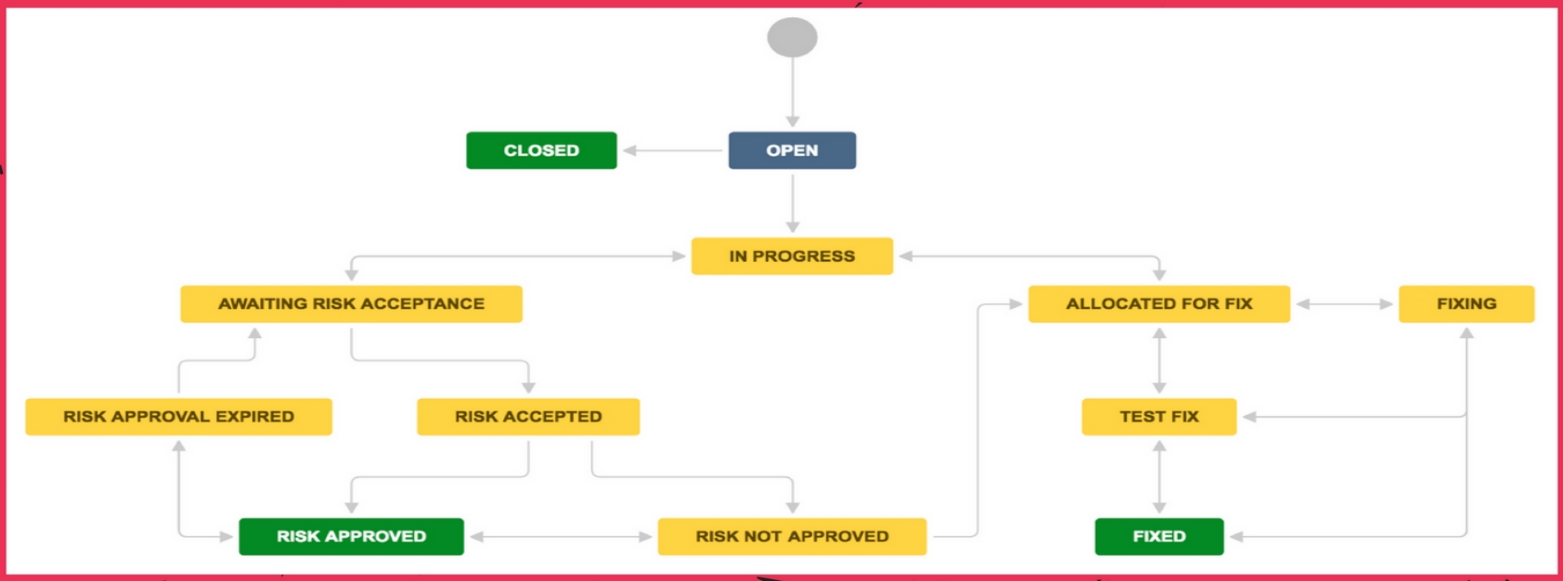


SecDevOps risk workflow

DINIS CRUZ

SecDevOps Risk Workflow

Risk Workflows to enable SecDevOps and manage Software Risks and Quality

Dinis Cruz

This book is for sale at http://leanpub.com/secdevops

This version was published on 2016-10-19

Contents

Introduction

- new introduction needed after changing name to 'SecDevOps Risk Workflow' (from JIRA Risk Workflow)
- new title make more sense since a lot of what the book is covering
- The JIRA Risk workflow fits as the way to implement SecDevOps
- Lots of good thinking on SecDevOps
- See this issue for more details on this decision Issue 15[1]
- Book is positioned on adding security to DevOps with a focus on Risk acceptance and Code Analysis workflows

Here[2] is good definition of SecDevOps is:

> *SecDevOps (Securing DevOps) The scenario here is an organisation is embarking on a DevOps and agile ways of working adoption journey. There is concern about security and we are asked to advise on how to embed security into the DevOps style of operation. And this means embedding and ensuring "secure by design" discipline in the software delivery methodology using techniques such as automated security review of code, automated application security testing, educating and empowering developers to use secure design patterns etc.*

(previous) Introduction

This book will give you a solution for the following common problems inside AppSec and Development teams:

- *"How do I get my manager to take security issues in my app seriously"*
- *"How do I get time to spend on non-functional requirements and refactoring"*
- *"We are product-driven development team and don't have time for anything that is not customer-driven"*
- *"We know our current development, testing and deployment environment is highly inefficient, but how can we prove that to management"*
- *"We constantly do hacks and compromises before deadlines, but we can't measure its real impact, and how they always tend to be a false economy"*

I find that collectively developers teams know what needs to be done, but they are not usually listened to or empowered to make important decisions about their development priorities and environment. There are always side effects of rushed code and hacks.

[1] https://github.com/DinisCruz/Book_Jira_Risk_Workflow/issues/15

[2] https://www.linkedin.com/pulse/devsecops-secdevops-difference-kumar-mba-msc-cissp-mbcs-citp

This book contains the materials created while using JIRA to manage RISK and measure those side effects

All information presented based on real work experience in creating, implementing and deploying multiple JIRA RISK Projects.

There are 2 main target audiences is AppSec and Developers.

Why AppSec

AppSec (Application Security) has massive problem of how modern developers work and therefor how to communicate with developers

I created this workflow in order to solve this problem

Why Developers

Because developers can use this workflow to control their development workflow, and handle the 'backlog pit of despair' problem.

Why JIRA

- It's market penetration (most companies I work for use JIRA)
- JIRA and GitHub are the only Issue tracking system that I have seen that have the kind of features that are required for the kind of complex workflows that are needed
- JIRA can be integrated with various other tools used in Agile/DevOps/CI allowing for further integration. (ie future ready)
- comment that this is very different from the JIRA Risk plugins that exist current in the JIRA marketplace

Why not infoSec

InfoSec tickets don't tend to work that well with JIRA tickets where InfoSec usually has their tools and dashboards to manage (for example Firewalls, Anti-Virus detections, Active Directory issues)

Could these same techniques be used to manage other types of Risk. Yes but since I don't have that much experience with it, I will leave it to the reader.

> AppSec is about: code, apps, CI, secure coding standards, threat models, frameworks, code dependencies, QA, testing, fuzzing, dev environments, DevOps, ….

> InfoSec is about: Networks, Firewalls, Server security, Anti-virus, IDS, Logging, NOC, Policies, end-user security, mobile devices, AD/Ldap management, user provisioning, DevOps, ….

Note that if your 'InfoSec' team/person cannot code (and would not be hired by the Dev team), then that is NOT AppSec. InfoSec is very important, but it is key to understand its limitations when there are no coding skills in that department (there are a number of conversions and activities that can only be done by professionals that understand development)

Is this view too restrictive? Can someone be a valuable AppSec specialist without actually being able to code. Would these qualities compensate for lack of coding skills: * understanding modern development, * being able to explain and discuss issues with developers, * being able to explain, configure, and use tools

Why I'm writing this book

- my struggle to communicate with developers
- explain how I arrived at the JIRA Risk project workflow
- brief O2 Platform history, and how I used to communicate with developers in a common language (in O2 it was via O2 scripts and mini-tools)
- O2 Platform REPL and how it changed how I code
  - what IDEs should be doing for Us (developers and AppSec professionals)

Change log

Here are the changes made (per version): * **v0.59** * added section on: * *“Getting Assurance and Trust from Application Security Tests”* * *“Hyperlink everything you do”* * *“Developer Teams Need Budgets”* * *“Developers Should be Able to Fire Their Managers”* * *“Every Bug is an Opportunity”* * *“Code Confidence Index”* * *“Getting Assurance and Trust from Application Security Tests”* * *“Chained threat models”* * **v0.58** * Started refactoring content into SecDevOps section * New content and multiple content fixes * **v0.56** * Changed title to *‘SecDevOps Risk Workflow’* (from *‘Jira RISK Workflow’*) * Added SecDevOps section * **v0.55** * First contribution via PR
* added license details (CC BY 4.0) * **v0.54** * Major section on *Security Champions* * **v0.52** ** * **Major content changes including import of multiple chapters that were in the Quality Book[3] * bumping to version 0.50 due to the current volume of content** * **v0.11** * Added automation using leanpub-book-site[4] tool * Lots of content added, first pictures in chapters

- **v0.10 (Oct 2016)**
  - Created Git repo and hooked Leanpub
  - Added first set of content files

[3] https://github.com/DinisCruz/Book_Software_Quality
[4] https://github.com/o2platform/leanpub-book-site

Contributions

The following individuals helped in the creation of this book with suggestions, comments and Pull Requests:

- Dave van Stein (https://github.com/davevs)

Disclaimers

- **(unit) Test** - For me a test is anything that can be executed with one of these Unit Test Frameworks: https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- **RISK** - Abuse the concept, found RISK to be best one for the wide range of issues covered by AppSec, while being understood by all players
- **100% Code Coverage** - not the summit, but base-camp (i.e. not the destination). And 100% code is not enough, we really need 500% or more Code Coverage)
- **AppSec** ~= Non Functional requirements - AppSec is about understanding and controlling appâ€™s unintended behaviours

License

Printed version

If you are reading this on a digital device, here are some reasons why you might want to consider buying the printed book [5]:

- Better way to learn the concepts in this book (see "Physical Books are the best technology for reading"[6])
- better graphs and diagrams (which are easier to understand and cross reference on paper)
- helps funding the development of this and other books (that I'm working on)

[5] the printed version of this book will be created after the first v1.0 release, and will be released at lulu.com and Amazon.

