The Rugged Handbook

Strawman Edition

August 2012

Acknowledgements

The idea of Rugged Software was conceived by Josh Corman, David Rice, and Jeff Williams in 2010.

The following experts and participants spent a week together, at the 2012 Rugged Summit, developing the details of a Rugged software development program. This Handbook is the result of the work at the 2012 Rugged Summit.

- Ramesh Balasubramanian
- Justin Berman
- John Bernero
- Nick Coblentz
- Josh Corman
- Gene Kim
- Jason Li
- Randy Meyers
- John Pavone
- Ken van Wyk
- John Wilander
- Jeff Williams
- Chris Wysopal

We acknowledge the "Emperor's New Clothes" bravery of these participants for their willingness to acknowledge the tremendous difficulties our industry faces and to seek out new and better solutions.

We gratefully acknowledge the support from Aspect Security, who hosted the 2012 Rugged Summit and drafted the initial version of the Rugged Handbook based on our work, and from the Federal Deposit Insurance Corporation, who sponsored the Summit and initial development.

Join the Rugged Project!

This handbook is a strawman version intended to generate discussion, comment, controversy, and argument. We expect significant development and changes to occur in future versions. The Rugged Project is actively seeking anyone interested in finding new and better ways to create secure software. Please join us and contribute your ideas.

http://www.ruggedsoftware.com

License

Table of Contents

Acknowledgements 2
The Rugged Manifesto 5
Introduction 6
Getting Rugged! 12
1. Monitor Threats Like a Town of Prairie Dogs 12
2. Work Together Like an Ant Colony 13
3. Unify Defenses Like a Herd of Muskoxen 14
4. Control Your Environment Like a Beaver Family 15
5. Defend Yourself Like a Honey Badger 16
6. Strengthen Yourself Like a Bighorn Ram 17
7. Adapt and Evolve 18
Telling the Security Story 20
What a Security Story Is Not 21
What Comprises a Security Story? 21
Writing a Security Story 22
Rugged Roles 24
The Rugged Executive 25
The Rugged Security Analyst 28
The Rugged Architect 31
The Rugged Project Manager 33
The Rugged Developer 35
The Rugged Tester 37
Getting Started with Rugged 40
Proving that You're Rugged 42
Slash Security (http://www.example.com/security) 44
Supporting Tools and Technology 45
How Rugged Fits 48
Case Studies 51

The Rugged Manifesto

I am rugged and, more importantly, my code is rugged.

I recognize that software has become a foundation of our modern world.

I recognize the awesome responsibility that comes with this foundational role.

I recognize that my code will be used in ways I cannot anticipate, in ways it was not designed, and for longer than it was ever intended.

I recognize that my code will be attacked by talented and persistent adversaries who threaten our physical, economic, and national security.

I recognize these things - and I choose to be rugged.

I am rugged because I refuse to be a source of vulnerability or weakness.

I am rugged because I assure my code will support its mission.

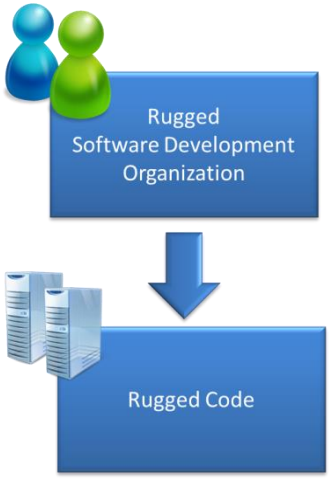
I am rugged because my code can face these challenges and persist in spite of them.

I am rugged, not because it is easy, but because it is necessary and I am up for the challenge.

Introduction

"Health is a state of complete physical, mental and social well-being,
and not merely the absence of disease or infirmity." – World Health Organization, 1948

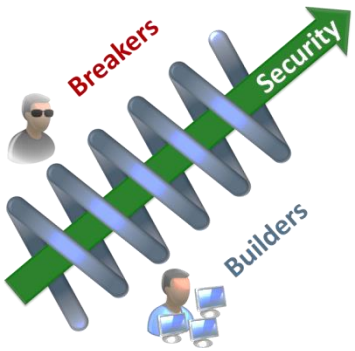
What *Is* Rugged?



"Rugged" describes software development organizations which have a culture of rapidly evolving their ability to create available, survivable, defensible, secure, and resilient software. Rugged organizations use competition, cooperation, and experimentation to learn and improve rather than making the same mistakes over and over. Rugged organizations also actively seek out threats and create defenses before they are a problem.

"Rugged" can also be used to describe the software applications produced by rugged organizations. These applications are not only well-protected against attacks, but are also able to analyze themselves, report on their own security status, detect attacks in progress, and respond appropriately.

Rugged is NOT a technology, process model, SDLC, or organizational structure. It's not even a noun. Rugged is NOT the same as "secure." Secure is a possible state of affairs at a certain point in time. But rugged describes staying ahead of the threat over time. Rugged organizations create secure code as a byproduct of their culture. You are rugged because you run the gauntlet, instrument your organization and your code, constantly experiment to see if anything breaks, and survive the process of hardening yourself through real-world experience. Rugged organizations produce rugged code – designed to withstand not just today's threat, but future challenges as well. Rugged also represents a balance between two approaches to establishing, verifying, and monitoring security – a positive control-based approach, and a negative attack based approach.



To encourage this culture, one of our key ideas is to make security visible with a specific and understandable "security story" that captures everything necessary to understand why the code and the organization that built it are Rugged. The security story is both a guide and a reflection of security in the development organization and the software. Making it explicit allows you to create a culture where "builders" and "breakers" can compete and cooperate to make it better. Without a unifying story, all you have are thousands of security fragments that are impossible to understand.

We don't view Rugged as a burden or cost. In fact, this approach should yield time and money savings, while reducing risk. Because the alternative is unacceptable, we choose to be rugged. Not because it is easy, but because it is necessary, and we are up to the challenge.

Why Rugged?

Whether we choose to or not, we all *trust* software with our lives, our finances, our safety, our government, our communication, our businesses, and even our happiness. This increasing dependence demands that we build our software to survive and thrive despite any threats and challenges we might face, now or in the future. While many brilliant minds have pursued this goal and discovered many powerful techniques and technologies, the complexity, connectivity, and criticality of our software is unquestionably increasing faster than our ability to understand the risks associated with that growth.

We envision a world where software can be trusted with the most important activities of humanity, where software enables us to fulfill a higher purpose without enabling those who would profit by harming others, and where the power of software benefits everyone.

Nature provides many rugged examples, none more popular than the unstoppable honey badger. Consider how nature has provided the honey badger the ability to shrug off bee stings and cobra venom. Or how lizards can lose their tail when caught by an attacker. Natural protections continue to evolve through experimentation, competition, and natural selection based on real world threats.

On the other hand, many systems built by humans are quite fragile in the face of advancing threats. The Titanic was engineered to be unsinkable yet couldn't handle a quite predictable threat. Dams, computer systems, skyscrapers, and other human creations generally fail badly when faced with earthquakes, tsunamis, hurricanes, and power outages, not to mention intelligent human adversaries. Yet we are racing into an era where software controls everything including medical technology, financial systems, the power grid, military defense, the water supply, and just about every other aspect of our lives.

We have a lot to learn from nature about what it really takes to design and implement truly rugged systems. We're proud of the software we create, and it hurts us to see it misused to harm others. We are excited when people use our software in ways we never intended and want to encourage this creativity without endangering them. That's why we are interested in pursuing Rugged.

Scope

Rugged and this Handbook are far from complete. In fact we've only just started. We are publishing this early version, a "strawman," with the intent of provoking you, challenging you, and inciting you to think differently about security, assurance, development, and operations.

For now, we have chosen to severely limit our scope in order to get to an initial release more quickly. In this strawman, we have chosen to focus only on application security for individual projects building new software. In the future, we hope to extend the ideas of Rugged to…

- Buying and selling software
- Understanding the entire software supply chain
- Network security, physical security, database security, etc…
- Other types of software projects, including legacy code, outsourced code, libraries, etc…
- Enterprise level security as opposed to individual projects

We have no *proof* that any of the things that we recommend in this handbook will work. Of course, nobody has any proof that *anything* actually works (or that we even know what "works" means). What we do know is that we need to try new approaches, for it is certain that the problem is scaling up much faster than our ability to apply our current techniques.

We invite you to participate in the Rugged Project. The simplest way is to give it a try and let us know how it worked for you. We would love your help further developing Rugged or figuring out how to extend Rugged concepts. You can start by visiting the website at http://www.ruggedsoftware.org. We challenge you to forget everything you know about application security and re-imagine how it could be reinvented to produce superior code.

Background - Security Challenges

Application security tools and techniques have both advanced in the last decade. However, any fair assessment must conclude that we are not making significant progress in our field. We are still struggling with simple vulnerabilities like cross-site scripting and SQL injection, while more complex vulnerabilities, such as access control and business logic flaws, go largely ignored. At the same time, we are relying on our applications for more critical activities, making them more complex, and interconnecting them at an unprecedented rate. Simultaneously, there is strong evidence that attackers are continuing to get better organized, more capable, and more aggressive.

We are convinced that negative and reactive approaches to application security cannot scale and are doomed to fail. These approaches primarily rely on finding vulnerabilities and fixing them. We believe that fixing holes without establishing strong defenses against the threats that matter does not make sense. Yet most organizations today use penetration testing or automated tools as their sole source of assurance. These organizations are not learning from their mistakes, they simply patch over the symptoms and continue introducing the same vulnerabilities over and over again.

The best projects today perform activities like threat modeling, security architecture, secure coding training, and security testing. However, it's generally unclear how these activities connect back to the business goals that are important to the organizations. That is, there's duplication, gaps, and no line of sight that explains the benefit of these activities in business terms. Frequently these activities simply report vulnerabilities or risks that do not become part of any sort of coherent security strategy. In fact, most of these efforts create no lasting value, and are simply repeated from scratch after some period of time.

The resulting lack of visibility has a chilling effect on the ability of developers, architects, testers, and many others to focus on quality (including security). This condition often leads to security being wrongly viewed as an added cost and effort unrelated to the needs of the business. Poor understanding of application security also creates another type of team problem: security specialists often exaggerate or over-hype the severity of risks in order to get attention. In a large number of organizations, this behavior has led to friction between development teams and security.

In many organizations, the complexity of the current approach to security is overwhelming, and they resort to building a compliance based program. While their intended goal is to improve security, these programs often result in a "race for the bottom" where the organization is motivated to perform the minimal amount to achieve compliance, and certifiers are motivated to "check the box" for the lowest price. Neither trend is encouraging nor do they reduce real business level risk.

In a world where a group like Anonymous can exploit 20 different organizations, all of which have a current PCI certification for their systems, is evidence that current approaches to software security are not working, that incremental approaches are not working, and that a radical rethink is necessary.

If you are tired of your organization making the same mistakes over and over, expecting tools to do more than is possible, only reaching a portion of your application portfolio, or expecting a process model to change your culture, perhaps Rugged is worth investigating.

Changing the Software Ecosystem

We recognize that the application security problems in our software ecosystem run very deep. They are not necessarily knowledge or technology problems - most of the security challenges we wrestle with have been well understood for decades. Instead, the problems in our ecosystem are cultural, economic, and social. There is no point in attempting to assign blame or liability for these issues. As Ice-T put it, "Don't hate the playa – hate the game." Rugged is our attempt at changing the game.

We have seen some valiant security efforts fail to make meaningful change in the software ecosystem. The Orange Book, compliance, full-disclosure, maturity models, open source, tools, security APIs, top ten lists, cheat sheets, and more have all essentially failed to affect the software development ecosystem. In the meantime, movements like Agile and DevOps have radically changed that same ecosystem, so we know change is possible.

We need an approach that is instantly engaging developers, project support staff, stakeholders, and consumers, focuses attention on the most important business issues first, enables buying decisions, scales to entire portfolios, and doesn't create blame or liability for anyone. We are absolutely *not* trying to catalog or measure the practices currently in use.

We are actively seeking new techniques for application security. But more importantly, we are attempting an entirely new approach to the problem. Literally nothing is off the table. We may cause some controversy by ignoring some widely practiced activities, but this is a opportunity for anyone with a security technology or technique to show exactly how it helps.

Evolving a Rugged Culture

We believe that the key to producing secure code is to change your software development culture. We have to get beyond looking at the technology and look at the software development organization that created it. We believe this evolution has to start with the people, process, technology, and culture of that organization.

Rugged is not a process model – it doesn't require any particular practices or activities. Instead, Rugged is about outcomes – you decide the who, how, and when. If you want to be Rugged, then the software development organization you create needs to reliably create software that is protected against the threats of today and well-prepared for the threats of tomorrow. You need to adapt and evolve to keep ahead of the ever-increasing threat.

Over time, we hope that Rugged will spread from individual projects to business units, to divisions, and eventually to the entire organization, including governance, architecture, infrastructure, operations, etc... We have not had time to fully envision how Rugged works at the enterprise level, so this document is (for now) focused on the individual software project. Below are a few quick thoughts on enterprise Rugged.

