

AGILE INFORMATION N SECURITY

Using Scrum to Survive In and
Secure a Rapidly Changing Environment



JAMES R. FITZER

Agile Information Security

USING SCRUM TO SURVIVE IN AND SECURE A
RAPIDLY CHANGING ENVIRONMENT

James R. Fitzer

DALLAS, TEXAS

Copyright © 2015 by James R. Fitzer.

All rights reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other academic or noncommercial uses permitted by copyright law. For permission requests, write to the publisher at the address below.

James Fitzer
james.fitzer@agilesecuritybook.com

Book Layout ©2013 BookDesignTemplates.com

Ordering Information:

Quantity sales. Special discounts are available on quantity purchases by corporations, associations, and others. For details, contact the “Special Sales Department” at the address above.

Agile Information Security/ James R. Fitzer. —1st ed.
ISBN 978-1511804240
ISBN: 9781483556185

Contents

Foreword

Introduction

Who Should Read This Book

What is Agile, Anyways?

Why Agile for IT Security?

Case Study Introduction: NBID

The Agile Philosophy

Agile Principles

Obstacles to Agile and Overcoming Resistance

The Agile Team and Building Relationships

NBID: Cross-Team Collaboration

Scrum for Information Security

Introduction to Scrum

The Product Backlog

Using Stories

Sprints

The Daily Scrum (or Stand-up Meeting)

Guidelines and Structure of a Scrum

Inter-team Meetings (The Scrum of Scrums)

NBID: The Daily Scrum and IT Staff

Security Risk Assessments

Back to the Basics

Basic Risk Assessment Reporting

The Agile Risk Assessment Process

The Agile Risk Assessment Report

What is Acceptable Risk?

NBID: Risk Assessment – A Seething Pit of Despair

Compliance and Agility

Regulations and Agility

Turning Standards into Stories

Metrics, Dashboards, and Automation

NBID: Balancing Agility and Compliance

Implementing Controls, Prevention, Monitoring, and Response

Vulnerability Management

Monitoring and Response

Unintended Consequences and Enforceable Controls

Prioritizing Vulnerabilities in an Agile Environment

Preproduction Systems Security

NBID: When Information Security and Operational Requirements Collide

Final Thoughts

[Share This Book](#)

[Origins of This Book](#)

[Return to the Basics](#)

[Be a Servant Leader](#)

[Constant Improvement](#)

[Bibliography](#)

Foreword

When I first met the author nearly 15 years ago, my first impression could be summed up with just two words: “Leadership challenge.”

He was a walking dichotomy; a towering tattooed guitar player that covered up a hidden nerd. I had no clue that this barely 18 year-old kid had already acquired industry recognized IT certifications, or that he would become one of my greatest successes as a leader, and one of my closest friends.

Over the next few years in the Army, I watched James grow from an unruly but somehow professional young Soldier into an outstanding example of what an Army non-commissioned officer should be. His eye for detail and desire to get things right the first time made for a winning formula, when combined with his genuine care for people and desire to help those in need. He went on to serve as a Drill Sergeant, a role of vital importance, as it is the first impression young Soldiers have of the army. And he excelled at it.

When James left the Army, he expressed an interest in restarting his IT career. With my own IT management career in full swing, I was able to offer him a position on my help desk in Washington D.C.

Can you tell I’m a glutton for punishment?

It was obvious that he was destined for a larger role from the very beginning. He managed to resolve several problems that our infrastructure team had been unable to fix, some of which had existed for several years. This was the first time that I really got to see James’ research abilities in action.

If you’re still reading this, you’ve noticed a common theme: Attention to detail, understanding and interpreting requirements, and thorough research. James’ ability to remove the “fat” from what the customer is asking for, hold jargon-free conversations with the business, and his dedication to quality research will be on display as you read through these pages.

I’ve set a pretty low bar for you here, so obviously it has to get better after this.

Right?

Thank you for reading, and please enjoy this book.

Stefan Still

Vice President of Service Delivery – Enstar US
Friend, Former Squad Leader, Manager, and Partner in Awesome
Tampa, FL

This work is dedicated first and foremost to my wife and daughter, whose support and love have kept me sane. And to the developers, management, and my team of engineers at the Northern Border Integration Demonstration, whose advice and hard work made this book possible.

The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts.

—Dr. Eugene H. Spafford, Executive Director, Purdue University Center for Education and Research in Information Assurance and Security (CERIAS), as quoted in A.K. Dewdney, “Computer Recreations: Of Worms, Viruses, and the Core War,” *Scientific American*, March 1989, 110.

CHAPTER 1

Introduction

One of the chief difficulties of the information security discipline is the need to balance the business or product with information security and assurance goals. All too often, particularly in government and financial sectors, information security personnel, developers, and system administrators and engineers are seen as competing interests, with divergent goals. This problem is compounded by the increasing prevalence of agile software development methodologies, which are seen by outsiders as a way to circumvent established processes with respect to system security and stability. Unfortunately in many environments, this misconception about agile development has resulted in an increase in the divide between development staff and the system engineers, security administrators, and IT staff charged with supporting their systems.

Rectifying this divide requires a return to the basics of information security and protection, from a philosophical perspective. The core goal of information security is summed up quite well in *Information Security Fundamentals*, by Thomas Peltier, Justin Peltier, and John Blackley:

Information protection should support the business objective or mission of the enterprise. This idea cannot be stressed enough. All too often, information security personnel lose track of their goals and responsibilities. The position of ISSO (Information Systems Security Officer) has been created to support the enterprise, not the other way around.¹

Their text is used in information security coursework throughout the country, and the statement above should be viewed as a guiding principle for all information security professionals, who need to understand that operational requirements and security guidelines can collide. Highly visible breaches, advancement of regulations and laws for the protection of data, and the explosion of internet-connected devices and services have brought an enormous amount of attention to the field of information security, resulting in not only a greatly increased focus on countermeasures, but “bureaucratic bloat” within organizations, particularly within the Department of Defense, where information assurance (IA) often ignores that very basic first tenet of supporting the mission rather than becoming it.

During my time at the Northern Border Integration Demonstration (NBID), I witnessed first-hand the clash between security standards and mission focus, between endless documentation and rapid change. While these problems were significant, they were not insurmountable, and over the years the NBID team found workable solutions, by distancing ourselves from traditional mindsets and working agility into our security process.

With all the books and articles on agile development, Scrum, and the larger topic of information security, you may find yourself wondering why this book matters.

Who Should Read This Book

This book is intended for information security engineers, information assurance officers, system engineers, or anyone who has been thrust into the arduous task of securing and maintaining a rapidly changing system. Throughout my experiences at the Northern Border Integration Demonstration, the challenges of maintaining adequate security posture (and complying with tomes of government regulations) became clear; when there's a new software build every few weeks, those charged with maintaining the system's security are always behind the power curve.

This book is not meant to be an exhaustive source of information on agile development, Scrum, or information security in general. The reader should feel free to take what's useful and discard the rest. This isn't meant to be a method strictly adhered to, but a rough framework and collection of ideas that you should modify as you see fit.

It is my hope that this book prevents you from experiencing the conflict and frustration I felt initially when being asked maintain security compliance on NBID. You will find sprinkled throughout this book lessons and anecdotes from that project that have led to the ideas and methods I present herein.

What is Agile, Anyways?

Agile development, simply, is a big community of software development methods in which software is produced incrementally, by cross-functional teams, with a philosophy that relies first and foremost on producing working software. The Agile philosophy promotes flexibility and the ability to quickly respond to change.

Although concepts related to agile development are not new, the origin of the agile "movement" is owed to a group of developers who got together in Utah in 2001 to discuss approaches to managing software development projects. The result of that gathering became known as the Agile Manifesto. It reads:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.²

From that simple statement, many methods and frameworks have sprung forth, offering solutions to management and organizational headaches that have plagued software development since the beginning. The authors enumerate 12 principles behind the Manifesto, which include some useful concepts for information security:

- Early, continuous delivery of a working product, with short intervals between releases.
- Cross-functional, self-organizing teams produce the best requirements, architectures, and designs.
- Face-to-face communication.
- Simplicity (the art of maximizing the amount of work not done)
- Regular reflection on how to become more effective, and adjustments to behavior and processes.

Another vital part of agile development is often referred to as a “feedback loop,” which may not be a term that information security professionals are familiar with. This simply refers to review and refinement of not only the product, but processes and methods over short intervals. Our product development is iterative and incremental, and improvements are gained through testing and immediate feedback. These processes are repeated with every delivery, and the same kind of feedback is built into meetings, planning, and retrospectives throughout a project’s lifecycle. In this way, we are constantly improving not only our product, but the processes by which we deliver it.

Various agile methods are in use all over the world to improve software development and break down barriers to communication in all kinds of industries. As these methods continue to grow in popularity, the likelihood of encountering a system or organization using agile methods increases.

Why Agile for IT Security?

Being agile isn't just for software developers anymore. In fact, the case can be made that agile philosophies and teams are useful in a variety of industries. The developers you work with don't even have to be using agile methods for your team to benefit from the principles in this book.

However, given the increasing popularity of agile development, you may find yourself one day charged with maintaining security posture on a system and software suite being developed by an agile team. The challenges that surface when you are asked to secure a constantly changing system can be daunting, and many of us will find that we always feel behind schedule; endlessly playing "catch up" to stay ahead of not only emerging security threats, but compliance with organizational information assurance policy, laws and regulations, and the needs of our organization.

Using agile principles for information security is just another way to accomplish our security mission and better support our enterprise. After all, despite the decrees handed down from above, established policy, or information assurance "directives," the core mission of information security remains: to support the organization's mission while protecting its assets.

Agile Adoption in the Enterprise

Each year, VersionOne conducts its "Annual State of Agile Development Survey," where respondents are asked questions related to development activities within their organizations, their own knowledge and background with agile methods, and institutional and organizational obstacles to agile development. The survey provides a great overview of agile adoption throughout the industry. In the 2013 survey, nearly 88 percent of respondents said their organizations were using agile development in the enterprise, with nearly 40 percent managing at least 75 percent of their projects using agile.³

The very same survey reports that IT staff accounted for just 3 percent of respondents, and in their measurement of which team roles are the most knowledgeable agile practitioners, IT staff did not even register on the scale. This raises two possibilities: that IT staff as a demographic don't have a lot of people knowledgeable in agile methods, or that agile methods aren't important to IT and operations personnel, and as a result they're not well represented in industry data.

Either way, agile principles can be used to improve a variety of business processes within your organization. Its mission oriented, iterative focus lends itself well to securing your systems and producing artifacts required for compliance with policy

and regulations.

Improving Business Processes with Agile and Scrum

Although agile software development has become very prevalent in the enterprise, many organizations have not adopted its methods elsewhere in their business. In the book “Scrum in Action,” by Andrew Pham and Phuong-Van Pham, the authors express their belief that current corporate culture has not been reorganized for agile and Scrum.

They highlight examples from Capital One, Texas Instruments, and other companies that have adopted agile methodologies, but have kept large pieces of their organization in place without changes. They also point out that there is “much rigor in the traditional approach (to project management) that even Agile and Scrum can benefit from,” and that a primary reason for failure with Scrum is that teams don’t know how to work the framework into the existing organizational structure.⁴

The reality is that, although Scrum may be traditionally used to produce software, the principles and philosophy behind the method can be used to improve any process which requires rapid change, regular delivery of a “working product” (in this case, patches, documents, and countermeasures), and involves multi-functional teams with varying areas of expertise.

The information security field is heavily laden with processes, procedures, and documentation. It’s easy to see how this field can clash with a philosophy that places interaction above process, and responding to change over planning. The growth of agile philosophy, and the increased importance of information protection, virtually guarantee these worlds to continue to collide.

Scrum and other agile methodologies aim to reduce complex processes to simple goals, decomposing tasks into small, manageable chunks that can be finished quickly. One of the core sources of resistance to agile methods is the fear that agility means cutting corners, or incomplete products. It’s helpful to remember that simplicity does not necessarily mean incompleteness. This and other sources of resistance are covered in chapter 2.

For now, recognize that the goal of this book isn’t necessarily to change everything about your organization and throw away large swaths of your current operational method; rather, we will focus on ways you can improve your business and better accomplish your security mission.

Case Study Introduction: NBID

The Northern Border Integration Demonstration

The Northern Border Integration Demonstration, or NBID, was a system built and installed in the Operational Integration Center, or OIC, at Selfridge Air National Guard base near Chesterfield, Michigan. The facility was built to enhance collaboration between various federal, state, and local agencies including the Border Patrol, Michigan State Police, U.S. Coast Guard, Customs and Border Protection (CBP), the Royal Canadian Mounted Police, and other agencies throughout the area. The goal was to increase situational awareness along a critical border region encompassing around 35 miles of territory including the St. Clair River. The NBID's Collaboration Network (CN) combined organic CBP capabilities, government and private video surveillance systems, commercial feeds (weather, traffic, geodata) with custom software to present a customizable common operational picture for analysts and leadership.⁵

NBID's development team came from the department of the Navy, and was based at the Naval Surface Warfare Center in Dahlgren, Virginia. The goal was to build the system and capability using a combination of commercial-off-the-shelf and proprietary systems, and the NSWC team was selected due to their established expertise in system integration. The team used agile processes heavily to enhance their ability to quickly respond to new feature requests and emerging customer needs, as is fitting with a demonstration of capability.⁶

Given the development team's split allegiances between the Department of Homeland Security and the Department of Defense, the system engineers (whom I led) experienced great challenges maintaining compliance with various security standards and accreditation, while still being able to support the mission of the system. Over the years, we discovered ways to overcome these obstacles, and this book is borne from those efforts. Throughout the course of this book, you'll find frequent referrals to NBID to illustrate key concepts, reinforce team-building ideas, and provide examples of how the techniques outlined within can help your organization become more agile.

Before delving into these techniques, however, we will examine the principles and philosophy behind agile development, and how these principles can help us better accomplish the mission, and more quickly react to change.

¹ Peltier, T., Peltier, J., & Blackley, J. (2005). *Information Security Fundamentals*. Boca Raton: Auerbach Publications.

² Beck, Kent, et al. (2001). *The Agile Manifesto*. Retrieved from <http://www.agilemanifesto.org>

³ VersionOne. (2013). *8th Annual State of Agile Development Survey*. Retrieved from <http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf>

- ⁴ Pham, A., & Pham, P. (2012). Scrum in Action: Agile Software Project Management and Development. Boston: Course Technology.
- ⁵ Cichowski, M. (2011, March 24). Northern border security goes high tech.
<http://liveshots.blogs.foxnews.com/2011/03/24/northern-border-security-goes-high-tech/>
- ⁶ Naval Surface Warfare Center – Dahlgren Division. (2011). Navy Engineers and Scientists Support New Customs and Border Protection Operational Integration Center.
<http://www.navsea.navy.mil/nswc/dahlgren/NEWS/NBID/NBID.aspx>

The Agile Philosophy

The first time I ever mentioned the idea of applying a more agile philosophy to security and information assurance to another person, he asked me if I needed help finding a place to buy patchouli oil and incense. He told me, “Security isn’t something you adopt a philosophy for. There isn’t any real leeway. You get standards from someone else, you meet them.”

We tend to forget that philosophy is simply the underlying ideas or theories in a given discipline. It refers to any set of beliefs or values applicable to whatever field you’re involved in, whether that’s law, the nature of reality, or security engineering. Our philosophy on something hinges not just on the standards of the discipline, but also our goals, our value system, and our personal point of view.

And yet, I’m willing to bet that not a lot of people give too much thought to their personal philosophy on information security. For the average user, security is merely an obstacle to their work, and something to impede their progress, versus a piece of the organization that supports their work. From an engineering or development perspective, security is often viewed as another chore to accomplish so we can “get back to work” installing the newest whiz-bang piece of equipment or producing more software. If either of those is true in your enterprise, then there are improvements to be made in your company’s security philosophy, ranging from simple tweaks to a complete overhaul of the program. Unfortunately, it’s common for people to think that their organization simply can’t do things a different way.

In a lot of organizations, it’s quite tempting to believe this. After all, every last one of us has worked in an organization where we heard things like “We could do it better... but this is how it’s done here.” Usually, a statement like this reflects either process, tradition, or simply laziness; not wanting to expend the effort to find another way. We’ll discuss resistance to change further along in this chapter, and some strategies to overcome it. For now, let’s focus on how adopting an agile philosophy can promote positive organizational change, within the security team and beyond.

The agile philosophy is reflected by a set of principles enumerated by the writers of the Manifesto⁷. Each of these provides guidance and direction for those working to

build agility into their teams and projects, and are presented here in the context of the agile information security program.

Agile Principles

Customer Satisfaction through Early, Continuous Delivery

No matter what particular flavor of agile development you subscribe to, one of the key concepts is customer satisfaction through delivering useful software continuously and early. Through the lens of an information security program, our “customer” is the organization we support; our “products” are the controls we implement, the required artifacts we deliver for compliance and risk, and the training and policy changes necessary to better accomplish our goals. The point of this principle is: get a useful product into the customer’s hands early. The customer receives an early product and has a chance to voice feedback and concerns that can be rolled into a later release. Once a task is finished, it’s onto the next piece of work, and the product should be released the moment it is operationally useful and functional. The customer gets to see continuous progress on the deliverables, whatever they may be, and provide feedback early. Meanwhile, your team can adjust to that feedback quickly. Everyone is satisfied, and development continues.

In the case of controls, once testing has identified that the risk associated with the control is mitigated, it’s time to move that control into effect. Further refinement and the ability to tune the product shouldn’t delay its initial implementation: a security appliance you’re testing on a non-production network isn’t stopping any threats. It is up to the individual team to determine at what point operational utility is high enough to justify implementation in production, but the premise behind this principle is “earlier rather than later.”

