhancements, each train can be related to a common platform (e.g., RHEL, Windows, etc.) or environment (e.g., Enterprise Apps, Customer-facing Apps, PCI apps, etc.).

As each month passes, new patches are released that result in priority and requirement changes. The Agile framework is intended for this condition of dynamic requirements. A backlog of patch requirements is continuously groomed by Security and IT Operations to ensure work is properly defined and prioritized. The backlog also serves as a simple risk register for patching. The backlog is presented to the Agile team to advance patches in risk priority order. This approach ensures the team is always dedicating their limited time and resources to the highest risk security updates. Many software manufacturers release new security updates to the general public every month with the recommendation to implement within 30 days (e.g., Microsoft Patch Tuesday). The recurring 1-4 week “time-boxed” sprints align well with today’s typical vendor patch release cycle.

One of the biggest objections to patching is the risk of “breaking” existing application programs (e.g., financial applications, human resources applications, supply chain, etc.). A patch might target a single piece of software. However, this one software item (e.g., Java, Internet Explorer, etc.) might be used by multiple application programs. Unless testing occurs, patching might cause unintended service availability and data integrity issues with application programs. For example, patching Internet Explorer might cause a financial reporting application to present errors for key menu items within the browser, or Java patch also includes functionality changes that cause HTTP errors on the website. Further, many patches do not uninstall easily. Quality Assurance (QA) and customer testing are incorporated into Agile with test planning, tracking and reporting during the entire sprint. This continuous testing approach provides organizations the

Author: Michael Hoehl, mmhoehl@gmail.com
ability to identify patch compatibility issues early before any unintended harm occurs from updates and patches.

The National Institute of Standards and Technology (NIST) provides patching guidance with Special Publication 800-40 v2.0 *Creating a Patch and Vulnerability Management Program* (NIST, 2005). This guidance aligns well with the Agile Scrum approach. For example, NIST proposes a team called Patch and Vulnerability Group (PVG). PVG is a formal group that incorporates representatives from Information Technology Security and Operations. In many ways, the NIST PVG is similar to the Agile Scrum Team. Both the PVG and Scrum Team are cross-functional and self-organizing. The PVG includes a member that collaborates with stakeholders (e.g., Business Process Owners, Legal, Internal Audit, Operations, etc.) much in the same way as the Agile Product Owner. The Agile Product Backlog is used to communicate and prioritize stories (new product features). The Product Backlog is similar to NIST Database of Remediations that is used to communicate and prioritize patches. The Sprint review event confirms the Scrum Team achieved the overall goal of the Sprint as the Product Owner intended. This is similar to NIST Patch and Vulnerability Group (PVG) demonstrating to Security/Audit patch is in-place as committed. The Sprint retrospective event aligns with NIST SP800-40 recommendation to measure the effectiveness of the patch management program consistently and apply corrective actions as necessary. These are just a few of the aspects of NIST patching standard and Agile that are similar in purpose and practice.

4. Prerequisites for Agile Patching

For all organizations, the first step to operate patch and vulnerability management effectively is a patch policy. This policy provides the authority to advance patch remediation when there are conflicting IT priorities and limited resources. The policy must contain a few key components to be effective. The policy must clearly articulate what must be patched and target completion period. Data classification, regulation, asset value, redundancy, location, and business purpose are all factors that determine patch eligibility and timing. These all must be considered in the policy so that anyone that reads the policy draws the same conclusion on scope and timing. Procedures for
obtaining an exemption and who can authorize the exemption (and ultimately accept risk) must also be included in the patch policy. Other key policies include asset management, change management, and release management. Without these policies, organizations will not have the necessary information to know what system is eligible for a required patch and how the required software update is to be delivered properly. Lastly, a software lifecycle management policy is vital. The good news is software that is no longer supported by the manufacturer will have no more patches to deploy. The bad news is End-of-Life (EOL) software might have unknown vulnerabilities or exploits in the wild in which a remediation will never be made available. Patching can’t fix EOL vulnerabilities.