We believe this evolution is a natural outcome of attempts to simplify and strengthen security stories. For example, by externalizing security defenses (e.g. providing security services) and using them on multiple projects, we can implement applications more quickly and increase security. Projects are free to innovate, and if the experiment is a success, the results can be repeated and promoted to an enterprise solution. This culture of "federalism" allows projects to experiment and share the benefits with the entire organization.

We hope that Rugged will affect the entire application portfolio, including new and legacy, web and non-web, internal and external, mainframe and mobile, and critical and trivial applications. As you discover defenses, procedures, and other security techniques that work for you, they can be adopted across the enterprise and shared across many projects. Individual projects won't have to reinvent those defenses, as long as they use the enterprise solution. We believe that using Rugged across the enterprise will result in software cost savings across the lifecycle from reduced effort required during design, requirements, implementation, testing, remediation, and even training.

In the long term, we hope Rugged will lead us to new ways of building software that result in secure code. Building on the ideas in Agile and DevOps, we are starting to see the outlines of an organizational approach to security that is positive, real-time, continuous, largely automated, and highly efficient.

Getting Rugged!

So how does a software development organization become Rugged? If you think of the organization itself as a system designed to produce rugged software, then software assurance is a quest to improve the business and the people in it rather than specifically the software it produces.

In this section we suggest some strategies that we think might help an organization start to produce rugged code. We have tried hard *not* to focus on the vulnerabilities in the software itself. We view those as a natural consequence of the organization that wrote the code. Instead, we imagine the software development organization *itself* as the system we are trying to secure.

So rather than focusing on technical vulnerabilities like cross-site scripting and SQL injection, we target problems in the software development organization, such as not providing secure coding training, failure to monitor threat intelligence, inability to create a security architecture, and failure to track and manage security bugs. These are the real vulnerabilities, as they are what really introduce risks into your business.

Again we challenge you to help us make this better. How do you build an organization that will reliably produce secure code using only people, processes, and technology? In this section, we suggest a few foundational principles for approaching the problem.

1. Monitor Threats Like a Town of Prairie Dogs

First, we think you need an organization that keeps aware of advances in the threats facing it. That means establishing a role or team with responsibility to monitor threat sources such as FIRST, SANS, OWASP, WASC, and other organizations that provide this information. You'll want to establish a process for reviewing the output from these groups, evaluating how it affects your business and technology, and an escalation path for threats that appear to affect your business.

Cross-site request forgery (CSRF) was first discovered in 2000, yet most organizations never heard about it until 2005. Even today, there are many organizations and developers who have never *heard* of CSRF, much less understand the risk it poses to their business. For such a simple and devastating attack it's surprising that so many remain unaware. When a new risk comes out it may take months or years of dedicated effort to fully address across an enterprise. That's why threat monitoring is so important.

One important source of these new threats to an organization is through the adoption of new technologies. We believe that every new technology should go through a security review before it is adopted by an organization. The goal of the review is to identify how the new technology can be used safely and how the technology should be configured to help ensure security. For example, before you adopt jQuery, you should set some standards around using it safely without introducing cross-site scripting (XSS).

We suggest that emerging threats should be tracked and periodically re-evaluated. As understanding of these threats grows, the aggregate risk to the organization may become clearer over time.

2. Work Together Like an Ant Colony

We think everyone in your IT organization has a role in making you Rugged. But to get them working together requires a way for them to communicate effectively. The problem is that security is so complex that every topic has a huge amount of context associated with it. For example, to understand an access control vulnerability, you really need to understand a lot about common access control vulnerabilities, how the application is designed and implemented, the network topology and protections, the roles and entitlements involved, the protected functions or data, the business context, the entitlement management process, etc…

We can simplify this communication problem greatly if everyone can share a simple model of all the security-relevant information about the application. This way, each issue that arises can be placed into context in a way that everyone can quickly understand. The model can't just be about the software. It should include the people, process, and technology that created the software – the entire software development organization.

In Rugged, we capture all the information relevant to security in a model we call a "Security Story." The story explains *why* all your security is in place. In fact, your story captures what you *mean* when you use the word "secure." As long as organizations think of security as a magical, unmeasurable, unattainable quality, it is by definition impossible to achieve. But once you start thinking of security as something tangible that you build and optimize over time, you can focus all your existing engineering skills to work on achieving it.

If you're good at building things, you're probably

already creating many of the pieces of your security story. You can use your existing engineering skills to build out your story quickly and efficiently. You need to take control of your security story. Don't let your project's definition of security be driven by the signatures in a tool, external compliance requirements, or what happens to be in a particular penetration tester's or developer's head.

We think this security story will create a critical and effective line-of-sight from the highest level security concerns in the organization, through the security defenses, and all the way to the evidence that demonstrates the effectiveness of these defenses. Some existing security activities will contribute to this story, others can be dropped, and new techniques will need to be developed. In this way, everyone's security efforts are coordinated, prioritized, and aligned.

We don't expect you to create your story all at once. It's something which evolves over time and grows as you uncover new risks and demand new evidence. Over time, you will capture and promote your story, ruthlessly challenge that story, strengthen and simplify the story, and justify the story with evidence. Throughout the process, your story will be explicit and visible. The story is both a reflection and a guide. That is, the story should be updated when we change how security works, and it should be used to show us where changes are needed.

We think this is a foundational concept in Rugged, and we've expounded on the idea in the "Telling the Security Story" section below.

3. Unify Defenses Like a Herd of Muskoxen

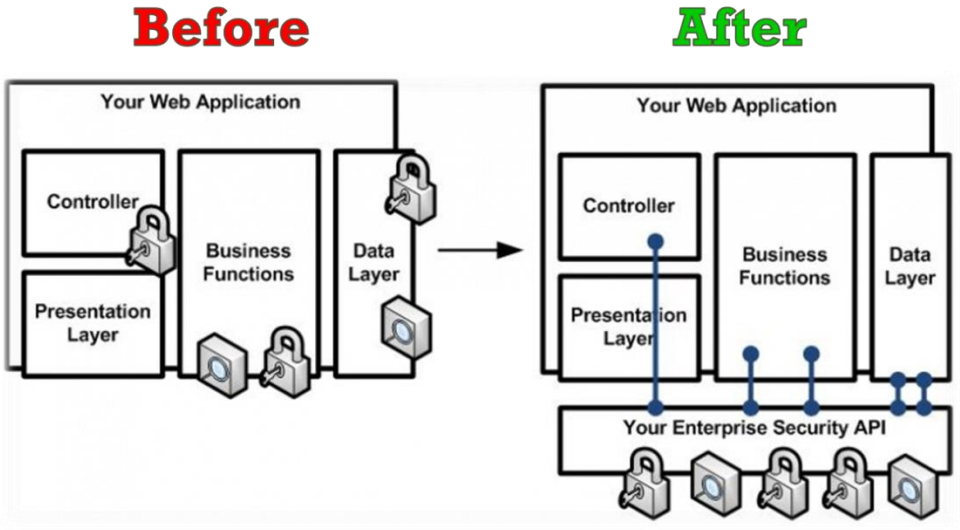
Most application security defenses have been custom developed, including authentication, session management, access control, input validation, output escaping, encryption, error handling, and logging. These defenses are tricky to write correctly. In fact, the MITRE CWE project (http://cwe.mitre.org) has already catalogued over 1000 different *classes* of software mistakes that developers might make.

We believe one of the keys to success is making it *easier* for software developers to write secure code. To do this, we suggest providing developers with a complete set of trustworthy and easy to use security controls. Standardizing security is a proven approach that has worked well for other security areas, such as encryption, where almost all developers now use NIST approved standard encryption components.

Benefits of using standard defenses include:

- Eliminate vulnerabilities related to missing and broken security defenses
- Reduce vulnerabilities related to misused security defenses
- Cut software and vulnerability remediation costs across all stages of development
- Builds on existing security defenses from both open and commercial sources
- Allows security improvements to be done centrally
- May facilitate more effective use of security assessment tools like static analysis
- Minimizes complexity and simplifies security requirements and training

These defenses might be offered as a shared library, web services, products, or other techniques. By centralizing these controls and externalizing them from the application, they can be tested, managed, and maintained at the high level of quality that they deserve.



Using standard defenses allows automated verification to work much more quickly and accurately. The reason is that it is relatively easy to scan code or test to prove that an application has properly used a standard defense. On the other hand, it is extremely difficult to verify that an application does not have any patterns that might represent a security weakness. There are simply far too many ways to write insecure code to perform an exhaustive search for negative patterns.

4. Control Your Environment Like a Family of Beavers

Modern software is not created by *your* developers alone. Today, almost every application is assembled from dozens or hundreds of components, including libraries, frameworks, runtimes, modules, components, utilities, and the like. In fact, 90% of the code in many modern applications is from external components. There are huge benefits to using these components, but at the same time we need to defend against the security risks associated with using code we didn't build.

These components often run with "full trust" meaning that they can do anything your application can do, including access your databases and filesystems, run system commands, or shut down applications. The entire software supply chain ends up running in your data center with full trust. So we recommend establishing guardrails to protect you against inadvertent insecure use of non-trusted third-party code. This is similar to establishing the secure by default approach for your own internal code.

There are several ways to help ensure secure component use. The most important is to establish a vetting process and centralized repository where approved components will be accessed. Beware transitive dependencies, as many libraries use other libraries. You'll need some intelligence about

updates and discovered vulnerabilities in all your components so that you can keep them up to date. Projects may need to change their code periodically to use the latest libraries.

In addition, you can try to limit the use of these libraries to approved functions. For example, you might not want your graphics library to be making native calls to the operating system. In this case, you might be able to create wrappers for third party libraries that only allow access to the approved features. Or perhaps you can enable a sandbox like Java's SecurityManager to prevent dangerous activity from a component. If wrapping is not possible, you can create a simple guideline for each component that details the secure use of that library. Both of these approaches can be verified with simple tools.

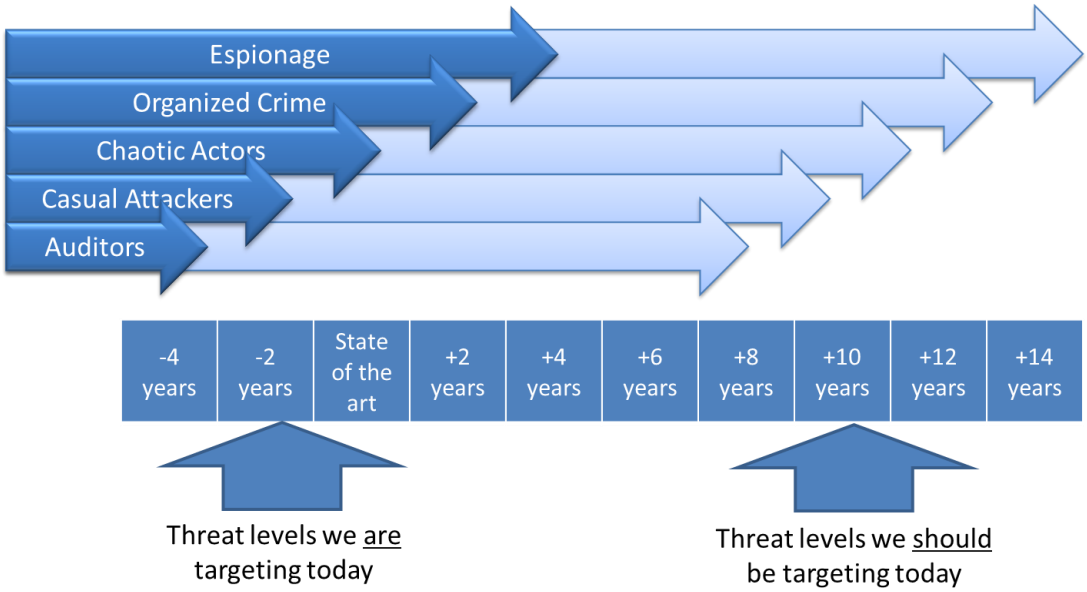
Ultimately, you're going to want to understand, protect, and monitor your entire software supply chain. You are responsible for the security of the applications you field, including all of the software components inside.

5. Defend Yourself Like a Honey Badger

In ten years, we know that the threat has advanced far beyond where it is today. Yet most organizations are building to threats that have already been understood for several years. When most applications are in production for a ten year lifespan, why are we building them using a threat profile that is two to four years out of date? We need to build defenses that are likely to withstand attack for at least ten years.

Take XML External Entity (XXE) attacks for example. This is a relatively new attack on XML parsers that has been presented at conferences for a few years. Many web applications that process XML requests (such as web services) are susceptible to this attack and a successful attack can result in stealing all the files off an application server, including all the code. Yet only a tiny fraction of organizations are working on this problem.