But what about the documents and artifacts we produce? In fact, we already know that agile in general values working products over documentation, so why not skip boring security risk assessments and regulatory requirements in favor of the newest, latest security gadgetry?

Remember, an information security program isn’t just about implementing technology. It’s also about providing decision makers the information they need to make good decisions about their technology. It also means ensuring our users are informed, well trained, and that the effectiveness of our programs are periodically evaluated.

So, in an agile security program, our “working product” includes documentation that furthers the goals of our department or organization. And in the spirit of this agile

principle, that documentation should be delivered as soon as it is operationally useful. If a security risk assessment, the results of a policy audit, or a new training module can guide decisions and set more work in motion, then it's time to deliver them. Early, continuous delivery of these decision-making and support products help our organization accomplish its mission.

Embrace Changing Requirements

The very phrase “changing requirements” strikes fear into the hearts of engineers everywhere. There's always trepidation involved with late-breaking changes to a product, particularly one that has been produced as the result of a great deal of manpower and effort. It may be tempting, especially once a great deal of infrastructure and documentation has been delivered, to avoid change or to discourage others from introducing change. This is particularly apparent in the information security world, where a small change in infrastructure could have a devastating effect on the accuracy and validity of risk assessments, accreditation documents, and security policy. Resist the temptation to avoid change, and learn instead to thrive on it.

Remember, agile processes were developed for the express purpose of responding well to change. We are delivering products at short intervals, so each sprint (a concept which will be introduced later) is not a monumental undertaking. Embracing change is simply about communication: listening to our organization's needs and responding to them. Anticipating those needs, and being ready for them. The security team that can react to rapid change in requirements becomes accustomed to shifting focus and refocusing resources. This is the team we want when there's a major incident, data spillage, or other high-level problem. Recall that information security exists to support the enterprise. What better way to do this than quickly and efficiently react to their changing needs?

Frequent Delivery

This agile principle lends itself particularly well to our business, given the fast-paced nature of emerging threats. Ideally, we should be delivering new features, patches, closing exploits, and updating relevant documentation every few weeks. In the case of a critical zero-day vulnerability, an out-of-band update immediately upon discovery is not outside the realm of possibility. Frequent deliveries also reduce the complexity of any individual delivery, simplifying troubleshooting, change control, and diagnosis of any regressions introduced, for example as a result of an operating system patch, network device firmware, or ACL change.

A fast cadence drives further progress, and frequent delivery contributes toward building a team culture that can shift focus fast, bringing us closer to our goal of not just tolerating change, but thriving on it.

Daily Collaboration

One of the crucial lessons I learned at NBID is that knowledge silos and missing or absent communication between teams can cripple a project and really harm morale. In our efforts to become a team that supports the mission of our organization, frequent (preferably daily) collaboration with other service teams, developers, customers, and other business units is crucial. Remember, our work involves making sometimes wide-reaching changes to system architecture in the form of patches, updates, and configuration changes. This can have a major impact on other parts of our organization.

Although collaboration is vital even in a relatively static business environment, this collaboration becomes even more important when the software on the system we are charged with securing is also developed by an agile team. If the configuration of our production environment is markedly different than developers' preproduction environments, we've just blocked their ability to do their jobs effectively.

Consider the software team delivering a new build in a 3 week sprint: major changes to operating systems, SDKs, or network infrastructure could very well cause them to have to re-work code during that sprint. Although one of our goals in building an agile team is to tolerate change, we prefer to react to change introduced by new features, functionality, and useful software from our developers. Introducing change to another team's workflow as a result of our own failure to communicate should be regarded as something to be avoided at all costs. Our goal is to never be the reason another team misses a task on their sprint; supporting them rather than becoming an obstacle for their progress.

That may require finding a new way to mitigate a vulnerability that doesn't cause negative impact to our enterprise. And as an agile security team, it's these kind of decisions that we need to make to support the mission.

Bring on Motivated People, Support Them, and Build Trust

An agile security team should be constructed around motivated, self-directed people who are given the tools and environment they need to thrive. The concept that an engineer needs permission to solve a problem is a concept that needs to be stricken entirely from the way we operate. Provide guidance and support rather than decrees,

and above all else, build trust.

The leader of an agile team should be fully committed to removing obstacles for their folks, whether those obstacles are technical or organizational in nature. Removing tedious security tasks through automation, redistributing workload to avoid overtasking your people, encouraging innovation, and fostering an environment where engineers feel a sense of ownership over their system ensure that the overall health and security of your program are foremost in everyone's minds.

Finally, build a culture of trust within your team, and throughout your organization. Trust is a two-way street, and is earned through time and experience. Drive home the idea that even failure presents an opportunity to learn and improve. Encourage your team (and other teams) to try new approaches, even if they don't result in the intended outcome. A software engineer developing a product on your system should never feel sheepish about coming to you for support. If a feature they implement adds complexity to the overall security program or architecture, you should be the resource that developer turns to for assistance. If you become the stereotypical security team, lambasting developers over the vulnerabilities introduced by a new feature or code base, your team is failing at the first mission of supporting the organization. Don't waste time listing all the reasons a given feature will threaten your accreditation or compliance requirements; instead help management and developers find ways to meet that requirement in other ways. Less "you can't do that," and more "how can I help you make this successful?"

From a leadership perspective, building trust is simple: Admit when you've made a mistake, keep your promises, and above all else: aggressively hunt down and eliminate anything that keeps your people from doing their jobs.

Face-to-Face Communication

Communicating in person is becoming an increasingly lost art in the world of mountains of email and misused "reply all." Creating a ton of written correspondence is one of the quickest ways to reduce the bandwidth of your team and your organization, as they work to find a "professional" response to every query or status update. In-person communication encourages a back-and-forth conversation, raising (and answering) new questions, and presents leadership that is concerned and invested in the daily activities of their team members.

Additionally, verbal communication promotes honesty, as often complications or mistakes are more easily discussed in the context of a small team, versus an e-mail thread where higher levels of the organizational chart may have visibility. It allows us to ensure that by the time an obstacle or problem leaves the sole visibility of our team,

we already have potential solutions in mind, further building trust and promoting that environment of support. A team member should never have to deliver the news of a failure single-handedly. If you're the leader, then the success and failure of your team rests directly on your shoulders; allow your team to express these challenges in person, and then bear the responsibility of delivering bad news to senior leadership or other teams yourself.

This sort of communication gets more challenging as the prevalence of geographically separated teams increases. Whenever possible, it's preferable that individual teams are co-located, but this isn't always feasible. In this situation, try to find ways to capture the same essence of a face-to-face interaction, such as videoconferencing or conference calling, rather than falling back on e-mail. Realize that for important events, such as planning a sprint or release, the cost of bringing geographically separated people together may be worth the trust and morale that face-to-face meetings provide.

Regardless of how the interaction is accomplished, the goal is for openness and honesty, delivering and receiving accurate information so that the team can move on to its next task.

Measure your Progress through Working Products

This is quite possibly the most important principle for those looking to build an agile information security team. The lack of a security incident isn't a good measure of a security team's effectiveness, particularly since there will come a time where you'll no longer be able to meet that measure. Similarly, compliance with a particular standard or policy means little without context: what have you done to make compliance with that policy less intrusive on your users and developers? What steps have you taken to test the validity of that standard, and if necessary, exceed it? How many artifacts have you produced that have enabled organizational decision-making, versus simply collecting dust on a server or being endlessly forwarded?

Remember, our deliverables are useless unless they mitigate a vulnerability, enable decision-making, remove an obstacle, or develop a more security-conscious user base. If your team produces a product or feature that lacks utility, you've produced nothing of value. Don't focus on how many risks you added to the assessment; instead ask yourself how many items on that assessment have a lower residual risk due to controls than before.

Further, a leader shouldn't measure his team's progress based on how they accomplish their tasks, but by the completion of those tasks alone. If you want to build a

team that innovates; you need to get out of the way and let them develop their own routine. Just because something isn't done in a traditional manner (or "your way"), doesn't mean it's ineffective. Teach your team to define "finished" as whether or not a piece of work accomplishes its purpose, not based on how your engineers got to that point.

Sustainable Pace

One of the key things to keep an eye on during your first few sprints is the cadence of your operation. Over time, you should develop a feature/document rhythm that is sustainable indefinitely. Not only will this ensure that you haven't over-tasked your team, but it provides some level of predictability to other service teams, developers, and senior management.

For any given product, regular releases will ensure at least some level of scheduling ease. When regularly recurring requirements are uniform in their pace, such as updates to risk assessments, patch tempo, or security awareness training, it becomes easier for our team (and others) to plan important new features and releases around an expected workload.

But perhaps more importantly, if you're building an agile team, you've gone to great lengths to attract and retain intelligent, innovative, and motivated people. The absolute last thing you want to do is lose them or harm their productivity by placing unrealistic expectations on their progress. A smaller amount of progress sustainable indefinitely is far superior to a great deal of quick progress that destroys your team.

Continuous Focus on Technical Excellence and Design

A major misconception about agile development's focus on working products is that these products are somehow less mature or inherently unstable as a result of the rapid pace of development. This is avoided by having a constant focus on technical excellence and good design practices.

As your security program evolves, it becomes vital to ensure that a balance is struck between new capability, and refinement of existing infrastructure and products. A complex and highly capable intrusion detection system is well on its way to meaningless if due care wasn't taken in the capacity and robustness of the system. Concerns like performance, stability, and capacity can be every bit as important for a security appliance as the actual feature set of the system. Testing and validation of our infrastructure, and focusing on "doing it right" the first time, enhance our agility, even if

it introduces a bit of necessary complexity or delay. The quality of our product is ensured through robust independent testing prior to delivery. At every stage of our work, we should be ensuring that the solutions we produce are robust and maintainable. One of the ways we can accomplish this is expressed by the next principle, simplicity.

Simplicity is Paramount

One of the key sources of instability in any piece of infrastructure or task is poorly managed complexity, and this is as true for a set of firewall rules or data loss prevention system as it is for a piece of software. If complexity is introducing instability into a system, it's time to refactor that system. If a task seems too arduous or out of reach, it's time to decompose it into smaller pieces.

Refactoring a piece of engineering is a crucial part of our agile toolkit. Within the software development world, this is nothing more than simplifying the code and maintaining its health without changing its external behavior. For the security practitioner, this could mean taking some time to build a meaningful nested structure into a permission set, or refining the roles by which we assign permissions. Refactoring increases understanding of the system, and ideally promotes reuse and collective ownership of features.

Similarly, another tool in our toolbox that drives good design is the process of decomposition. Within the agile security team, we will express many high level tasks in the form of "security stories" or "abuse cases."

A security story is a tool for defining new features or competencies based on the perspective of the user or organization's desires. It generally comes in the form of a statement, desire, and justification. An example might be, "As a user, I want my communication with clients to be secure, so that I can freely and easily discuss confidential or sensitive information with the customer." These stories become the basis by which we construct high-level security tasks and items on our risk assessment.

Another related tool is the abuse case. Similar to a security story, this is a statement from the perspective of an attacker or disgruntled insider, and take the same form, such as "As a disgruntled employee, I want to steal valuable designs from the company, so that I can sell them to competitors or produce a product on my own." These type of stories further guide countermeasures like access control and data loss prevention infrastructure.

As you may expect, these stories can produce high level tasks that are fairly complex and nuanced. The goal of decomposition is to reduce these complex tasks into manageable, simple chunks. One good way to determine if a task has been sufficiently

decomposed is whether or not it can be accomplished in a small period of time, such as 1-3 days.

Finally, within any agile team, ensuring simplicity involves making sure we are managing scope creep within a given feature. We should endeavor to keep unrelated features and deliverables separate from one another to manage the complexity of our project.

Through a focus on simplicity, we can avoid adding unnecessary complications to the security program and infrastructure, which enhances maintainability and lightens the workload of the security team.

Self-Organizing Teams

If you've built an effective agile team, it's composed of motivated, intelligent engineers, who trust one another and their leadership, in a larger organization that supports their needs and removes obstacles to success. The team knows to refactor their work and enhance simplicity, follow sound design standards, and decompose larger tasks into manageable chunks. The team is cross functional, with varying areas of expertise, and communicates effectively with one another and other organizations to produce their product and stay abreast of other teams activities.

This is a team that knows how to accomplish their immediate goal with very little direction, and this is the essence of a self-organizing team. The leader of an agile security team, therefore, needs to focus very little on directing the minutiae of their daily activities, and can focus on a leader's primary duty: removing obstacles impeding their team's success.

The agile philosophy holds that such a team produces the best products, articulates the best requirements, and builds the best architectures.

Regular Reflection and Process Improvement

Finally, one of the key principles of an agile process is the ability to regularly reflect upon and improve how we do business. Within Scrum, this is called a retrospective, and will be covered in more detail in Chapter 3. In the military, this takes the form of an "after-action review," or AAR, and accomplishes the same goal. Through this reflection, we seek to establish what went right, what went wrong, and how we can improve. The goal of this reflection is to continue to build our effectiveness and drive efficiency, allowing us to focus more on the work that we do: protect the organization's infrastructure and assets to help them accomplish their mission.

These reflections or “feedback loops” throughout our entire process are the most valuable part of agile development. Constant improvement is the centerpiece of agility.

Obstacles to Agile and Overcoming Resistance

The information security discipline has become exceedingly rigid in many organizations. As with any major change to process, resistance will be encountered when you make the effort to transition into a more agile framework for your security program. The key to managing and overcoming this is recognizing the sources of resistance and coming up with a strategy to overcome it through an understanding of the needs of your organization, examining the obstacles with an eye toward making all concerned see the value in agility.

Resistance is Guaranteed

There is no wide-reaching change you can make in an organization that will avoid resistance. It's human nature to become accustomed to the status quo, and we are naturally creatures of habit. Over time, our work habits and organizational culture become a source of comfort, and adopting a philosophy such as agile will upset the status quo. The more adventurous within your organization will embrace the opportunity to try something new, but those voices are often not as vehement as people who have become comfortable with their way of doing business. To overcome resistance, we need to examine the sources and see this resistance not as an obstacle to overcome, but as a problem to understand. The key to understanding this problem lies in identifying the needs of those who resist agile, and finding ways to use agile processes to address those needs.

Sources of Resistance

Whenever organizational change happens, there are those whose jobs become more important (increasing power, clout, or respect), and those whose importance lessens. Regardless of any desire for the organization's success, human nature dictates that power and control are prime motivators in behavior.

Another source of resistance to agile will come from those for whom open, face-to-face communication is difficult, or whose personality may seem incompatible with some of the team methods agile processes use. Some people prefer to work alone, and collaboration can be seen as an obstacle to their success, or the fear of losing ownership of their work could be a factor.

Finally, resistance to agile can also come from those who believe that implementing a framework such as Scrum will be insufficient to comply with regulatory, legal, or policy goals. Within the information security discipline, this resistance is a cultural one, resulting from perceptions on the “necessary rigidity” of information security standards.

Let’s examine each of these sources to determine how to combat unwarranted concerns, and address concerns that have more merit.

Power and Control

Possibly the most difficult source of resistance to any change are people whose power and control (real or perceived) will decrease as a result of this change. If these sources of resistance actually possess a great deal of power within an organization (managers and executives), the obstacles they will put up can be challenging indeed. In order to manage power and control concerns, these people need to be made aware of the value of change to the organization, and need to be reassured of their continued importance to the company’s needs, regardless of whether their individual control and power decreases.

People who will lose control by a change to agile will hopefully still see that the success of the organization is the primary focus in any change. In this situation, appeal to their pride in their company, and the satisfaction in knowing that helping to spearhead change makes them responsible for organizational improvement. A manager who loses staffing or budget to an agile change may be persuaded to concede these things if they know their name will be attached to a company’s shift to agile processes. If necessary, appeal to this vanity and ask that person, “Don’t you want to be the person who helps make us more successful?”

In some cases, the loss of power or control can be replaced by increased importance in other ways. A project manager who is intimidated by the prospect of losing control of some aspects of design needs to benefits of agile explained in a way that frames the change to highlight his or her benefit. Perhaps this person will lose some control over design, but he or she will also be given much greater insight and visibility into ongoing work, be better able to respond to organizational needs, which translates into this person looking better to internal management and external customers and stakeholders. The iterative processes we build by adopting agile give this person a greater ability to accurately answer questions about deliveries, features, and the status of the project as a whole.

Personal Resistance and Personality

The frequent, face to face communication required by agile processes can be daunting for some people. They may find it difficult to communicate effectively in person, or may have a belief that the others on the team will co-opt their ideas and innovations as their own, resulting in this person being perceived as extraneous. Difficulty in social interactions is particularly common in disciplines such as ours, where intellect and a knack for complex technical design often go hand-in-hand with social difficulty.

For these sources of resistance, communication on the benefits and purpose of face-to-face communication is essential. Remind them that the purpose of frequent interactions isn't to force them into an uncomfortable situation or embarrass them, but to validate and bring visibility to their concerns, their ideas, and their point of view. Explain that we're not seeking to bring them into a stand-up meeting to marginalize them, but to boost their influence and importance. By fostering an environment where the ideas and innovations are the focus, we are able to give the introvert a venue by which they can highlight their accomplishments and ideas.

Another personality trait that is particularly common within competitive information security programs is the tendency to believe that they'll lose "credit" for their ideas by bringing them out in the open in a team environment. In this situation, we simply fall back on honesty about the ownership of features in any agile project. New countermeasures, risk documents, and training products are owned by the team, not any one person. While this necessarily means that the team will gain credit for a feature whose primary source was a single person, remind them that their name will be attached to other deliverables as well, even ones that they didn't have a personal involvement in. In the agile security team, credit is shared.

Personality types affect agile processes and interaction, and when building your team it's important to ensure that your team's talents and ideas are the central point. Personality concerns, while sometimes daunting, are no match for a caring, enthusiastic team who are committed to finding commonality.