[6] http://blog.diniscruz.com/2013/09/physical-books-are-best-technology-for.html

1. Sec-DevOps

1.1 Concepts

1.1.1 SecDevOps or DevSecOps

- Which one is better
- Which one sends the better message
- DevSecOps seems to be getting a couple more google queries : https://www.google.co.uk/trends/explore?q=sec
- DevSecOps has the advantage that puts Dev (Development) first (which is good since that is most important part)
- SecDevOps is a good extension of DevOps which is already a known practice
  - I also like the idea that when Security becomes embedded in the SDL and Sec-DevOps just becomes DevOps
- I really like the definitions here
- DevSecOps & SecDevOps - Is there a difference? https://www.linkedin.com/pulse/devsecops-secdevops-difference-kumar-mba-msc-cissp-mbcs-citp

1.1.2 Good resources on DevSecOps and SecDevOps

DevSecOps & SecDevOps - Is there a difference? https://www.linkedin.com/pulse/devsecops-secdevops-difference-kumar-mba-msc-cissp-mbcs-citp

SecDevOps

- https://www.reddit.com/r/secdevops/
- https://twitter.com/hashtag/secdevops
- SecDevOps: Embracing the Speed of DevOps and Continuous Delivery in a Secure Environment https://securityintelligence.com/secdevops-embracing-the-speed-of-devops-and-continuous-delivery-in-a-secure-environment/
- SecDevOps: Injecting Security Into DevOps Processes https://blog.newrelic.com/2015/08/27/secdevops-rugged-devops/
- SecDevOps - Source Code Review Consultants (at speed) https://www.seek.com.au/Job/32037282?cid=advlink
- The 12 Days of SecDevOps http://blog.threatstack.com/the-12-days-of-secdevops
- Advanced Cloud Security and applied SecDevOps https://www.blackhat.com/us-16/training/advanced-cloud-security-and-applied-secdevops.html
- Swimming application security upstream with SecDevOps http://www.esg-global.com/blog/swimming-application-security-upstream-with-secdevops
- https://devops.com/tag/secdevops/
  - Murphy's DevOps: Is Security Causing Things to Go Wrong? https://devops.com/2016/02/29/murphys-devops-is-security-causing-things-to-go-wrong/
  - Flash Mob Inflection: Rugged DevOps Revolution https://devops.com/2016/02/19/flash-mob-inflection-rugged-devops-revolution/

  - Security Breaks DevOps – Here's How to Fix It https://devops.com/2015/07/08/security-breaks-devops-heres-how-to-fix-it/
  - The devOpsSec Dilemma: Effective Strategies for Social Networking https://devops.com/2015/05/28/devo dilemma-effective-strategies-social-networking/
  - Automated Security Testing in a Continuous Delivery Pipeline https://devops.com/2015/04/06/automate security-testing-continuous-delivery-pipeline/
  - It's time security pros shake their DevOps fear, uncertainly, and doubt https://devops.com/2015/05/18/its- time-security-pros-shake-their-devops-fear-uncertainly-and-doubt/
  - ChatOps: Communicating at the speed of DevOps https://devops.com/2014/07/16/chatops-communicating-speed-devops/
- SecDevOps: The New Black of IT http://www.slideshare.net/CloudPassage/sec-devops-webinar-deck

DevSecOps

- https://github.com/devsecops/awesome-devsecops (more links)
- https://twitter.com/hashtag/devsecops
- http://www.devsecops.org/
- http://www.devseccon.com/
- A primer on secure DevOps: Why DevSecOps matters http://techbeacon.com/devsecops-foundations
- Why Did We Need to Invent DevSecOps? http://blog.threatstack.com/why-did-we-need-to-invent-devsecops
- DevOpsSec Securing Software through Continuous Delivery http://www.oreilly.com/webops-perf/free/devopssec.csp
- DevSecOps: The Marriage of SecOps and DevOps www.tripwire.com/state-of-security/security-awareness/devsecops-the-marriage-of-secops-and-devops/
- A Look Back at DevOpsDays Austin 2016 http://blog.threatstack.com/a-look-back-at-devops-days-austin-2016
- MASTERED DEVOPS? WHAT'S NEXT? DEVSECOPS, THAT'S WHAT! https://www.cloudreach.com/us-en/2014/11/devops-devsecops/?topic=aws
- Gartner: DevOps is good; DevSecOps is better http://searchcio.techtarget.com/tip/Gartner-DevOps-is-good-DevSecOps-is-better

1.1.3 History of Sec-DevOps

1.1.4 Making the Sec part invisble

- concept that a lot of the work done today (2016) in AppSec and SecDevOps is part of a transition into just DevOps
  - when security in embedded into the Dev SDL

  - when security is invisible
- Case-Study: The biggest advantages of Microsoft's Security push has been the quality and robustness of their products, and big improvement on their SDL Story: IE body count from *Renegades of the Empire* book)
- add reference (and some content from) *Secure coding (and Application Security) must be invisible to developers*[1]

1.1.5 Rugged Software

- https://www.ruggedsoftware.org/
- Rugged Manifesto
- add explanation of what it is (and its history)
- why it didn't really work (as least according to the original expectations)
  - lack of community adoption
  - 'Security Driven' vs 'Developer driver'
- The Docker case study
  - why Docker was so successful (as an idea and adoption)
- lessons learned

[1] blog.diniscruz.com/2012/04/secure-coding-and-application-security.html

1.2 Dev-Ops

1.2.1 Don't kill the Ops

- be careful that DevOps is not a cost saving exercise, where it is seen as a way to kill/reduce Ops.
  - Dev + Ops -> DevOps -> Dev
- seen cases where there is an crazy idea that the 'Ops' team will be made redundant

1.2.2 History of DevOps

1.2.3 Infrastructure as code

- all changes and scripts and clicks:
  - are code
  - need to be stored in git
  - need tests (can run locally and in production)

1.2.4 Patch speed as an quality metric

- patching shouldn't be a problem
  - if it is a problem:
    * then that needs to addressed
    * means that there are problems in deploying the app (or rebuilding the server)
    * means hard rollbacks
  - open JIRA ticket on 'lack of patching'
    * do this before an incident, so that when problems occur (due to lack of patching), there is a way to capture the incidents
    * history (and Murphy) should provide evidence on the cost of non-patching
- lack of patching are 'canaries on coal mine' (point to a bigger problem)
- think of patching as 'fire drills'

1.2.4.1 Patching is easy

When: * it is easy to run the new version in an isolated environment (with and without production data) * there is a good understanding of what is going to change (files, ports, behaviour, inter-dependencies, schema's changes) * i.e. a diff with the current version of everything * there are Tests (e2e and integration) that run though the affected systems success criteria and confirm (if any) what changed (i.e. the side effects) * it easy to rollback to previous version(s)

1.2.5 Performing root-cause analysis

- some environments make it hard to perform root-cause analysis, because it is seen as a blame exercise, instead of a learning opportunity
- Root-cause analysis are key for any bug (namely one the ones with Security implications)
  - lack of 'Root-cause analysis' is a risk (which needs to be accepted)
    * means business owners doesn't want to spend the time to find other (similar issues)
- add story of project manager that asked "please don't find more security issues during the retest (they are out of scope)"
- in an DevOps world, it is key for the root-cause analysis that there is a way replicate the issue (in an simulated environment).
  - the end-result of a root-cause analysis should be a Test (very close to a script) (that passes when the issue can be replicated correctly and reliably)

1.2.6 When devs are cloud admins

- When moving to the cloud, Devs are becoming SysAdmins
  - which is a massive issue and is creating a large number of security issues
- in some cases this move is literally *'throwing the baby with the bath water'* where lack of innovation, speed and features from the local admin/IT teams, pushed the developers to have free reign in the cloud (like AWS) where they can create servers and VMs in seconds (vs days, weeks or months)

1.3 Sec DevOps patterns

- provide examples

2. Risk Workflow

2.1 Concepts

2.1.1 Abusing the concept of RISK

As you read this book you will notice liberal references to the concept of RISK, especially when I discuss anything that has security or quality implications.

The reason is I find that RISK is a sufficiently broad concept that can encompass issues of security or quality in a way that makes sense.

I know that there are many, more formal definitions of RISK and all its sub-categories that could be used, but it is most important that in the real world we keep things simple, and avoid a proliferation of unnecessary terms.

Fundamentally, my definition of RISK is based on the concept of 'behaviours' and 'side-effects of code' (whether intentional or not). The key is to map reality and what is possible.

RISKs can also be used in multi-dimensional analysis, where multiple paths can be analysed, each with a specific set of RISKs (depending on the chosen implementation).

For example, every threat (or RISK) in a threat model needs to have a corresponding RISK ticket, so that it can be tracked, managed and reported.

Making it expensive to do dangerous actions

A key concept is that we must make it harder for development teams to add features that have security and quality implications.

On the other hand, we can't really say 'No' to business owners, since they are responsible for the success of any current project, and they have very legitimate reasons to ask for those features. Business wishes are driven by (end)user wishes; saying no to the business means saying no to the customer. The goal, therefore, is to enable users to do what they want, with an acceptable level of risk.

By providing multiple paths (with and without additional or new RISK) we make the implications of specific decisions very clear.

What usually happens is that initially, Path A might look easier and faster than either of Paths B or C, but, after doing a Threat Model (i.e. creating a better brief) and mapping out Path A's RISK, in many cases Path B or C might be better options due to their reduced number of RISKs.

It is important to have these discussions before any code is written and while there is still the opportunity to propose changes. I've been involved in many projects where the security/risk review only occurs before the project goes live, which means there is zero opportunity to make any significant architectural change.

2.1.2 Accepting risk

Accepting risk is always a topic of discussion, mostly caused by lack of information. The JIRA Workflow aims to make this process more objective and pragmatic.

Risks can be accepted for a variety of reasons:

- the system containing the risk will be decommissioned in the short term
- the risk can only occur under very specific circumstances; these would already have caused a bigger risk
- the cost of fixing (both money and other resources) is much higher than the cost of fixing operational issues
- the business is okay with the probability of occurrence and its financial/commercial impact
- the system is still under development and it is not live
- the assets managed/exposed by the affected risks are not that important
- the risk represents the current state of affairs and a significant effort would be required to change it

Risks should never be accepted for issues that

- threaten the operational existence of the company (e.g. losing a banking license)
- have significant financial impact (e.g. costs, fines due to data leaks)
- have significant business impact (e.g. losing customers)
- greatly impact the company's reputation (e.g. front page news)

Not being attacked is both a blessing and a curse. It's a blessing for a company that has not gone through the pain of an attack, but a curse because it is easy to gain a false sense of security.

The real challenge for companies that have NOT been attacked properly, or publicly, and don't have an institutional memory of those incidents, is to be able to quantify these risks in real business and financial terms.

This is why exploits are so important for companies that don't have good internal references (or case studies) on why secure applications matter.