We think organizations can build applications that will be largely resistant to the threats of the future. For example, take the broad class of injection attacks, currently responsible for the vast majority of serious breaches. A simple strategy using strong, white list, input validation, in-application attack detection, and safe interpreter use can eliminate these flaws forever.



Some might think it is unreasonable to expect people to build applications that are resistant to threats that don't even exist today. However, the threat doesn't advance as fast as the press would have you believe. While there are new attacks all the time, the fundamental techniques used do not advance that fast. For example, the STRIDE model used by Microsoft has not changed in a decade. Similarly, the OWASP Top Ten has remained largely unchanged in the last decade.

Therefore, we expect the next decade to include incremental advances in attack techniques as well as a small number of new techniques. This is not so impossible to plan for and defend against. A great example of a success story here is the cryptographic community, which regularly uses Moore's law and other projections to determine the strength of defenses required today to protect for the expected lifetime of the system.

6. Strengthen Yourself Like a Bighorn Ram

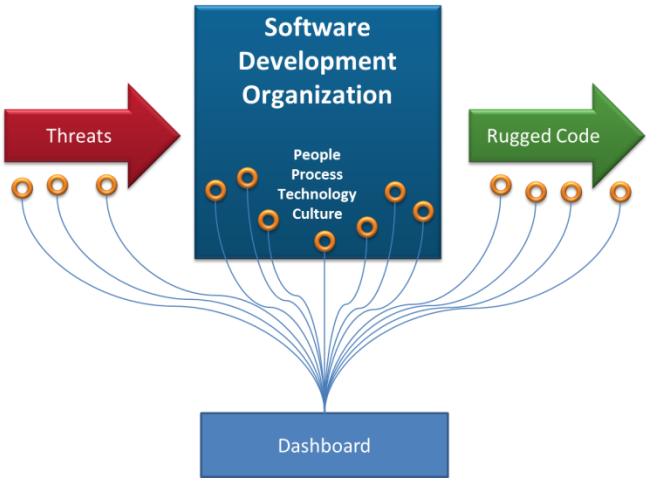
A software development organization is a complex machine with many moving parts and complex interactions. In order to be sure that this machine is operating properly and producing secure applications, we suggest instrumenting your organization with "sensors" to monitor the inner workings of the machine. You'll want to verify and monitor both software development organization and the code it produces.

We envision an organization where security is constantly tested, analyzed, and monitored for potential security vulnerabilities: during coding, during testing, and during operation. What we are imagining is considerably different from the industry-typical pre-production penetration test that dominates the software industry. We expect defenses to be continuously verified and monitored all the way through development and into production. We imagine security dashboards for the software development organization; so that executives can immediately see that defenses are working correctly.

Organizations should calculate *security coverage* by looking at the percentage of applications analyzed, the range of security areas assessed, and the depth of assurance of those assessments.

Imagine that a project tailors the organization's standard security test suite for their project. These tests ensure that all output is properly escaped by the user interface. So if a developer forgets to use the standard output escaping controls, they get feedback in seconds or minutes, not months or years. We are envisioning *positive* tests verifying the proper use of controls, rather than attempts at exhaustive negative testing. In many cases, positive tests are simple to create, run, and interpret.

We also expect the "root causes" in the organization, such as failures in training, development tools, and process execution, to show up. You might think of these root causes as a type of "organizational exposure" as they open your business to risk. Identifying these flaws and eliminating them will make your organization stronger and, as a side-effect, improve your software.



To monitor the code, we recommend using your security story as a guide and identify the most cost-effective ways to verify each defense is in place, correct, and properly used. This could include reports generated by the build tools, results from automated test cases, and other direct metrics. By integrating security testing of all kinds into the build and deployment process, your project will generate evidence automatically. Seek out ways to make your code easy to verify, such as having tools that will report how the security defenses are provisioned and used.

To instrument and monitor the development organization itself, you'll want to verify that all the activities necessary to generate security are actually being executed. We think organizations should perform experiments to see how the software development organization responds when a problem is introduced. For example, try intentionally weakening the code of an encryption algorithm, or tampering with the binaries produced by the build chain, or inserting malicious code into a component, or deleting a critical part of the security story. You may want to create or use automated tools that introduce this type of problem randomly into your organization to help verify that your defenses are in place and working properly. How well does your organization handle and recover from these "incidents?"

This monitoring extends into operations as well. There are many opportunities to install "sensors" into operational systems, and systems to read and analyze the data produced. By understanding the behavior of your operational applications, you will have much deeper insight into what types of attacks are actually happening and what defenses might need to be shored up.

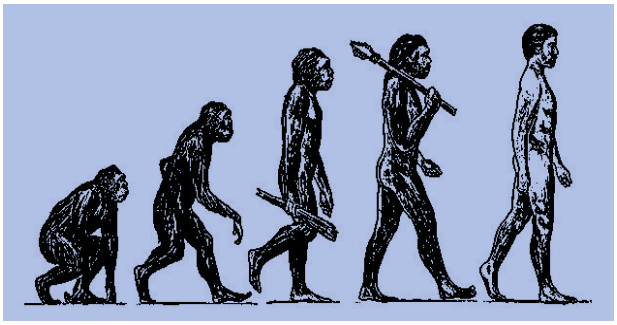
The security story is the thread that ties together all the output from these "sensors." The reports all can be tied to the verification of one or more of the defenses. Eventually, we hope that much of this can be automated, so that the story is constantly up-to-date.

7. Adapt and Evolve

Rugged is about more than just producing secure code. It's about establishing a culture of evolving quickly to keep up with advancing threats. We think that much of the progress in security comes from

the interaction between “breakers” and “builders.” These two roles push each other to improve all the time, and the result is an application and security story that have been “battle-hardened.”

We view vulnerabilities as a symptom of a deeper organizational problem. When you find a vulnerability in software, it’s really just a symptom of some problem in the software development organization that produced it. Rugged organizations take the opportunity to improve, and prevent that mistake from reoccurring or being reintroduced.



As soon as a software vulnerability is discovered, we think you should evaluate the scope and seriousness of the problem. By linking each root problem into the security story, everyone will quickly and clearly understand the scope and seriousness of the issue. This should be performed, not only for the specific problem, but for the aggregate problem across all of your application portfolio. For example, if you discover an XXE flaw, you should attempt to estimate the likely number of those flaws in the entire codebase, and calculate the risk of the aggregate vulnerability.

The next step is to evaluate the organizational causes of the problem. Look at the people, processes, development practices, and tools that allowed the vulnerability to be introduced and determine how it can be avoided across the entire organization in the future. The long term solution may be a technical change, such as a standard architectural or implementation pattern. But more likely, the problem is rooted in a broken development process, people without proper skills, training, or motivation, or a tool that failed to provide needed support.

For example, when you discover a SQL injection flaw, you might get a recommendation to move to a parameterized query. But perhaps the root cause is that an object relational mapping layer that doesn’t require SQL to be written should have been used. Or perhaps the developer wasn’t trained properly in the organization’s standard coding patterns for databases. Or perhaps the QA testers should add some tests to detect and prevent these flaws. Being Rugged means that you constantly patch and refactor our software development organization to eliminate the organizational bugs that are causing insecure code.

Telling the Security Story

Imagine an *artifact* that anyone (developer, manager, business owner, user, etc…) can read and feel assured that their security concerns are addressed. For example, here is the front page of the security story showing the six major security lifelines for a fictitious Rugged Bank. The full story fleshes out each of these high level lifelines into a full description of defenses and verification.

The Rugged Bank Security Story

Commitment: *Brick-and-mortar banks protect money with vaults, police, and alarms. As we move these operations to cyberspace, everyone should expect even better protection of their money and financial information. RuggedBank is committed to the highest standards of application and network security for our RBOnline application. We secure our infrastructure, ensure data is always protected, build defenses against injection and other application attacks, practice rugged software development, and carefully verify our code. At the core of our security is a commitment to transparency – across our protections, processes, and even potential problems.*

Infrastructure Control

RBOnline security depends on maintaining control of our physical and network infrastructure. We are hosted in a secure data center, managed by trusted staff, and our systems are kept hardened, patched, and up-to-date. Our network is defended and segmented by firewalls and we detect and block both network and application attacks.

Learn more…

Application Control

All our data protections depends on having software that is resistant to injection or other attacks that rob us of control of our systems. We have strict rules around data handling and interpreter use to prevent injection. In addition we ensure the application is always available and ready for use.

Learn more …

Data Protection

Protecting your data is our highest priority. Our primary defense is universal authentication and access control. As a secondary defense, we also use strong encryption everywhere data is transmitted or stored. To ensure that these protections are not bypassed, we have established extensive protections against injection and other attacks.

Learn more …

Rugged Development

To make sure that our application is designed and implemented securely, We follow a secure development lifecycle. All our developers are trained in using our standard security defenses. We have performed extensive threat modeling and minimized our attack surface. We use powerful tools to protect and manage our source code, user stories, and software artifacts.

Learn more …

Security Verification

An independent team performs architecture analysis, code review, and penetration testing on our application. We use a custom test suite and automated security tools to double-check for vulnerabilities. We are committed to finding and eliminating vulnerabilities throughout our secure development lifecycle.

Learn more …

Transparent Security

Without information, good security decisions are impossible, so we are committed to ensure that you understand the protections that we have provided. We will notify you of any security issues that might affect your business. You can export your data from our application at any time, and we will purge it from our systems.

Learn more …

If you have questions, comments, or want to report a security vulnerability, please send email to security@ruggedbank.com.

The *document* is not the point – Rugged is about creating a shared understanding. The journey to create it, challenge it, and improve it is what produces security. In this way, the story is the catalyst that allows builders and breakers to work together both competitively and collaboratively. In the cryptography

community, this basic approach has driven researchers on both sides to advance the state-of-the-art rapidly.

A security story is a *collaborative* effort that highlights how the implementation, application design, service infrastructure, organization processes, and the business environment itself protect what's important to the business. This visibility allows all stakeholders to ensure any activities undertaken have a clear line-of-sight back to something of value.

Parts of the security story already live in many forms within organizations. These fragments go by names like "protection requirements," "security architecture," "security requirements," "abuse cases," "scan coverage," "findings," "risks," and many more. They are almost always disconnected, recreated at many stages across the lifecycle, and often conflicting and with multiple gaps. Together, this chaotic incomplete mess is literally the only definition of "secure" for an application, which leads directly to the situation we are in today.

Why call this a "story?" This name acknowledges a few truths about security. One of the primary uses of the story is to convey a complex set of information to many stakeholders, both internal and external. We accept that the story is not a formal model of security, but a practical way of communicating a large number of details in a structured way. We believe that, at least today, using informal stories is the best way for organizations to learn how to achieve security, and that their stories will become more detailed and sophisticated over time.

What a Security Story Is Not

Many security activities are unrealistically advertised as all you need to achieve security. Being Rugged means being honest about an application's capabilities and so in that spirit, we believe in being honest about a security story's capabilities. A security story is not:

- A generic checklist of standards, security defenses, requirements, or activities
- A collection of penetration test results
- A certification for developers, architects, and testers
- A one-size-fits-all framework, architecture, or solution for security
- A product or a process model
- Even the collective set from this list

A fixed security model or process is limiting as each individual application will have its own unique assets and risks. For example, an e-commerce website might be most concerned with availability of the application whereas a financial institution might be most concerned about an accurate audit trail. How an organization addresses these concerns depends greatly on the purpose and design of the application and so a regimented guideline of static principles has limited value. Instead, a security story is something that stakeholders create based on what's important to their application.

What Comprises a Security Story?

There is no fixed format or content for a security story as we expect a security story to evolve over the life of an application. A story may contain text, images, diagrams, spreadsheets, links, and other formats. Since a primary goal of the story is communication, however, the format is important and the story is not just a dumping ground for documents.

Initial security stories may focus solely on capturing the concerns of application stakeholders, including both application providers and application users. Simply by identifying those concerns, organizations bring visibility and awareness at all levels of application responsibility. Architects can design against those concerns, developers can develop to those concerns, and testers can test against those concerns.

The natural evolution of a security story is to capture what is being done to address the stated concerns. Not all concerns need be addressed through technical defenses - the flexibility in a security story is that it allows all types of strategies. The only limitation on strategies captured in a security story is that you must be able to explain and defend them. Even this "limitation" is open as descriptions can take many forms including narratives and diagrams.

Writing a Security Story

Just like real story writing, putting together a security story can be done in many ways. We outline one possible technique but it is by no means the "only" way or even the "right" way to put your security story together. A security story is always a work in progress and is subject to iterative refinement.

Identify the Lifelines

The most important step is to identify what matters in your application. Every application has unique features, functionality, assets and users. Perhaps it is protecting client data, or preventing fraudulent transactions. Maybe it's establishing an audit trail or ensuring limited downtime. Whatever these lifelines are, start by writing one sentence that concisely describes the specific concern an application stakeholder may have. The temptation may be to fixate on a specific type of vulnerability or attack (e.g. focus on cross-site scripting). However, such narrow focus on a negative theme typically makes for a poor story. Note that generally speaking, stories about positive themes (e.g. protecting sensitive data) are superior to stories about negative ones (e.g. focusing on attacks or vulnerabilities).