Risk assessment is necessary to determine the severity of security vulnerabilities, scope of the vulnerability, the likelihood of being the target of an exploit, and organizational appetite for risk duration. When conducting a risk assessment, it is important for organizations to consider the following factors (AustraliaDOD, 2016) that could lead to increased risk:

- high value or high exposure assets are impacted
- assets historically targeted are impacted
- patch was released outside of a vendor’s regular patch release schedule
- exploits related to a security vulnerability are wormable or can be automated

Patching demands a high volume of unique risk assessments as new patches are released seemingly every day. Further, risk assessment can require a significant investment of time and resources. Organizations are advised to adopt a uniform standard to evaluate and report patch risk levels across multiple products and vendors. Risk ratings can be acquired by an authoritative source (e.g., NIST, FIRST CVSS, FAIR, or OCTAVE), by software vendor (e.g., Microsoft DREAD), or patch management system (e.g., Ivanti or Flexera). These ratings are then evaluated and revised in the context of the organization and target systems.

NIST SP 800-40 advises, “Inventory the organization’s IT resources to determine which hardware equipment, operating systems, and software applications are used within the organization” (NIST, 2005). This aligns with the Center for Internet Security Critical Security Controls Top 20 (CSC, 2016) in which inventory of authorized devices and

Author: Michael Hoehl, mmhoehl@gmail.com
software are top priority controls. Having an accurate inventory is necessary for many phases of patch management. First, the software distribution infrastructure requires the asset inventory to confirm agents and access are in place to deliver the software patches. Next, when the patch is announced by the vendor, an initial risk assessment requires asset inventory to determine how pervasive the vulnerability is within the organization. The same inventory is necessary to scope and prioritize order of patch delivery. Management reporting on progress and residual risk will require the asset inventory. Lastly, Audit and Regulatory Compliance teams require the asset inventory to select test samples and confirm the operational effectiveness of control.

NIST advises organizations to implement infrastructure to perform automated deployment of patches to allow rapid update of hundreds or even thousands of systems (NIST, 2005). Prior to any patch notification, this infrastructure must be in place to receive the software vulnerability remediation package and deliver to the collection of systems that are eligible for the vendor code fix. This includes updating, removing, configuring, and disabling software. Ideally, the software deployment infrastructure is integrated with the asset management system. Lastly, most organizations have a mix of technology in place. This makes it impractical to expect a single product to be able to deliver all patches (and software) to all assets. Therefore, Security, Application Development, Operations, and Infrastructure teams should collaborate to establish an enterprise-wide approach to automate software delivery that covers all technology. This is a key strategic and tactical conversation that should occur independent of any specific vendor patch notification and occur repeatedly.

5. How to implement Agile Patching - People

For patch management to operate continuously and effectively, organizations must consider people, process, and technology. The next three sections of this paper explore these areas as they apply to Agile Patching. This section focuses on people, roles, and organization.

Agile Team and PVG (NIST Patch and Vulnerability Group) roles map well together. As mentioned earlier, Agile has essentially 3 unique roles. The Agile roles are mapped to NIST patching framework in the table below.

Author: Michael Hoehl, mmhoehl@gmail.com
### 5.1. Scrum Team = NIST Patch and Vulnerability Group (PVG)

Scrum Teams are not assembled and then disassembled as each new project is approved. New work is presented to teams instead of new teams presented to work. A Scrum Team typically remains together through multiple projects (even multiple years). This approach optimizes staff performance and product quality. Patching gains these same benefits with a dedicated team having a common mission to rapidly implement patch remediation and predictably reduce risk. NIST SP800-40 identifies this team as the Patch and Vulnerability Group (PVG). PVG is a dedicated team that incorporates representatives from Security, IT Operations, and QA. According to NIST, the burden of implementing and testing patch remediations should be shifted from local IT system administrators to the PVG. The reason for this cross-functional, dedicated team approach is to avoid overwhelming local IT system administrators. Except for small IT environments, it is a complex and difficult endeavor for local IT administrators to perform all patch and vulnerability remediation in a timely manner (NIST, 2005). PVG is proposed to overcome this issue.