As one staff member or manager accepts risk on behalf of his boss, and this action is replicated throughout a company, all the way to the CTO, risks are accumulated. If risks are accepted throughout the management levels of a company, then responsibility should be borne in a similar way if things go wrong. Developers should be able to ensure that it is their manager who will get fired. That manager should make sure that his boss will get fired, and so on, all the way to the CTO and the CISO.

- business owners (or who controls the development pipeline) make large number of decisions with very little side effects

- add explanation of why pollution is a much better analogy than 'Technical Debt' (in measuring the side effects of coding and management decisions).
  - Risk is a nice way to measure this polution
- *"...you are already being played in this game, so you might as well expose the game and tilt the rules in your favour..."* *

There is a video by Host Unknown, that lays out the consequences of thoughtlessly accepting risk. It is a comic piece, but with a very serious message. (Host Unknown presents: Accepted the Risk[1])

2.1.3 Creating Small Tests

When opening up issues focused on quality or security best practices (for example doing a security assessment or code review), it's better to keep them as small as possible. Ideally these issues are placed on a separate bug tracking project (like the Security RISK one), since they can be quite problematic with project managers which like to keep their bug count small.

If those nuggets of information are not captured, what happen is that a lot of knowledge will be missed/lost, since that information will only exists at the moment when AppSec Developers are looking at code. This needs to be captured or is lost (sometimes forever or until that bug/issue becomes active)

This also helps to capture the high/critical security issues that are made of multiple low/medium ones, and visualize patterns that occur across the organization (for example an issue that affects dozens of teams or products)

What is very important is that you have a place to put all those little things and little examples, and little changes.

This will also provide way to measure exactly what will occur during a quality pass (or sprint). i.e. what to do when focusing on cleaning and improving the quality the application.

The smaller the issue, the smaller the commit, the smaller the tests will be. All make the whole process much more smoother and easier to review.

It will also create a great two-way communication system between Development and AppSecs, since it's great place for the team to collaborate.

When the new Developers joins the team, they should be starting with fixing those bugs and creating tests for them (which will really help them to understand what is going on and how the application works).

I also find that fixing these kind of easy tests are a good warm up for more complex Development tasks.

[1] https://www.youtube.com/watch?v=9IG3zqvUqJY

2.1.4 Creating Abuse Cases

- dev teams tend to focus on the Happy Paths
- In Agile environments creating evil user stories linked to a user story can be a powerful technique.
- think malicious 'Murphy' [2]

Evil user stories (Abuse cases)

Evil user stories have a dependency with threat modeling and can be an effective way of translating higher level threats and mitigations.

Take for example a login flow. After doing a threat model on this flow one should have identified the following information:

Attackers: * registered users * unregistered users

Assets * credentials * customer information

With this information the following threats can be constructed * unregistered users bypassing the login functionality * unregistered users enumerating accounts * unregistered users brute-forcing passwords * registered users accessing other users' information

At the same time the team has constructed the following user stories that should be implemented: *"as a registered user I want to be able to login with my credentials so I can access my information"*

Combined with the outcome of the threat model the following evil user stories can be constructed: * *"as an unregistered user I should not be able to login without credentials and access customer information"* * *"as an unregistered user I should not be able to identify existing accounts and use them to attempt to login"* * *"as a user I should not be able to have unlimited attempts for logging in and hijack someone's account"* * *"as an authenticated user I should not be able to access other users' information"*

2.1.5 Deliver PenTest reports using JIRA

- much more efficient that creating a PDF
- allows quick validation of findings and direct communication with developers and AppSec team
- There are a number tools that act (if customised properlY) as a triage tool between pentest results and JIRA
  - 'defect dojo'
  - 'bag of holding'
  - ThreadFix
  - Other…
- Mention how Threat Modeling tools could fit in this

[2] Murphy's law: *'Anything that can go wrong, will go wrong'*, https://en.wikipedia.org/wiki/Murphy%27s_law

2.1.6 Email is not an Official Communication Medium

Emails are conversations, they are not official communication mediums. In companies, there is a huge amount of information and decisions that is only communicated using emails, namely:

- risks,
- to-dos
- non-functional requirements
- re-factoring needs
- post-mortem analysis

This knowledge tends to only exist on an email thread or in a middle of document. That is not good enough. Email is mostly noise, and once something goes into an email, it is often very difficult to find it again.

If information is not at the end of a link, like on a wiki page, bug tracking system or source control solution like git, it basically doesn't exist.

It is especially important not to communicate security risks or quality issues in email, where it is not good enough to say to a manager, *"... by the way, here is something I am worried about..."*.

You have to create and send a JIRA RISK ticket to the manager.

This will allow you to track the situation, the information collected and the responses; in short, you can track exactly what is going on. This approach also gets around the problem of someone moving to other teams or companies. Their knowledge of a particular issue remains behind for someone else to use if they need to.

Emails are not a way to communicate official information; they are just a nice chat, and they should be understood as such. Important official information, in my view, should be hyperlinked.

The hyperlinkability of a piece of information is key, because once it has a hyperlinked location, you can point others to it, and a track record is created.

The way I look at it, if information is not available on the hyperlink, it doesn't exist.

The Slack revolution

There is a Real-time messaging revolution happening, driven by tools like Slack or Matter-most, which are quickly becoming central to development and operational teams (some still use old-school tools like Basecamp, Jabber, IRC, IM or Skype)

One the real powers of this new generation of collaboration tools, is the integration with CI/CD and they capability to become the glue between teams and tools.

The problem is that all data is short-lived and will soon disappear again (there are some limited search capabilities, which are as bad as email)

Using Wikis as knowledge capture

JIRA issues can also be hard to find, specially since they they to be laser sharp focused on specific topics.

Labels, queries and reports help, but the best model is capture their knowledge, by linking to them on Wikis (like MediaWiki or Confluence) or Document Management tools (like Umbraco or Sharepoint). The idea is to document lessons learned, how-to's and best practices.

2.1.7 Hyperlink Everything you do

Whether you are a developer or a security person, it is crucial that you link everything you do, to a location where somebody can just click on a link and hit it. Make sure whatever you do is hyperlinkable.

This means that what you create is scalable, and it can be shared and found easily. It forms part of a workflow that is just about impossible, if you don't hyperlink your material.

An email is a black box, a dump of data which is a wasted opportunity because once an email is sent, it is difficult to find the information again. Yes, it is still possible to create a mess when you start to link things, connect things, and generate all sorts of data, but you are playing a better game. You are on a path that makes it much easier in the medium term for somebody to come in, click on the link, and make sure it is improved. It is a much better model.

Let others to help you.

If you share something with a link, in the future somebody can proactively find it, connect to it and do something about it. That is how you scale. Once you send enough links out to people, they learn where to look for information.

Every time I write something that his more than a couple of paragraphs long I try to put in a link so that my future self, or somebody else in the future, will be able to find it and propagate that information without my active involvement.

Putting information in a public hyperlinked position encourages a culture of sharing. Making information available to a wider audience, either to the internet or internally in a company, sends the message that it is okay to share.

Sharing through hyperlinking is a powerful concept because when you send information to somebody directly it is very unlikely that you will note on each file whether it is okay to share.

But if you put data on a public, or on an internal easy-to-access system, then you send a message to other players that it is okay to share this information with others. Sending that link to other people has a huge impact on how that idea, or that concept, will propagate across the company and across your environment.

The thing to remember is that you are playing a long game. Your priority is not to get an immediate response, it is to change the pattern, stage the flows.

2.1.8 Linking source code to Risks

- add links to risk as source code comments
  - links are added to root cause location and all places it exists
- very powerful technique
- makes the risk visible
- reinforces the concept that there is a cost (i.e. pollution) to write insecure and less-quality code
- positive model where fixes will delete the comments
- allows AppSec team to detect when security reviews and needed (in this case when the comments are removed)
- build tools can scan for these comments and provide a 'risk pollution' indicator
- *add coffee-script example from 'Start with tests' presentation*

2.1.9 Mitigating Risk

- One strategy to handle list is to mitigate it
- sometimes external solutions (like a WAF) can be a more effective solution
  - If it fixed the issue and there are Tests that prove it, then it is a valid fix
  - in this case a new RISK needs to be created which says *"Number of security vulnerabilities are currently being mitigated using a WAF, if that WAF is disabled, or there is an WAF bypass exploit, then the vulns will become exploitable"*

2.1.10 Passive aggressive strategy

- Keep zooming in on the answers (and refine the Issue scope and focus)
- not finding something is also an Risk
- not having time to research something is also a Risk
- sometimes we just have to open tickets that state the obvious:
  - when are not using SSL (or have an HTTP to HTTPS transition), then we have to open an Risk to be accepted with: *"Our current **SSL** protection **only works when the attacker** we are trying to protect against (via man-in-the-middle) **is not there**"*)

2.1.11 Reducing complexity

The name of the game is to reduce the moving parts of a particular solution, its interactions, its inputs and behaviours.

Basically what we want is to reduce the complexity of what is being done.

2.1.12 Start with passing tests

- add content from presentation

2.1.13 Ten minute hack vs one day work

- 10m hacks vs 1 day proper coding will create more RISK tickets and source comments
- I get asked this a lot from developers
- There has to be a quantitive difference, which needs to be measured

That code sucks

- Ok can you prove it?
  - Code analysers in IDE can help identify known bad patterns and provide awareness
- Why is it bad?

In new code legacy - if it also hard to refactor and change, then yes

2.1.14 Triage issues before developers see them

Issues identified in vulnerability scans, code analysers, and penetration tests, must be triaged by the AppSec team to prevent overloading developers with duplicate issues or false positives. Once triaged, these issues can be pushed to issue trackers, and exposed to the developers.

Unfortunately, this is still an underdeveloped part of the AppSec pipeline. Even commercial vendors' UI don't really support the customisation and routing of the issues they 'discover'.

The following tools are attempting to do just that:

- Bag of Holding https://github.com/aparsons/bag-of-holding
- Defect Dojo https://github.com/OWASP/django-DefectDojo
- ThreadFix

With these triage tools, AppSec specialists can identify false positives, accepted risks, or theoretical vulnerabilities that are not relevant in the context of the system. This ensures that developers should only have to deal with the things that need fixing.

Create JIRA workflow to triage raw issues

The creation of a security JIRA workflow for the raw issues to act as a triage tool, and push them to team boards after reviewing, would be another good example of the power of JIRA for workflows.