Create a Strategy for Defending Each Lifeline

The initial temptation for any organization that has undertaken some security activities will be to simply regurgitate the results of those activities. But a description of security activities or other countermeasures alone does not make an effective security story. Organizations must connect all the defenses and assurance activities to the strategies and create a narrative that ties these elements together. Typically, the strategy provides some reasoning or logic for why certain defenses address the identified concerns. Remember that the audience of the story will be varied and the story should be written to speak to multiple audiences. As a result, a story may be broken down to varying degrees of specificity. Some readers will prefer an abridged story whereas others will want the full verbose version. Regardless of the length and format, a well written story should contain clear, concise, meaningful, compelling, measurable, actionable, and plain language.

Build the Defenses

For each area of concern, describe each of the defenses that address the concern. These defenses could be design principles, technical defenses, manual processes, legal mechanisms, or any other describable mechanism that addresses the concern. But your story should be specific about exactly what you are actually doing. Whether your software project is driven by requirements, stories, or use cases, this is where they come from. So rather than saying you use strong encryption, be specific and say that you are using the AES/CBC/PKCS5Padding algorithm implemented in the BouncyCastle v3.1 encryption library along with some custom code wrappers to ensure secure use.

Make sure you provide all necessary details for robust and effective protection. For example, transport layer security provided via SSL is a commonly used defense to provide an encrypted channel for sensitive data transmission. There are a number of elements related to SSL that should be addressed to ensure effective security, such as proper certificate issuers, strong cipher support, protection of keys, etc.

Prove It!

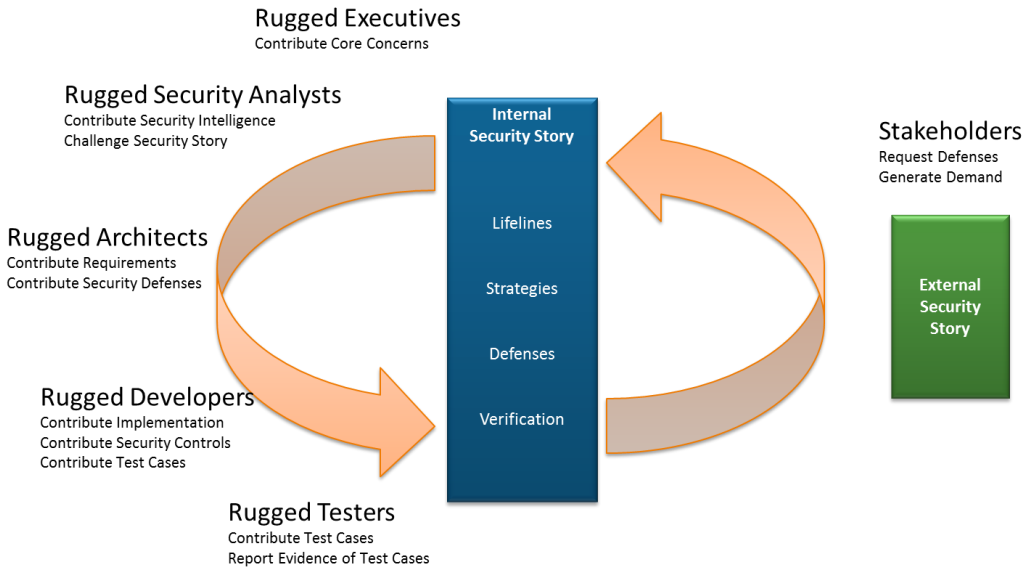
Simply making claims about your security measures is not enough. Your story has to include the reasons to believe that these defense actually work. You should share the details of how you validate each of the defenses in the story. There are many ways to verify defenses, including code review, test cases, scans, automated tools, human analysis, and even attestation. Your goal is to choose a verification technique that will do a good job of showing that the defense is present and works in the most cost-effective manner. This may evolve as you develop your ability to verify in more efficient ways. Your story must include the proof that these defenses are actually effective.

Revise and Repeat

As an organization continues to evolve and mature, an application's security story may change and expand. New concerns, activities, or defenses may evolve and should be added to the security story. Additionally, circumstances may require additional detail about an element of the story. We believe organizations should remember that their security story is not static and must be willing to update the story as conditions change.

Rugged Roles

Every organization has their own set of organizational roles, each with a unique set of responsibilities. So Rugged doesn't attempt to define exactly who should do what. However, it's helpful to understand the activities that some typical roles might perform. You should feel free to assign these responsibilities to people in the organization as you see fit, possibly redefining roles or creating new ones.



Rugged Roles – See following sections for full descriptions

This diagram shows how each role contributes certain information to the story, and receives back information that allows them to perform their responsibilities. This process occurs throughout the software development lifecycle. At some point, you may decide to expose your security story, or portions, to external stakeholders (e.g. customers, partners, constituents) - anyone that might have an interest in the security of your application. You may gain valuable feedback or marketing advantages by sharing your story this way.

In this section, we consider common roles and how each interacts with the security story. For purposes of this section, we are presuming a typical large software project with a few dozen developers, testers, and other staff. In large organizations there are also likely to be enterprise-focused roles that spend their time working out the enterprise-level application security story.

Whatever your role, we strongly encourage you to try a Rugged Experiment! There's nothing wrong with creating a personal security story for your project and using it to help explain the need for defenses, a risk decision, justify testing, or organize your own thinking around security. You will likely be able to win others over and build support for the approach. Communicate inside your organization and to outside groups about the Rugged approach. Share your stories about how it worked in your organization and what you did to make it better.

The Rugged Executive

As a Rugged Executive, I want to bring products and services to market quickly and efficiently without being blind to the risks in my software applications. I need clear insight into the threats facing our organization, how our software measures against them, and I need it aligned with our business priorities and budgets.

To achieve my goals I have come to value...

- *Visibility into application security over Blind trust*
- *Line-of-sight over Disconnected activities*
- *Choosing appropriate assurance levels over Random rigor*

Your Mission

The job of the Rugged Executive is critically important to achieving application security across your application portfolio, but it's not difficult. All you have to do is express the organization's priorities and concerns and require full visibility and line of sight to the assurance of your security story. Organizations have many different executive structures, and many different roles may fulfill Rugged Executive responsibilities. Bill Gates' Trustworthy Computing Memo is a shining example of the importance of this role in helping organizations become Rugged.

You need to provide the business information that will enable your team to make great decisions. Development projects need to understand the strategic importance of their project to the business, the criticality of the information they are processing, and how much risk the business can afford to take. You can start to capture these priorities by brainstorming about outcomes that would be very damaging to your business. You might consider the disclosure of sensitive information, the inability to perform critical activities, the corruption of important data, or a loss of accountability on transactions. Based on these outcomes it is important to require development teams to be Rugged by delivering applications that provide visibility into how these concerns were met, and a line of sight with measurable and accountable assurance to substantiate your organization's security story.

These top level concerns will guide development to make great design and implementation choices throughout their development process. Your clarity identifying the key security concerns for an application is critical. Only you can set these clear parameters around what's important to protect and how far you are willing to go to protect it.

You are also responsible for providing business level oversight of Rugged projects. You'll have to evaluate the story and decide whether it's compelling. You can use the story as a framework to explain your security to your stakeholders and get their feedback. Over time, this will improve the completeness of your story. You don't need to understand all the technical details of the story, but only you can tell whether the security concerns for your business and your users have been addressed.

By sharing your priorities and guidance, you enable other roles to succeed with their responsibilities. Through completed stories, these roles will provide you with the data you need to weigh the costs and benefits of security investments. You'll also gain a competitive advantage, as you'll be able to clearly articulate the security features and assurance associated with each of your applications.

Over time, you'll start to recognize that improved confidence in security gives you the freedom to innovate. As your confidence in security grows, you will be able to deploy more sophisticated functionality with more sensitive data.

Ideas for Being an Effective Rugged Executive

Issue an Executive Memo. Let your organization know that security is a priority for your business. Emphasize that security is important and that over time you expect projects, software, infrastructure, operations etc. to be Rugged. Let them know what needs to be protected and why it's important to the business.

Understand Security Demand. Engage with your customers, business partners, and other stakeholders to understand their security concerns and needs. That information will help you make strategic decisions about where to invest. You are responsible for accurately representing these concerns to the project team in the story.

Capture Business Security Concerns. Communicate what the important security priorities are for each application. Keep the overall number of concerns as low as possible. Also assess what level of assurance you need and help projects understand what level of evidence is required.

Create a Strategic Security Budget. With Rugged, most security costs will be part of your ordinary software budget. However, enterprise level improvements can be made that will reduce and eliminate the biggest risks across your entire application portfolio.

Demand Visibility. Ultimately, you are responsible for ensuring your organization's technology strategy. You need insight in order to make smart decisions and investments.

Rugged Management in the Future

Joe is an executive for his organization, which has built and operates several hundred applications. Joe just issued his annual security memo. To set priorities, Joe reviews the organization's enterprise security story, which summarizes all the application stories and includes enterprise security architecture. His review showed their standard approaches to authentication, database queries, and encryption had resulted in dramatic reduction in vulnerabilities in these areas as well as significant reductions in time to market. However, it was clear that the complexity of their multiple access control systems had become a costly problem.

So in this year's memo, Joe emphasized the criticality of their effort to unify access control across the organization. Their enterprise security architecture has already been updated with standard defenses, but many applications have not been updated to use them. Joe decides that by the end of the year, all use of non-standard access control systems will be tracked as a needed improvement. By the end of two years, all systems will be required to be updated.

Metrics

The Rugged Executive measures success based on the level of understanding and control the organization has over application security. You'll need to gather some basic statistics:

- Trend number of apps (monitored, unmonitored stacked)
- Trend number of servers monitored
- Trend total lines of code (library, custom stacked)
- Employees trained with role-specific security training
- Security tools available to developer population

You can also gather one view of your security by gathering "negative" metrics, based on your risk and vulnerability data, such as:

- High and critical risks by division

- Most prevalent risks (need for better security defenses)
- Average number of vulnerabilities over time (trend)
- Top 5 most insecure applications
- Top 5 biggest library problems
- Trend overall vulnerability counts by category (Low, medium, high stacked)
- Vulnerabilities by division and security category
- Vulnerability flow (new vulnerabilities - remediated vulnerabilities)?

You should also consider gathering “positive” metrics. These are often easier and more meaningful than their negative brethren.

- Compliance with using standard defenses
- Success rate using standard defenses (e.g. 204 of 210 SQL queries done correctly)
- Compliance with security activities in the lifecycle
- Coverage of application portfolio
- Depth of testing of application portfolio

Some higher level questions that you might try to answer include: How easy is it to describe the application security program? Is the investment in the program paying off? Are you using security as a differentiator in the market? How are clients reacting to improved visibility?

The Rugged Security Analyst

As a Rugged Analyst, I want to equip and support my enterprise in achieving their security stories. I want to assist teams in build cutting edge software quickly that is secure rather than tell them what they can and can't do. I'm focused on providing threat intelligence and sustainable security patterns that encourage and enable rather than limit.

To achieve my goals I have come to value...

- *Sustainable, continuous security patterns over Reactive bug hunting*
- *Encouraging innovation and improving time to market over being a road block*
- *Security research and intelligence over Fear, uncertainty, and doubt*
- *Proving security with positive verification over Exploiting vulnerabilities*
- *Seeking threats over Being surprised*

Your Mission

In order to do your job effectively, you are going to need information about your enterprise and what is important to your business. You should work with your organization's technology executives in order to find out your organization's sensitive assets, the critical business functions, and your organization's risk tolerance. You make sure they are captured in the security story. There are many different types of security analyst, and many different roles may fulfill Rugged Analyst responsibilities.

In addition to aligning yourself with the technology executives, it would also be prudent for you to interact with the business analyst eliciting functional software requirements from the business sponsor. It's likely that this person will be responsible for developing use case diagrams to depict core software functions. This will not only provide you with the opportunity to learn more about how users and system interfaces interact with the software, but it will also support your development of security requirements and abuse cases.

You are also going to need information about the technologies and architectures that your organization is currently using, how they are being used, standards in place, and known risks. In particular, focus on the standards and frameworks in use, and learn how the security defenses work in all different types of applications.

Your mission is to identify gaps and weaknesses in your project's security story. Your special skill is that you can think like an attacker and identify novel weaknesses in your application. Your focus is not to find multiple instances of well-understood vulnerabilities – that's for the Rugged Tester. Your job is to use your skills to uncover new threats and unknown vulnerabilities, and perhaps work out new ways to check for these issues (preferably automated).

Sometimes these gaps will arise as a new threat is identified and understood. For example, when an intranet application is moved onto the public Internet, a new set of threats emerges and the security story is likely to have numerous gaps that need to be addressed. New threats may also emerge through your own security, evaluating other people's research, and monitoring threat intelligence channels.