As mentioned earlier, Agile is not exclusively for software development. When Scrum is applied to patch and vulnerability management, the majority of members are not developers. Security, IT Operations, and QA represent the majority of the PVG membership. With PVG, there is very little coding done. The coding is done substantially by the software manufacturer (e.g., Microsoft, Adobe, etc.). However, coding in the form of scripting (e.g., shell scripts, Microsoft PowerShell, etc.) and installation packaging is often necessary for patch automation. Therefore, there is a
requirement for software engineering expertise within the PVG. The Security representative confirms IT assets and software eligible for patch, effectiveness of patch, and order of deployment. The IT Operations representatives in the Scrum lead software packaging, scripting, and deployment efforts. The QA representatives in the Scrum confirm the patches do not break applications that reside on the target systems. Testing is an integral part of the Agile framework and necessity for PVG.

An example organizational structure for Patch and Vulnerability Group (PVG) is provided below:

![Proposed Agile Patching Organization](image)

**Figure 4: Proposed Agile Patching Organization**

It is common for large organizations practicing Agile to have multiple Scrum teams. Just like with Agile, large organizations may need to have several PVGs. These PVGs can work together in a confederation or could be structured hierarchically with an
authoritative top-level PVG (NIST, 2005). This structure is also common for Agile and is known as Scrum of Scrums.

5.2. Scrum Master = Patch Service Owner

A Scrum Team can face many hurdles on their journey thru each Sprint. External interference (e.g., IT operational incidents, other projects competing for the same resource, etc.); team confusion about objectives; conflicting stakeholder demands; and, poor meeting etiquette all threaten progression of Sprint. A team member is necessary that represents the management of Scrum to sustain the principles and pace of Agile. This is the purpose of the Scrum Master. The Scrum Master facilitates communication, builds consensus, enables cooperation across all roles and functions. In addition, the Scrum Master embraces innovation and is a change agent by welcoming continuous improvement. Just as the Scrum Master keeps the Agile Team focused, the Patch Service Owner does for PVG. This role’s primary focus is to ensure PVG practices are understood and followed to achieve the patching objectives. The Patch Service Owner and Security Manager collaborate to ensure the right patches are being advanced in the right order. In addition, they come to an agreement of Definition of Done (DoD). DoD is the acceptance criteria for a Story and is used to assess that a Story has been completed as required. This is vital as exemptions, resource limitations, and timing all can affect the outcome. Patching can face many impediments including change freezes, end-of-life software, and dependencies on surrounding people, process, and technology. The Patch Service Owner works with the Security Manager (like Scrum Master works with Product Owner) to overcome these challenges so that they do not affect the Patch and Vulnerability Group (PVG) performance. It is important to understand that the Patch Service Owner (like Scrum Master) is not the staff manager of the PVG. The Patch Service Owner ensures the patching process is followed and understood—but has no formal authority over the PVG like a traditional staff manager. Separate staff managers are still required for each member of the PVG. The staff managers plan resourcing and rotation of the PVG (Scrum) members so that there are no key functions (i.e., Security, IT Operations, etc.) missing or unavailable for Sprints to advance. The Patch Service Owner facilitates collaboration and tries to drive decision-making at the team level—not staff

Author: Michael Hoehl, mmhoehl@gmail.com
manager level. Lastly, the Patch Service Owner drives cadence so that the PVG is all in step together and advancing in the right direction. Ensuring the PVG is functional and productive is a key duty of Patch Service Owner. If the PVG is struggling with communication, coordination, blockers, or Definition of Done, the Patch Service Owner is responsible for applying Scrum-like disciplines and practices to help PVG advance predictably.