But, Scrum is Insufficient to Meet our Requirements!

One of the more common objections to agile methods, and Scrum in particular, is that it lacks the rigor and discipline required for an organization's needs. Within the information security field, we find rigidity and concrete guidelines, and many perceive agile methods as haphazard, or too flexible to meet what we perceive as "black and white" requirements. In his book "Succeeding with Agile, Software Development Using Scrum," Mike Cohn identifies two sources of misconceptions about agile,

“Waterfallacies,” and “Agile Phobias.”⁸

Waterfallacies

A Waterfallacy is, according to Cohn, a “mistaken belief or idea about agile or Scrum created from working too long on waterfall projects.” Many software and systems projects fall into the waterfall category, where development is broken down into distinct stages: Requirements, Design, Implementation, Testing, Installation, and Maintenance. Contrary to Scrum (where a given product is released the moment it is operationally useful), waterfall projects don’t deliver the product until every associated feature is complete, tested, and the “final” product is finished. When your security team has operated under such a system for so long, Cohn’s “waterfallacies” can be a significant source of misconceptions about agile. Some of Cohn’s examples of these include:

- *Scrum teams don’t plan, so we’re unable to make commitments to customers*
- *Scrum ignores architecture, this is not compatible with our type of system*
- *Scrum is fine for simple projects, but our system is too complicated*⁸

These objections are easily refuted, but the challenge lies in making sure that the source of these fallacies understands and believes your reassurances. Explain that an iterative approach introduces more predictable progress, not less. Remind these people that agile is architecture agnostic, and that one of the key principles of scrum is decomposition of complex systems into simpler, independent tasks. These reassurances can be coupled with a pilot program or project to demonstrate the utility and benefits of Scrum, and often first-hand experience shattering these fallacies is the greatest evidence we have against them.

An agile phobia, according to Cohn, is “much more personal,” and more difficult to counter with rationality and evidence. Some of these concerns include a fear of being seen as useless, a fear of making decisions independently, or (for managers), a fear of allowing one’s subordinates to make decisions. Cohn suggests that often, these people simply need to know that their fears have been noted.⁸

Countering irrational fears is difficult, and the only real action you can take is to continue to reassure these sources and demonstrate your commitment to the organization as a whole, which includes them.

Compliance is Process Independent

Within any organization, your information security program is governed by policies, organizational goals, legal and regulatory requirements, and industry best practices. One of the chief drivers of resistance to agile is that agile is somehow insufficient to meet these requirements

Recall from our discussion on agile principles that our primary measure of progress is a working product. For any individual project or task, the measure of success isn't how that task was accomplished, but *simply that it was completed*. This principle can and should be adapted to the larger goals of compliance and meeting large information security requirements.

Compliance with these policies, regulations, or best-practices is a goal, not a concrete method. How we get there is less important than arriving. I cannot come up with a single information security requirement that cannot be accomplished iteratively with agile methods. Whether we use the waterfall approach, where the goal isn't met until the "end," or an iterative approach where useful features are delivered incrementally, our desire is the same: to meet that particular requirement. In fact, given the rapid change of information security requirements, it's very easy to argue that early, frequent delivery in an iterative fashion (with large tasks decomposed into small, manageable pieces) is superior for the fast-moving landscape of information security.

Addressing Resistance

Resistance to change can tell us a lot about our organizational health. Regardless of the source or nature of the resistance, overcoming it requires understanding, and a good faith effort to address those concerns. The last thing we should do is marginalize or dismiss these concerns outright, whether they're valid or invented. Dismissing a person's concerns as unimportant will only increase their fight against your change, because now you've not only proven the accuracy of their concern, but you've told them that their input simply isn't important. Always endeavor to satisfy the concerns of resisting factions within your organization, because it's far easier to address the resistance than it is to drive forward with constant reverse pressure.

The Agile Team and Building Relationships

The way a team plays as a whole determines its success. You may have the greatest bunch of individual stars in the world, but if they don't play together, the club won't be worth a dime.

- Babe Ruth

Our agile security team is a cross-functional, self-organizing team of professionals, who operate in an environment where they trust one another, have direct ownership of the product they produce, work at a sustainable pace, and have been empowered to accomplish their goals, being committed to the team's success.

In order to form such a high-performing team, we structure that team around building relationships with people, both within the team and externally. We communicate effectively, we take pride in our work, and we operate as a unit. To further illustrate the building blocks of our agile team, let's examine each of the roles within a Scrum team through the lens of the information security discipline.

Team Lead or Scrum Master

The agile team leader, or Scrum Master, is responsible for the facilitation of the team's success. This person is not a micromanager, a "boss," or a taskmaster handing out decrees. The chief responsibility of the agile security team leader is removing obstacles to the team's success, and driving his team's investment in their product. Remember, the team leader doesn't drive the "how" of accomplishing a task, the team leader trusts members of the team to do that. As a team leader, you protect the team from outside forces, facilitate communication, and resolve issues between other teams.

One of the crucial concepts within agile and Scrum is the idea of a "servant leader." Although much has been written on servant leadership, the core premise is that the leader's commitment rests upon the welfare of the team. The servant leader puts the needs of his team and his organization ahead of his or her own. The servant leader isn't seeking power or control, but rather leads by example, exhibiting a "let's do this together," attitude that is a sharp contrast from barking orders. In our agile team, leader serves the members of the team, not the other way around. The servant leader builds trust by caring for the needs of the team and keeping promises.

From a practical standpoint, further tasks of a leader will be discussed in chapter 3 when the Scrum framework is detailed. For now, it is sufficient to say that the agile security team leader is someone who facilitates decision-making and progress by the members of the team, allowing them to direct themselves, and eliminating obstacles to their progress.

Team Members

The heart of our agile security team is made up of the team members. They've been selected for their expertise, and now it's time to allow them to succeed. The agile team

must be allowed the autonomy to do amazing work. It is critical that the team be allowed to take requirements, regulations, or a specific risk identified on your security risk assessment, and have the authority to determine how best to meet that need.

Our team members should be accountable for their work; not to the team lead or upper management, but to the rest of the team. Frequent peer interactions, collaboration, and review allow our team members to learn from one another. One of the concepts behind our self-organizing team is shared mentorship and ownership of the product. The agile security team can share knowledge and make collective decisions far more efficiently than a single leader can.

The shared ownership, cross functional nature, and collaboration of the agile security team ensure team adaptability. When an obstacle or challenge arises that one member cannot solve, the team lead or another member is there for fresh perspective or the removal of obstacles.

Driving a culture where people are empowered to make decisions and adapt to circumstances is crucial to the adoption of agile within any team, especially information security.

Product Owner

Within Scrum, another role is the Product Owner, whose main purpose is to represent the interests of the stakeholders. For security purposes, depending on the organization, stakeholders can include individual users, other teams, senior executives, legal and compliance officers, or anyone else with a significant stake in the success of our information security program.

A product owner is the medium by which communication with stakeholders occurs. This is the person solely responsible for the construction and maintenance of the overall product backlog, which for us will include required documents like the risk assessment, controls, training artifacts, and policies that the information security program produces. The product owner also prioritizes items on the product backlog, defining the scope of the project and its requirements. Serving as a facilitator, the product owner can (and should) certainly take input on the product backlog and requirements from stakeholders and from the agile security team, but the final responsibility rests solely with this person. Put another way, the product owner drives the content and goals of the product, while the agile team drives the implementation.

Building Relationships Externally

Very few teams in an organization have the far-reaching impact of a team charged with information security. The controls you implement, and especially the policies that you produce, can have a massive effect on the work conducted throughout your organization. One of the key characteristics of an agile security team will be to build relationships with external teams, the user base, and leadership. Thus, the product owner becomes a very significant part of how our team interacts with the rest of our organization.

Remember that our goal is to protect the information and assets of our company, in order to better support the mission of the enterprise. If our team has negatively affected the productivity or the mission of the organization, then we have lost sight of our purpose. Ultimately, it isn't up to us to determine whether the risk of a vulnerability or threat is greater than the negative effects a given control will introduce. We are to present options and scenarios, and then once we receive requirements, we implement them. We do not have the right or authority to dictate policy to those who we serve.

Information security as an enabler versus an obstacle is one of the toughest lessons to learn within our field.

NBID: Cross-Team Collaboration

My entry into the NBID project occurred after the first major build of the software and system had been installed. The network and system engineers, developers, and testers had already delivered and validated the first phase of the project. In very short order, I had to orient myself to the various pieces and how they interacted, and that included learning about the development team I supported.

In the beginning, the relationship between NBID developers and the engineering staff was contentious at times. The “blame game” was frequently played, both by the system engineering/infrastructure side (myself included), and the developers, contributing to a pretty difficult relationship that often put both sides at opposing or competing interests. A performance problem within a SharePoint customization would immediately be blamed on “bad code,” when it was every bit as possible (sometimes certain) that the problem lay in poor infrastructure design decisions.

Sometime after I became more comfortable with the program, and we moved further along in the development of the system, I began to attend the development team's daily stand-ups. My hope, with a limited (if not nonexistent) knowledge of agile methods at the time, was that I could at the very least be aware of what they were working on so that I could plan my activities accordingly. Further, I hoped that by letting them know what patching and system maintenance activities we were engaging in, I could avoid becoming an obstacle for their progress. If they planned to deploy and test a

new feature, then I'd be able to postpone or accelerate a patch or change required on our side.

The NBID development lab was on a military facility, while the production system was part of a CBP operation center. This meant that each system had separate information security standards to follow (from DISA and DHS, respectively), and often caused difficulty moving the software to production. One unfortunate side effect of varying security requirements between facilities was that, on a semi-regular basis, something that worked fine in development was horrendously broken when deployed to production. The project required a build night "deadline," by which we made a decision whether we could finish, or whether we would have to roll-back the new build. This led to some frantic nights where we were dangerously close to hitting that deadline, and some pretty hectic efforts to fix the problems.

To this day, I don't feel I went far enough to facilitate cross-team collaboration. However, over the course of the project, each production software deployment became less of a "witch hunt" to find out who was responsible for a problem, and more of a collaborative effort to identify and solve the problem. In the end, although we still weren't perfect, I feel like NBID became an effort that both sides took great pride in, despite occasionally opposing interests.

⁷ <http://www.agilemanifesto.org/principles.html>

⁸ Cohn, M. (2010). *Succeeding with Agile: Software Development Using Scrum*. Upper Saddle River: Addison-Wesley.

CHAPTER 3

Scrum for Information Security

Scrum is currently the most widely used agile framework for software development, and it's easy to see why. It's a highly mature set of processes with a great deal of flexibility, and is adaptable to a wide range of industries and projects. According to VersionOne, it is also the most widely used agile methodology in the enterprise, with Scrum and its variants being used in 73 percent of agile projects.⁹ As such, it's a natural framework to base our agile information security program on. In this chapter, we'll give a brief introduction to the Scrum method, and then describe a few key parts of the framework as they pertain to our security program, including the product backlog, sprints, the daily stand-up (or "Scrum") meeting, and a brief example of how IT staff participation in developer Scrums contributed to inter-team cooperation at NBID.

Introduction to Scrum

Scrum was developed in the 90s by Jeff Sutherland and Ken Schwaber, and it was formally delivered in 1995 at a conference in Austin, Texas. Both of these men were also part of the creation of the Agile Manifesto in 2001. The name comes from the game of Rugby, where a Scrum is a restart of the game that involves a bunch of players close together to regain possession of the ball.¹⁰



“Whoever gets the ball doesn’t have to update the risk assessment!”¹¹

The “Scrum” in this case is the daily Scrum, also known as a stand-up meeting. This is a hallmark of Scrum and other agile methodologies. The conduct and tone of the Scrum has a massive effect on the effectiveness of the program, so we’ll spend a lot of time on Scrum dos and don’ts later in this chapter.

As the most used agile framework, Scrum represents not only the most likely framework an information security professional may encounter in the development teams he works with, but a great base to construct our agile information security program.

The Product Backlog

The product backlog defines the product we will deliver to our customer. It should be located in a place where everyone involved in the team can view it, and is an ever-changing piece of the project that will be modified as new deliverables are discovered. This is the source of the items that will appear on each sprint, once high level tasks (usually expressed as stories) are decomposed into smaller tasks.

What makes Scrum different than traditional (or waterfall) development methods is that these items are not defined in excruciating detail. Rather, these items are defined somewhat loosely, with more detailed, iterative definitions defined progressively as items are placed into a sprint and development work begins. The details of how the backlog is physically displayed vary greatly. Some prefer to use a large whiteboard with handwritten items, sticky notes, or something similar to represent items. Others prefer to keep track of it digitally using general purpose software such as Excel, or a

purpose-built product. Regardless of how your team manages the backlog, there are a few elements that will be common to any team's backlog.

Content of the Product Backlog

Within a product backlog for a software development team, items on the backlog can include new features and functionality, bugs, other related technical work (including infrastructure needs and hardware acquisition), and any tasks related to knowledge-building or research related to a new feature or bug. Recall that these are high level items that we will express in the form of stories (more on this later).

For our security team, what are some of the things we might consider adding to the backlog?

- New hardware, software and other infrastructure pertaining to the mitigation of a risk on our risk assessment.
- Policy and configuration changes, such as Windows group policies, firewall ACLs, audit settings, log collection settings, and permissions.
- Modification of existing controls to fix bugs or unintended problems.
- Documentation updates, including changes to the risk assessment, updates to user training, or acceptable use policies.
- User training deliverables, such as training sessions themselves, the associated materials, and any other efforts to increase user awareness.
- Artifacts and activities related to auditing and compliance.

In the case of an information security program, the stories that make up our backlog can come from a variety of sources. Obviously, our chief concern is the protection of organizational assets, so a great deal of these may come in the form of user stories or abuse cases related to the protection of operational data. However, there is one key way in which a security program differs vastly from a software development project, and this is a vital difference.

Tell Your Users What They Need

The vast majority of our backlog items, even if concerning a user story, **will not actually come from users**. This is an absolutely crucial difference from a traditional software development project. Our users may be very able to articulate to a software team a function that they want in the product. However, recall that the majority of our

users see security as an obstacle, or at the very least don't understand the importance of our security program. As such, you're unlikely to get actionable backlog items from the majority of your users.

This is where your team's expertise becomes the driving force for construction of the backlog. The team, the leaders, and the product owner need to make assumptions on user work needs, and use those needs to determine what vulnerabilities and threats exist in your system. This is done through the creation of a risk assessment, which will become the **primary source** of items for the backlog from the perspective of user stories and abuse cases. In short, your users generally do not know their security needs; it's up to you to tell them.

Other Sources

We also may express these items as stories from executives or our legal department, if the feature or task relates to compliance with regulation or law. Software developers may have a new feature or piece of connectivity that we may need to secure. Keep in mind, though, that dealing with external teams can produce some complex, interdependent stories on our backlog.

Earlier we discussed keeping open lines of communication with software developers, and this is one situation in which this communication is essential. It's likely that at some point you'll receive a requirement from a development team that involves relaxing or changing an existing control or policy, to enable required functionality that your developers need to deliver. This can be one of the most challenging requirements you'll receive as a security team, and requires a great deal of involvement from not only your team and developers, but critical decision makers in your organization. The change needs to be assessed, risks enumerated, and the benefit of the new feature evaluated. Remember, it isn't your job to deny such a request *carte blanche*; you need to make a good faith effort to evaluate the change and make a recommendation to senior management. Whenever possible, if a control must be relaxed or disabled, try to find other, non-impacting ways to mitigate the given vulnerability.

Product Owner's Role

As we discussed previously, the owner of the product backlog is the product owner. Although our team and external users/teams will have input on the backlog, it is ultimately the product owner who determines how these needs are expressed as stories in our backlog. Open communication between the product owner and external sources,

as well as our security team, is vital to ensure we are all on the same page.

The product owner is responsible for prioritizing the backlog. The items which are most important (based on business needs, risk severity, etc) are placed at the top of the backlog. Additionally, if an important item has a dependency, this dependency must necessarily be put first. The owner is responsible for communicating with the teams, ready to answer questions.

The product owner should be someone who can articulate the importance of these items, both to external entities (senior management, concerned users, developers) and to the team itself. Remember, our team of security engineers have a different perspective than a typical user, and the product owner can help translate those perspectives.

Using Stories

The primary source of stories to build your security product backlog will be items on the risk assessment, which will be described in more detail in chapter 4. However, so that we can provide a good overview of the Scrum methodology, it's worthwhile to discuss what stories are and how to express them here. Later, we will discuss integrating them into your risk assessment to produce actionable items for the product backlog.

Stories are the primary form that your backlog items will take. A story is nothing more than a statement of a need, framed through the perspective of a person. In the security world, we have user stories and “attacker” stories, usually referred to as “abuse cases.” Here are a few examples of these security stories:

- As a user, I want my antivirus to be robust, so that I don't experience downtime due to virus infections and stability problems.
- As a user, I want our public website or document repository to be safe from DDoS attacks, so that I'm not interrupted when presenting information to potential clients.
- As a scientist, I want my research to be protected, so that it is not stolen and misused
- As the legal department, we want our employee personal data to be protected, so that we will not be held liable for identity theft due to a breach of our system.

As you can see, these stories take the generic form of “As a <type of user>, I want <some situation or end state> so that <the reason for the end state or situation>”. This form ensures brief, concise stories. Remember, we expand on what these stories entail

later on, as we decompose these stories and put them into the release and sprints. Additionally, as previously discussed, many of these stories will come directly from the security team, who is basically expressing these desires on behalf of the user, largely based on the risk assessment. The source of a given story isn't as important as whether the story is valid, actionable, and feasible.