By identifying a new threat, analyzing it, and adding it to the story as an unfinished chapter, you notify other roles that there is a new requirement to solve. Other roles, like architects, developers, and testers will figure out the best design and implementation for the new defenses.

Other times, these gaps and weaknesses are uncovered through analysis of the story itself. For example, as you evaluate the data protection aspects of your story, you may notice that SQL injection was never

considered. Your job is to raise this concern, prioritize it, explain it to the stakeholders that need to know, and propose defenses.

You are also responsible for performing the triage and analysis on vulnerabilities to determine the root cause. This analysis looks at the software development organization critically, to see what could be changed to avoid, prevent, or deter whatever caused the vulnerability or allowed it to exist from ever reoccurring.

Your understanding of what is important to your business and the level of assurance required is *critical*. Your ability to make informed security decision is what will make you effective in your job. Decisions made without this context are likely to cause friction with the rest of your team. Be sure to actively engage the business analysts of the team and explore meaningful approaches to commingling business concerns with security objectives.

In return for your efforts, you'll be rewarded with close relationships with the rest of your development team, greater ability to influence the architecture and implementation, and more focus on novel risks.

Ideas for Being an Effective Rugged Security Analyst

Seek Threat Intelligence. Form relationships with security analysts from other organizations, particularly those in the same and similar sectors. Review trade information about the types of attacks that are occurring and get ahead of them before they strike your organization. Collaborate on ways to help defend against attacks as a sector instead of as an individual company. Read and understand incident response data sources like the Verizon "Data Breach Investigations Report" and apply the intelligence to your organization.

Join your local OWASP Chapter, attend security conferences, and read all the technical papers, blog entries and other research you can get your hands on. Understanding threat intelligence is the best way for you to predict future threats to the business.

Understand Attackers and Their Methods. One of the effective ways to represent business and adversarial concerns is through analyzing attackers against your business and the ways they can actually attack your systems. Taking it a step further, connect these attacks with the real impact to the business. It provides a canvas that brings together business functionality, technology use and security requirements. It facilitates priority in effort and can be leveraged to discover secure defenses.

Identify Abuse Cases. Using your understanding of how security vulnerabilities happen in both security controls and business logic, create security tests that will ensure that the controls are properly designed and implemented. Be sure your abuse cases cover not only the flaws in what is present, but also the missing defenses, the "dogs that didn't bark."

Work toward Enterprise Solutions. Your enterprise should not have to fix the same problems time and time again. Work with architects and developers to integrate good solutions into your frameworks, libraries, and services, so that the root cause is eliminated and your enterprise does not need to continuously reinvent the wheel.

Research Constantly. Application security is a relatively new field, and we need to devote much more of our attention to constructive research. As applications become more critical, attacks will cause more than just financial damage. Attacks on healthcare systems will cause people to die, as will attacks on our power generation and distribution applications. Not only does this provide constant stimulation, but offers you the chance to publish papers and make an impression on the world.

Continuously Improve and Optimize. The security analyst alone makes up a very small population of staff within a company. He or she cannot "do security" by themselves. The security analyst should always look for a way to operationalize, automate, or optimize security activities. Teach, mentor, equip, and support the other project team member to accomplish the security story rather than doing it all yourself.

Rugged Analysis in the Future

Alice is a Lead Security Analyst at a video game industry company that has adopted Rugged. Today, Alice receives an alert from one of their threat intelligence services that there are increasing attacks on XML based services in the video game industry. Alice investigates and discovers how these attacks work and recognizes that they don't have a standard defense in place to counter these attacks.

Alice writes a custom test case and runs a few tests to help determine how widespread the problem is. Unfortunately, their organization doesn't have standard approach for parsing XML and many of their applications look like they are vulnerable. Alice evaluates the risk of this weakness, not for a single application, but aggregated across the enterprise and it is clearly a high priority.

Alice updates her project's security story with the risk and notifies the enterprise architecture group along with recommending a few defenses to shore up the organization's XML parsing strategy. In this case, the solution will likely have to involve some tweaks to the frameworks they are using and some guidance for developers building XML based services. Eventually, the development teams will take over Alice's prototype test cases so that all projects will test themselves for this flaw. Everyone updates the story as the solution progresses.

Metrics

The Rugged Security Analyst should evaluate success based on the simplicity and strength of the security stories. Some of the metrics that we are evaluating include:

- Number of new threats researched or considered for the security story
- Number of threats that represented gaps or weaknesses in the existing defenses.
- Overall size of security stories (not that bigger is necessarily better)
- Complexity of security story, based on structure, duplication, centralization, etc…
- Completeness of the story against industry standard threat sources
- Number of recommended solutions adopted by architecture
- Freshness of security knowledge relative to state-of-the-art

The Rugged Architect

As a Rugged Software Architect, I want my software architecture to make security strong, simple, and clear. My goal is to provide a comprehensive set of strong security defenses and secure usage patterns that makes security simple for developers to build and use, and easy for testers and tools to verify.

To achieve my goals I have come to value...

- *Visible security over Blind trust*
- *Strong and simple standard defenses over Custom coded security*
- *Accountable security over Infrastructure reliance*
- *Adversary awareness over Incident response*
- *Defense in depth over Single defenses*

Your Mission

The goal of the Rugged Architect is to create a software security architecture that designs and positions security defenses in a way that will meet the expected threats. This architecture is captured as part of the security story. Organizations manage software architecture in many ways, and many different roles may fulfill Rugged Architect responsibilities.

Combined with your deep understanding of the organization's technology infrastructure, your mission is to design a comprehensive set of secure defenses as actionable development guidance for your rugged developers and testers to use. You are responsible to ensure that the organization provides common, secure services to other applications. The defenses represent secure patterns and anti-patterns that align with your organization's architectural strategy and principles. You should offer your expert training and guidance to developers about how security architecture works and why it is important to the business.

You also should help create security support roles within the development community. These support roles facilitate the adoption, understanding, proper implementation and verification of security defenses within their respective applications.

Ideas for Being an Effective Rugged Software Architect

Profile Applications. Examine your application portfolio and evaluate the importance of the data and functions provided. Use your organization's security story and the intelligence provided by the security analyst to profile the organization's current applications according to risk.

Create an Enterprise Security Architecture. To the maximum extent possible, make security as simple as possible for developers and testers. Embedding security defenses in the programing language, frameworks, development and configuration environments utilized by developers and testers will have higher probability of adoption. Additionally, this means creating the simplest defenses that correctly address gaps. Defense in depth helps to ensure that vulnerabilities in one protection doesn't expose the business. We believe that most secure settings or patterns should be the default, as this will make easy things secure and complex things possible.

Establish "Guardrails" for Safe Use of Technologies. Work with analysts to evaluate the security of new technologies and determine how they can be used safely. Offer specific guidance about the configuration and coding patterns for the safe use of these technologies.

Design for Security Verification. Your architectures need to include a way for them to be easily and completely verified. For example, imagine a complex access control scheme implemented entirely in

assembly language. It might appear to be fast, correct, and effective. However, the work to verify the implementation will be extremely difficult and completely ineffective from a Return on Investment (ROI) perspective. In fact, we believe that something that is very difficult or impossible to verify is a security bug and should be tracked and fixed.

Prioritize Improvements. As an application architect you have the ability to see technology usage and design patterns across many applications. Utilize the vulnerability intelligence from your security analysts to develop secure defense patterns and anti-patterns for the most prevalent and damaging threats to the business.

Train Security. Provide specialized training to support the adoption, proper design, implementation, use, and testing of security defenses and architectural patterns.

Rugged Architecture in the Future

Scott is an experienced software architect in an organization with a diverse technology profile from mainframe to mobile. Scott's current assignment is to integrate the organization's standard approach to input validation into a recently adopted web application framework. Scott wants the solution to be as automatic as possible for developers, requiring the least amount of effort from developers, and impossible for developers to avoid.

Scott specifies an architectural approach that will wrap their existing input validation services into the API used by their new framework. This specification will be implemented by his development group, along with test cases to verify that the validation is correct and properly used. If this experiment goes well, then this solution will be mandatory for any project using the new framework.

Metrics

The Rugged Architect should evaluate performance based on the successful integration of standard security defenses and custom security defenses into their architectures. Are the solutions easy to understand and explain? Are the defenses used in a way that makes them easy to configure and manage?

The Rugged Project Manager

As a Rugged project manager I want the software my team delivers to be secure. I use the security story to make sure my project is on task and that the result will be demonstrably secure.

To achieve my goals I have come to value...

- *Security assistance over Exaggeration and penalties*
- *Clear security requirements over Surprise vulnerabilities*
- *Security rationale over Dogma and mandates*

Your Mission

You should understand enough of the security architecture and requirements that you can manage the scope of what you are being asked to implement and the skills required. Understanding the rationale behind the requirements will help you explain why they are necessary to your project team. Project management is handled differently in different organizations, and many different roles may fulfill Rugged Project Manager responsibilities.

You need an accurate understanding of the costs associated (both in time and dollars) required to close the gaps in your existing project, or ensure no gaps are deployed in your new project. Building these costs and scheduling and measuring the tasks into the project ensures that security is treated as a capability, and not a last minute requirement foisted on development teams.

Your key responsibility is to oversee the creation and maintenance of the security story for your project. You are the key link that aggregates and evaluates the security story. Just like the software being developed, the Rugged Project Managers should coordinate the work between the business, analysts, architects, developers, and testers. You will build time into your project to produce a high-quality security story at whatever resolution is needed.

Ideas for Being an Effective Rugged Project Manager

Account for Security Costs. One of the biggest reasons security falters in development organizations is that it is not treated as a capability. Time and money is allotted to deliver capabilities for applications, and security is just tacked on at the end. Ensure your organization does not fall into this trap by ensuring that security is treated as just another set of requirements.

Build Rugged into Your Project Plan. You have a security story to create. If your teams are not progressing toward delivering this target by the time they get to acceptance testing, adjust their priorities to make sure it is completed.

Ensure the Story Is On Track. The project manager should ensure that the story is being built according to the schedule and plan. Risks to the successful completion of the story should be identified early so that there is time to make the necessary changes in the application and the organization.

Expect Gaps and Weaknesses in the Story. The project manager should expect security issues to arise during a project. If not, then there is most likely a detection problem. Once identified, those bugs should be handled and closed in the same way as other application problems.

Rugged Project Management in the Future

Mary is a new Project Manager without much of experience in security. Today she has a review meeting with the business where they are certain to ask whether the application has implemented the proper

security. Unfortunately, today she has some bad news. The security analysts have identified a new security vulnerability in the framework Mary is planning to use, and the Architect has determined that it will take significant effort to accommodate the fix.

Since this change is going to delay the project's release date, Mary wants to confirm that the fix is necessary for their application. Upon further review, the development team determines that they are not going to be using the vulnerable portion of the framework. Mary agrees to address this in her project's security story, so that it will not be forgotten, and agrees to implement the standard pattern before the next major release.

Metrics

The Rugged Project Manager should evaluate success based on the level of Rugged integration into the project process. How many activities must occur outside the normal project process in support of the security story? How accurate is the final security story compared to the final architecture and application? Are explicit security requirements part of the project? Are developers aware of the motivation behind these requirements?

The Rugged Developer

As a Rugged Developer, I want my software to be secure against attacks, interference, corruption, random events, and more.

To achieve my goals I have come to value...

- *Software Quality over Security Products*
- *Defensive Code over Patching*
- *Ruggedizing Your Own Systems over Waiting To Be Hacked*
- *Something about avoiding extensive pen test results, or wait until security testing with fingers crossed, or…*

Your Mission

Good news – you're already Rugged! ...In part. We developers do all sorts of things to ensure our code is robust and maintainable. To become a fully rugged developer you only need to add security to the quality goals you try to achieve.

The interesting part of security is that you're protecting your code against intelligent adversaries, not just random things. So in addition to being robust against chaotic users and errors in other systems, your code also has to withstand attacks from people who really know the nuts and bolts of your programming languages, your frameworks, and your deployment platform.

Being a Rugged developer means you have a key role in your project's security story. The story should tell you what security defenses are available, when they are to be used, and how they are to be used. Your job is to ensure these things happen. You should also strive to integrate security tests into your development life cycle, and even try to hack your own systems. Better you than a "security" guy or bad guy, right?

Ideas for Being an Effective Rugged Developer

Add Security Unit Tests. Perhaps you've been to security training or you've read a blog post on a new attack form. Make it a habit of trying to add unit tests for attack input you come across. They will add to your negative testing and make your application more robust. A certain escape character, for instance ', may be usable in a nifty security exploit but just as well produce numerous of errors for benign users. A user registration with the name Olivia O'Hara should not fizzle your SQL statement execution. You as a developer have the deepest knowledge of how the system is designed and implemented and thus you are in the best position to implement and test security.