2.1.15 Using AppSec to measure quality

- add content from presentation AppSec and Software Quality[3]
- academic research
  - https://insights.sei.cmu.edu/sei_blog/2016/06/using-quality-metrics-and-security-methods-to-predict-software-assurance.html[4]

[3] http://blog.diniscruz.com/2016/05/appsec-and-software-quality.html

[4] UsingQualityMetricsandSecurityMethodstoPredictSoftwareAssurance

2.1.16 Graduates to manage JIRA

One of the challenges of the JIRA RISK workflow is managing and maintaining the opened issues. This can be a considerable amount of work, especially when there are 200 or more issues.

Note that, in large organisations, the number of risks opened and managed should be above 500, which is not a lot, and in fact, is the level when visibility into existing risks really starts to happen.

The solution isn't to have less issues.

The solution to help improve and manage these issues is to allocate resources, for example to graduates, or recently hired staff.

These are inexpensive professionals that want to go into app sec, or just want the job to get a foot in the door at the company. It's one of those easy, win-win situations which will allow them to learn massively, meet a lot of key people and really get their heads around what is going on.

This will mean that the developers can actually spend time fixing the issues instead of maintaining JIRA.

The maintenance of issues is critical for the JIRA RISK workflow to work, because one of its key properties is that it is up to date and that it behaves as a 'source of truth'.

It is key that risks are accepted, followed up on, and issues never moved into the dev's backlog (where they will be lost forever).

We can't have security RISKs in backlog; either issues are being fixed or they are being accepted.

2.1.17 Risk Dashboards and emails

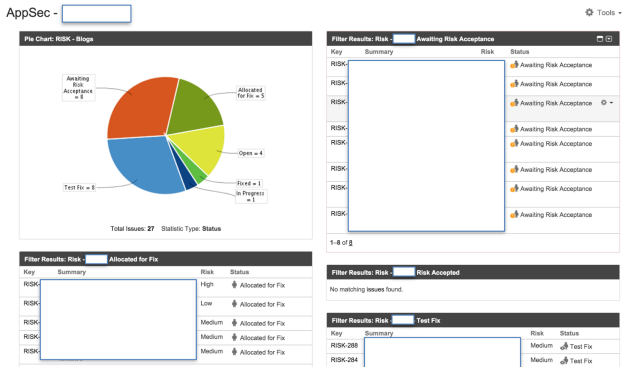
It is critical that you create a suite of management dashboards that map the existing security metrics and the status of RISK tickets:

- *Open, In Progress*
- *Awaiting Risk Acceptance, Risk Accepted*
- *Risk Approved, Risk not Approved, Risk Expired*
- *Allocated for Fix, Fixing, Test Fix*
- *Fixed*

Visualising this data makes it real and creates powerful dashboards. These need to be provided to all players in the software development lifecycle.

You should create a sequence of dashboards that starts with the developer (mapping the issues that he is responsible for), then his technical architect, then the business owner, the product owner … all the way to the CTO and CISO.

Each dashboard makes it clear which risks they are responsible for, and the current status of application security for those particular applications/projects.



Jira Dashboard

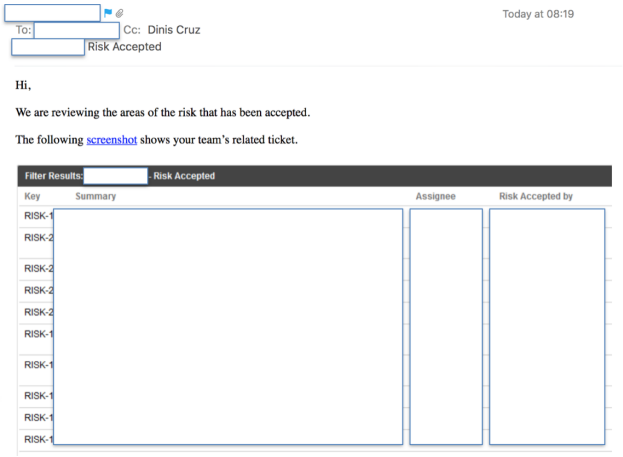
To reenforce ownership and make sure the issues/risks don't 'disappear', the solution is to use either the full dashboard, or specific tables/graphs, to create emails that are sent regularly to their respective owner.

The power of risk acceptance is that each layer is accepting risk on behalf of the layer above. Which means that the higher one is the company structure, the more risks are allocated and accepted (all the way to the CISO and CTO).

The CISO plays a big part in this workflow, since it is his job to approve all 'Risk Accepted' issues (or raise exceptions that need to approved by higher management).

This workflow drives a large number of security activities because each player is motivated to to act on his best interest (which is to reduce the number of allocated Risk Accepted items)

The idea is to social-engineer the management layer into asking the developers to do the non-function-requirements tasks that they know are important, but are seldom given the time to do.



Emails to management

2.2 For Developers

2.2.1 Backlog Pit of Despair

In lots of development teams, especially in agile teams, the backlog has become a huge black hole of issues and features that get thrown into it and disappear. It is a mixed bag of things we might want to do in the future, so we store them in the backlog.

The job of the product backlog is to represent all the ideas that anyone in the application food chain has had about the product, the customer, and the sales team. The fact that these ideas are in the backlog means they aren't priority tasks, but are still important enough that they are captured. Moving something into the backlog in this way, and identifying it as a future task, is a business decision.

However, you cannot use the backlog for non-functional requirements, especially the ones that have security implications. You have to have a separate project to hold and track those tickets, such as a Jira or a GitHub project.

Security issues or refactoring needs, still exist, regardless of whether the product owner wants to pursue them, whether they are a priority, or whether customers are asking for them.

Security and quality issues should either be in a fixed state, or in a risk acceptance state.

The difference is that quality and security tickets represents reality, whereas the backlog represents the ideas that could be developed in the future. That is why they have very different properties, and why you shouldn't have quality and security tickets in the backlog *Pit of Despair* [5].

2.2.2 Developer Teams Need Budgets

Business needs to trust developer teams.

Business needs to trust that developers want to do their best for their projects, and for their company.

If business doesn't learn to trust its developer teams, problems will emerge, productivity will be affected and quality/security will suffer.

A great way to show trust is to **give the developer team a budget**, and with it the power to spend money on things that will benefit the team.

This could include perks for developers such as conference attendance, buying books, or buying things which are normally a struggle to obtain. It is often the case that companies will purchase items for a team when requested, but first the team must struggle to overcome the company's apathy to investing in the developer team. The balance of power never lies with the team, and that imbalance makes it hard to ask for items to be purchased. Inconsistencies are also a problem: sometimes it can be easier to ask for Â£5,000, or for Â£50,000, than it is to ask for Â£50.

[5] Princess Bride - Pit of Despair, https://www.youtube.com/watch?v=mBaDcOBoHFk

Companies need to treat developer teams as the adults they are, and they need to trust them. My experience in all aspects of organisations is that it is difficult to spend money.

When you spend money, especially in an open-ended way, your expenditure is recorded and it becomes official. If your investment doesn't yield good results for the company, you will be held accountable.

Therefore, allocating a budget to the developer teams will keep the teams honest, and will direct their focus to productive investments. Purchases could range from buying some tools for the developers, to buying a trip, or even to outsourcing some work to a freelancer.

Spending on the operational expenses for the team will yield benefits both to developer teams and to business.

2.2.3 Developers Should be Able to Fire Their Managers

Many problems developer teams deal with arise from the inverted power structure of their working environment. The idea persists that the person managing the developers is the one who is ultimately in charge, responsible, and accountable.

That idea is wrong, because sometimes the person best-equipped to make the key technological decisions, and the difficult decisions, is the developer, who works hands-on, writing and reading the code to make sure that everything is correct.

A benefit of the 'Accept Risk' workflow, is that it pushes the responsibility to the ones that really matter. I've seen cases when upper-layers of management realise that they are not the ones that should be accepting that particular risk, since they are not the ones deciding on it. In theses cases usualy the decision comes down to the developers, who should use the opportunity to gain a bigger mandate to make the best decisions for the project.

Sometimes, a perverse situation occurs where the managers are no longer coding. They may have been promoted in the past because they used to be great programmers, or for other reasons, but now they are out of touch with programming and they no longer understand how it works.

Their job is to make the developers more productive. They work in customer liaison, they manage the team and its results, they organise, review, handle business requirements and expectations, and make sure everything runs smoothly. That is the job of the manager, and that manager also acts as the voice of the developer team.

This situation promotes inefficiencies and makes the managers more difficult to work with. They don't want to share information, but they do want to take ownership of developers' work or ideas that they didn't have themselves. This environment gets very political very fast, and productivity is effected.

The manager I describe above should ideally be defending the developer team, and should act like an agent for that team. Logically, a developer, or a group of developers, should therefore have the power to nominate, appoint, and sack the manager if necessary.

The developers should hold the balance of power.

Developers should also be able to take decisions on pay, perks, and budgets. Business should treat the developer teams as the grownups they are, because the developer teams are ultimately accountable for what is created within the company.

2.2.4 Every Bug is an Opportunity

The power of having a very high degree of code coverage (97%+) is that you have a system where making changes is easy.

The tests are easy to fix, and you don't have an asymmetric code fixing problem, where a small change of code gives you a nightmare of test changes, or vice versa.

Instead, you get a very interesting flow where every bug, every security issue, or every code change is an opportunity to check the validity of your tests. **Every time you make a code change**, **you want the tests to break**. In fact you should worry if the tests don't break when you make code changes.

You should also worry if the tests break at a different location than expected. This means that the change you made is not having the desired effect. Either your tests are wrong, or you don't have the right coverage, because you expected the test to break in one location but it broke somewhere else, maybe even on a completely different test that happened to pass that particular path.

To enforce the quality of your test, especially when you have a high degree of coverage, use every single bug fix, and every single code change as opportunities to confirm that your tests are still effective.

Every code change gives the opportunity to make sure that the understanding of what should change is what did change. Every bug allows you to ask the questions, *"Can I replicate this bug? Can I correctly identify all that is needed to replicate the creation, or the expectation of this bug?"*

2.2.5 Make managers accountable

Managers need to be held accountable for the decisions they make, especially when those decisions have implications for quality and security. There are several benefits to increased accountability. Organisations will be encouraged to accept risk, and, before they click on the 'Accept Risk' button, managers will be compelled to 'read' the ticket information. However, the introduction and development of a more accountable management structure is a long game, to be played out over many months, or even years. Sometimes, it requires a change in management to create the right dynamic environment.