5.3. Product Owner = Security Manager

In Agile, the Product Owner is responsible for managing relationships with stakeholders and Product Backlog grooming. This includes defining new features, prioritization, scope, and timing. The PVG Security Manager is equivalent to the Agile Product Owner. PVG Security Manager monitors vendor patch notifications, performs preliminary risk assessment, leads discussion with stakeholders, identifies dependencies, and considers compensating controls. Each patch remediation is equivalent to an Agile Story. Each story contains the risk, remediation guidance, scope, timing, dependencies, and Definition of Done. The results of this effort are captured in what NIST defines as a remediation database (NIST, 2005) that is similar to the Agile Product Backlog of Stories. As new vulnerabilities are identified and patch remediations are available, grooming is required of the Remediation Database. What was priority #1 yesterday, might no longer be today. The Security Manager considers the appropriateness of exemptions and engages all necessary stakeholders to accept risk. At the end of each Sprint, the PVG Security Manager leads a review of the Sprint deliverables to confirm vulnerability has been properly mitigated. Metrics are discussed and possible sample testing is performed. The Security Manager might then engage Internal Audit for additional control testing. Lastly, the Security Manager participates in a lessons learned meeting with the PVG after every sprint. This is equivalent to Agile Retrospective Review and is intended to sponsor continuous improvement of the patching process.

5.4. Additional PVG members

Depending on the size of an organization, the PVG might benefit from additional members not typically found in the Agile model. For example, an individual responsible for communications might be appropriate. This individual could routinely collaborate
with change and release management to obtain approval for patch deployment. Out-of-band or emergency patches might demand a communications representative from PVG to explain roles and approach to remediation. Employee communications might be necessary when software is being intentionally removed to remediate a serious vulnerability or when patching requires manual intervention (e.g., restarting computer). Technical documentation and instruction guides might be necessary for IT local system administrators so that they don’t unintentionally remove patch remediations.

Certification and Accreditation procedures might need updated reflecting the new mandatory patches that must be in place prior to production deployment of new systems. Lastly, this individual might share lessons learned with other teams that are also performing software distribution (e.g., new version upgrades of Microsoft Office).

Another member to consider for PVG is a representative from the software distribution infrastructure team. The scope of the PVG does not include the “weeding and feeding” of the infrastructure used to deploy software. PVG is not intended to build a software deployment solution or ensure new systems have properly operating agents. Patches and security updates must blend into established software lifecycle management processes and technology standards. This includes the technology platform used to deliver the software or patches. The PVG is a customer of the automation technology platform used to deploy software—not owner or administrator. As mentioned earlier, prerequisites for patching include asset inventory management, configuration management, and automated software deployment infrastructure. These IT services are important and must to be in place and operating effectively independent of PVG. As these IT services are shared, having a representative from these external teams can help to avoid resource collisions and patch deployment delays. These representatives can speak to current non-patch software deployments in the pipeline, readiness of infrastructure to deploy software patches, patch eligibility discovery, and gaps between what needs to be patched and what can be patched automatically.

6. How to implement Agile Patching - Process

Now that the key roles have been identified, this section proposes the necessary processes for patching using Agile Scrum framework.

Author: Michael Hoehl, mmhoehl@gmail.com
6.1. Stories = Patches

As mentioned earlier, customers define features and requirements for code development at a high level in Agile using Stories. Risk, dependencies (also known as surround), and initial estimate of effort are also included in each Story. The Scrum team then selects Stories to advance in 1-4 week Sprints. For NIST SP800-40 Patch and Vulnerability Group (PVG), each Story represents a patch remediation. The Security team communicates through these Stories the requirements necessary for successful patching including initial risk assessment, system eligibility, vendor instruction, location of patch code, priority, scope, timing, exemptions, and dependencies.

6.2. Product Backlog = Remediation Database

Agile development planning and project requirements are defined using a collection of stories within the Product Backlog. The Agile Product Backlog is equivalent to NIST SP800-40 proposed Remediation Database. As the Agile Product Backlog conveys to development teams what development needs to be done and when, the NIST Remediation Database conveys to IT what patching needs to be done and when. As new patches are released from vendors, a risk assessment is performed by the Security Manager (Agile Product Owner). Once high-level requirements for patching are determined, the patch and associated information is added to Remediation Database (i.e., Agile Story representing the new patch is placed in the Product Backlog). The NIST Remediation Database is a centralized collection of instructions for removing vulnerabilities by installing patches, changing configuration items, or performing workarounds (NIST, 2005). The information found in the Remediation Database is often input to common IT processes such as change and configuration management.