Model Your Data Instead of Using Strings. Almost nothing is just a string. Strings are super convenient for representing input, but they are also capable of transmitting source code, SQL statements, escape characters, null values, markup etc. Write wrappers around your strings and narrow down what they can contain. Even if you don't spend time on a draconic regular expressions or check for syntax and semantics, a simple input restriction to unicode letters + digits + simple punctuation may prove extremely powerful against attacks. Attackers love string input. Rugged developers deny them the pleasure.

Hack Your Own Systems. Even more fun, do it with your team. If management has a problem, tell them it's better you do it than someone on the outside.

Get Educated. There are many materials available to help you learn secure coding, including websites, commercial secure coding training, and vulnerable applications like WebGoat. Also, although top lists

can be lame, the OWASP Top 10 and CWE Top 25 are great places to start. As luck would have it, most of the issues in these lists are concrete and you can take action in code today. There are a lot more good materials available at both OWASP (https://www.owasp.org) and MITRE (http://cwe.mitre.org).

Make Sure You Patch Your Application Frameworks and Libraries. Know which frameworks and libraries you use (Struts, Spring, .NET MVC, etc) and their versions. Make sure your regression test suite allows you to upgrade frameworks quickly. Make sure you get those patch alerts. A framework with a security bug can quickly open up several or all your applications to attacks. All the major web frameworks have been found to have severe security bugs the last two years so the problem is very much a reality today.

Rugged Development in the Future

Simon is a relatively junior developer working on a new web application for a financial organization. Today, Simon is working on a new feature that will allow users to update their profile information more easily. The user interface has been approved, and Simon is building the service that will update the backend storage with the new information.

Simon takes advantage of their persistence layer that uses parameterized queries to defend against SQL injection. He uses some standard test cases for storage components also, and includes a test that is intended for use when sensitive information is being modified. The test requires re-authentication when changing information that has security importance, like the email and mailing address in the profile. The test fails because Simon didn't know about this requirement.

Simon updates his user interface to require the user's password, adds the appropriate logic, and the test case passes. The test cases become part of the test suite that runs as part of the build process, generating the evidence needed to demonstrate security.

Metrics

The Rugged Developer should evaluate success based on how well their code stands up to both internal and external stresses. How many weaknesses are discovered after the code is released for testing? How often are mistakes repeated? How long does it take to remediate a vulnerability? How many security-related test cases are associated with a project?

The Rugged Tester

As a Rugged Tester, I want to find security weaknesses before they cause harm. I will ensure that the software delivered to me faithfully executes the protections described in the security story.

To achieve my goals I have come to value...

- *Comprehensive testing over Random exploration*
- *Verifying positive behavior over Negative hacking*
- *Automated verification over Manual review*

Your Mission

As always, your primary responsibility is to verify that the software produced by your project meets the functional and non-functional requirements. Rugged simply adds the verification that all aspects of the security story are implemented correctly. This includes ensuring that security defenses are present, correct, and used properly. Testing is managed in many different, and many different roles may fulfill Rugged Tester responsibilities.

Your verification work provides the proof behind the security story. Every project, security defense, framework and integral component will have appropriate associated tests used to generate proof that all the engineering which has been done has culminated in a Rugged system and software development project.

Early in the requirements specification and design stage of the development process, get involved with the software development team and help create reusable test case templates or models that can be attached to each security defense. In addition, work with the business stakeholders and developers to determine what constitutes “success” or “failure” for a particular test case. The security analyst should assist in creating these test cases or perform research or tools for the testing team. This will ensure the software delivered to you during the testing phase has realistic goals and successful verification is attainable.

Security features will be tested in the same way all other functions and features of an application are tested. You will integrate automated and manual methods in order to test in a repeatable and effective manner, and the results produced will actually prove the correctness of the implementation of security along with all other features. The tests should be derived from the story, as opposed to randomly exploring security without a model.

Your testing of security features is no different than testing functional features and can be managed and implemented with your existing testing methodologies and tools. The other good news is that as your enterprise delivers security defenses used repeatedly throughout the organization you will be able to reuse the tests which target those features in order to ease your own burden.

Some Rugged Testers may have used penetration testing, security code review, or other types of security verification in the past. Those specialists should read the section about the Rugged Security Analyst. Your security skills are immensely valued, and your organization needs them to be focused on new vulnerabilities and novel risks.

Ideas for Being an Effective Rugged Tester

Create a Library of Reusable Test Cases Patterns. As security defenses and secure coding, configuration, and architecture patterns are defined to satisfy security requirements, testers should strive to create automated and manual test case patterns that are directly linked to each. A library of test cases

patterns and tools should be available throughout the organization. Repeatable test cases are an excellent way to help organizations avoid making the same mistake over and over.

Use both Positive and Negative Tests. The goal of the Rugged Tester is to verify that the security defenses are strong and effective. Test plans should focus on positive test cases that verify the presence, correctness, and proper use of the security defenses. Negative test cases should also be included that aim to show that the defenses cannot be tampered with or bypassed. For example, if the application performs an access control check before allowing an administrative function, the positive tests should verify that that authorized users are allowed and unauthorized users are blocked. The negative tests should test whether the authorization relies on any user controlled information, fails open, is incomplete, etc…

Use Vulnerabilities to Improve. If a standard defense for the vulnerability does not yet exist, work with the architect, security analyst, and developers to help them produce one. The existing codebase should be searched for other instances of this vulnerability that could be remedied with the new standard defense. For example, when the first instance of Cross Site Request Forgery (CSRF) is reported, the organization should take the opportunity to analyze the potential aggregate harm to the organization and decide whether investing in a standard control makes sense.

Relate Failed Security Defenses to the Security Story. When a test case fails for a security defense, relate that failure back to the security story to help determine the defect priority. To put the finding in context, it should include the lifeline, strategy, and defense that were compromised. The security story can be used to bridge the technical and non-technical gap between the development team and the business stakeholder. For example, instead of reporting "SQL Injection in report.jsp," a better finding would describe a failure to use the organization's standard database access pattern that relies on parameterized queries. In addition the finding should include a description of how this undermines the organization's data protection strategy, which is critical to the organization's reputation.

Create Repeatable Test Cases. Testers should strive to create repeatable test cases that can be used throughout the application and perhaps across the portfolio. Repeatable test cases are an excellent way to help organizations avoid repeating the same mistake. The OWASP ESAPI project contains a large library of test cases for common security defenses that can be tailored for an organization's own security mechanisms.

Create or Leverage Security Robots. Part of being Rugged is being prepared for unknown and unpredictable situations. Automated tools that cause random outages, damage, and other chaos will help to understand how the organization and applications will respond. Netflix has developed some open source tools that continuously challenge their infrastructure to fail early and often. These failures help them become Rugged. Security versions of these tools can simulate the failure of various defenses to see whether other layers of security, such as intrusion prevention, are sufficient to protect the organization's interests.

Rugged Testing in the Future

Denise is a software tester with many years of experience. She uses a variety of tools and procedures to verify that the software behaves as it should. She does not know much about security, but has access to an extensive library of test cases and knows how to choose the applicable cases in her assigned scenarios. Today, she is verifying a new shopping cart in an e-commerce web application. Denise has run the standard test procedures on the e-commerce website, and has found a few issues.

First, she noticed that a URL contained an integer parameter named cartId. She followed her standard procedure and changed the parameter to various values, and noted that other people's carts were being

updated. Next she noticed that entering certain special characters were corrupting the HTML for the cart. Upon investigation, she realized that her input was being rendered inside a block of Javascript without proper escaping.

Denise captured the exact details of how she found these issues, and identified secure coding patterns and example implementations on the organizations security portal. She reported them the vulnerabilities back to the development team. They recognized the problem and implemented several fixes. They also established automated test cases, to ensure that these problems are eliminated across the entire application, not just this particular instance. If successful, these tests will be promoted across the entire enterprise.

Metrics

The Rugged Tester should evaluate success based on the level of success in testing the project. How much code coverage is achieved in testing? Are there tested abuse cases for each area of concern? Are previously identified vulnerabilities incorporated into the test plan? How often are test cases run? How well automated are security tests?

Getting Started with Rugged

The good news is that you don't have to achieve an imaginary level or pass some arbitrary test. Getting started with Rugged is as easy as writing down as much as you know about your security story right now. Then you can start hardening your story by seeking information, prioritizing issues, and closing gaps.

The Rugged approach is totally compatible with existing software development approaches. With "Waterfall" style projects, the story tends to evolve from top-to-bottom where the top-level security concerns are identified early, architecture and defenses are defined, implementation details are added, and finally evidence is generates. On "Agile" style projects, a single defense is fully articulated and then additional defenses are added to the story over time. "DevOps" projects are particularly amenable to Rugged, because Rugged doesn't specify processes or practices. That means you can build your story and supporting evidence however you want.

Anyone Can Launch Rugged

Perhaps the simplest way to start down the Rugged path is to take a shot at capturing your security story. It doesn't have to be complete, accurate, or even legible. And you don't have to start it at the beginning of a project. Having a concrete story will allow you to start adding, deleting, rethinking, and refactoring to make the story more accurate, simple, and compelling. You should show the story to as many people as possible and get many different perspectives.

Start your story by brainstorming about the things that are most important to your business. What are the outcomes that you could not afford to have happen? Capture a list of all the threat agents that might attack your business. Think what harms they might be able to cause and start to prioritize them in terms of severity.

You be able to find business concerns by identifying the security defenses that are in place and asking why they are required. You may find a security defenses that are present but aren't really protecting your application from realistic risks. Or, worse, you may see defenses that haven't been enforced uniformly across the application, or defenses that rely on too much human behavior to get implemented correctly.

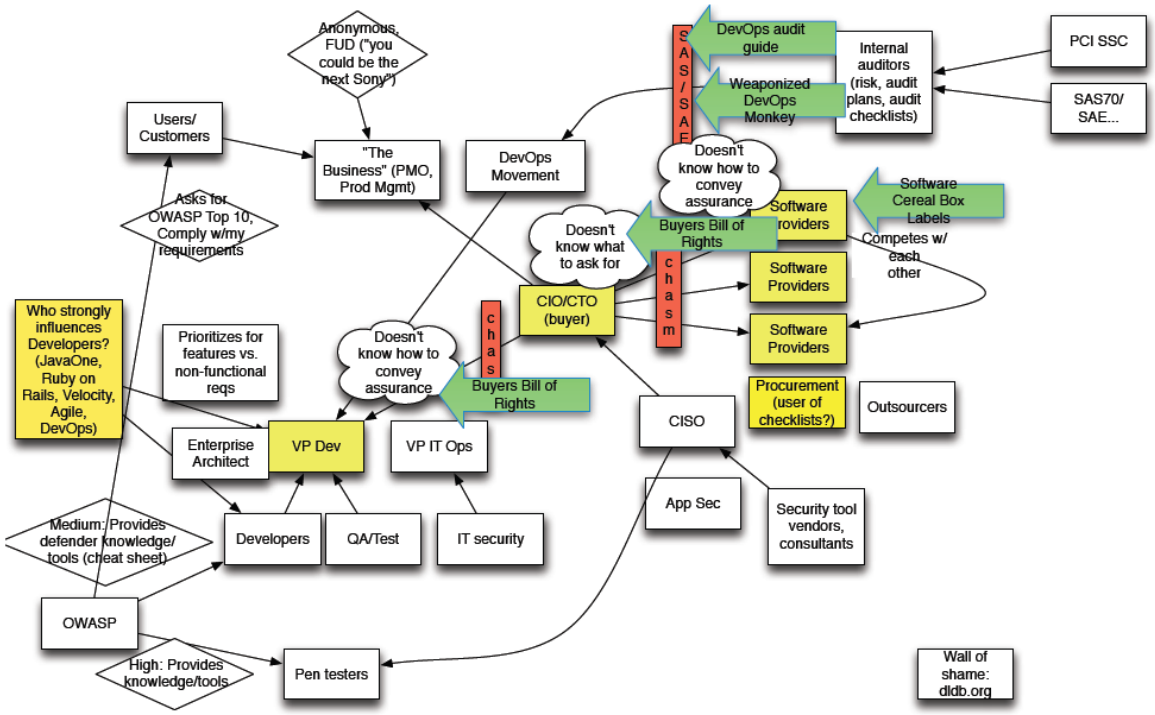
In some cases, you may get help by starting at the bottom with the security tests being performed and reverse engineering the business reason for them. You'll probably see many tests being performed that don't make sense for your application. This doesn't hurt anything, but we can do much better if the tests are aligned with the security story.

Along the way, you are going to notice gaps, overcomplexity, and inconsistencies in your story. It may take some time before your story is as clean and clear as it should be. We recommend that you treat this as an engineering problem with a current story and a future story. Use the future story to scope and guide enhancements to the security of your application.

Rugged Value Network

This value network was created at the 2012 Rugged Summit to capture the various roles and entities in a typical software development organization and how they provide value related to application security.

As you can see in the diagram, there is a large chasm between the executives and both internal and external software providers.



This chasm means that there is not a strong information flow between development groups and technology executives. We designed Rugged to help overcome these barriers to help both sides work together to make smart decisions about software risk.