Usually, a new manager will not be too enthusiastic when he sees the risks that are currently accepted by a business and are now his responsibility. He knows he has a limited window, of a few months perhaps, where he can blame his predecessor for any problems. After that time has passed, he 'owns' the risk, and has accepted it himself.

This workflow creates historical records of decisions, which supports and reinforces a culture of accountability.

In agile environments, the Product Owner is responsible for making the right decisions.

2.2.6 Risk lifecycle

Here are the multiple stages of Risks

```
digraph {
  size ="3.4"
  node [shape=underline]

  "Risk"                   -> "Test (reproduces issue)" [label = "write test"]  \

  "Test (reproduces issue)" -> "Risk Accepted"          [label = "accepts risk"]
  "Test (reproduces issue)" -> "Fixing"                 [label = "allocated for \
fix"]
  "Test (reproduces issue)" -> "Regression Tests"       [label = "fixed"]
}
```

It is key that a Test that replicates the issue reported is created.

The tests results for each of the bottom layer mean different things:

- **Risk Accepted** : Issues that will NOT be fixed and represent the current accepted Risk profile of the target application or company
- **Fixing**: Issues that are current being activity address (still real Risks)
- **Regression Tests**: Past issues which are now being confirmed that they don't exist anymore. Some of these tests should run against production

2.2.7 Risk profile of Frameworks

Many frameworks work as a wrapper around a 'raw' language and have built in mechanisms to prevent vulnerabilities.

Examples are:

- Rails for Ruby
- Django for Python

- AngularJS, Ember, ReactJS for javascript
- Razor for .Net

Using these frameworks can help less experienced developers and act as a 'secure by default' mechanism.

This means that when using the 'secure defaults' of these frameworks, there will be less Risk tickets created. Ideally Frameworks will make it really easy to write secure code, and really hard (and visible) to write dangerous/insecure code (for example how AngularJS handles raw HTML injection from controllers to views).

This can backfire when Frameworks perform complex and dangerous operations 'automagically'. In those cases it is common for the developers to not really understand what is going on under the hood, and high risk vulnerabilities will be created (sometimes even using the code based on the Framework's own code samples)

- Add case-studies on issues created by 'normal' framework usage:
  - SpringMVC Auto-Binding (also called over-posting)
  - Razor controls vulnerable to XSS on un-expected locations (like label)
  - Ember SafeHtml
  - OpenAM SQL Query Injection on code sample (and see in live app)
  - Android query SQL Injection (on some parameters)

2.2.8 Test lifecycle

- explain test flow from replicating the bug all the way to regression tests

```
digraph {
  size ="3.4"
  node [shape=box]
  "Bug"-> "Test Reproduces Bug"
       -> "Root cause analysis"
       -> "Fix"
       -> "Test is now Regression test"
       -> "Release"
}
```

2.2.9 Creating better briefs

- Developers should use this JIRA workflow to get better briefs and project plans from management (Threat Models are also a key part of this strategy)

- Allow the devs to find the time to do the non-functional requirements that they want to do, but are seldom given it (the time)
- basically this workflow can help developers to solve lots of problems that they have in their own development workflow (and SDL)

2.3 For management

2.3.1 Annual Reports should contain Info sec Part

Annual reports should have an info sec and app sec section, they should list to the business their activities, and they should provide very detailed answers to what is going on.

So, most companies have Intel dashboards of vulnerabilities, basically awareness of what is going on. That data should be published because only then, you can actually make the market work and reward companies. And make them understand that they need to invest and understand the pollution that happens when you have rented projects with crazy deadlines and don't really have the right resources but somehow manage to deliver.

Which in a way is due to some heroic effort from some developers, who sometimes work their ass off to deliver something spectacular, and the management get rewarded by pushing them really hard. And then what you end up having is this highly spectacular vulnerabilities being created and highly spectacular risks that are now being bought by the company. And they don't even realise until it blows up one day.

In Agile environments it's important to also provide relative numbers like:

- Risk issues vs velocity
- Risk issues vs story points

By analysing these numbers over time, the tipping point (where quality/security is no longer in focus) can become painfully clear.

2.3.2 Cloud Security

One way in which cloud security differs from previous generations of security efforts, such as software security and website security, is that in the past, both software and website security were almost business disablers. The more features and the more security people added, the less attractive the product became. There are very few applications and websites that actually make the client need more security in order to do more business, which results in the best return on investment.

What's interesting about cloud security is that it might be one of the times where security is actually a business requirement, because a lack of security would slow down the adoption rate and prevent people from moving into the cloud. Accordingly, people care about cloud security, and they invest in it.

While thinking about this I realized that the problem with security vulnerabilities in the cloud is that any compromise doesn't just affect one company, it affects all the companies hosted in the cloud. It's a much bigger problem than the traditional scenario, where security incidents resulted in one company being affected, and the people in that company worked to resolve the problem. When an

incident happens in the cloud, the companies or assets affected are not under the control of their owners. The people hosting them must now manage all these external parties, who don't have any control over their data, but who can't work because their service is down or compromised. The problem is horizontal; a Company A-driven attack will affect Companies B, C, D, E, F, all the way to X, Y, Z. As a result, it's a much tougher problem to solve.

While catastrophic failures are tolerated in normal websites and applications, cloud-based worlds are much less tolerant. A catastrophic failure, where everything fails or all the data is compromised or removed, means the potential loss of that business. That hasn't happened yet, but surely it will happen. Cloud service companies will then have to show that they care more about security than the people who own the data.

Cloud provides care more than you

One of the concepts that Bruce Schneier talked about at the OWASP IBWAS conference in Spain is that a cloud service actually cares more about the security of their customers than the customer does. This makes sense since their risk to is enormous.

In the future, cloud companies will be required to demonstrate this important concept. They will have to be able to say, "Look, I have better systems in place than you, so you can trust me with your data. I can manage more data for you than you can". This will be akin to the regulatory compliance that handles data in the outside world.

One way to do this is with publishing of RISK data and dashboards (see OWASP Security Labelling System Project[6]).

You can imagine a credit card company wanting or needing to demonstrate this. But you can also see the value of it to the medical industry, or any kind of industry that holds personal information. If a company can't provide this type of service itself it must outsource the service, but to be able to do that the security industry must become more transparent. We need a lot more maturity in our industry, because companies need to be able to show that they have that kind of security controls. It is not sufficient to be declared compliant by somebody who just goes for the lowest common denominator, gets paid, and tells the company it's compliant.

Genuinely enhanced visibility of what's going on will allow people to measure what's happening, and then make the big decisions based on that information. The proof of the pudding will be not how many vulnerabilities the cloud companies have, but how they recover from incidents.

The better a company is able to sustain an attack, the better that company can protect data. A company who says, "I received x, y, z number of attacks and I was able to sustain them and protect them this way" is more trustworthy than the company that is completely opaque. The key to making this work is to create either technology, or standards of process that actually increase the visibility of what is going on.

[6] https://www.owasp.org/index.php/OWASP_Security_Labeling_System_Project

2.3.3 Code Confidence Index

Most teams don't have confidence in their own code, in the code that they use, in the third parties, or the soup of dependencies that they have on the application. This is a problem, because the less confidence you have in your code, the less likely you are to want to make changes to that code. The more you hesitate to touch it, the slower your changes, your re-factoring, and your securing of the code will be.

To address this issue, we need to find ways to measure the confidence of code, in a kind of **Code Confidence Index** (COI).

If we can identify the factors that allow us to measure code confidence, we would be able to see which teams are confident in their own or other code, and which teams aren't.

Ultimately, the logic should be that the teams with high levels of *Code Confidence* are the teams who make will be making better software. Their re-factoring is better, and they ship faster.

2.3.4 Feedback loops are key

An error often made is not spending enough energy and focus in finding the root cause of issues discovered/exploited.

Incidents get identified by operational monitoring or incident response teams, are send to a security department, and after some analysing are directed to development teams as tickets. The development teams have no knowing of the original incident and cannot put the request in the right perspective. This can lead to sub optimal fixes with undesired side-effects.

It is beneficial to include development teams in the root cause analysis from the start to make sure the best solution can be identified.

2.3.5 Getting Assurance and Trust from Application Security Tests

When you write an application security test, you ask a question. Sometimes the tests you do don't work, but the tests that fail are as important as the tests that succeed. Firstly, they tell you that something isn't there today so you can check it for the future. Secondly, they tell you the coverage of what you do.

These tests must pass, because they confirm that something is impossible. If you do a SQL injection test, in a particular page or field, or if you do an authorization test, and it doesn't work, you must capture that.

If you try something, and a particular vulnerability or exploit isn't working, the fact that it doesn't work is a feature. The fact that it isn't exploitable today is something that you want to keep testing. Your test confirms that you are not vulnerable to that and that is a powerful piece of information.

You should have tests for both the things you know are problems, and the things you know are not problems. Such tests give you confidence that you have tested your application for certain things.

Be sure that you have enough coverage of those tests. You also want to relate this to code coverage and to functionality, because you want to make sure that there is direct alignment between what is possible on the application and what is invoked by the test they should match.

The objective is to have much a stronger assurance of what is happening in the application, and to detect future vulnerabilities (created in locations that were not vulnerable at the time of the last security assessment)

2.3.6 I don’t know the security status of a website

Lack of data in making decisions about application security.

Recently, I looked at a very interesting company that provides debug-card for kids, which allows them (the kinds) to get a card which the parents can will control their budget online. There is even a way to invest in the company online via a crowdfunding scheme.

I looked at this company as a knowledgeable person, able to process security information and highly technical information about the application security of any particular web service. But I was not able to make any informed security decision about whether or not this service is okay or safe for my kids. I couldn’t understand the company’s level of security because they don’t have to publish it and, therefore, I donâ€™t have access to that data.

As a result, I have to take everything on the company’s website at face value. And because there is no requirement to publish any real information about their product, the information given is shaped by the companyâ€™s marketing strategy. I have no objective way of measuring whether the company has credit capability, if there are any issues I should know about, or if my kids’ data is going to be verified and protected.