One type of story not commonly used in software development is an abuse case. This is a story from the perspective of a hacker, disgruntled employee, or other malicious entity that drives controls and safeguards. Some examples include:

- As a competitor, I want to be able to look at confidential designs, so that I can beat you to market.
- As a scam artist, I want to obtain employee names, addresses, and social security numbers, so that I can steal their identity and finance a Corvette under their name.
- As a hostile foreign government, I want to breach your military systems so that I can download classified intelligence.
- As a disgruntled employee who will soon be fired, I want to delete permanently as much data as possible, so that I can cause chaos and just be a generally mean person.

Construction of abuse cases will come through the team's knowledge of common threats, evaluation of how those threats are applicable to our company, and best practices. Their placement on the backlog will depend largely on the residual risk as evaluated after existing controls are considered, a task more closely discussed in chapter 4.

As a security engineer, you may find it difficult to express a given item on your risk assessment as a user story. Feel free to modify the format of a story to fit your needs, as the point is that it is brief, and high level. However, keep in mind that for a security or infrastructure engineer, your systems have needs too. There is nothing wrong with a "user story" that looks like this:

"As a web application that connects to an external data source, I want all input I receive to be well-formed and validated, so that one of those meanies from the "abuse cases" can't crash me and execute code remotely."

Not only is this particular story very specific, it immediately identifies this story as more closely related to infrastructure needs and best practices than to a specific user need.

Sprints

A sprint is a concrete period of time during which items derived from the product

backlog are completed. These can be as long as 4 to 6 weeks, however many teams prefer sprints of 1 to 2 weeks. The idea is that at the completion of each sprint, your team will have produced a working, useful product. Sprints can be part of a larger group of related items (which will have their own sprints). This is called a “release.” The details of this organization will depend largely on your needs. The key characteristics of a sprint are:

- The time period is concrete, and cannot be extended. Any unfinished items will return to the product backlog.
- Once the sprint backlog is defined, no more work can be added to that sprint. Dependencies need to be accounted for before the sprint begins.
- Each sprint begins with a planning meeting, and ends with a review and a retrospective, and the daily Scrum “stand-up” meeting is the prime method of communication.
- At the end of a sprint, a working product has been produced.

Sprint Planning Meeting and the Sprint Backlog

During the sprint planning meeting, the team, Scrum Master, and the product owner meet to discuss what will go into the next few sprints. The details for successive sprints may be murky, but it is important to identify what work will follow the current sprint, so that the team can begin looking forward and identifying dependencies. At the end of the sprint planning meeting, the sprint should have a goal and an associated sprint backlog.

The sprint goal is a simple narrative, written as a team, describing in a few sentences what the goal of this sprint is. This can be as simple as “install and configure the new VPN gateway and configure remote-access services” or “conduct a review of firewall ACLs as part of our ongoing compliance audit.”

The sprint backlog, then, is a list of tasks associated with that goal, including any dependencies. This will not always be strictly technical in nature; implementing a new feature or system may require meeting with other teams to gather information about existing infrastructure, and that would be a task on the backlog. The backlog should represent small, decomposed tasks that can ideally be completed in a single day. The sprint backlog should also have a rough schedule for each item, and there are many ways this can be expressed; choose the method that works best for your team.

In terms of the size of the sprint, remember that the product owner is responsible for determining priority. This person does **not** dictate the total amount of items that go into a sprint. The team, and the team alone, determines how much work they can

accomplish during the sprint.

Decomposing Stories

As we construct the sprint backlog, recall that our stories are fairly high level tasks which can potentially be complex and have a number of other sub-tasks and dependencies. We need to simplify these into groups of one-day tasks for the sprint backlog, and this is done through decomposition. Ideally, any story on our backlog should be fully decomposed before the sprint planning meeting is over, and we should try to avoid unexpected items appearing during a sprint. An example of a user story being broken into tasks via decomposition is below. Depending on the task, this could take multiple levels of decomposition:

Story	Tasks
As a user, I want the drives on my work laptop to be encrypted, so that if the device is lost or stolen, nobody can misuse it.	Determine required group policy settings for enforcement of BitLocker Implement GPO for test organizational unit Validate functionality of GPO with test users Push policy to production Audit company laptops for compliance with new GPO

Each larger story should be similarly composed. If the scope of a given story is large enough (sometimes called an “epic”), then that story is broken into smaller stories, which are then decomposed into individual tasks.

Acceptance Criteria

For each story, there should be some way of expressing whether or not the work satisfies the story. In Scrum, these are called “Acceptance Criteria.” These are the requirements that must be satisfied in order to call a given story “complete.”

Acceptance criteria do not have to mean that a feature is 100 percent finished. Remember, Scrum is both iterative and incremental. We produce a working product at the end of each sprint, but working is a different concept than perfection. Improvements to any given story can always be identified for future sprints.

However, this does not mean we accept an inferior piece of work for a sprint, and that's why acceptance criteria are important. We can exceed the acceptance criteria, but a given sprint should never fall short. There is no such thing as partially meeting these criteria. These acceptance criteria are negotiated between the team and product owner, and once decided, they are inflexible.

We can develop these acceptance criteria in many ways, but one that I like in particular is a test plan. For a given user scenario or abuse case, we can lay out a number of actions to be taken by testing, and a desired result for each. If we meet the criteria for these tests, but an ancillary problem emerges (such as a performance issue, false positive on a security alert, or other scenario), we can address those items in future iterations. Imagine a story involving the implementation of an IPS rule to detect and prevent a new flaw in Windows Server. Let's assume that the flaw either has no patch yet, or that another story covers the remediation of this flaw. Acceptance criteria for this scenario could look like this:

- IPS rule has been implemented on both production and disaster recovery environments.
- IPS rule successfully detects and blocks 100% of exploit attempts.
- The IPS rule results in less than 5 percent false positives compared to actual attempts.
- Performance tests of the infrastructure pass successfully.
- Risk assessment, design documents, and testing documents are updated.

Sprint Review

At the end of a given sprint, the team is required to deliver their working product. The review meeting is the team's opportunity to showcase that product to the product owner, management, representatives from other teams, and other stakeholders. As a rule, these should be very informal, and organizations adopt various rules about length, materials needed, and conduct of the meeting based on their needs.

The sprint review meeting will identify whether a sprint has met the goal of the sprint, by evaluating these stories against their acceptance criteria. Should a given piece of work not be accepted, it is placed back into the product backlog for inclusion into a further sprint, perhaps with any additional dependencies identified during the sprint that may have contributed to the lack of progress.

Sprint Retrospective

One of the axioms of agile methods in general, and Scrum in particular, is that there is no such thing as a perfect team. No matter how accomplished your security team is, or how robust a product produced during a sprint is, there is always room for improvement. Contrary to the sprint review, the retrospective generally only includes the team, the Scrum Master, and the product owner. I have found that the U.S. military's After Action Review (AAR) format is quite well suited for the sprint retrospective.

When conducting an AAR, the goal is to identify what our goal was, what went right (aspects of the sprint we should “sustain,” or keep doing), and what went wrong (things we need to improve). Improvement can be expressed by things we need to start doing, things we need to stop doing, or general weaknesses in our process.

Once these items are identified, the team then collaboratively discusses items on the list to focus on for the next sprint. Successive sprint retrospectives should also evaluate whether or not we accomplished the goals set out by the previous retrospective, in essence, “Did we improve from the last sprint?”

Sprints Provide Repeatability

In any information security program, consistency and repeatability are hallmarks of a healthy program. Using Scrum, we achieve that process consistency through the use of sprints, which provide a level of predictability to our processes. The end result of this method is that we are constantly improving, producing working products (through controls or decision-making documents) at a rapid cadence, and are endlessly improving our processes and capability.

The common view within the information security field is that our process must be tedious and difficult in order to be effective. Through Scrum, we can achieve the same level of rigor and sound design practices, through a process that cuts through the extraneous and delivers on the core needs of our enterprise.

It's fairly easy to see how false various misconceptions about agile methods are. Such an agile and repeatable process lends itself well to the rigor and repeatability required by a robust information security program.

The Daily Scrum (or Stand-up Meeting)

One of the most important parts of Scrum is the daily stand up meeting from which the method gets its name. A Scrum has several purposes, and some key features and characteristics that distinguish it from “just another status meeting.”

Meetings are dreaded in many workplaces these days. Every one of us has sat through a multi-hour “status meeting,” usually at a regular interval. Traditional meetings result in a lot of time wasted reiterating things the team already knows, a lot of problem-solving while uninvolved people listen, and are typically a productivity killer. What makes Scrum different is the very specific and defined guidelines for time, location, and content. Although modifications can (and should) be made to suit your needs, the end result should always fit the “spirit” of the Scrum: a quick way for a team to highlight what will be accomplished today, and set the Scrum Master to the task of removing obstacles.

Who Attends the Scrum?

The Scrum is attended by the entire team, without exception. This includes the product owner. Remember, even though the product owner is a conduit by which we interface with other teams and stakeholders, he or she is still a core member of our team, responsible for the product backlog and defining the items in it.

In some organizations, members of external teams, or higher level management, may express a desire to attend a Scrum. While this isn’t necessarily a problem, it’s important to set a few expectations with any external attendees. First, make them aware that they’ll be involved as listeners only. If a piece of information is vitally important for them to deliver through the team, they really should deliver that information through the product owner, to avoid disrupting our process. Second, ensure that there is an absolute need for that person to attend, and that it doesn’t become a habit. The Scrum is a very important productivity event, and should be limited to the team responsible for delivering the product.

There’s an old joke that gets mentioned in nearly every book on Scrum when it comes to the difference between a Scrum team and external stakeholders. Who am I to buck tradition by not mentioning it here?

“When it comes to breakfast, what’s the difference between a chicken and a pig?”

The chicken is only involved, but the pig is committed!”

Within the context of a scrum team, your team is the pig. For your purposes, everyone else is just a chicken.

Guidelines and Structure of a Scrum

The Scrum is one of the few things within an agile framework that shouldn’t be modified

too much beyond the core basics. The more changes and modifications you add to the meeting, the less effective it is at its goal: to keep the entire team aware of what people are working on, and to let the Scrum Master know what obstacles face the team so that he can begin removing them. Ideally, this meeting should begin the work day; this ensures that the team moves out with a sense of purpose and a clearly defined direction for their daily work. It is limited to 15 minutes, and this is achievable in nearly any team as long as the guidelines below are followed.

Time Limit, or “Why the heck are we standing?”

The Scrum is meant to be a short meeting, with only the vital information communicated amongst the team members. The 15 minute guideline should be seen as a limit that is only exceeded in very unusual circumstances.

This is why we refer to it as a “stand-up” meeting. Conducting the meeting standing, as opposed to sitting around the conference table, subconsciously reminds us to be brief, as we’re not intended to be here long.

Location, Location, Location

The Scrum should be held as close to the work area as possible, preferably in the middle of the work area. Conference rooms and other locations that signify a “formal” meeting should be avoided, because we’re creatures of habit, and locations like that will subconsciously signify that we need to return to the old, dreary way of conducting our meetings. By holding the meeting in the workplace, we’re immediately prepared to start our day and be productive.

An additional benefit of holding the meeting in the workplace is that our mind is on the work at hand. A Scrum team’s workplace should include a board that defines tasks for the current sprint, perhaps even the product backlog. This board and any additions to it can be an easy reference for the team as they move through their status update.

One option that is particularly helpful is the inclusion of an “improvement” or “problem” board. This is where any obstacles raised during the Scrum can be recorded for the Scrum Master to start working on after the meeting. This board should also be used in between Scrums if any obstacles arise.

Speaking Order

There are various methods to determine the order of who speaks, and you should feel

free to choose a method that works for your team. Keep in mind, however, that the speaking order should reinforce the concept of a self-organizing team; the Scrum Master shouldn't call on people by name, directing the flow of the meeting. Whatever method you choose, it should result in either the team determining the order, or randomization. The meeting is for the team, not the Scrum Master.

One common method is to start the meeting with the person who arrived last. This not only establishes a self-directed method of selecting the first speaker, but also gives your team incentive to both arrive on time, and be prepared for the meeting. The first person is given some sort of item which is then passed or tossed around at the speaker's direction until everyone has finished.

Another common method is the use of numbered note cards, which are shuffled and handed out as team members arrive. This method leaves the speaking completely to chance, and the team can focus on content.

The Three Questions

The basic information delivered in a Scrum can be expressed as three questions that each team member must answer. In the interests of keeping the meeting brief, it's ill-advised to deviate from this format, although the order can be arranged as you see fit.

- What did I complete yesterday?
- What will I complete today?
- What obstacles or problems are interfering with my progress?

These questions provide all the team needs to know about the current status of any given work item, and allow the Scrum Master to take note of the obstacle and begin immediate steps to remove it after the meeting.

Answering these questions daily also ensures that information is rarely missed by the team. The daily frequency of the meetings ensures that a team member will rarely be in a position where he or she has "too much" going on to share. The brief, specific nature of the questions helps ensure brevity, and that vital information is being passed to the team and the Scrum Master.

Some teams prefer to use their progress board and the sprint backlog to discuss progress task by task. Although this is described in much of the agile world as acceptable, I still prefer the questions. It puts each team member on equal footing in terms of importance, and avoids the perception of a leadership-controlled meeting. Remember, a self-organizing team generally produces the best product.

Do This!

- Start the meeting definitively – Once everyone has arrived, the first person in the speaking order should immediately begin. Silence is the enemy of momentum; if necessary the Scrum Master can prompt the first person in some way.
- Keep the meeting moving – The moment someone is finished speaking, the next person should start.
- Time the meetings – If you have difficulty with ever-lengthening Scrums, time them and publish the results. The team should hold itself accountable for brevity and progress.
- Definitively end the meeting – If you've conducted your Scrum right, everyone should have an immediate task to begin. Don't lose that momentum! The Scrum Master can make this as simple as "OK, let's get going," or as cheesy as "Thundercats, HOOOO!" The point is, end the meeting and get to work!

Don't Do This!

- Don't tell stories – Nobody needs to know how arduous yesterday's task was, the research you conducted to complete it, or how much of a firewall superstar you are. Unless a problem is still blocking you, it's irrelevant to the Scrum. Going into a long winded story destroys the brevity that keeps our progress moving. Keep it simple!
- Don't solve problems – If a technical obstacle is blocking you, the Scrum isn't the place to solve it. Note it as an obstacle and move on. Similarly, if someone has the solution to that problem, now is not the time for a technical discussion. A simple "I can help you with that offline," is sufficient to provide post-Scrum direction.
- Don't plan new requirements – If something new has emerged, schedule a separate planning meeting. The purpose of a Scrum is to move out on currently defined tasks.
- Don't report to the leader – You're giving an update to the team. Scrum Master, if your team is directly looking at you the entire time, put an end to that practice.
- Don't record the minutes and email out a summary – The whole purpose of a

Scrum is that it's a non-formal, progress oriented meeting. The instant you begin recording it and sending out minutes, you've lost the very essence of why we use the Scrum.

Tradition and Consistency

Remember, a good security program is all about consistent, repeatable processes. This applies as much to the conduct of your daily meeting as it does to patching and update schedules. Once you've decided on the particulars of how your Scrum will run, stick to it, and hold each other accountable. Establish traditions, even if it's as simple as choosing a "token" to pass from speaker to speaker that has special significance to your team. The Scrum should be the ritual by which your team moves out and accomplishes their mission.

Inter-team Meetings (The Scrum of Scrums)

Recall that as a security team, the work you do has far reaching effects across your organization. Communication with other teams is essential for your team to operate effectively.

In many agile projects, the product owner is the conduit by which this communication happens, and this can be a very effective way to pass information along to and from your agile security team. However, if you're lucky enough to be in an organization where multiple teams are using Scrum, consider a "Scrum of Scrums" as well.

The Scrum of Scrums will happen sometime after the team's daily Scrum. Each team will select an ambassador to attend, and this can be the Scrum Master or any other team member. The format is much the same, except that the obstacles raised are scoped specifically to impediments regarding coordination between teams. These items are tracked via their own backlog.

In large agile teams, a Scrum of Scrums can be used to break the team into smaller sub-teams, perhaps focusing on a specific area of the product. Within the security world, this could be network infrastructure, authentication, antivirus, or monitoring, to name a few.

NBID: The Daily Scrum and IT Staff

Fairly early on within my involvement with NBID, I realized that information flow

was a big problem. During a particularly busy timeframe, the developers had issues with a deployment as a result of performance and stability problems during some pretty intense change work on the systems side. One of the developers, who later became the lead after his predecessor was promoted, suggested that I begin attending their stand-up, even if I couldn't be a daily attendee. At the very least, this would allow me to notify them of upcoming work from a security and maintenance perspective.

Although I didn't understand the full importance of these meetings at the time, having no real exposure to agile methods, I was immediately struck by the difference between the stand-up and other meetings our team was required to attend. Where is the half-hour long discussion on risk assessment items that have no change in status? Where are the notifications of situations and new rules that we already received via email? What about discussions on larger-organization happenings that don't pertain to our team?

The meeting content moved fast, and though it changed, the format was fairly consistent when I didn't muddy the waters with some storytelling. I never left one without a good idea of what every developer was working on. Often, developers would follow me down the hall so we could deploy a new bit of code to the test lab, so work was beginning immediately.

NBID also reminded me that, even outside of my military life, small bits of tradition and habit were important. We had an "Easy" button from Staples, and during deployment nights, at the completion of each installer or module, we'd hit the button, resulting in a cheerful, "That was easy!" It became something that signified our progress throughout a deployment night.

One evening, we didn't have the easy button because we were deploying from an alternate location. Without even realizing it, I began to have bad feelings about the lack of that "easy button." That deployment was one of the worst, and we nearly hit the rollback deadline as a result of various complications.

The easy button had nothing to do with that, and certainly didn't **cause** the complications, but I'm convinced that abandoning tradition negatively affected my response to the complications. Rather than "OK, this is nothing, let's figure this out," I was thinking, "Man, I wish I brought that damned easy button."