As you consider your own organization's path towards application security, you should consider building out your own value network to help identify and evaluate the best opportunities to effect change. Seek out the paths where you believe information and value are not flowing freely. These might be opportunities for targeting your efforts to get the maximum return.

Proving that You're Rugged

How do you know if you are Rugged enough? When do you have enough security defenses? How do you know when you are done? These are questions that have plagued security for decades.

Know the Goal

Some applications require extreme levels of security, others not so much. Before you can decide whether a security story is strong enough, you need to know how much security you need. Generally, this is driven by the level of "inherent risk" in the business. To determine your risk, consider the people, assets, and functions involved in the business without considering the technology and assess the worst possible outcomes if a security disaster, of any type, should occur.

Rugged organizations will identify these risks early in the process and build their story around them. However, you still have to decide how much uncertainty you can tolerate. Do you need mountains of evidence and formal proof that your story was properly implemented? Or are you comfortable with periodic spot-checking and sampling across your story?

Rating a Security Story

It's fairly easy to tell whether an application is Rugged enough. Either there's a compelling security story or not. We have no need for complex appraisals, interviews, and assessments. We will let the story do the talking.

We haven't defined a formal way to evaluate security stories yet. Perhaps that will be developed in the future as patterns and practices for great security stories emerge. However, we offer the following as a rough guide to scoring a security story.

Level	Description
Fairy Tale	Generally a marketing document, the "fairy tale" outlines a few unclear defenses without a clear architecture or any line-of-sight to business security concerns. No evidence whatsoever.
Short Story	Usually available as a white paper or data sheet, the "short story" describes a few of the key security features of the application, but misses several obvious business concerns. Evidence is haphazard and disconnected from the rest of the security story. Might reference compliance with standards.
Novel	The "novel" describes a set of security features matched to most of the business security concerns. The story provides some evidence of security testing and process, but may show some gaps in coverage.
Documentary	In the "documentary," business security concerns are present and mostly complete, the security architecture and coding patterns are captured, implementation and testing evidence are clear. The work is satisfactory but possibly tedious and overwhelming.
Manifesto	The "manifesto" is the highest form of the security story. It's an appealing, clear, and compelling story that covers all the major business security concerns. The architecture

	and defenses are simple and strong and explained clearly. The correct operation of all the security measures is well-established by easy to understand evidence and backed up by third-party assessments.

There is always the possibility that someone might fabricate evidence or attempt to mislead others in their security story. This might create legal liability for an organization attempting this, so we also recommend that normal audits and spot-checks be alert and test for this type of deception. For particularly critical applications, the story can be evaluated by a third party. The results should, of course, become part of the security story.

Doing Battle

Do realistic hacking exercises generate assurance? We believe that they can, but only after you have built a compelling security story. There is a level of assurance you can't achieve without having some battle-testing in realistic scenarios.

This is not to imply that battle-testing is in any way a replacement for all the preparation necessary beforehand. Despite the widespread use of penetration testing as the sole security activity for an application, sending completely unprotected applications into battle to find out whether they are secure is just silly.

Another option if you are confident in your defenses is to allow security researchers to test your applications without danger of prosecution. You can even offer a "bug bounty" program that rewards researchers for their participation.

Slash Security (http://www.example.com/security)

So far, we've discussed the usefulness of a story to getting your software development organization aligned and focused on your security goals. In this section we discuss the power of disclosing your security story to the world.

For some reason, probably having to do with lawyers, almost every web site has a privacy page. We believe that most applications should provide their users with insight into how the application protects the user's data. Don't you want to know how your online bank protects your financial transactions? However, most websites are devoid of anything beyond vapid marketing claims of item like "military-grade encryption."

But you *could* also publish a version of your security story publicly. You may want the public to be aware of what you have done to secure your applications. Some organizations may even want to disclose things that they have *not* done to make it clear that those are the responsibilities of the user. With full details, users can make informed choices about whether to trust you with their data. You don't have to publish details of the environment that would help attackers in any way. The fact that you have published your security story may even protect you from litigation, so long as you actually do what you claim.

Your story can make a powerful marketing tool that quickly differentiates your application from other applications in a similar space. The elements of the story that explain how stakeholder concerns are addressed are the same elements that will convince clients and customers that using your application is safe. As newspapers and media already know, there's nothing more effective than a compelling story. The challenge is to ensure that the external security story contains the right level of detail that is convincing to customers and clients, without revealing sensitive details about the application. Consider placing your security story in a standard location so that interested parties can easily find your security story. We suggest the following locations:

Application Type	Location
Web Application	/security
Library	security.txt
Web Service	?security
Desktop Client	About -> Security

Supporting Tools and Technology

There is nothing about Rugged that demands the use of any particular tools or technology. However, we envision Rugged organizations leveraging technology to manage their security stories, instrument and monitor their software development organization, and respond to inquiries about security. While there are no tools specifically designed for Rugged, there are many that can contribute. In this section we consider the possibilities for supporting technologies and outline some requirements for their use. We encourage organizations to innovate in this space and contribute to the Rugged landscape.

1. Creating a security story

Currently there are numerous tools that could be leveraged to help construct a compelling security story. There are requirements tools, rudimentary threat modeling tools, architecture tools, test dashboards, and other related automation, but nothing that can truly manage the whole story.

We envision a system that would allow all the roles to collaborate on the argument concurrently. For example, while executives are contemplating their core business needs, security analysts are updating threats, developers can be generating code metrics, testers can be adding test results, and scanning tools can post their results all in a way that automatically links evidence into the security story in the right place.

Imagine an organization wrestling with their data protection requirements. Executives want to be sure that client data will not be exposed and embarrass the company. The security analysts have identified a number of ways that attackers might be able to cause this exposure. The architects are designing and positioning security defenses to stop these threats. These roles need to collaborate on the approach to crafting the security story.

The system should allow for all threats to be evaluated and managed. There may be threats that are addressed outside the system. We encourage you to model those threats as well, and capture why they are not a risk. This will help to ensure that threats have been considered and prevent wasted work.

The system should allow the rationale for the particular set of defenses selected to be captured. The rationale is typically a discussion of why the defenses selected address the threat adequately and with the appropriate level of assurance. The defenses might not prevent the threat, but may have to be deterrent or detective defenses. Not all defenses have to be technical, some can be procedural as well. In most situations, the story will be stronger with “defense-in-depth” or multiple defenses for a single part of the story. Whatever the mechanism, the value must be measurable and auditable.

2. Performing tradeoffs

The system should also enable organizations to evaluate alternative approaches for addressing threats. This will allow organizations to compare and contrast different approaches, choosing the one that makes the most sense for the particular business. Maintaining these alternative approaches will help justify the decisions that were made and explain why security works the way it does.

3. Generating evidence to support the story

Most projects already generate quite a bit of assurance evidence, including test results, penetration test reports, code review reports, and design documentation. The problem is that this documentation is rarely linked together into any kind of coherent structure.

We envision multiple different types of sensors that can gather evidence from all parts of the development process. The first type is the "direct sensor" that gathers data directly from the application itself. Some of these direct sensors work directly on the source code, such as static analysis, quality metrics, and security code review. Others work off of running code, such as test cases, dynamic scanning tools, and manual penetration testing. Notice that sensors can be generate their data automatically or they can require a human to gather and submit data.

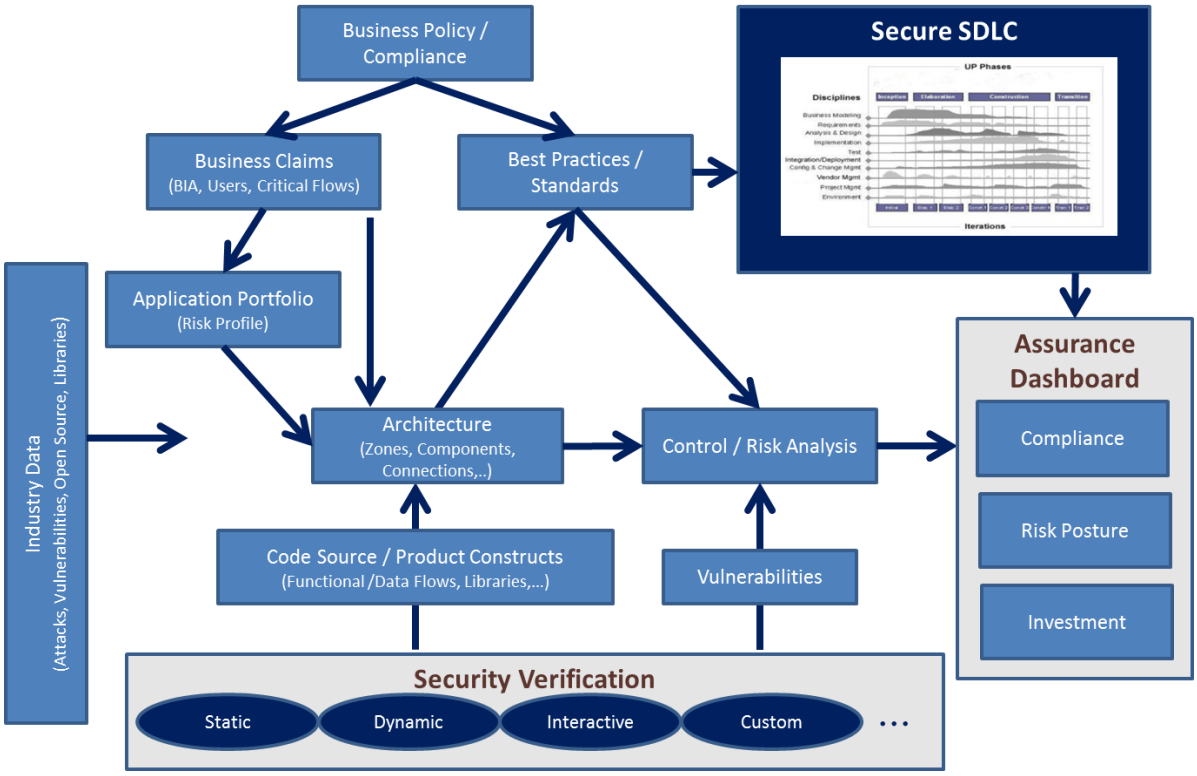
The second type of sensor is the "indirect sensor" that gathers information from the software development organization. These sensors don't measure the software directly. Instead, they pull information about the people, process, and tools that are used to build the software. For example, one "sensor" might gather information about which developers have taken training for certain topics. Another might gather information the exact tools used to build the software.

Organizations should focus on their stories to determine which types of sensors are most important and which deserve to be automated. We expect that many types of evidence will continue to be generated manually and then be added to the security story. Later, we hope these manual processes can be replaced with "scripts" or other automation, developed by security specialists and run by testers. Eventually, we hope that software components will include security "testability" as a feature, and allow easy verification of their proper operation. One simple example is the ability to produce a report of access control rules and entitlements that can be verified.

4. Managing the story and evidence

Imagine if every piece of evidence generated was submitted to a (hypothetical) Security Story Management System (SSMS) in a way that would automatically link it to the appropriate part of the story. The implications of a failed test or newly discovered security weakness could be very quickly diagnosed and addressed. This is similar in some ways to vulnerability management tools or governance, risk, and compliance (GRC) tools. However, these tools aren't (yet) able to model the full line-of-sight needed to represent a full application security story.

The entity model below shows an approach for modeling a security story in a database. A structure of claims is represented and supported by evidence from both security verification and the organization's secure SDLC. All the information will be viewed through an assurance dashboard.



Ultimately, the organization will be able to use this model and dashboard to answer important questions, such as:

- What are the threats to my organization?
- What are the most important defenses to add to my application?
- Are the applications across our enterprise compliant with standards and regulations?
- Which vulnerabilities are getting introduced into our portfolio most frequently during coding?
- What topics would be most useful to train developers in?
- And many more...

How Rugged Fits

Rugged doesn't really propose any new application security techniques or processes. Rugged is about assembling what we already know how to do into something that makes sense. You can think of Rugged as a type of framework built around the unifying concept of a security story that leverages much of what already exists. You'll prove that you're Rugged by putting your cards on the table in the form of a story. In this section, we'll discuss some of the major tools, trends, and techniques in application security and interpret their role in a Rugged organization.

Compliance

Compliance simply does not equal security. In many ways, subscribing to a compliance regime is the exact opposite of being Rugged. Where compliance breeds contempt for security, Rugged organizations simply do what they need to do to protect themselves.

There are a number of compliance regimes that address application security, including the PCI DSS, NIST 800-53, DISA Application Security STIG, and more. These standards are generic, and tend to result in a lowest common denominator" attitude towards security. Security stories, on the other hand, are closely tailored to your organization, technology, and business. If your project demands the use of a security standard, you should carefully interpret each requirement in light of your story.