This means that my friends who recommended that service to me are even worse off than I am. They are not knowledgeable users and they can only rely on the limited information given on the company’s website.

If there are three or four competing services at any moment in time, they will not be able to compete on the security of their product. If a company only invests in security in case security becomes a problem, or causes embarrassment in the future, then that is not good enough. It is like saying, “Oh, let’s not pollute our environment because we might get caught”.

In business today, security issues are directly correlated to quality issues. Application security could be used to gain a good understanding of what is going on in the company, and whether it is a good company to invest in, or a good company to use as a consumer.

This approach could scale.If I found problems, or if data was open, I could publish my analysis, others could consume it, and this would result in a much more peer-reviewed workflow for companies.

This reflects my first point: if I canâ€™t understand a company’s level of security because they don’t have to publish it, this should change. And if it does, it will change the market.

Having a responsible disclosure program or public bug bounty program is also as strong quality and security indicator.

In fact, a company that doesn't have an public bug bounty program is basically advertising to the world the fact that they don't have an AppSec team.

2.3.7 Insurance Driven Security

- insurance companies are starting to operate in the AppSec work
- they will need objective way to measure RISK
- workflows like this one could provide that

2.3.8 Lack of action is a risk

- lack of time to perform 'root cause analysis' or lack of understanding of what cause the problem is a valid risk in itself

2.3.9 Measuring companies' AppSec

Reviewing a company's technology allows us to understand its future. As more and more companies depend on their technology, and as more investment is directly connected to software developed by companies, there is a need for companies to provide independent analysis of their technology stacks.

Typically, this is something an analyst does, where they should be measuring both the maturity of the company's software development, and the maturity of its environment. The analyst should use public information to compile their evidence, and they should then use this evidence to support their data. Having good evidence means having explicit data on the record which clearly supports the analysis.

The power of the analyst is that they can take the time, and have the ability to create peer-reviewed, easy-to-measure, easy-to-understand analysis of companies.

Let's look at investments, as it is now easier and easier for people to invest in companies. If your idea is to invest in technology at the earliest stages of a company's growth, you want to be really sure it is going in the right direction. You should be able to get good metrics of what is going on, and you need a good understanding of what is happening inside these companies. Questions to ask include the following:

- Are they using an old technology stack?
- How much are they really investing in it?
- How much are they really busking the whole thing? (because you know that is something that can easily happen, where marketing paints a very different rosy picture of reality)
- Are they having have scalability problems caused by past architectural decisions?
- Are they able to refactor the code and keep it clean/lean and focused on the target domain models?

If you have a company which is growing very fast, it makes a massive difference if the company is on a scalable platform or not.

In the past, companies could afford to let their site 'blow up' and then address problems after they occurred. That is no longer possible, due to the side effects, and particularly the security side effects, of any crashes or instability. Today, the more successful a company becomes, the more attackers will focus on it. If there are serious security vulnerabilities, they will be exploited faster than the company can address them, and this will affect real users and real money. We can see this happening with cars and the IoS (Internet of Things).

There is still a huge amount of cost in moving away from products/technologies, which is I call the 'lock-in index', and is whatever 'locks' the customer to the software.

The idea is that investors and users should have a good measure of how 'locked-in' a company is, and the technological quality of the application/service they are about to invest into.

2.3.10 OWASP Top 10 Risk methodology

- add details about Risk methodology that in included with the OWASP Top 10
- mention other Risks methodologies
  - applying OWASP ASVS allows for a more risk based approach
  - OwaspSAMM and BSIMM can measure the maturity level of an organisation

2.3.11 Relationship with existing standards

- Mention how this maps to PCI DSS , HIPAA and other standards
  - most of companies will fail these standards when their existing 'real' RISKs are correctly identified and mapped (which is why there is a big difference between being 'compliant' and being 'secure')

More and more external regulatory bodies and laws require some level of proof about the implemented security controls.

To prevent unnecessary delays or fines, these requirements need to be embedded in the process in the form of regular scans, embedded controls (e.g. in the default infrastructure), or in user stories.

For example in DevOps environments all requirements related to OS hardening should be present in the default container and not be something that development teams need to think about.

2.3.12 Responsible disclosure

Having a responsible disclosure process and/or participating in programs like HackerOne or Bugcrowd have several advantages:

- it shows a level of confidence to the customers
- it provides additional testing cycles
- it sends a message that they have an AppSec program

2.3.12.1 JIRA workflow for bug bounty's submissions

- add diagram (from XYZ)
  - real world example of using an JIRA workflow to manage submissions received

2.3.13 Third party components and outsourcing

- these dependencies need to be managed
- lots of control lost to 3rd parties
- open source and close source are very similar
  - at least with open source there is the chance to look at the code (doesn't mean that somebody will)
  - companies need developers that are core contributors of FOSS
- outsource code tends to have lots of 'pollution' and RISKs that are not mapped (until it is too late)
  - liability and accountability of issues is a big problem
  - see OWASP Legal project for an idea of AppSec clauses to include in outsourcing contracts.
- copyright infringes will become more of a problem in the future; this can be hard to identify in 3rd party components

2.3.13.1 Tools to help manage dependencies

- OWASP Dependency checker can identify vulnerable components in use
- several commercial tools exist that can identify copyright infringements in components
  - Veracode has it
  - Sourceclear
  - Sonar (check which version)
  - nsp node security
  - ... add more (see 13 tools for checking the security risk of open-source dependencies[7])

2.3.14 Understand Project's Risks

- know what risk game you (manager and devs) are playing
  - what is the worse case scenario for the application
  - what are you defending (and from whom?)
- what are your incident response plan
- take advantage of the cases when you are not under attack (and you have some time to actually address the issues)

[7] http://techbeacon.com/13-tools-checking-security-risk-open-source-dependencies-0

2.3.15 Using logs to detect risks exploitation

- are your logs and dashboards good enough to tell you what is going on?
- you should be able to know when existing and new vulnerabilities are discovered and exploited
- this is a really hard problem that requires serious resources and technology
- specially important for the known risks, can you at least detect them
  - this one of the reasons why a suite of Tests needs to be created that run against live servers
  - not only those Tests will confirm the current status of those issues (across the multiple environment), they will provide the NOC (Network Operations Centre) case studies of that they should be detecting

2.3.15.1 beware of the security myth

- often no special software or expertise is needed to identify basic potential bad behaviour.
- usually companies already have all the tools and technology needed in house (prob is making them work in the company's reality)
- for example if someone is accessing 20 non-existing pages per second for several minutes, it is most likely someone brute-forcing the application.
  - This can easily be identified by monitoring for 404 and 403 errors per IP address over time.

2.3.16 Using Tests to Communicate

Teams should talk to each other using tests. Teams usually have real problems communicating efficiently with each other. You might find an email, a half-baked bug-tracking issue opened, a few paragraphs, a screen shot, and if you are lucky, a recorded screencast movie.

This is a horrible way to communicate. The information isn't reusable, the context isn't immediately clear, you can't scale, you can't expand on the topic, and you can't run it automatically to know if the problem is still there or not. This is a highly inefficient way to communicate.

The best way for teams to communicate is by using Tests.

Both within and across teams; top-down and bottom-up, from managers and testers to teams.

Tests should become the lingua franca of companies. Their main means of communication.

This has tremendous advantages, since in order for it to work, it requires really very good test APIs, and very good execution environments.

One must have an easy-to-develop, easy-to-install, and easy-to-access development environment in place, something that very rarely occurs.

By making tests the principal way teams communicate, you create an environment that not only rewards good APIs, it rewards good solutions for testing the application. Another advantage is that

you can measure how many tests have been written across teams and you can review the quality of the tests.

Not only is this a major advantage to a company, it is also a spectacular way to review changes. If you send a test that says, *"Hey! I would like XYZ to occur"*, or *"Here is a problem"*, then the best way to review the quality of the change is to review the test that was modified in order to solve the problem.

In this scenario, you have a test that reflects the fixed state of a behaviour, and you have a test that communicates the changes you want to make. A review of both tests is often the best way to evaluate if the requested changes have been made and are working as desired.

2.3.17 Who is actually making decisions?

One of the interesting situations that occurs when we play the risk acceptance game at large organisations, is how we are able to find out exactly who is making business and technical decisions.

Initially, when a 'Risk Accepted' request is made, what tends to happen is that the risk is pushed up the management chain, where each player pushes to their boss to accept risk. After all, what is at stake is who will take responsibility for that risk, and in some cases, who might be fired for it.

Eventually there is a moment when a senior director (or even the CTO) resists accepting the risk and pushes it down. What he is saying at that moment in time, is that the decision to accept that particular risk, has to be made by someone else, who has been delegated (officially or implicitly) that responsibility.

In some cases this can now go all the way down to the actual developer/team doing the coding. Paradoxically, usually the developer didn't realised until that moment, that he is one that is actually deciding how and when to fix it (or not).

Developers often hold a huge amount of power, but they just don't know it.

Playing the risk acceptance game, and identifying who is deciding on risk, is a way of empowering the actual decision-maker. Once the person realises their role and power, they can make sure they have realistic time-scales to fix the issue accordingly, and when required, make the case for more time, more resources, or even a more defined role as a decision-maker.