⁹ VersionOne. (2013). 8th Annual State of Agile Development Survey. Retrieved from <http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf>

¹⁰ <http://www.scrumguides.org/history.html>

¹¹ Image source (Author: Zegreg63:

http://commons.wikimedia.org/wiki/File:M%C3%AAI%C3%A9e_ASM-MHRC.jpg Licensed under Creative

Commons Attribution-ShareAlike 3.0 Unported license, <http://creativecommons.org/licenses/by-sa/3.0/deed.en>.

Security Risk Assessments

One of the key tasks of an information security program is the creation and maintenance of a valid security risk assessment. It is possibly the most important document within your team, and guides a great deal of organizational and infrastructure change.

It's also, in many organizations, the single most irritating part of the job. The security risk assessment has become not just the central document by which decisions are made, but the central document on which resources are used. In some organizations, particularly within the government, there are entire teams whose sole job is the care and feeding of the risk assessment, and the amount of bloat and bureaucracy that comes with this effort is staggering. This situation exists precisely because of how important the risk assessment process is to our security program. Within our agile security team, this importance is highlighted by the fact that we use the items on our risk assessment as one of the key sources of stories to construct our product backlog and assemble our sprints.

The challenge, then, is how to eliminate as much of the administrative overhead from the creation and maintenance of the risk assessment as we can, without decreasing its effectiveness. The risk assessment process isn't just the creation of the document, but a great deal of data collection and interpretation combined with industry best practices and current known vulnerabilities, coupled with a distinct knowledge of the needs and asset values within your organization. Recall that our agile security team defines a "working product" as not only technical controls and training/policy outputs, but documents that enable decision-making. The risk assessment is a crucial piece of our decision-making process.

For a very well-written and complete guide to the risk assessment process, I recommend Doug Landoll's book, "The Security Risk Assessment Handbook."¹² His text is around 450 pages of sound and solid techniques for every conceivable piece of your risk assessment. However, in an agile project, a 2 or 3 week development sprint gives us little time to conduct a full assessment every time our developers drop new code. Therefore, we have to come up with a reasonable framework for our assessment that can be done as quickly as possible, and is tolerant to constant change. Before deciding how we are going to conduct the assessment, let's take a step back and define

what our assessment should consist of, and take the entire process back to the basics, and examine some of the core questions answered by a risk assessment.

Back to the Basics

What are our Goals?

The risk assessment, in some companies, has become a gargantuan effort; usually involving multiple months of data gathering, penetration testing, interviews, and evaluation of system hardening efforts and compliance. The problem with such a massive effort on a risk assessment is that, in any product developed iteratively, the risk assessment could be obsolete within a matter of weeks. Various methods for conducting these assessments, while thorough, will leave us constantly behind schedule, endlessly updating the assessment leaving very little time for implementation of controls. If your organization is lucky enough to have the budget for a dedicated security assessment team, then my hat is off to you. However, most of us will not be in that situation, so we need to step back and honestly ask ourselves: What do we wish to accomplish with our risk assessment?”

At its core, the risk assessment is simply a document that evaluates our existing controls against known and unknown threats and vulnerabilities, and an estimate of the probability and severity of loss to our assets, expressed as “risk.” Some common practices, such as interviewing every single employee in a department, or conducting a detailed security scan of every single system in the enterprise, are not necessary to accomplish these goals. Such activities may have value elsewhere, such as for auditing compliance with a given control, or spot-checking employee policy, but too often we lose sight of our goals and the scope of the assessment creeps out of control. For our purposes, a security risk assessment enumerates various threats/vulnerabilities, and evaluates the effectiveness of our controls. An audit (a completely separate process), would be the process by which we ensure those controls are implemented correctly. When we’re dealing with a rapidly changing system, distinguishing the two is imperative.

A risk assessment answers a few basic questions about our overall system, which will allow us to drive decision-making and place appropriate stories onto our product backlog by which we will develop controls. These questions are:

- What do we need to protect?
- Who and what are the threats and vulnerabilities we face?

- What is the impact if our assets are damaged or lost (the value to our company).
- What can we do to minimize exposure to these losses?

These questions can be answered by many different risk assessment formats based on your needs, but for now let's examine these questions and what they mean.

What Needs to be Protected?

Scoping an assessment will be a very important part of our agile security team's risk assessment. Given time constraints and the constantly changing nature of our systems, it's essential we realistically break down what needs to be protected, and how far our bubble of responsibility extends.

Depending on how your organization is structured, topics which affect your team may be outside of the scope of your assessment. If your system is part of a larger organization, the software and systems you support may rely on infrastructure maintained and owned by other parts of your organization. Some examples of these include physical security and higher-level network infrastructure. In the case of a development system, it is important to determine what infrastructure is organizational in nature (and used as a tool), versus the system your team supports. As a rule, you can include external dependencies in your assessment, but keep in mind that you may have little to no control over. Controls associated with these dependencies may either be further downstream from the infrastructure in question, or that particular risk may be transferred to another part of the organization (more on this later). When assessing external infrastructure, if intrusive scans or penetration testing is involved, it could even result in a breach of policy or legal ramifications, depending on the organization (particularly in the government).

But perhaps most importantly to us, over-scoping a risk assessment results in wasted time and resources; a condition unacceptable for rapid development, and incompatible with an agile security framework. For example, if an organization already has a robust document destruction policy (and your team follows it), it could be a waste of resources to assess that policy further, as it's not directly related to your system.

You are responsible for protecting the system, assets, and information your team "owns." The assessment should contain any potential threats and vulnerabilities either directly applicable to your system, or external dependencies your system relies on.

Threats and Vulnerabilities

Our risk assessment will be expressed in terms of the threats and vulnerabilities our system and organization will face. The difference between a threat and vulnerability is subtle, yet important. A threat is a negative event that has some sort of unwanted impact to our organization or system, whereas a vulnerability is weakness in our infrastructure or controls that can allow that threat to become a reality. A common way to enumerate both is to express the risk assessment in terms of individual vulnerabilities and the threats they will enable.

It's important to remember that not all threats and vulnerabilities are related to infrastructure alone. Weaknesses in policy and security procedure can also expose us to vulnerabilities, such as a password policy that is so strong that it results in users writing down their passwords en masse. We will talk about unintended consequences more later.

Value of Assets and the Implications of Losses

In order to understand what needs to be protected, we need to understand what our assets are, what they're worth to us, and the implications of losses related to that asset. For the security practitioner supporting infrastructure for a development team, we may only be concerned with the development system we support. In this case, however, we should consider the value of the intellectual property our team is developing, as well.

Loss, for our purposes, refers not just to the cost of the infrastructure, or man hours required to recover data or fail-over to a DR site. Loss also covers intangibles, such as loss of consumer confidence or damage to reputation in the event of a high profile breach. Often, these intangibles can more greatly damage a company than a physical loss.

The total cost of an asset, in terms of money, manpower to restore the asset, and reputation costs should all be considered carefully, as this will drive our risk management decisions and help us determine which controls ultimately make it into our product backlog.

Minimize Exposure to Loss or Damage

If we accept that our mission as an agile security team is to protect our organization's information assets while supporting the mission of the organization, and we further accept that there is no such thing as a completely secure system, then it logically follows that our goal is to minimize risk as much as possible without degrading our organization's capability. Unfortunately, returning to risk management basics means

coming to terms with the fact that executives, security managers, and other personnel have lost sight of that last clause.

It will take effort to return to a mindset of supporting the mission, versus becoming it. Whenever possible, when given a choice between operational capability and a given security standards, err on the side of operational capability. Standards can be adjusted, vulnerabilities mitigated in other ways. But a loss of operational capability is harder to work around.

In order to assist us in minimizing the exposure, and seeing the reduction in risk provided by our controls, a risk assessment format must be simple, easy to read, and applicable to a wide range of situations.

Basic Risk Assessment Reporting

There are dozens of different methods to enumerate and document risks, but they all have several pieces in common. The first is some sort of matrix by which risk is assessed based on the likelihood and potential severity of a given event. The second is a format by which these events are listed, along with the controls associated with them, and the residual risk, or remaining risk after the control has been taken into account. Usually, this is combined with a bunch of other system and assessment information into the risk assessment report.

The formats provided here should be modified as you see fit, to suit the needs of your program.

Risk Matrix

This format is used to assess the risk for a given event. Essentially, the severity of an incident and its likelihood are evaluated together, and where the two values meet is the overall risk for that event. Your organization or your team can determine what the criteria is for likelihood and severity, and you should feel free to modify the overall risk determination to fit your needs and your tolerance for loss.

Likelihood (How likely is the event?)	Severity (potential damage / financial impact if event occurs)				
	Insignificant	Minor	Moderate	Major	Catastrophic
Almost certain	Moderate	High	High	Critical	Critical
Likely	Moderate	Moderate	High	High	Critical
Possible	Low	Moderate	High	High	Critical
Unlikely	Low	Moderate	Moderate	High	High
Rare	Low	Low	Moderate	Moderate	High

Risk Matrix

This matrix is also used to determine the residual risk once a control is in place. Often, those conducting a risk assessment will simply “bump down” the risk a notch for residual risk. It’s important to re-evaluate the severity and likelihood properly with respect to residual risk. If your control will not truly reduce the overall risk, then its usefulness is in doubt.

Risk Assessment Report

The risk assessment report is the subject of great scrutiny within organizations, and this has led to some of them being pretty voluminous in nature. Within some organizations, it’s common to include some of these items:

- Executive Summary – A brief overview of the system being assessed, the method by which the assessment was conducted, and a quick breakdown of vulnerabilities and their residual risks.
- Risk Assessment Methodology – This details not only the methods by which the assessment was conducted, but also defines terms, risk levels, and the steps involved in conducting the assessment.
- Characteristics of the system being assessed – Functional and techno-logical details describing the system. Usually, this contains specific op-erating system, hardware data, and information about the nature of any custom code. This can also include a description of remote user loca-tions and other characteristics.
- A risk management worksheet – This can be done in one large sheet, or by specific tasks or system components. This details each vulnerability or

threat, a risk level, a control or mitigating factor, and the residual risk.

- Plan of Action and Milestones (POA&M). A list of issues needed to bring the system to an acceptable risk level. Commonly generated based on implementation of identified controls.

The problem is, most risk assessment reports are heavy on the filler (Methodology, system characteristics), and by the time a decision-maker gets to the risk management worksheets and POA&M, they're no longer paying attention.

Our challenge, being an agile security team who wishes to produce useful decision making products with a minimum of filler, is to strip everything from the risk assessment that isn't actionable. But first, we need to figure out how we're going to conduct the actual assessment in an iterative, agile way.

The Agile Risk Assessment Process

Scoping the Assessment

The initial assessment will necessarily take more time than future updates. In addition to avoiding over-scoping the entire assessment, fitting the assessment into a time-boxed sprint may be difficult. There are a few options you can take for the initial assessment:

- Don't try to fit the assessment into a sprint – At this point, if you're conducting an initial assessment, you probably haven't constructed your project backlog or had any sprint planning meetings. Recall that the first few sprints will set the tone for your sprint length, so without having conducted one already, your timeframe is nebulous anyway. Consider taking the assessment **out of** the sprint process, and having it be a part of your product planning.
- If you are supporting a development system with an established sprint cadence, consider breaking the assessment into pieces based on the work in progress within developer sprints. Perhaps your assessment activities could take place in a sprint following the one where your development team deployed the code you're assessing. Overall system risk assessments should be conducted as above: prior to the project kicking off (or as soon as feasible).

Data Gathering

When gathering data, we are attempting to get a solid understanding of the current state of the system. This phase will also include identifying critical systems and assets, and reviewing the mission of the system or business unit we're working with.

Identifying critical systems and assets may already be done for you: if the organization has an existing asset classification system, then this could be a trivial process. Remember that this is a risk assessment, not an inventory or audit. It is sufficient for our purposes to identify assets in broad groups and categories.

We will also gather data on existing infrastructure and controls, including procedures and policies. These can be divided into three broad categories, **administrative, technical, and physical**, and examples are below:

Data Categories

Administrative	Technical	Physical
HR Policies (termination/hiring)	Network topology	Server access / locks
Change control process	Data at rest/encryption	Surveillance of server rooms
Organizational Structure	Permission structures	Security of workstations from theft
Incident response/disaster recovery	Group policies	Building access
Account provisioning Process	Intrusion detection Systems	Infrastructure security (wiring closets, etc)
Information Classification	Log collection/archival	Fire suppression
Auditing procedures	Input validation / error checking	Building security systems

Once the state of these categories is known, then the process of determining vulnerabilities and inspecting controls can begin. Once the vulnerabilities are enumerated and controls assessed, we can begin to actually enumerate the risk for any given event.

Vulnerability and Threat Analysis

Our analysis of vulnerabilities and threats needs to be done with an eye toward creating our user stories and abuse cases that will drive the product backlog. Listing these

vulnerabilities could be as simple as running through each major component of the system and brainstorming ideas for the stories, at least initially. Remember, Scrum is iterative, and our risk assessment report is a living document. We can certainly clarify detail later.

The most important piece of vulnerability and threat analysis is re-evaluation of these vulnerabilities after controls are taken into account. Too often, this portion is an afterthought, but an honest re-assessment is key to determining if a control is worth implementing.

When determining impact and likelihood, remember to take into account the complete cost of recovery (manpower, post-incident auditing, etc), including the negative business cost (if any) because of lost reputation. These cost estimates will help us determine the Return on Investment (ROI) for a given control.

The Agile Risk Assessment Report

Previously, we discussed the common items included in a risk assessment report. For our agile security team, we concern ourselves only with the pieces that provide actionable items and enable decision-making. Therefore, our Risk Assessment Report will be significantly different.

Namely, we're going to ignore the system characteristics. Obviously we will be well aware of the topology due to our data gathering and collaboration with other teams. However, this information is present elsewhere in almost every organization, and our assessment does not need to regurgitate it to enable decision-making. The risk assessment methodology will be left out as well. Obviously, we want to ensure our methodology is sound, and we do this through the established improvement processes common to any agile team. We will probably even have a published SOP for conducting the assessments. However, the inclusion of this data in the assessment itself is a waste of time and space.

We simply want to produce a risk assessment report that enables decision-making, and drives further work in the form of stories for our sprints. We need to provide the rest of our team, and leadership, with a high level direction for our security program. This leaves us with three sections for our Agile Risk Assessment: The executive summary, the Risk Management worksheet(s), and the Plan of Action and Milestones (POA&M).

Executive Summary

The executive summary is simply a high level view of what the assessment revealed. It should include a brief description of the system being assessed, the scope of the assessment effort, and should mention any compliance or regulatory guidance used to assist in preparing the assessment. The executive summary should also identify any caveats to the assessment, any circumstances that could have skewed the results, or other information pertinent to the reader.

The executive summary should end with a count of identified vulnerabilities, a count of controls recommended, and a breakdown of the residual risk of the system and an overall risk level for the system.

Try to keep this section brief; one page if possible. The reader should be digging into the rest of the assessment if more detail is needed. The executive summary should be the section of the assessment you would brief to an executive who doesn't care about the details.

Risk Management Worksheets

When enumerating risk, we've already constructed high level stories to do so. In a large system, a single risk management worksheet will not be sufficient, and there are several ways to decompose these into smaller worksheets. One method is to create a worksheet for each system component, however this isn't always easy, as the boundaries between components can be murky.

Another method is a risk management worksheet for each high level user story. These tasks would be based on function rather than component boundaries. Abuse cases that we have developed can be used to enumerate specific risks within these high level tasks.

Additionally, care should be taken to also include process and policy risks, not merely technical ones. The lack of a defined account revocation process upon employee termination is just as much of a security problem as an unpatched device.

Whichever method you choose, we should endeavor to keep the format as simple as possible. Remember that the assessment report isn't a compliance audit, and we will list (in a single table) our plan of action later in the POA&M. The risk management worksheets are a very simple breakdown of risk, associated controls, and residual risk.

Risk Management Worksheet Example				
Policy, System Component, or Task	Vulnerability	Initial Risk Level	Controls	Residual Risk
Remote users working on sensitive documents from home	Laptop / data theft	High	At-rest encryption, device management features allowing remote wipe.	Low
	Credentials or data intercepted in transit	Medium	VPN / DirectAccess with smartcard authentication, IDS on remote access gateway	Low
	Employee data theft	Medium	File level auditing on all sensitive documents, permissions granted only to required users	Low
Corporate Password Policy	Written down passwords due to complexity	Medium	Replace complex passwords with two-factor authentication	Low

This template is a very basic idea of what needs to be included, and can be modified to suit your needs. Some organizations also include a column for who owns a given control implementation, or technical details on how the control will be implemented. However, we will define those details iteratively as we plan our sprints, so including them in the assessment is a duplication of effort.

Plan of Action and Milestones

The Plan of Action is nothing more than an expression of vulnerabilities needing controls, prioritized by severity, and assigned a set of milestones and completion dates. This is used throughout the DoD as a way to plan activities after a risk assessment, and with a few modifications it can benefit our program as well.

If possible, manpower and costs for a given control should be included as well, along with detailed milestones and completion dates. Unlike the risk assessment worksheet, vulnerabilities here are not separated by component or story, but are ranked by severity for the whole system. Further, vulnerabilities we identified in the worksheets which have no current control, we have chosen to accept the risk for, or otherwise are un-actionable will not be included on the POA&M.

By now, assuming you have a good understanding of the product backlog and sprint planning concepts from chapter 3, this should look extremely familiar. The items on the POA&M from an initial risk assessment will become the basis by which we begin to construct our backlog. We have a high level list of vulnerabilities and their potential controls: to begin planning sprints, we simply need to express these as stories which we can decompose and begin working on.