6.3. Grooming = Patch Prioritization

The Agile Product Backlog is dynamic as new customer demands and features are identified with new Stories. Because of this, Stories in the Product Backlog are routinely examined by the Product Owner to ensure that the most important features are being advanced first for development. The patches listed in the NIST Remediation Database also require this continuous grooming by the Security Manager as new patches are released frequently. New patches often change work priority and effort. This Agile

Author: Michael Hoehl, mmhoehl@gmail.com
approach with grooming stories ensures all IT teams are on the same page with new business conditions, new requirements, and appropriate priority of feature development. Patching must have this same recurring grooming practice to ensure all IT teams are on the same page with evolving risk conditions, new patching demands, and appropriate priority of patch remediation.

6.4. Sprint Zero = Infrastructure and Organization Preparation

Many organizations conduct a Sprint Zero event before formally beginning an Agile project. The purpose is to work out environmental, organizational, and technical issues prior to the code development. This way the Scrum project work is optimized and value is maximized. Prior to any patching being performed, there is the same need to prepare environmental, organizational, and technical issues. Sprint Zero is essential for PVG and Agile Patching. Scrum team member rotation and organizational changes can occur during this event. All technologies and standards used by the PVG are examined during this event to ensure they are ready when called upon for patch preparation and deployment. During this event, the IT Operations team confirms that the automation tools are working as intended and prepared for deployment of software to computer assets. If the software deployment infrastructure is only partially ready, then the patches will be deployed to only a fraction of the intended targets leaving unintended residual risk. Further, during Sprint Zero the Security team confirms that vulnerability scanners are working as intended to identify configuration and code weaknesses. Vulnerability Scanning is necessary for confirming the patch remediation design is effective during QA and in-place after deployment to PROD. All of these actions during Sprint Zero are independent of any new patches being released. Essentially Sprint Zero for PVG means keeping the organization ready for patch demands. Sprint Zero is blended into the Scrum lifecycle. Scrum practitioners either include Sprint Zero before starting a Sprint or dedicate an entire Sprint.

6.5. Sprint Execution = Patch Preparation and Testing

Once a story or stories have been selected by the Scrum team, a Sprint begins. During the Sprint no new stories can be introduced and the functionality requirements may not change. During the Sprint, the Scrum team is dedicated and works uninterrupted
on a single Sprint. The Scrum team breaks down the Stories into tasks necessary to complete within Sprint. These tasks are posted within the Sprint Backlog. Scrum team decides on the actions, dependencies, and time required to complete the tasks. Scrum team members then select and commit to advance these tasks from the Sprint Backlog. Daily Standup meetings keep the team in sync with what is to be accomplished. A Burn Down Chart is used by the Scrum Master (Patch Service Owner) to track progress with tasks as all Scrum team members work toward a common completion date.

Sprint Execution is equivalent to Patch Preparation and Testing. Vendor requirements for patching are broken down into common tasks. The Scrum team then estimates effort (typically in units of days) to complete patch task. Each member of the team selects patching task(s) from the Sprint Backlog that they are committing to advance. Daily standup meetings occur to answer what was completed yesterday; what will be done today; and, are there any blockers. The daily standup is not intended to be a brainstorming session for problem-solving. It is for communicating the problem to all and identifying the necessary resources to overcome blockers. Separate breakout sessions during the day will focus on the problem and identify the necessary problem busters. PVG member attendance is mandatory for these 15-30 minute meetings to enable optimum communication and collaboration. Other teams are also welcome—though only PVG members are to be primary communicators. If too many questions are asked of the PVG instead of the PVG leading discussion, a Communication representative should be invited to capture daily standup discussion items (meeting minutes) and disseminate updates to Management. This is especially important for emergency or critical patches. After the daily standup, the Scrum team then advances the commitments for the day. This includes creating patch packages, establishing eligible target assets, testing patch compatibility with application programs, verifying compensating controls, scanning for vulnerability, processing exemptions, and preparing communications.