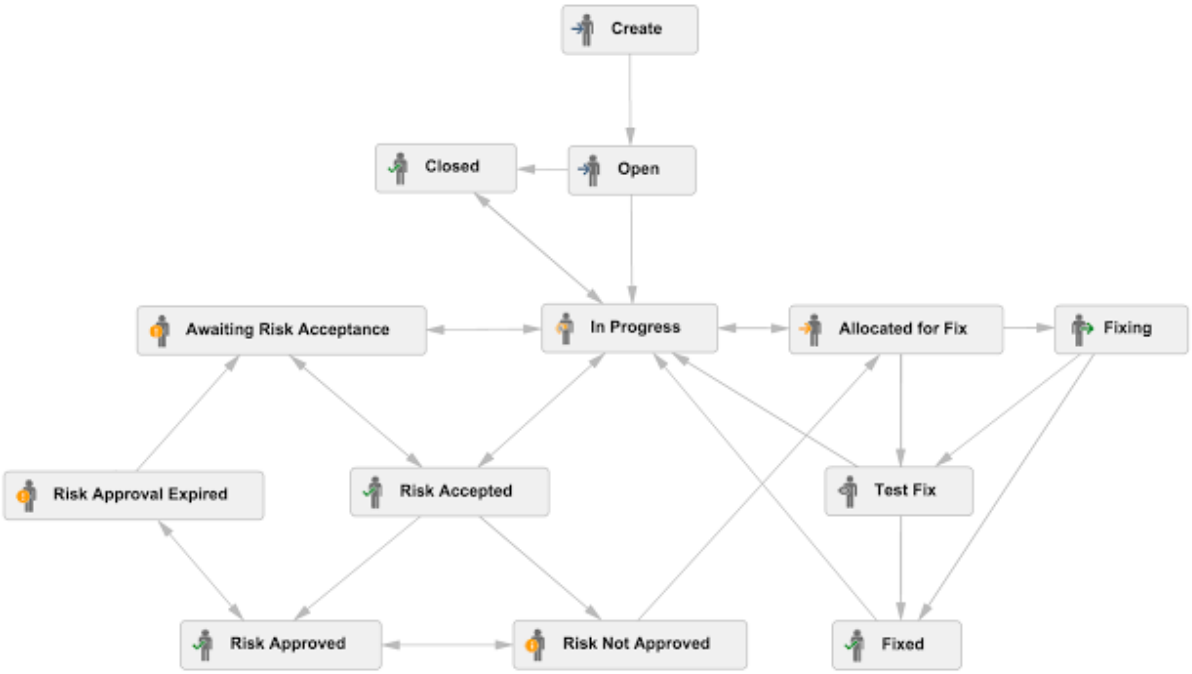
This exercise is very important because, until you discover who clicks on the 'Accept Risk' button, there is little knowledge about who (if anybody at all) is making important decisions.

Risk issues can only be on one of two moving paths: *'Fixing path'* and *'Risk Acceptance path'*. Since deciding not to act on a particular issue is a decision in itself, the Risk workflow makes sure that issues don't 'disappear' in informal conversions, emails or documents.

This is also a good example why *Pollution* is a better analogy than *Technical Debt*. Features that are pushed with un-realistic deadlines or bad briefs will create a higher number of Risk tickets (i.e. pollution) which will have to be accepted by the business owners (who agreed on the development brief and time-scales)

2.4 For security Teams

2.5 JIRA RISK Workflow



Key concepts of this workflow

- All tests should pass all the time
- Tests that check/confirm vulnerabilities should also pass
- The key to make this work is to:
  - Make business owners understand the risks of their decisions (and click on the 'accept risk' button)
- it is not about blame ,it is about assigning responsibility and about empowerment
- note that I'm using JIRA for these examples, but these concepts apply to any Bug tracking system

2.5.1 Capture knowledge when devs look at code

- it is key that when a development is looking at code, he has the easy capability to create tickets for 'things noticed'.
- for example methods that need refactoring, complex logic, weird code, hard to visualize architecture, etc...
- if this info/knowledge is not captured, it will be lost
- note that the developer that notices the issue (and opens the ticket) is not able to do something about it (at that moment in time), since he will be focused on solving another bug.

- this can be facilitated by more junior devs (or graduates/work-experience) which can take the responsibility to open and manage these tickets
  - they could even have a go that addressing them (with the developer being the one that merges the PRs)

2.5.2 Describe Risks as Features rather than as Wishes

When opening up a risk JIRA ticket, it is key to describe the exact behavior of that issue **as a feature**, versus how you would like to see happening (i.e your wish list).

For example:

- instead of saying *'application should encode XYZ value'*, you should say that *'XYZ value is not encoded'*
- don't say an *'application shouldn't be vulnerable to XSS or SQL injection'*, you say *' application is vulnerable to SQL injection'*. In this case SQL Injection is a **feature** of the application, and while the application allows SQL Injection, the application is working as designed (whether that is intended or not, that is a different story :))

When describing vulnerabilities, what we are doing is describing features.

Vulnerability is a feature of an application

If an application has a direct object reference vulnerability (OWASP Top 10), then that is a feature that allows *User A to access user B data* (by design, using capabilities of the application).

For each of these cases, you need to open Risk tickets, since the idea is that those risks represent existing features. Sometimes you open multiple risks for the same issue, allowing technical and business audiences to understand what is going on (SQL Injection doesn't mean a lot to management, but *'Access and modify all customer data'* does).

I remember a funny story where we found SQL injection in a pentest, and when we presented the findings, the business actually said: *'... well, that is not a critical issue, we have good backups, so that SQL injection is not that dangerous'*. When we asked *'what about if we can dropped all tables?'* , they said *'ok, but we can recover from that very fast, so no problem.'*

We would argue *'well ... we can modify data'* and they will go, *'well ... we have read-only access and we can protect it from there.'*. But then, when we showed them that *'we were able to log in as any user with a typical SQL Injection payload of: or 1=1*' that really connected the dots and they said *'... ahh, yeah ... we will fix that ASAP'*

The reason that example clicked, was because we showed them how to bypass the business logic of the application using the SQL Injection '**feature**'. They were ok with data corruption or content changes (sort off), the problem was bypassing the application's business logic and break their non-repudiation[8] capabilities (i.e. losing the ability to understand what a user actually did on the site).

[8] https://en.wikipedia.org/wiki/Non-repudiation

2.5.3 Git for security

- Why Git and DVCS (Distributed Version Control Systems (check name)) are so important for Quality and Security
- Why migrating to git is a good idea
- Analogy with Docker

How git workflows can be used for security

- add explation

git and svn

- The Git Svn story
  - 'Kinda' the same workflows, just different
  - Git developers to decision to make the opposite decision of Svn
    - why? ... not just a dig at Svn. concept represents different approaches and focus
  - speed of git checkouts
  - virtual file systems
  - nightmare of moving files in svn ()
  - the power of one .git folder

Note: find better chapter for this content ('git and svn')

2.5.4 Issue tracking solution

- Why the chosen Issue tracking solution is so important
- Required features for maximum productivity
  - copy and paste of screenshots
  - simple hyperlinking of issues
  - use of markdown or wikitext

2.5.5 Risk accepting threat model

If you are having trouble getting the dev teams to do threat models with you, or for them to spend the time on those threat models, then the solution is to make them accept the risk incurred from having not done a threat model for the application.

The idea is not to be confrontational, but rather to give a very pragmatic and focused statement, which is that this feature doesn't have a threat model.

The idea is that they have to accept that they don't have a threat model. The logic is to create a ticket that says we don't have a threat model and we are going to close it by doing it, or if the developers and their management team don't want to spend the time, you make them accept the risk that you don't have a threat model.

It is very objective and should be very pragmatic, which is quite hard to accept for them, but that's part of the exercise.

2.5.6 Storing risk issues on JIRA

Is a JIRA ticket system a security risk

- Are they zero days?
- Is it increasing the attack surface
- 'If it is on fire, fix it and document it later, if not on fire, document it'
- … better to know about it and to make it explicit that it will be exposed
- … in reality that info already exists on JIRA (maybe not so obvious)
- Monitoring the JIRA access could provide info about an attacker

2.5.7 The smaller the ticket scope the better

For bugs and tasks, the smaller the bug the better

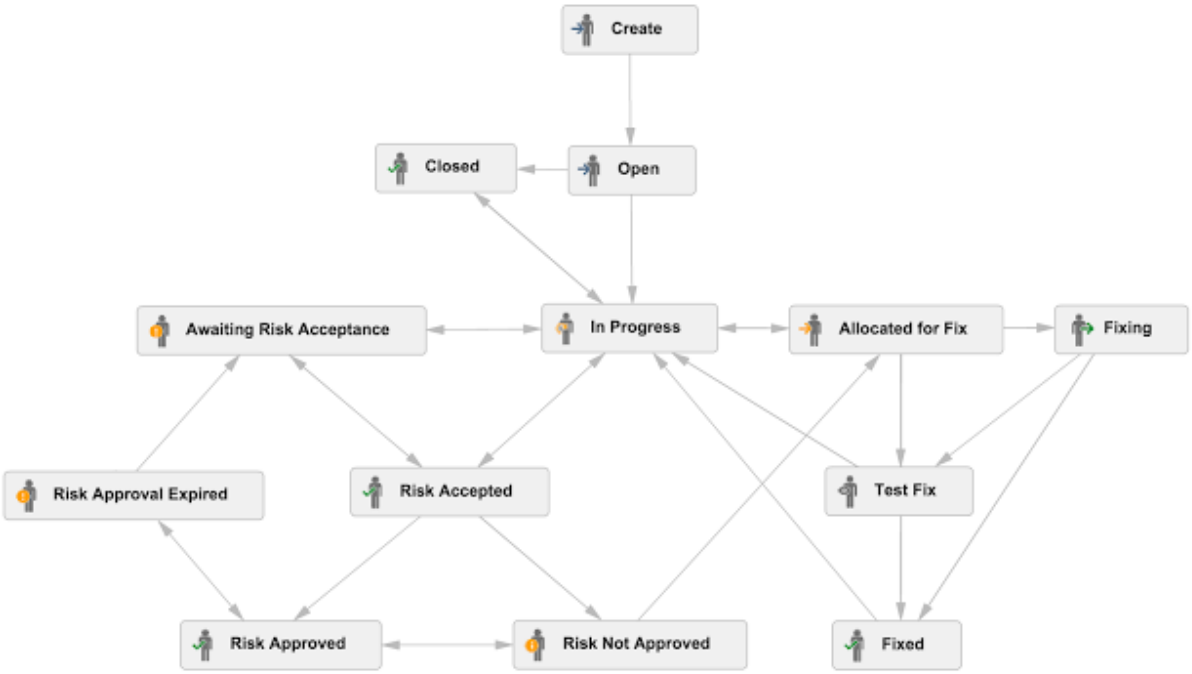
Lots of small bugs and issues is great

- easier to code
- easier to delegate (between developers)
- easier to outsource
- easier to test
- easier to roll back
- easier to merge into upstream or legacy branches
- easier to deploy

It is better to put them is a specially JIRA project(s) which can be focused on Quality (or Non-functional requirements)

- of course that this needs to be rational and kept into context
- you should only create a couple of each instance/pattern (specially when they are not being fixed)
    - in this cases you create a ‘holding ticket’ that will store references to all the individual issues (this is good for systemic vulns)
        * Aggregate issues in Stories