For example, consider NIST 800-53 SI-10 - "Information Input Validation," which reads: "The information system checks the validity of information inputs." While this is fairly clear in intent, it is nowhere near specific enough for developers to implement or testers to verify. The application architects should determine which technologies will be used, how they will be configured, and what programming idioms should be followed by developers. In addition, the developers will have to resolve how to apply validation to every input to the application.

Instead, determine where each of the requirements fits into your story. Perhaps there's an important threat that they are trying to protect against which you did not consider. Or perhaps the requirement doesn't apply to your application and needs to be tailored or dropped. By forcing the assimilation of compliance requirements into the story, the necessary analysis happens early and yields great benefit downstream.

Development Approaches (e.g Waterfall, Agile, DevOps, etc)

Rugged should be easy to use in conjunction with a variety of development approaches. In many organizations, the development process is already generating many aspects of the story and all that is required is collecting them and making them fit together. Other organizations will need to add the necessary activities to generate the information necessary to build out the story. See below for sources that have information about how to build out these pieces.

In traditional waterfall projects, the security story is likely to develop from top to bottom. In the early stages of the project, you should identify business security needs and make sure they are addressed in the architecture. Later, during development and test, you will generate the evidence to prove that the story has been fulfilled.

An Agile project may choose to develop their security story in a depth-first fashion. The project might build out the security story associated with the current sprint. This story would be narrow, but would include the full line-of-sight from business concern all the way down to detailed evidence and test results proving that the appropriate security is in place.

Organizations using DevOps may choose to be even more aggressive about automatically creating the security story. We envision a fully instrumented software development organization, so that the evidence supporting the security story is continuously and automatically generated. We envision security and chaos "monkeys" that constantly attempt to break security defenses and encourage the project to create ever-stronger defenses.

Resources

OWASP

The Open Web Application Security Project (OWASP) has extensive information around application security. You might want to start by investigating OpenSAMM (see below) for activities, but OWASP has many more useful resources. You'll find tools, such as Zap security testing proxy, O2 to help with automation, and CSRF Tester. You can meet with the OWASP community at conferences and join your local chapter. You'll also find documentation on important application security issues, including the OWASP Top Ten, the Application Security Verification Standard (ASVS), the OWASP Testing Guide, and the OWASP Risk Rating Methodology.

MITRE

MITRE is focused on making security measurable, and they have a number of very useful resources to support that goal. The Common Weakness Enumeration (CWE) provides a vulnerability taxonomy and is a great source for understanding vulnerabiliites. They also have related standards for weakness risk analysis and scoring.

ESAPI

The Enterprise Security API (ESAPI) project is an effort to define what application security support is needed by developers. ESAPI includes a programming API and a number of reference implementation in various languages. You can use the ESAPI as a starting point for building out your own set of enterprise security defenses. In the long term, this has significant payoffs in terms of simplifying, externalizing, and strengthening your enterprise security story.

Automation

Automated application security tools are an important part of the current approach. Tools like dynamic vulnerability scanners, static analysis tools, and manual tools are all widely used. To achieve any kind of scale, automation almost has to be part of the future solution as well. Rugged provides some structure around exactly what these tools are supposed to produce. Automation results have to meaningfully contribute to some part of the security story. Otherwise, there is no point to using them. Therefore, we encourage tool vendors to be more explicit about exactly what they are testing, the level of rigor they achieve, and specifically what is not tested.

Process Models

The Open Software Assurance Maturity Model (OpenSAMM) is an "open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific

risks facing the organization." The Microsoft Secure Development Lifecycle and Build Security in Maturity Model also attempt this same goal. We strongly encourage you to investigate the activities that are detailed in these models and determine how they can be integrated and tailored for your organization. If you stay focused on your ability to produce your security stories efficiently and effectively, you'll be able to see which activities will fit into your organization and culture.

Case Studies

As mentioned previously, security stories can be a powerful marketing tool. In fact, several companies already tell their security story in varying forms. We examine a variety of examples to point out strengths in the story process.

Apple: Supplier Responsibility at Apple

The idea of a story is not limited to the realm of security. The Apple website has a section devoted to supplier responsibility that communicates Apple's efforts in this area. The site provides a high level overview of various areas of concern: labor and human rights, worker health and safety, environmental impact, etc. The site also cites evidence that Apple addresses these concerns along with additional detail that readers can drill-down into. These elements are not unlike a security story that identifies lifelines, strategies, defenses, and verification. The Apple Supplier Responsibility page is a good example of a powerful story organized into manageable, meaningful chunks.

http://www.apple.com/supplierresponsibility/

At the same time, the Apple story points out a major danger of using stories for public consumption. Because they are produced by the company, there will be temptation to distort facts and leave out important information. While there may be legal ramifications for this type of exaggeration, we're primarily focused on the benefits of using these stories for internal use. There's no better way to ensure that a complete picture of security gets assembled, and establishing some pressure to strengthen and simplify the story over time.

Box: Security Leadership in the Cloud

Box is one of many cloud-based storage service for files. The service has a page devoted entirely to security which outlines the various countermeasures in place at Box.

https://www.box.com/enterprise/security-and-architecture/

This page provides a great breadth of information including statements about account management, single sign-on support, reporting and auditing, network protection, data center configuration, and privacy. All of this information is relevant to a security story. However, unlike the Apple Supplier Responsibility story, the business-level lifelines are not explicitly defined, nor is any verification evidence discussed or provided.

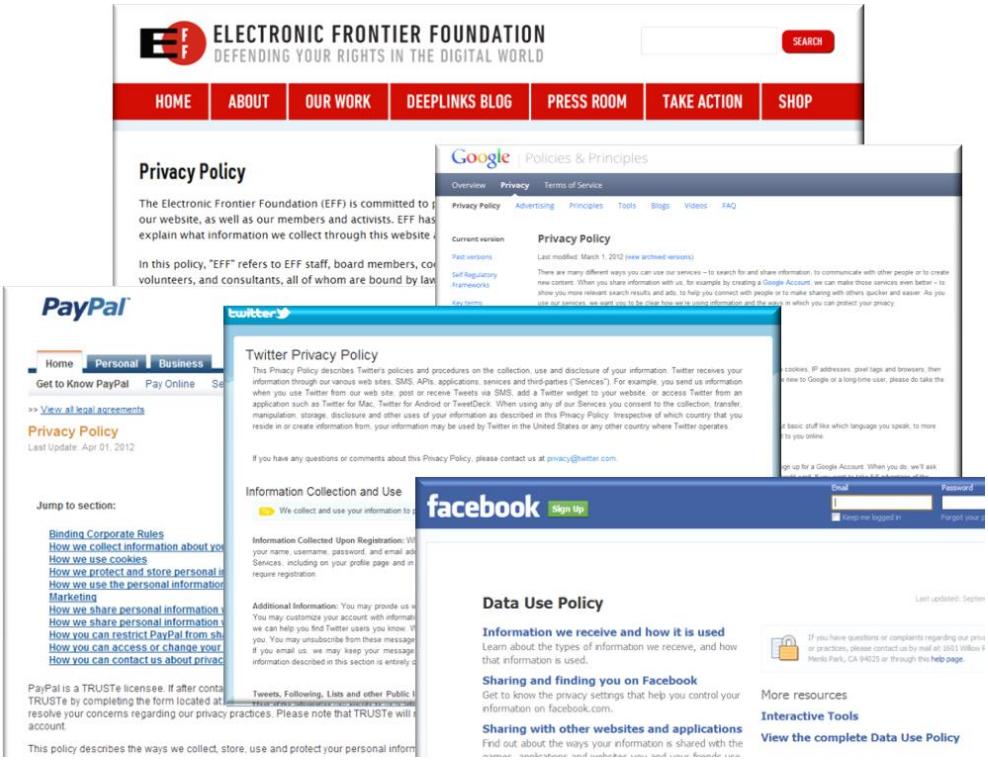
The focus on account management and single sign-on suggest a concern about securely establishing identity and enforcing access control to sensitive data. The reference to data encryption and privacy point to a concern about protecting the secrecy of sensitive data. Mentions about network protection and data centers allude to a concern about ensuring the accessibility of the service. However, a less informed user will not get much benefit to these story elements.

A more compelling story would first explicitly identify these concerns. The story would then explain how countermeasures like data encryption and auditing address those concerns. For the curious readers, additional detail could be provided.

The Box security story nonetheless provides a good foundation for their security story. This level of visibility shows the company considers security a priority and may very well be in the process of adopting a Rugged mindset.

Privacy Policies

Like security, privacy has been a legal concern long before the days of the Internet. With the advent of online web sites, companies began posting their privacy policies. These policies are ubiquitous across virtually every online web site. Like security stories, privacy stories share several common themes: information gathered, information usage, information sharing, etc. Security themes will not be as uniform as each application has concerns that are fairly unique to the functionality, whereas privacy concerns affect organizations in much the same way. The prevalence of such security stories across a wide spectrum of sites demonstrates how the simple concept of a consistent story can speed widespread visibility of an issue.



https://www.eff.org/policy
http://www.google.com/policies/privacy/
https://www.paypal.com/us/cgi-bin/marketingweb?cmd=p/gen/ua/policy_privacy-outside
http://twitter.com/privacy
http://www.facebook.com/about/privacy/

Hypothetical Network Monitoring Portal

The hypothetical application is a network management system that includes a web portal. The system aggregates network monitoring information from a variety of sources and creates actionable tickets for computer emergency response teams. This portal allows responders to "drill-down" into record sets that are drawn from the application knowledge base. The knowledge base is offered as a paid service charged by usage. Below, we have started an example security story in order to provide a concrete example. Note that this example is intentionally incomplete and should not be construed as an exemplary story.

Their story starts with a statement of their commitment to being Rugged and why it's important to their business and then outlines each of the major lifelines the company relies on.

Network Monitor Security Story

"You trust your network devices with all the information in your enterprise. Monitoring your network the only way to control a business running at Internet speed. Network Monitor is committed to the highest standards of application and network security for our Portal application. We secure our infrastructure, ensure data is always protected, build defenses against injection and other application attacks, practice rugged software development, and carefully verify our code. At the core of our security is a commitment to transparency – across our protections, processes, and even potential problems.

Confidentiality of Client Data

As a network monitoring service, we process and store sensitive information about any data packet that matches are known pattern signature. We know these packets can contain extremely sensitive information and we take precautions to ensure the data is protected.

- We use SSL for every connection in our system
- All of our databases use industry standard encryption
- Every server uses an encrypted file system
- All backups are made uses images of already encrypted file systems

Availability of Service

You depend on our service for your system so we know that any downtime, regardless of the reason, has a huge impact on your business. Not only have we designed our systems scalably to meet demand, we take proactive measures to guard against intentional misuse or degradation of the system.

- We have a distributed cluster of server farms to deal with changing levels of load
- Each user has an enforced threshold that throttles the number of requests and the amount of data they can access to mimimize how much they can affect other users
- Unauthenticated functionality is physically and logically separated from authenticated functionality so your actions are never affected by anonymous users
- We have a hot standby system ready to deploy at a moment's notice if the main system is compromised

Fraudulent Transactions

Our service is charged by usage so we realize it's important to know that we actively prevent fraudulent usage so you are never charged for transaction you didn't make

- We keep extensive audit trails of every action
- Our accounting team performs a manual review of all accounts against averages and variances based on previous usage
- Applications use transaction-based tokens to prevent spoofing

Integrity of Records

We know you depend on the accuracy of our service so we put great care in ensuring only authorized parties can modify any data

- All database accounts use the principal of least privilege
- Original records are always kept read-only; data changes are done throgh differential updates so all changes are always undoable
- Each record is tagged with an owner and only that owner can make any data changes

The organization started their story by identifying the things that mattered to the stakeholders. The external story focused on the user stakeholder and as a result, the organization identified four major lifelines:

- Users want their network monitoring information to be confidential
- Users want the portal to be available
- Users want to pay only for services used
- Users want to ensure that there was no unauthorized access to tickets
- They considered others, such as compliance,

The organization identified these concerns as variants on the confidentiality, availability, fraud, and integrity lifelines. Then the organization identified business processes, security defenses, and other mechanisms in place that specifically address the areas of concern. The organization limited the story to the lifelines that they thought was most suitable for the audience. However, the "behind the scenes" story (i.e. the internal security story) also included elements such as vulnerability testing, security architecture reviews, encryption algorithm decisions, key storage practices, and other pertinent details that were more appropriately kept within the organization.

The security story then connects these elements to the concerns through narrative. Observe that, in some cases, the audience can readily understand the effect of a chosen element. For example, the story describes how the application enforces a threshold and how the action addresses the concern of availability creating a coherent narrative. On the other hand, the story describes how SSL is used on every connection, but does not indicate what effect SSL has on the confidentiality of the client data.

A good external security story balances the level of detail with the intended audience. Note that this security story provides links on several key terms. Presumably, these links resolve to more detailed explanations of the element. Ideally, security stories are layered so that audiences that are inclined to more detail can obtain the relevant information. For example, each bullet in this story could be considered a "chapter summary" of the story whereas the link to greater detail could be considered the detailed chapter content.