6.6. Sprint Review = Patch Remediation Verification / Audit

When a Sprint is for developing functionality improvements and new features, a demonstration called Sprint Review is done to show how Scrum team delivered on its

Author: Michael Hoehl, mmhoehl@gmail.com
commitments. The Product Owner and Stakeholders can observe the new features at the Sprint Review and confirm the new code is working as intended. This Sprint Review ceremony must also occur for patching. The challenge for the Scrum team is how to credibly demonstrate non-functionality improvements like code fixes or security patches. Definition of Done is critical for non-functionality requirements. As the “new features” of patches are not easily observable, the DoD provides the only means of qualifying the work is complete as committed.

For patching, the following must be addressed during the Sprint Review:

- Patches included in scope
- Collection (systems in scope for patch)
- Compensating Controls and Approved Exemptions
- Necessary surround changes and dependencies
- QA results (application program capability)
- Vulnerability Scan results

The Patch Process Owner should lead this conversation with support from the Security Manager. Also, blockers that were discovered during the Sprint should be discussed when relevant to Release Management. The Burn Down Chart should be presented so that Management has a better understanding of task estimate accuracy and baseline for the duration of time (velocity) to prepare patches for release. This is insightful for emergency or out-of-band critical patch events.

The Sprint Review artifacts are critical to demonstrating the continuous and operational effectiveness of patching. Auditors (e.g., Internal Audit, PCI Qualified Security Assessor, etc.) may require samples of the Sprint Review to satisfy PCI Data Security Standard Requirement 6: Develop and maintain secure systems and applications (PCI, 2016). These same artifacts might be requested by application program vendors when the security patch causes unexpected results identified during QA.

### 6.7. Release Management = Release Management

Once the Sprint Review is completed and the Security Manager has accepted the work, the patch package is ready for release. This is similar to Agile practices for software development. The Scrum team does not actually implement the code—they
only create it. The PVG does not actually implement the patch—they only prepare it. Ideally, the organization already has a software distribution infrastructure or release management automation in-place. The patch packages are then blended with the other software lifecycle management activities. This approach avoids conflict by having competing software distribution standards.

6.8. Sprint Retrospectives = Patching Lessons Learned

One of the major advantages of the Sprint methodology is continuous learning and improvement. Discussions about what is and what is not working are held after every Sprint. Eligible topics include practices, estimates, communication, recurring blockers, environment, tools, etc.. The whole team participates. Short, iterative Sprint cycles allow improvements to be incorporated easily and quickly. The PVG lessons learned from patching are also valuable for software lifecycle management. The benefits realized by optimizing patching also can be realized directly with general software updates (e.g., software packaging) and indirectly (e.g., disaster recovery test scripts).

6.9. Agile Patching Sprint and Release Calendar

Below is a proposed calendar of events based on the Agile Patching approach.

Author: Michael Hoehl, mmhoehl@gmail.com
7. How to implement Agile Patching – Technology

This paper is intended to provide helpful guidance on the subject discussed. Any mention of product or service is not to be interpreted as vendor recommendation or endorsement.

7.1. Agile Toolkit Essentials

There are many tools and technology that are commonly used by Agile practitioners that also provide great value with patching. There are three key types of tools that are essential with Agile: Product Backlog, Sprint (iteration) Backlog, and Burn Down Charts (RUBIN, 2013). These serve as the hammer, screwdriver, and wrench in the Agile toolbox. If your organization has these Agile tools in place, consider adopting these for Agile Patching. This provides economy of mechanism and community support.

Author: Michael Hoehl, mmhoehl@gmail.com
to get started quickly. If Agile is not in practice, this section provides a general orientation about the three essential Agile tools.