The Risk Assessment Report as a Living Document

As you may have guessed, the process for conducting a risk assessment can be long and difficult, and collecting artifacts related to the assessment can be time consuming. If we had to repeat this process regularly, we could quickly find ourselves unable to accomplish much else.

Treat the risk assessment report as a living document, updated any time the state of the system changes a threat, a control, or adds new vulnerabilities. Since the risk assessment is part of our security product, we should treat this no differently than any other agile product: making iterative, continuous improvements to the product. One good strategy for this is to make updates to the assessment as part of stories that change our protective posture. Instead of having a separate task to update the assessment based on all the work that has completed in a given time frame, updating the assessment becomes a sub-task for any other story, such as installing a new IPS rule or changing backup policies.

What is Acceptable Risk?

The Myth of Risk-Free

The biggest contributor to information security becoming an obstacle in the enterprise is the erroneous idea that a system can be 100 percent secure. The truth of the matter is, even the most secure systems can be breached, and history proves this time and again. Further, the more secure a system is, as a rule, the less usable it becomes to the everyday user.

As an agile team of professionals, who are interested in enabling our organization's mission, and removing obstacles, we must come to learn that a risk-free system is a myth. If we wish our users to accomplish anything at their jobs, there will always be vulnerabilities and threats, and the landscape of these threats changes daily.

The security assessment will identify vulnerabilities, but the more difficult task is determining what to do about them. There are four general possibilities for us to deal with a given risk: Avoid, control, accept, or transfer.

Avoid the Risk

Risk avoidance involves finding a way to accomplish the mission that renders the risk inapplicable entirely. This is often interpreted as removing a given piece of functionality or prohibiting a certain activity in order to avoid the risk, but as a security team who wishes to enable our organization, this should be an option only used if necessary, and if business is not significantly hampered.

Using the previous example: The risk of sensitive data being stolen from a laptop that a user has taken home. We could prohibit remote work on any document above a certain information classification. Whereas data-at-rest encryption would be a mitigation (control) for that risk, ending the practice of working on sensitive data from home would be avoidance.

If a capability or user story introduces risk, but the capability is not particularly useful or required for our business, then that capability is a prime candidate for avoidance through elimination of the capability.

Another way to avoid risks is to find a new way to accomplish the same task. Using the same example, suppose we implemented a virtual desktop infrastructure system, and strong two-factor authentication? The sensitive data no longer leaves our enterprise, and the user's home laptop merely becomes a terminal. Assuming strong authentication with a physical token, there is no longer any major data theft risk if that laptop is taken.

Control the Risk

Many of our risks will fall into this category. Controlling a risk simply refers to finding a method to mitigate either the likelihood or severity (or both) of a risk. Intrusion prevention systems, robust antivirus, firewalls, and most of our other technical measures mitigate a risk. Policy changes can also mitigate risk, including user awareness training and account management policies.

When designing a control, the agile security team seeks to find the least business-disrupting way to control the risk.

Accept the Risk

Risk acceptance seems to be one of the more difficult concepts for security professionals and senior leadership alike to grasp. Nobody wants to be responsible for a breach, but there will always be vulnerabilities for which the outcome of a breach doesn't justify the expense or loss of capability from the mitigation.

Should you encounter personnel who are leery of accepting a given risk, remind them that they do it all the time. Even risks with strong, robust controls do not result in complete elimination of that risk. Every time a "high" or "medium" risk is mitigated, and the residual risk is "low," you are still accepting that reduced risk. Accepting a risk with no associated control is no different.

There is no such thing as risk-free. Risk management is just that: managing risk. At the end of the day, we will always have to accept some level of risk.

Transfer the Risk

Transferring the risk means we have decided that the burden for dealing with the risk falls on someone else. There are a number of reasons for us to transfer a risk, but the most common is that the risk encompasses a system or capability that is simply outside of our realm of control.

For example, if we are securing a development system, we may identify risk related to the physical security of the servers themselves. However, if our organization has a separate physical security team, we may have absolutely no control over physical security. As such, the risks associated with physical security would be transferred to that team. At that point, our responsibility turns to ensuring that team has exercised due care in their efforts.

Another way to transfer the risk is through the purchase of external services or insurance. Within our business, this is most commonly seen in the form of offsite backup services, hosting solutions (hybrid clouds and disaster recovery services), or insurance

for physical loss of infrastructure.

Return on Investment

For any given control, careful thought must be given to whether or not the mitigation is “worth it” from an ROI perspective. The potential cost of a breach must be analyzed:

- What is the total tangible cost of repair to the system if the threat event is realized?
- What costs will I incur for overtime, outside consultants, and other personnel expenses (including loss of productivity for affected users)?
- How badly will this hurt my brand?

These costs must be evaluated against the likelihood of the event to determine whether the expense of the control is worth potential cost to the organization. If the control is significantly more expensive than the cost / frequency of the threat event, the control isn’t worth implementing. These calculations can be done over a multi-year basis: if a threat agent happens frequently enough to cost us money regularly, then a more expensive control will eventually pay for itself. This timeframe for return on investment should become part of our mitigation decision.

NBID: Risk Assessment – A Seething Pit of Despair

During the course of working on NBID, the POA&M that came out of the initial risk assessment never did get closed out. Much of our time was devoted to figuring out how a given action item could be completed without completely breaking our capability. Often, the answer was “It can’t,” so that item was left incomplete on the POA&M. Long running issues on the risk assessment were never fully resolved, including transitioning our traffic to a system that was so restrictive, that avoidance of that system was a good portion of the reason our system was created in the first place.

The failure within NBID, at levels much higher than mine, was the failure to realize that risk acceptance is a valid, and sometimes necessary, risk strategy. The concept of seeing a vulnerability, recognizing that the control was potentially capability-breaking, and refusing to even consider alternate mitigation or risk acceptance was a practice that we could not seem to break for our senior IA personnel involved in the program.

We used to joke that eventually our system would be so secure, nobody would be able to access it, not even legitimate users. The unfortunate reality is that, had the

program continued, that wouldn't have been far from the truth. Ultimately, NBID was a great system of ideas produced by a talented development team, delivered to a customer who lacked the vision to use it, and the flexibility to keep it from becoming yet another government IT system bogged down by onerous security requirements, and more of a liability to its users than a benefit.

¹² Landoll, D. J. (2011). *The Security Risk Assessment Handbook*. Boca Raton: CRC Press.

Compliance and Agility

We've spent quite a bit of time talking about bringing information security back to the basics of supporting the mission through protecting our business assets, rather than becoming a great, lumbering behemoth that becomes an obstacle for our users. Our goal is to strip away pieces of our program which do not further that goal. And while this is certainly a worthy goal; leaning our information security program to just the bare essentials of good protection and sound practices, the reality is that, in any organization, there will be parts of the security program that simply aren't flexible, due to policy or regulation. If we can't modify a standard or regulation to more closely align with our needs, then we need to find a way to work it into our agile security program, finding ways to meet those regulatory or policy guidelines without negatively impacting our business.

Compliance within most security programs simply means verification of adherence to a given standard. Recall that we made the distinction in chapter 4 between a security risk assessment and various other activities, including audits. An audit of our program would be a part of our overall compliance strategy; the challenge now becomes doing this within the confines of a rapidly changing system, and fitting it into an already busy agile security program.

Regulatory compliance is going to vary greatly depending on the particular industry your company is in. Various laws and regulations exist for health care, financial sectors, government, and other sectors. We will attempt to take a standard-agnostic approach here, and illustrate some ways to work compliance into your agile security program.

Regulations and Agility

Increasingly high-profile breaches have demonstrated clearly the need for proper information protection, and in the wake of some very costly (and very public) losses of consumer data, the regulatory environment is increasingly strict and difficult to navigate. As a result, our company may have, as a matter of law, certain requirements. The Payment Card Industry (PCI) Data Security Standard, HIPAA, or FISMA standards are

examples of regulatory burdens borne out of continued failures by firms and IT departments to keep their data secure.

This is an opportunity for agility, believe it or not. Not only do agile methods have the rigor required for compliance goals, but it's actually quite easy to work compliance related activities into our agile program. The first step is to change the way we think about regulatory compliance.

Throughout this chapter, I'll use the Payment Card Industry (PCI) Data Security Standard as an example. This standard is granular and detailed, with responsibilities not only for companies, but vendors supporting them as well. It serves as a good base for which to discuss compliance in the context of agile security. The standards and supporting documentation can be found at <https://www.pcisecuritystandards.org>

Compliance is a Challenge, not a Burden

We need to stop thinking of regulatory compliance as an obstacle, and start thinking of it as an opportunity to showcase our team's effectiveness and talent. We have to use it as an opportunity to shine, and find ways to reduce the burden that compliance brings to our team and organization.

In short, we need to stop thinking of compliance as a burden, even when it is.

Despite the truth of the burden that regulations like HIPAA bring, it does us no good to dwell on these burdens. Compliance with a regulation is no different than implementing any other control. The only real difference is that in some cases, our method for controlling the given risk is dictated to us, rather than something we'll have a bit of flexibility on.

Ultimately, the role of compliance is to demonstrate sound best practices; something our program should have as a goal anyway. And remember, a regulatory guideline doesn't mean you can't find an innovative way to meet (or even exceed) that standard. Allow your self-organizing team to surprise you.

Turning Standards into Stories

The difficulty with some security regulations is that in many cases they weren't written in a way designed to be comprehensible to normal people. Take, for example, the section from NIST Special Publication 800-53 related to rules of behavior/acceptable use policies:

RULES OF BEHAVIOR

Control:

The organization:

- a. Establishes and makes readily available to individuals requiring access to the information system, the rules that describe their responsibilities and expected behavior with regard to information and information system usage;*
- b. Receives a signed acknowledgment from such individuals, indicating that they have read, understand, and agree to abide by the rules of behavior, before authorizing access to information and the information system;*
- c. Reviews and updates the rules of behavior [Assignment: organization-defined frequency]; and*
- d. Requires individuals who have signed a previous version of the rules of behavior to read and re-sign when the rules of behavior are revised/updated.*

Supplemental Guidance: This control enhancement applies to organizational users. Organizations consider rules of behavior based on individual user roles and responsibilities, differentiating, for example, between rules that apply to privileged users and rules that apply to general users. Establishing rules of behavior for some types of non-organizational users including, for example, individuals who simply receive data/information from federal information systems, is often not feasible given the large number of such users and the limited nature of their interactions with the systems. Rules of behavior for both organizational and non-organizational users can also be established in AC-8, System Use Notification. PL-4 b. (the signed acknowledgment portion of this control) may be satisfied by the security awareness training and role-based security training programs conducted by organizations if such training includes rules of behavior. Organizations can use electronic signatures for acknowledging rules of behavior. Related controls: AC-2, AC-6, AC-8, AC-9, AC-17, AC-18, AC-19, AC-20, AT-2, AT-3, CM-11, IA-2, IA-4, IA-5, MP-7, PS-6, PS-8, SA-5.

All of this, to say, essentially, that our users need to know what is acceptable use, the rules for that use, that they need to sign a statement, and that it needs to be re-signed when updated. There's a bit of a bonus that establishes the revolutionary idea that privileged users may have different rules than general users.

This is the essence of why compliance with regulations seems daunting, half the battle is distilling the information into something that makes sense.

Generally, security standards put out by various regulatory bodies take the form of lists such as NIST SP 800-53, and fairly granular standards are dictated for a wide variety of categories. Within the PCI Data Security Standards, the readability is much enhanced, and the guidance comes in a format that not only establishes the requirement, but the procedures by which an auditor should test compliance, and additional guidance for that control.

At a high level, our tasks to get these items into our backlog include:

- Read and interpret the regulatory requirement
- Identify dependencies and their associated stories
- Decompose the task as necessary

- Express the tasks as stories
- Prioritize the story or stories created above and place them into the project backlog

To illustrate this, let's take an item from the PCI Data Security Standards and decompose it into stories for our backlog.

PCI Data Security Standard – Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters.

If you're smart, or thorough, or not insanely busy like most of us, then this will already be completed and you can commence to patting yourself on the back. However, one key piece of compliance is that the actions are only half the battle. Verification is required to say that you're compliant with a given standard. Assuming a medium or large sized business spanning a few floors to a few buildings, and taking into account wireless access points, network devices, storage devices, network printers with built in print-servers... this begins to seem like a much less simple task than it did initially.

So the task now is to break that statement into separate tasks, expressed (if you wish) in story format, and prepare them for inclusion into our backlog.

Read and Interpret the Requirement

Luckily, the PCI DSS is much more readable than some other standards. However, we still need to examine the requirement to ensure we are clear on all that compliance with it will encompass.

The requirement is fairly simple: do not use vendor-supplied default passwords and other security parameters. The tricky bit there is "other security parameters. What precisely do they mean? There seems to be more ambiguity in that statement than we'd normally be comfortable with when dealing with compliance?

I would interpret "default security parameters" to mean any bit of security configuration within the device which is widely known and documented. So, in addition to vendor-default passwords, I would include SNMP community strings, vendor-default account names, as well as default values for any administrative login ports, gateway ports, or other network configuration (with the exception, of course, for intended outward facing ports such as 443 on a web server). Further, enabled default accounts that we do not need should be disabled entirely.

The subtask in this requirement is directly related to wireless networks is significantly more granular, and does not need more work to properly enumerate the

requirements. It states that all wireless vendor defaults are changed, including SSIDs, encryption keys, community strings. It also suggests disabling SSID broadcasts and enabling WPA/WPA2.

The goal is, if the requirement is too vague, develop it. If the requirement is lengthy and incomprehensible, simplify or decompose it.

Identify Dependencies

When evaluating a requirement for compliance, it is necessary to determine what other tasks the requirement depends on. We have a fairly broad mandate in our example to change vendor supplied defaults, and it can be fairly easy to decompose that, as we did above. However, there are some tasks that need to be accomplished before you can set to your task of changing defaults.

- Identify configurable infrastructure and equipment
- Research through vendor literature or login to the devices and determine what default accounts and settings exist.
- Before changing defaults, we need to identify which devices still have defaults enabled, so an audit of some kind is helpful if the information can be remotely queried.

Decompose the Task as Necessary

Identifying dependencies may have helped begin the decomposition work, but if the tasks are still too high-level, they need further decomposition.

In the case of our example, we may need to proceed with some of the prerequisite tasks before decomposing this requirement. For example, we'll need to identify what configurable equipment we have, and any documentation on current settings, so that we can begin to break the very large requirement into smaller pieces.

An easy way to decompose this task could be to simply compose a smaller task for each type of technology: wireless equipment, network routers/switches, storage devices, printers.

Another way could be to break the tasks up by business unit / department (such as finance, human resources, advertising, etc.).

Express the Tasks as Stories

If a task is sufficiently decomposed, it should be fairly easy to express in a small story. Recall that we can express these in terms of equipment as well as users, so take the form you find easiest to use, or eschew the format altogether. As long as the end result is a small, one/two sentence task, it can be put into the backlog. Some stories that could come out of our PCI DSS requirement could include.

- As the security team, I want to conduct an audit of networking devices so I can know what equipment still has default values.
- The accounting department wishes their multifunction machines to have default passwords and settings changed to ensure confidential data cannot be stolen via these devices.
- As a mobile user, I want the wireless encryption settings to be at the maximum supported protocol so that my information cannot be stolen.

Prioritize Tasks and Place into the Backlog

The product owner will take the tasks that our previous steps have identified, and place them into the backlog. Particular attention should be taken to ensure that the priority makes sense and doesn't create extra work. For example, if we have come up with a task to inventory our network devices and determine configurable settings, we should probably do that before we send someone around changing default accounts so that nothing is missed.

Once these items are in the backlog, our compliance tasks become no different than any other sprint. They can even be intermixed with other items on a sprint, depending on the team's workload.

Metrics, Dashboards, and Automation

Trust, but Verify

I would wager that almost no information security program can manually check every system and piece of infrastructure within their enterprise for compliance with the multitude of standards we must adhere to.

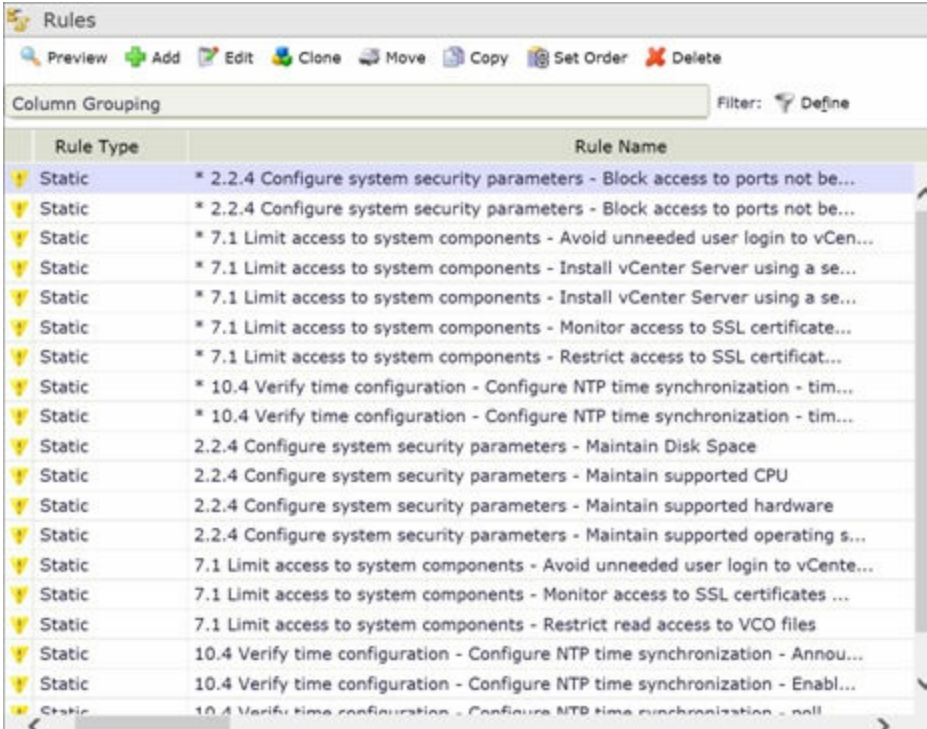
We need to find ways to verify compliance that are more efficient, repeatable, and less troublesome than frequent 100 percent audits. The good news is, enforcing standards in our environments has never been easier.