As mentioned earlier, Agile work begins with the Product Backlog. Teams should begin with identifying software that helps the Product Owner manage all the Stories and convey them to the Scrum team. Each Scrum Product Backlog has certain properties that differentiate it from a simple to-do list (SCRUM INSTITUTE, 2017):

- an entry in the Scrum Product Backlog always add value for the customer
- the entries in the Scrum Product Backlog are prioritized and ordered accordingly
- the level of detail depends on the position of the entry within the Scrum Product Backlog
- all entries are estimated
- the Scrum Product Backlog is a living document
- there are no action-items or low-level tasks in the Scrum Product Backlog

Every patch cannot be priority #1, so a Product Backlog is vital for the Security team to convey the order of patch remediation to the IT teams (and Scrum). Therefore, for Agile Patching, the Product Backlog is equivalent to the NIST Remediation Database. There are many commercial and open-source Agile technologies to manage Product Backlogs. Popular examples of technology include Atlassian Jira and Scrumpy.

Once stories (patches) have been selected by the Scrum (PVG) from the Product Backlog, further decomposition is done to identify action items and define all necessary tasks. Scrum members then chose from the tasks to be advanced. Though the tasks vary between Sprints for more conventional Agile practices, for this patching approach the tasks are substantially the same for each Sprint. Therefore, a time reporting system or project management system that visually tracks activities, effort, duration, progress, and resources might be sufficient. If the Sprint tasks are not so predictable or task execution varies from Sprint to Sprint, several technologies are available for Sprint iteration planning and activity management. These include Atlassian Trello and TargetProcess.

A Burn Down Chart is a very helpful way to visually track planned versus actual progress on Sprint activities. If blockers are preventing team members from fulfilling commitments on time, the Burn Down Chart reveals this quickly and historically. The

Author: Michael Hoehl, mmhoehl@gmail.com
Burn Down Chart reveals what specifically is affecting velocity of the Scrum team. Conditions including scope change, optimistic forecasting, and resource starving can all be identified by the Scrum Master with a Burn Down Chart during the Sprint as well as part of Lessons Learned for future Sprints. In many cases, these charts are available as part of the aforementioned project management information system. There are also stand-alone and add-on products that offer Burn Down Charting including Tableau and Artezio.

7.2. Additional Tools for Agile Patching

In addition to the Agile toolkit essential items mentioned above, there are some additional technologies to consider for Agile Patching. The QA team will depend on technology for several purposes including software packaging, deployment, and test automation. Most organizations have a “living” technology ecosystem that is continuously growing and changing. New software, new versions of software, new configurations of software, etc. all demand flexible QA/Test lab that can rapidly change to represent these changes as well as the new patches that are intended to be deployed. For this reason, many organizations should consider virtual technology and sandbox environments.

Much in the same way the QA team automates functionality testing for new releases of software, test automation technology should be considered forpatching. One of the major objections to patching is the concern that the security update will “break the app”. Though some organizations employ early adopters of the patch to identify software compatibility problems, the feedback from this approach to testing can be slow and unreliable. Having automation in place to perform scripted testing of essential application functions is ideal. Consider partnering with the QA Team for process and technology implementation. Modern software changes frequently, so maintaining the testing scripts can be a real challenge. Though technically the QA Team’s primary focus is functionality testing, they are a great partner when there is a risk that non-functionality updates (patches) could affect the software functionality unintentionally.

Vulnerability Scanners can serve as an important quality assurance tool, too. For several patches, there are post-implementation activities such as restarting the system.
before the patch remediation is effective and vulnerability removed. Once the patches are deployed, an authenticated vulnerability scan of the target systems can confirm the software update is in place as intended. This QA is especially valuable during the early phases of patch deployment preparation to ensure the patch package script executes all necessary actions for vulnerability to be removed. Further, the Vulnerability Scanners can determine if multiple instances of the vulnerable software are present. For SDK and run-time engines (e.g., Java), the binaries that need to be patched might be embedded in multiple locations within the filing system. This is because the application install package/script might include third-party binaries that are part of the entire application composition. This software “surround” can be missed by the patch package script if the location of the binary is not well known, leaving the target system with residual vulnerabilities that were intended to be patched.