In addition to tools that actively scan our networks for problems, management suites (including products from VMware and Microsoft) have gotten more robust in their ability to passively maintain a dashboard of data. In most cases, these dashboards can be configured to evaluate your system against a specific standard, much as Nessus plugins are available for Department of Defense Security Technical Implementation Guides (STIGs).

Automating Test Plans, Dashboards, and Compliance Metrics

For every measure you take to comply with a standard, it needs to be centrally, and automatically, verifiable. This means that not only should it be possible to check that particular configuration remotely without physically interacting with the infrastructure, but it should be done automatically on a regular bases, and included in reporting or alerting, depending on the nature of the control.

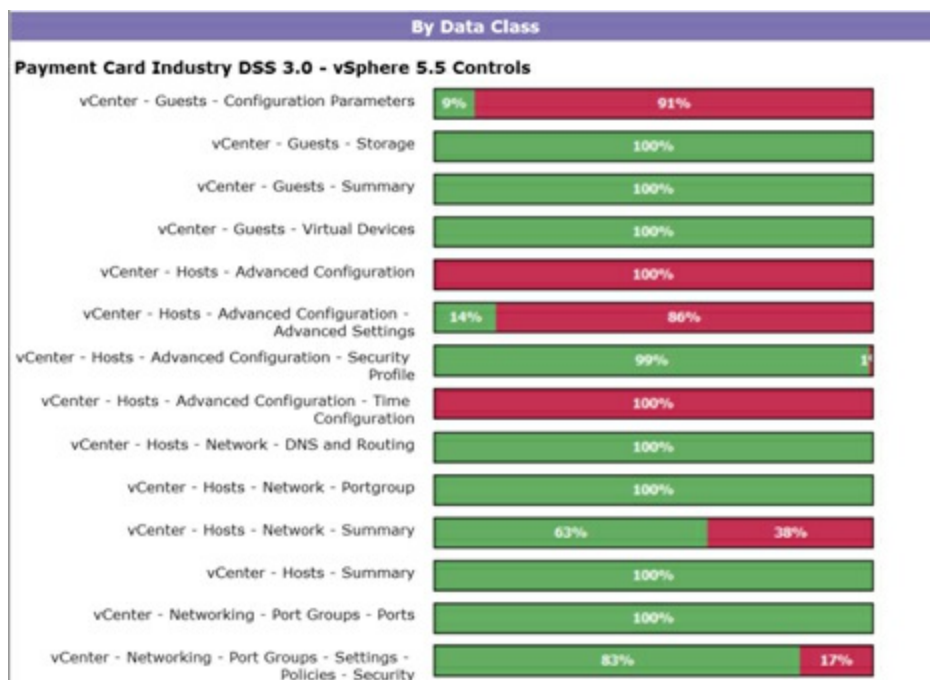
Here is an example of such a dashboard, specific to the PCI Data Security Standard. In this case, the DSS compliance toolkit for VMware's vCenter Operations Suite software provides configurable rules and reporting based specifically off the standards set aside in the DSS, and enumerated by requirement number and subcategories.



Rule Type	Rule Name
Static	* 2.2.4 Configure system security parameters - Block access to ports not be...
Static	* 2.2.4 Configure system security parameters - Block access to ports not be...
Static	* 7.1 Limit access to system components - Avoid unneeded user login to vCen...
Static	* 7.1 Limit access to system components - Install vCenter Server using a se...
Static	* 7.1 Limit access to system components - Install vCenter Server using a se...
Static	* 7.1 Limit access to system components - Monitor access to SSL certificate...
Static	* 7.1 Limit access to system components - Restrict access to SSL certificat...
Static	* 10.4 Verify time configuration - Configure NTP time synchronization - tim...
Static	* 10.4 Verify time configuration - Configure NTP time synchronization - tim...
Static	2.2.4 Configure system security parameters - Maintain Disk Space
Static	2.2.4 Configure system security parameters - Maintain supported CPU
Static	2.2.4 Configure system security parameters - Maintain supported hardware
Static	2.2.4 Configure system security parameters - Maintain supported operating s...
Static	7.1 Limit access to system components - Avoid unneeded user login to vCente...
Static	7.1 Limit access to system components - Monitor access to SSL certificates ...
Static	7.1 Limit access to system components - Restrict read access to VCO files
Static	10.4 Verify time configuration - Configure NTP time synchronization - Annou...
Static	10.4 Verify time configuration - Configure NTP time synchronization - Enabl...
Static	10.4 Verify time configuration - Configure NTP time synchronization - sell

Each rule, then, can result in a passing or failing result, which (depending on your organization) could trigger either an alert or simply a non-compliant entry in a report, as

seen here:



Exceeding the Standard Can be Easier

Remember, a standard for compliance is a minimum guideline. There's nothing that says we cannot exceed it, and in fact we **should** exceed standards if it can better protect our enterprise and support our mission, and is operationally feasible.

One fun side effect of exceeding some standards is that it occasionally will require less periodic maintenance effort than simply meeting the standard. Consider, for example, the DoD's password complexity minimums:

- Passwords must be at least 15 characters long
- Passwords have to contain upper case letters, lower case letters, numbers, and symbols
- Passwords must not contain names, telephone numbers, account names, or dictionary words
- Passwords expire every 60 days
- Passwords must not be the same as any of the previous 10, and must differ from the previous password by at least four characters.

Imagine, if you will, not only the complexity in implementing these rules, but the ongoing effort in resetting passwords and assisting users in choosing passwords that meet these requirements. Further, the frequency of changes means that users will have a

difficult time remembering them. The complexity and frequency means that these passwords will end up on a post it note on a desk in an alarming number of cases, completely destroying the intent of the password policy.

Now, this is a minimum standard. Let's say we exceed it, with a little bit of hardware expense. Assume we've purchased smart cards for our users, and readers for the computers. Two factor authentication with smartcards in the DoD has a six to 8 digit pin requirement, which is much easier for people to remember. Further, the PIN never has to change. We can force smartcard login, and bypass password authentication entirely. Now, we have a more secure method of authentication (two factor authentication), which is less of a burden on our users, exceeds our security requirements for compliance, and results in a massively smaller headache for our administrative staff.

Exceeding the standard when possible is a great thing, but it's even greater when it means less work for you or your team.

Metrics: What Should We Measure

While metrics are important to verify your compliance with applicable laws, metrics related to compliance are not the only things your program should measure. The effectiveness of a security program cannot be measured without supporting data. As with risk enumeration, the collection of metrics within Agile projects (where timeframe and relevance are key) must be prioritized based on their utility to improving the security posture of the environment or enabling decision-making. Again, we produce a working product.

The collection of metrics can fall into two very broad categories: metrics easily analyzed without processing, and those that require detailed analysis (not only of the data itself, but trends and correlation with other metrics). In other words, high effort and low effort metrics. You can also categorize metrics as either objective or subjective. Evaluate carefully the utility of any metric versus its cost (in collection and interpretation effort).

Metrics which can be highly automated or which are easy to interpret and use for decision-making should be prioritized.

Remember, the purpose for gathering metrics is decision support or evaluation of control effectiveness. If a metric doesn't meet those criteria, get rid of it.

“Mean Time” to Whatever – An Often Misused Metric

I'd like to take this opportunity to highlight an often used and seriously abused type of metric, that in nearly every situation I've encountered has had a negative effect on the team it's used against.

Mean Time to Resolution.

The problem with this metric, and "mean time" to anything, really, is that it completely strips circumstance and individual nuance out of the equation. In the interests of driving down this metric, engineers resolve tickets as fast as possible, either deliberately or inadvertently giving the particular problem illustrated by the ticket a cursory, shallow view.

On a help desk, I have seen the efforts to drive this number down result in a user calling many times for the same issue, and the engineer quickly does it for the user and resolves the ticket, instead of spending a longer amount of time helping the user fix their process so that the problem stops occurring.

Within the security field, resolving an alert quickly without a deep inspection of the issue could mask a breach in progress. Repeated alerts from an IDS dismissed as false positives could result in a missed breach.

In general, measuring average times and driving them lower (without context) through management pressure results in teams that are afraid to devote significant time to a problem.

NBID: Balancing Agility and Compliance

The chief difficulty at NBID is evident in its name, Northern Border Integration Demonstration. This meant that infrastructure and processes to ensure compliance with standards were not put in place from day 1, and as a result the entire project was playing catch-up from a security compliance standpoint. We never did have a true operations manager suite. Luckily, the system was smaller, so manually validating various pieces of configuration wasn't an onerous task.

The use of scheduled tasks and automation with PowerShell helped, as certain configuration data could be easily dumped to CSV or PDF for upload to another site where a team of professionals would take that e-mail, check a box on a paper, and send the e-mail to another person, presumably ad infinitum.

The biggest lesson I learned about compliance and reporting with NBID was that it's better to put the infrastructure in place correctly the first time, rather than try to build it in later. Although some of our reporting requirements were addressed in reasonably easy ways, the entire process would have been easier if implemented from the

beginning.

Vulnerability scans were a particular source of pain. As mentioned previously, the development system and production environments fell under two different organizations. Additionally, each cared about the state of the other, so we often found ourselves serving two masters, and attempting to apply two different standards to the same systems.

As a demonstration of capability, this proved to be a significant consumption of our already limited resources. I had an engineer at the production facility in Detroit who frequently stayed late to complete the scans, and probably hated me for passing along whatever the scanner configuration discrepancy of the week was.

The central problem was that our software project was meant to be agile, but the security program we operated under was more rigid than concrete, with no room for finding alternate paths to risk reduction, finding creative solutions to information security problems, and balancing usability and capability with information assurance concerns.

That's why this book was written. That mindset needs to cease to exist.

Implementing Controls, Prevention, Monitoring, and Response

At its core, implementing controls in an agile security program is no different than any other task. We have already discussed expressing risk as stories, and using the plan of action from the risk assessment to put controls into our product backlog. This chapter will instead highlight some concepts behind managing vulnerabilities, as well as discuss the roles of prevention, monitoring, and response.

We will also talk about enforceable controls, and unintended consequences for controls, as well as prioritization, waivers, and security in preproduction/test environments. All of these topics will help us make more intelligent decisions about what goes into the product backlog for implementation.

Vulnerability Management

Managing vulnerabilities is a function of not only our ability to respond to these vulnerabilities, but our ability to effectively identify them, as well. For every vulnerability that makes tech news, there are many which are largely under the radar.

Remembering that agile processes are iterative and incremental, we may see vulnerability scans and software patch verification become a regular, repeating part of our sprints. These processes need to be repeatable, as automated as possible (particularly with reporting), and low-impact to operations

Key pieces of vulnerability management include scanning technologies, patch management processes and products, and monitoring and response (which will be covered later)

Effective Security Scanning

There are a number of robust, effective scanning products out there, and each has its own benefits and devotees. However, vulnerability scanners are also frequently misused and misunderstood.

The first problem is the misconception that a clean scan equals “secure.” The complacency that this misconception breeds is very dangerous to the overall security of a system. Remember, a clean scan means that, for the plugins/modules loaded, the scanner was unable to find a defined set of indicators for any of the vulnerabilities in that plugin or module.

The scanners can be configured to scan for vulnerabilities in many different devices and operating systems, but if your scanner is not configured to use a given plugin, you’ll miss any of the vulnerabilities in that product.

One other advantage of a well configured scanner is the ability to scan for specific compliance issues. For example, commonly used modules for Nessus will scan for nearly every configuration item in the DoD’s Security Technical Implementation Guide (STIG) for a number of products.

Good scanning infrastructure should meet a few requirements:

- Complete – The scanner should support every operating system, device, or other product in your enterprise.
- Easy to scale – The scanner should easily accommodate expansions to your network or changes in infrastructure.
- Customizable – The scanner itself, and reports generated, should be highly customizable to meet your business needs
- Accuracy – The scanner should function well, correctly identifying problems and avoiding false positives.

Patch Management

Patch management can be a difficult challenge in any enterprise, and is especially difficult when other components of the system are constantly changing. The days of installing a product or operating system and then forgetting it entirely are long gone. With the rate at which new vulnerabilities are discovered and subsequently patched, along with the limited lifetime of some products, keeping your systems up to date will be a key task of your agile security team.

Effective patch management starts with sources and data. Not only do you need a method for becoming aware of patch and vulnerability data from vendors, you need to know how it is applicable to your systems. A comprehensive inventory, updated with any additions/changes, is key to understanding what assets are under your control.

Further, for any products or devices in your environment, subscriptions to

vulnerability/patch notification lists or other publications are a must. Wherever possible, use operations management suites to accomplish both functions. Maintaining inventory and awareness of patch data are functions of many modern system management packages.

Ideally, you will first deploy patches to a preproduction environment, however this is not possible in every organization. If you don't have a preproduction or test system available to you, patch low impact systems first and observe for any negative changes.

Monitoring and Response

Many security teams do a good job at prevention through vulnerability scanning, good design practices, and countermeasures. Fewer, however, are as robust in their monitoring and response efforts as they are with prevention.

The problem here is that monitoring and response are every bit as important (if not more so) than prevention. All efforts to prevent breaches, data leakage, or other security incidents will eventually fail, either due to new vulnerabilities, improper implementation, or the biggest information security weakness: people. Therefore, to paraphrase the agile manifesto: while there is great value in prevention, we value monitoring and response more. Ideally, we've done a good enough job in our prevention efforts so that we rarely have to put our response plans into action, but we should always assume that we will be breached, and plan accordingly.

Are We Secure?

I don't know, and neither do you.

Richard Bejtlich, in his book *The Tao of Network Security Monitoring: Beyond Intrusion Detection* reminds us that no organization can truly be called "secure" for any significant length of time beyond the last verification of adherence to policy.¹³

Security is not the complete absence of risk, but rather, according to Bejtlich, "the process of maintaining an acceptable level of perceived risk." I've found Rich's definition to be immensely helpful in keeping the scope of my efforts reasonable over the years.

An agile security team must have a realistic expectation regarding end-state and security posture. Security is a continuous cycle of assessment, countermeasures, detection, and response, and agile methods are perfectly suited for this kind of worldview.

Contrary to popular belief, there is no such thing as a completely hardened system. The ability to recognize and come to terms with that fact will drastically change how you approach information security. Ultimately, it's the management of risk (including the acceptance of residual risk) that will guide our programs.

Incident Response

There are two types of enterprises: those that have had a breach or security incident, and those that will. The differences between good and bad information security programs are directly reflected in their ability to respond to a breach or incident. Your monitoring infrastructure ties directly into this incident, as forensics and monitoring provide vital information used for decision-making.

Though a complete monitoring solution is outside the scope of this book, at a minimum your monitoring infrastructure should allow you to identify in-progress anomalies and issues, as well as answer the following once a breach has happened:

- What was breached?
- When was it breached?
- Is it still compromised?
- What else is compromised as a result?

The answers to these questions guide the next steps in the response process, which depend heavily on the nature of the intrusion. The role of your agile security team is to decide quickly what steps need to be taken, and placing your response plan into action.

Before we dive into the response plan, I'd like to recommend taking a look at Bejtlich's book for further information on monitoring, and in particular call out his sections on what makes a network defensible. *The Tao of Network Security Monitoring* is a fantastic title that is a permanent fixture in my workspace.

Bejtlich's Characteristics of Defensible Networks

Defensible networks can be watched: they must be designed with monitoring in mind.

Defensible networks limit an intruder's freedom of movement: once an intruder is in, they should not be able to get much farther.

Defensible networks offer a limited number of services: Much as a good permissions policy operates on the principle of least privilege, your network or servers should offer only the minimum number of services to accomplish the mission, nothing more.

Defensible networks can be kept current: the OS, core services, and other software must be kept

current. If a piece of custom infrastructure does not support the latest operating system or patch, then that custom infrastructure needs to be changed.

Ensuring that your systems are defensible will go a long way to helping you avoid impromptu testing of your incident response plans.

Incident Response Plan

There has been a great deal of discussion on how a Computer Security Incident Response Team should handle an incident, with a great deal of guidance on various methods. Common to all, however, are components like gathering of evidence, minimizing the damage, and communication of the incident to users and law enforcement. The most basic incident response plans contain, at a minimum, the following steps:

- Initial Notification and Assessment
- Contain / Stop the incident
- Gather Evidence and Information
- Recovery
- Post-Incident Assessment
- Notification and Communication (to include external agencies and law enforcement if necessary).

Initial Notification

The initial notification of an incident may vary from company to company, but at a minimum, it functions to allow senior resources, engineers, and leadership to be informed of and immediately engage on a security incident. The initial assessment should validate whether an actual incident is occurring, provide as much detail as quickly as possible as to the nature of the attack, and begin thorough documentation of the incident throughout its lifecycle.

Containment

The first priority once an incident has been recognized is to stop the attack or breach as soon as possible. In the vast majority of cases, this will mean disconnecting or disabling resources and services from the network. Sometimes, you may need to keep a service

online even during a breach, so you may need alternate ways to mitigate the impact. The highest priority should be removing the attacker's ability to damage your network.

Gathering Evidence

Once the incident is stopped or contained, the process of gathering evidence can begin (and this may be done in conjunction with the recovery). The evidence gathering process goes beyond investigating event logs. Detailed inspection of the systems breached for unauthorized changes to configurations, groups, changes to software installed or startup items, and other data are essential bits of the evidence gathering process.