A summary of common technology helpful for Agile Patching is provided in the table below:

<table>
<thead>
<tr>
<th>Product</th>
<th>Providers</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patch Alerting</td>
<td>US CERT Flexera</td>
<td>monitors vendor patch notifications</td>
</tr>
<tr>
<td>Product Backlog / Remediation Database</td>
<td>Jira Scrumpy</td>
<td>conveys to IT what patching needs to be done and when</td>
</tr>
<tr>
<td>Sprint Backlog</td>
<td>Trello TargetProcess</td>
<td>identify Sprint action items and define all necessary tasks</td>
</tr>
<tr>
<td>Burn Down Charts</td>
<td>Tableau Artezio</td>
<td>visually track planned versus actual progress on Sprint activities</td>
</tr>
<tr>
<td>Risk Register</td>
<td>RSA Archer IBM</td>
<td>tracks assessments, compensating controls, and exemptions</td>
</tr>
<tr>
<td>Test Automation</td>
<td>CA Selenium</td>
<td>when manual testing is too slow</td>
</tr>
<tr>
<td>Vulnerability Scanner</td>
<td>Tenable Rapid7</td>
<td>confirms patch is in place and audit</td>
</tr>
</tbody>
</table>

8. Conclusion

Patching is complex, people expensive, and time-consuming. Many resources are needed to prepare and deploy critical security updates. Unfortunately for most security updates, no new software functionality is being introduced. Management might perceive

Author: Michael Hoehl, mmhoehl@gmail.com
this activity as having a high cost but no or limited tangible benefit. The cost of deploying the patches is not typically funded with new revenue streams or consumer markets. Further, the liability of not rapidly deploying patches can be significant. Risk is transferred from the software manufacturer to organization/individual when a defect repair is not implemented in the manner and timeline required by the manufacturer. Lastly, the requirements and priorities for patching are quite fluid. Organizations want patching that is fast, frequent, and without fail. By combining NIST and Agile methodologies, organizations have a new approach for patching that saves time, optimizes resources, reduces risk rapidly, and increases credibility.

9. References


Author: Michael Hoehl, mmhoehl@gmail.com


Author: Michael Hoehl, mmhoehl@gmail.com
<table>
<thead>
<tr>
<th>Course Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Canberra Spring 2019</td>
<td>Canberra, AU</td>
<td>Sep 02, 2019 - Sep 21, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich September 2019</td>
<td>Munich, DE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Oslo September 2019</td>
<td>Oslo, NO</td>
<td>Sep 09, 2019 - Sep 14, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Network Security 2019</td>
<td>Las Vegas, NVUS</td>
<td>Sep 09, 2019 - Sep 16, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dubai September 2019</td>
<td>Dubai, AE</td>
<td>Sep 14, 2019 - Sep 19, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Raleigh 2019</td>
<td>Raleigh, NCUS</td>
<td>Sep 16, 2019 - Sep 21, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Rome September 2019</td>
<td>Rome, IT</td>
<td>Sep 16, 2019 - Sep 21, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>Oil &amp; Gas Cybersecurity Summit &amp; Training 2019</td>
<td>Houston, TXUS</td>
<td>Sep 16, 2019 - Sep 22, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Bahrain September 2019</td>
<td>Manama, BH</td>
<td>Sep 21, 2019 - Sep 26, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Osaka September 2019</td>
<td>Osaka, JP</td>
<td>Sep 09, 2019 - Sep 14, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Network Security 2019</td>
<td>Las Vegas, NVUS</td>
<td>Sep 09, 2019 - Sep 16, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dubai September 2019</td>
<td>Dubai, AE</td>
<td>Sep 14, 2019 - Sep 19, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich September 2019</td>
<td>Munich, DE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels September 2019</td>
<td>Brussels, BE</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Ottawa September 2019</td>
<td>Ottawa, CAUS</td>
<td>Sep 02, 2019 - Sep 07, 2019</td>
<td>Live Event</td>
</tr>
</tbody>
</table>