If you have a requirement to pursue legal action against the intruder, or have decided to take that course of action without a requirement, preserving evidence is vital. Consider a complete, static, bit-by-bit copy of the hard drives of the entire systems affected.

Recovery

This stage is where your disaster recovery plan comes into play, as there's very little difference in a breach due to a security incident and breaches due to other disasters. The key piece of this step is identifying precisely the scope of the breach, and perhaps more importantly, how far back the breach went.

Remember, we've only just detected the incident. It is possible that this sort of activity was happening for some time. Verify and validate the scope of the breach, so you know how far back your "clean" backups are.

Although there are methods to recover from breaches and malware that do not require a full system rebuild, I nearly always prefer erring on the side of caution. If a particularly complex and nasty bit of malware was introduced during the breach, failure to remove it entirely could cause the problem to come screaming back in short order. Whenever possible, wipe and restore the system entirely.

Post Incident Assessment

After the systems are recovered, and the breach has been stopped, the evidence and information from throughout the incident need to be analyzed and interpreted to give your team the ability to do a thorough post-incident assessment.

This assessment will not only distill and organize the evidence from the incident,

but also assess incident damages and costs. These costs include the actual manpower/material cost of the incident, as well as lost productivity, damage to reputation or consumer confidence, legal costs, and any other cost that would not have been incurred otherwise.

Remember to ensure that proper evidence handling is used during these phases. Evidence rules vary by location, so your team should ensure that the proper handling procedures are followed for your locality. The last thing you want is to be unable to pursue legal action against an attacker because the evidence was somehow compromised and is now inadmissible.

The post-incident assessment should identify not only the details of the incident and identify pending actions, but also should be used as a springboard for improvement. Our team needs to know what we did right, and what was not effective, from the perspectives of prevention, monitoring, and response. If we handled the incident well, but our system was breached due to an exploit that should have been patched already, our prevention systems and processes need improvement.

Most importantly, now is not the time for management to berate their security teams and look for a scapegoat. A breach is an event that can happen to any organization, and playing the blame game doesn't assist us in improving and stopping the next one. Ultimately, if we've done our jobs correctly, our response plan will be good enough so that when prevention does fail, the impact is limited. When evaluating how bad the incident was, try to remind yourself how bad it could have been? If your team is doing their jobs, the answer will be "much worse."

Notification/Communication

Communication and notification are events that ideally will be happening throughout the incident. Not only do users and management need to be made aware of the incident, but communication must be established with external partners, law enforcement, or anyone else who could be affected by the incident.

At the conclusion of the incident, a summary communication is a good idea. Throughout the incident, new information will come to light that could invalidate previous communication efforts. As a result, a summary of the total incident can provide clarity and wrap the incident up.

Unintended Consequences and Enforceable Controls

When evaluating potential controls (or controls already in place), it's imperative to include not only an evaluation of what unintended consequences the implementation of that control could have, but also whether the control can be enforced.

Unintended Consequences

Earlier, I used the example of two factor authentication being easier to implement, and having less of an ongoing support requirement, than 15 character complex passwords. On paper, from a strictly technological standpoint, this seems like a smart call. After all, the longer and more complex the password, the harder it is to brute force the password and compromise the account. Therefore, complex passwords are good, right?

The reality is that our average users are inconvenienced by this, and you'll start finding sticky notes all over the office, which will eventually make their way into the wrong hands. By increasing the requirements in an effort to become more secure, we became less secure.

This is a prime example of an unintended consequence, and situations like these are precisely where an agile process for information security really shines. Our product is incremental and iterative, and we are constantly refining our products. Of course, our preference would be to think about and properly predict these unintended consequences, but the rapid deliveries we produce as members of an agile team ensure that consequences such as these are detected quickly, either by our testing process or by users themselves, and we can immediately produce an action item to correct them.

In the case of the passwords, we have a story ready to go: "I want my authentication scheme to be simple, so that I don't put my account at risk by writing down an enormous password."

Enforceable Controls and Standards

Earlier in this book, we defined what "working software" in agile development meant in terms of a security program. Obviously, we had to redefine the terms a bit, and use the phrase "working product," defining that as controls and policies that mitigate risk, documentation that supports that mission, documents that enable decision-making, and training artifacts that make our users more security conscious. I believe that in an agile security program, products that don't meet those criteria have little value.

There is another quality, however, that can render controls and standards useless: being unenforceable.

This could be a written policy for which verification is impossible, or policies that are so difficult to live with, that employees avoid the technology involved entirely. Recently, I heard about a “complex pin” policy on a mobile application for enterprise mail on iOS devices. Essentially, the old standard of a simple numeric pin was deemed unacceptable. This pin was entered by the iOS numeric keypad, and was relatively fast, unobtrusive, and generally accepted by the user base. The new policy required not only a longer PIN, but one that contained letters as well as numbers.

The side effect of this was that instead of the easy to use iOS numeric keypad, the application now used the full keyboard. Any time the application was launched, the user was forced to enter this pin on much smaller keys, which proved to be too tedious for normal users. The result was that usage of this application plummeted immediately as users decided to simply avoid their e-mail on iOS devices. The only measure by which this control could be considered a success would be if the goal was to improve the company’s work/life balance.

Ineffective policies include policies designed to address problems that don’t exist, policies with no clear consequences or ownership, and policies which are so superfluous that your users simply don’t care. Additionally, enforcement of policy is entirely dependent on your team’s capacity to monitor them; if your enterprise insists on monitoring every single web page attempt by users, you may find that your team has no bandwidth for anything else but reviewing web filter logs.

You should always endeavor to write policies that make sense for your environment. If the policy brings business hardship to your users, it’d better be mitigating a very important threat; and the importance of that control needs to be communicated clearly and effectively to your user base.

Prioritizing Vulnerabilities in an Agile Environment

Your agile security team’s greatest asset is that its members possess not only the expertise, but the authority to make intelligent decisions on how to mitigate a given vulnerability. The product owner prioritizes backlog items, and the execution of these items is left to the team. This relationship requires that the product owner takes input from the team, senior management, and other stakeholders about the relative importance of a given vulnerability, and makes decisions accordingly.

The very nature of a fast moving project (such as NBID, or any other system in which change is constant) means that your team will necessarily need to occasionally push a given mitigation further down the overall product backlog. Not only do these decisions require measure against other vulnerabilities and items on the backlog, but

they should be measured against the user impact of a given control. If, for example, immediate mitigation of a low-impact vulnerability will cause a high degree of difficulty for your users, there may be an opportunity to find alternate ways to mitigate the vulnerability without adverse effects.

This is where an agile team really shines: finding alternate, creative paths to solve a problem and minimize impact on users. Absent legal and compliance concerns (which can make strict implementation of a control unavoidable), a potential control should always be evaluated not only in terms of ROI and residual risk, but the negative impacts on the business. Remember, information protection exists to support the mission of your enterprise, not become the sole mission.

Preproduction Systems Security

Within many environments, preproduction is a valuable source of test data for your IT department, where patches and updates can be rolled out in a near-production environment, providing an opportunity to catch any negative consequences prior to implementation in production. It's also a great place for power users to try new things without affecting their normal work environment.

The complication that arises when you're in a rapidly changing environment is that your developers will be using this environment for testing as well, which can introduce serious security challenges. You can't be sure of the true impact a new control or patch will have in production if your preproduction environment differs significantly.

The good news is, the robustness of enterprise-grade virtualization products allows additional environments to be stood up with relative ease. If your infrastructure allows, consider having multiple small preproduction environments, each scoped to testing from a different perspective. For example, consider a SharePoint 2013 farm, with SQL servers, web front ends, and application servers supporting not only out-of-the-box services such as search and publishing features, but custom web parts, workflows, and other third-party add-ons:

- One environment could be completely identical to production from a patching and custom code standpoint, allowing your user base a test bed that will leave them no surprises when changes are rolled to production. This environment can be kept relatively small if the user testing need is minimal. This environment, assuming custom code is the same, can also be a place for your security team to test and validate sprint backlog items prior to implementation in production.
- Another environment which maintains a state of patch and control parity, that

isn't used for user testing, which your developers can use as a sandbox for new code deployments, ensuring them reasonably high confidence that no unexpected problems will crop up in production

Not every enterprise will have the ability to maintain more than one preproduction environment. In this case, some decisions will have to be made on whose preproduction testing is the highest risk, and that team or activity will have to have priority within this environment.

Ideally, your team's security controls should be tested last. Any controls you intend to implement in production that will occur in conjunction with a custom code deployment need to be evaluated not on what is **currently** in production, but what will be in production when you implement these controls.

The determination of preproduction systems and procedures surrounding them should be decided early on, and codified into policy and change control mechanisms to ensure repeatability. The last thing your team needs is a breakdown in process which leads to unexpected surprises when custom code and your team's controls reach production.

NBID: When Information Security and Operational Requirements Collide

The NBID development team was extremely talented, but they were also small and overworked. New features, enhancements, and other work were piling up fairly steadily. Obviously, with limited resources, prioritization is a must, and certain concerns are pushed lower in the priority of work.

At one point, work was underway to update a particular piece of code for a new version of Java, as the previous version had gone end of life. There were not yet any exploits in the wild on that previous version of Java, but since it was not getting updated, the senior information assurance officers involved in the project mandated that we upgrade.

The piece of software in question was a centerpiece of NBID, a valuable safety and situational awareness tool for agents and leadership, and was a crucial piece of our overall strategy. The potential risk to not having this functionality was massive, while we felt the risk of a Java exploit was small. However, the government Information Assurance Machine is a difficult force to resist against.

We had other countermeasures involved, including firewall ACLs and other networking infrastructure that severely minimized the risk of any breaches. We were learning, in real time, about assessing residual risk and deciding to accept it. The

resistance we encountered was fierce, and we were unsure for some time if a waiver would be granted.

To his credit, our team's information assurance officer, Charlton, was absolutely on our side. He did a detailed assessment of the risk, all controls in place, and worked very hard to make sure that the senior leadership on the program knew precisely how important this feature was. Additionally, his role as our IASO never felt adversarial; not only was he our greatest advocate from a security standpoint, but he fought hard when colliding security standards from the Navy side and the DHS side caused difficulties within our team. His standards and commitment to our security and accreditation was high, but he never let that commitment cloud the mission of NBID and what we were trying to accomplish. He represented how I believe an IASO should act, and he embodied the servant leadership principles and other qualities that I highlighted in chapter two, always seeking to find a way to reconcile security standards with new capabilities, instead of just defaulting to security policy and flat-out telling us "no."

Unfortunately, his outlook did not extend to the more senior IASOs involved in our project. This was one of the key times where I knew that something was wrong in the world of information assurance and security. In some folks' minds, a not-yet-found vulnerability in Java, mitigated by other controls in the system, was more important than the complete loss of a crucial piece of our Northern Border technology. Information Assurance, in this case, had lost its way entirely. The seeds of this book were planted over a Shiner Bock and a burger as I sat confounded wondering how people could be so entrenched in their information assurance worlds that they had forgotten about the real world missions and people they supported.

¹³ Bejtlich, R. (2005). *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Boston: Addison-Wesley.

CHAPTER 7

Final Thoughts

It is my hope that somewhere in this brief volume, you've been able to find ideas, concepts, or an anecdote or two that can help you avoid some of the challenges I faced with a fast-moving environment like NBID. Although I believe the concepts in this book to be widely applicable to a variety of teams, both within information security and elsewhere, there will always be elements of your information security program that are simply inflexible, due to either operational requirements

My intent is that you take from this book what is useful, modify it to suit your needs, and use what I've written within to improve your teams. I welcome feedback, and my hope is that any future editions or expansion of this book can reflect real-world feedback from folks who have found this writing helpful to them. I encourage you, the reader, to contact me at james.fitzer@agilesecuritybook.com with feedback, personal lessons from your own security teams, or any other responses you'd like to provide.

As I wrap this up, I'd like to reiterate a few final thoughts that I feel are central to this book, and are applicable regardless of how many agile principles or methods you are able to implement within your own teams.

Share This Book

I didn't write this book from the perspective of someone who wants a writing career. Nor did I write it from the perspective of a foremost authority on agile development in general. I wrote this in the hopes of providing some advice and guidance to people and teams struggling with the same challenges I experienced at NBID.

To that end, I encourage you to share this book to others if you've found it useful. If you have a colleague in your team or another who you feel could benefit, pass it their way. I would rather see a few hundred people download the eBook and benefit from what is inside it than to see thousands of people buy it and leave it sitting on a shelf. I'd consider it a success if one day I ran across a dog eared, abused copy of it with obvious signs of use.

Origins of This Book

What you're reading began as my final capstone project for my Masters in Information Assurance and Security. My original intent was a comprehensive thesis on using agile methods to construct an information security program. What I found, however, is that although there's a great deal of literature on both agile development and information security, most efforts to unify the two consisted of academic papers that were fairly short and ambiguous, which didn't deal with the conflict between agility and the compliance and organizational challenges which oppose it.

The book reflects not an end-all expertise on my part, but a struggle to unify the mindset and organizational changes I felt NBID needed with the security standards and realities that I was faced with. As I worked my way through the thesis process, I realized that the best way to do that was not by producing another academic paper that would be cited by undergrad students and used by nobody else, but in a book that I could get out into the field.

The faculty and staff at American Military University put a great deal of faith in my pursuit of a creative project versus a traditional thesis, and throughout the process I was fortunate enough to also receive constant feedback and reality checks from NBID developers I trust, who are not only talented developers, but have a far better grasp of agile and Scrum than I do.

Return to the Basics

If there is one concept above all others that I hope my readers gain from this book, it's the desire to return information security to its roots. As *Information Security Fundamentals* reminds us, information security needs to be a "business enabler," and while the protection of our organization's assets is vital, I feel that along the way many information security personnel have forgotten their mandate to support the mission of their organization.

This principle is reiterated in information security texts across the country, and yet repeatedly I find examples within the industry where this concept seems to have become lost within a sea of bureaucracy.

The beautiful thing about agile principles is that they can be applied to nearly any business. While you may not have the ability to focus all your work into short sprints, the overall principles of responding to (and thriving on) change, measuring your progress on functional products, and creating a self-organizing team can be applied to any team whose members are competent, whose leaders are supportive, and who believe in their organization.

Don't use your security program to lord over your user base. Instead, treat every conflict between security and usability as an opportunity to bridge that gap, making your team a valued resource and trusted partner, rather than an enemy and an obstacle.

Be a Servant Leader

Whether you are a leader in your organization now, or are simply a member of the rank-and-file attempting to create positive organizational change, the goals of servant leadership will benefit your team and make you an asset to an organization. At every opportunity, find ways to make your team more effective and remove obstacles from their path. As a leader, you exist to benefit your team, not the other way around. I guarantee that if you approach your job with a desire to improve your organization, and make life easier for those you support, then you'll find yourself assuming the mantle of leadership eventually, even if you aren't a leader right now.

Constant Improvement

No matter which agile framework you choose to operate in, or even if you don't adopt an agile method at all, the "feedback loop" of constant evaluation of your product and processes is a must for a healthy team.

Develop yourself and your team through constant improvement in how you do business. Don't allow yourself to stagnate and be comfortable with the status quo. Regardless of how effective your team is, there is always room for improvement. Treat every problem and obstacle as an opportunity to better support your enterprise, learn lessons, and protect your organizations assets.

Always be an advocate for positive organizational change, despite the resistance. If you present ways to change your business that will benefit your enterprise, eventually someone will have to listen.

But above all else, never accept merely meeting the standard as a success. In order to combat the bloat, bureaucracy, and red tape that has made security a productivity killer within so many organizations, it's vital that we refuse to accept simply adhering to established standards as "good enough." Adopt the agile manifesto as part of your overall outlook on all your information security needs, and trust that by focusing on delivering a solid product, all the rest will fall into place.

Bibliography

- Beck, Kent, et al. (2001). *The Agile Manifesto*. Retrieved from <http://www.agilemanifesto.org>
- Bejtlich, R. (2005). *The Tao of Network Security Monitoring: Beyond Intrusion Detection*. Boston: Addison-Wesley.
- Cichowski, M. (2011, March 24). *Northern Border Security goes High Tech*. Retrieved from FoxNews Liveshots: <http://liveshots.blogs.foxnews.com/2011/03/24/northern-border-security-goes-high-tech/>
- Cohn, M. (2010). *Succeeding with Agile: Software Development Using Scrum*. Upper Saddle River: Addison-Wesley.
- Landoll, D. J. (2011). *The Security Risk Assessment Handbook*. Boca Raton: CRC Press.
- Naval Surface Warfare Center - Dahlgren Division. (2011). *Navy Engineers and Scientists Support New Customs and Border Protection Operational Integration Center*. Retrieved from <http://www.navsea.navy.mil/nswc/dahlgren/NEWS/NBID/NBID.aspx>
- Peltier, T. R., Peltier, J., & Blackley, J. (2005). *Information Security Fundamentals*. Boca Raton: Auerbach Publications.
- Pham, A., & Pham, P. (2012). *Scrum in Action: Agile Software Project Management and Development*. Boston: Course Technology.
- VersionOne. (2013). *8th Annual State of Agile Development Survey*. Retrieved from <http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf>

ABOUT THE AUTHOR



James Fitzer is an infrastructure engineer for Microsoft's SharePoint Online service, a 15 year veteran of the United States Army, and a freelance author and security enthusiast. Prior to his work with Microsoft, he was the lead system engineer for the U.S. Navy's Northern Border Integration Demonstration project, a three year demonstration of system integration and rapid development technologies for the Department of Homeland Security.

He holds a Master of Science degree in Information Technology with a concentration in Information Assurance and Security, VMWare Certified Professional – Datacenter Virtualization, Cisco Certified Network Associate, and other certifications.

His hobbies include flying small airplanes, playing guitar, riding his Harley, and virtually any other hobby that involves turning money into noise in creative ways.