2.6 JIRA RISK Workflow



Key concepts of this workflow

- All tests should pass all the time
- Tests that check/confirm vulnerabilities should also pass
- The key to make this work is to:
  - Make business owners understand the risks of their decisions (and click on the 'accept risk' button)
- it is not about blame ,it is about assigning responsibility and about empowerment
- note that I'm using JIRA for these examples, but these concepts apply to any Bug tracking system

2.7 JIRA Technologies

This section covers the multiple technologies used

2.7.1 Â Confluence

- Atlassian wiki solution
- Tight integration with JIRA
- useful for reporting and Threat modeling
- used to create the materials used in weekly/monthly reports to owners of RISKs (i.e. the ones that have accepted the Risk)

2.7.2 JQL Query Language

- how it works
- security implications
- power examples of queries

2.7.3 Jira components

- used to map issues to specific projects
- allow easy filer per project

2.7.4 Copy and paste of images

- only available in the most recent version of JIRA but a feature that makes a massive difference
- the file attachment of images also works quite well

2.7.5 JIRA dashboards

- how to create them
- pie chart and issues list

2.7.6 JIRA Filters

- how to use them
- saving them

2.7.7 JIRA Kanban boards

- really good to track specific projects
- show couple examples
- add how to create on

2.7.8 JIRA Labels

- show examples
- how to use them
- common workflow
- using them to map OWASP Top 10 issues

2.7.9 JIRA workflows

- explain what they are what makes them powerful
- key components of it
- a good workflow tells a story and guides the user down specific paths

Tidy up your diagrams

It is important that your diagrams have layouts that look good and make sense.

The Diagram UI allows a bit of flexibility on where the Stages appear, so make use it and create nice diagrams

- show where they can be seen in the UI
- makes big difference when they are easy to read and understand
- I hide the labels since I find that they are harder to position and don't provide that much more information
- add example of two diagrams diagrams. Same content: one messy and one clean
  - note how the clean one is much easier to read

Naming of states

- This is very important since they need to convey the desired action
- names have to be short or they don't look good in the UI

Other workflows

Add screenshots for: - bug bounty - development - other ...

A simple workflow

- add example of a simple workflow

2.8 Security Champions

- The Security Champions concept[9]
- What is an Security Champion and what do they do?[10]
- Becoming A Security Champion[11]
- Do you have an heartbeat, you qualify![12]
- If you don't have an Security Champion, get a mug[13]
- Make sure your Security Champions are given time[14]
- Making Security Champions AppSec Specialists[15]
- Involvement in senior technical discussions[16]
- Security Champions Don't Take it Personally[17]
- Supported by central AppSec[18]
- Public references to Security Champions teams[19]

2.8.1 AppSec memo from God

Having an Board level mandate is very important since it sends a strong message of AppSec importance.

The best way to provide a mandate to the existing AppSec team is to send a memo to the entire company, providing a vision for AppSec and re-enforcing its board-level visibility.

Sometimes called the 'Memo from God' the most famous one is Bill Gates 'Trustworthy computing'[20] memo from January 2002 (responsible for making Microsoft turn the corner on AppSec)

2.8.1.1 Example of what it could look like

Here is a variation of a memo that I wrote for a CTO (in a project where I was leading the AppSec efforts) which contains the key points to make. Note that the contents of this book are released under an Creative Commons licence (CC BY 3.0), which means you can reuse this text in its entirely on your organisation.

[9] The-security-champions-concept.md
[10] What-is-an-security-champion.md
[11] Becoming-a-security-champion.md
[12] Do-you-have-an-heartbeat-you-qualify.md
[13] If-you-dont-have-an-sc-get-a-mug/If-you-dont-have-an-sc-get-a-mug.md
[14] Make-sure-your-Security-Champions-are-given-time.md
[15] Making-Security-Champions-AppSec-Specialists.md
[16] Involvement-in-senior-technical-discussions.md
[17] Security-champions-dont-take-it-personally.md
[18] Security-champions.md
[19] Public-references-to-Security-Champions-teams/Public-references-to-Security-Champions-teams.md
[20] https://news.microsoft.com/2012/01/11/memo-from-bill-gates

From: CTO (or jointly with CEO)

As you must have noticed from the news, 'cyber security' is becoming an very hot topic, with daily reports of companies being exploited and assets compromised. At XYZ we have been lucky to not have (yet) been the target of such attacks, but as we grow and increase our visibility (and assets) we will become a target.

We are a digital company and everything we do happens via the applications we write and use. Historically the focus of development has been in getting things done as fast as possible, with security being an after thought and down on the 'real' priority list (which is typical of fast growing companies like XYZ)

This is about to change and we will be putting significant efforts in improving the security of all aspects of XYZ operations and development.

In addition to the current network and physical security activities that we already have in place (firewalls, anti-virus, password management, user accounts restrictions, etc…), we will start a new Application Security practice which will focus on the enablement of our developers to write Secure and Resilient code/applications.

The key to write secure code is to embed security into the SDL (Software Development Lifecycle) and to ensure that Application Security is an enabler (i.e. allows development to happen faster and more efficiently).

Usually security is seen as a TAX and always saying NO (which is sometimes the reason why it is avoided/bypassed). For Application Security the analogy I would like you to think of is 'Brakes in Cars' (i.e. technology that allows the car to go faster).

What this means is that for us to do security right, we will need to improve (even more) our current development and deployment practices (aka Continuous Integration/Deployment) so that all/most changes are small, incremental and fully tested.

The test(ing) component is an area where significant efforts will occur, where high-levels of code-coverage and testing are not just important, but a key requirement in order to have Application Security Assurance (i.e. we can't protect something we don't understand, visualise and control)

From a practical point of view (i.e. day to day development) we are kick starting a number of activities which I'm going to outline bellow:

1) Security Champions

Key to making Application Security scale, is the existence of Security Champions in each team.

Security Champions are active members of our development teams and act as their 'voice' on security issues.

We already have a number of Security Champions (see wiki) and if your team doesn't have one, I strongly advise you to step-up and put yourself forward to become one (this will be massive opportunity to learn and to improve your skills)

2) Secure Coding Standards

One of the most important development questions that needs to have a simple/quick answer is *'how do I code xyz securely?'*. Using our existing development stack/tools (and maybe new ones) we aim to create an environment where such guidance is available in the IDE or at the distance of a link.

This kind of guidance only works if it is actively maintained, and I expect everybody to share their knowledge and help in creating highly focused and accurate secure coding guidances (i.e. relevant to XYX coding practices)

3) Application Security Automation

Since we ship code everyday, we need to do security reviews everyday, which means that we need to automate as much as possible the discovery (and even mitigation) of security vulnerabilities.

The key objective is that when (not if) one of you create a security vulnerability (which is as inevitable as bugs) we have systems, technology and workflows in place to detect it before that code hits a live server.

This means that we will be introducing Static (SAST) and Dynamic (DAST) software analysis tools and will be looking to expand our current Unit/Integration/E2E testing to incorporate 'attack patters' and 'architectural checks'

4) Security data classification and Attack Surface

Another area we need to focus is the mapping of the data we use and what are the inputs/outputs of each of of our applications

When looking at the application you are working at the moment (or have worked in the past), I want you to think:

- 'Do I need this data?
- 'What would happen if this data was exposed to the public' (or sold on the Dark Net)
- 'What is the impact to our brand'
- 'Will this impact our parters and suppliers?'
- 'How much money will we lose?'
- 'What happens if this data is modified?'
- 'Do I know my attack surface?'
- 'Do I trust the data that I receive?' (very relevant for 'internal' systems)

Note the reference to 'internal' systems, since Application Security is not only something that applies to our more outward facing code. We need to remember that our attackers are getting more sophisticated and the real dangerous ones will go after our money and assets.

Ironically, the further you go into the network the more the more sophisticated and focused the attacker will be, and what matters is the value and exploitability of the

application's assets (unless the code lives in a completely isolated network and doesn't talk with anything else)

5) JIRA based Risk/Issue acceptance and security visualisation

One of the key challenges when dealing with Application Security are *'understanding what needs to be done?'*, *'what are the risks?'* and *'what should be the priority?'*.

To help answering these questions a new JIRA project (called RISK) has been created and will be used to hold all currently known security issues/vulnerabilities/compromises.

The workflow of these issues is the following:

1. issue is opened (by anybody)
2. issue is reviewed and expanded (for example to provide risk mappings, exploit details, references to similar issues)
3. technical/business owners decide if a) the issue should be fixed asap (one to two weeks) , or b) the issue is NOT going to be fixed (in the short term) and its risk needs to be accepted
4. The issues to the fixed will be:
    - moved into a 'To Fix' JIRA issue stage,
    - linked into issues that are focused in fixing the issue (in the respective repos)
    - closed when there is verification (by Security Champions) that the issue has been fixed
5. The issues NOT to be fixed will be:
    - moved into a 'To Accept Risk' JIRA issue stage
    - assigned to the respective business/technical owner (who will have a button called 'Accept Risk' to click)

The key of this workflow is improve visibility into the real-world of Application Security compromises. As a company our objective is not to create 100% secure applications, but to create applications/code whose risk is aligned with the current threat landscape and business risk appetite

The other major advantage of mapping security issues like this, is that we will be able to have an accurate visualisation of the risk profile of our applications. For example, we will expand our use of tools like ELK to visualise (in quasi-real-time) the exploitation of these 'accepted risk' security issues (and ideally detect similar security vulnerabilities as they are being exploited)

6) Secure architecture, threat models and nonfunctional requirements

Application Security reviews and practices will also raise/highlight a number of 'non functional requirements' or 'technical debt' issues. These will not have the 'the house is on fire' risk profile, but will need development time/focus in order to ensure the quality and resilience of our code base

One practice that we will be introduction is the creation of Threat Models for existing applications and new features. The Security Champions will help in the creation of these

Threat Models, and I expect you all to contribute, since although security focused, they tend to promote a better understand the 'real architecture' (i.e. they will help us to understand better how out technology works and interacts)

7) Hack anything that moves at XYZ

On the topic of security vulnerabilities we need a culture change were we celebrate and reward the discovery of security issues. We need to (collectively) understand that every issue we discover (and mitigate) is less one available to our attackers.

So, you have my permission to hack (responsibly) anything XYZ related (from external websites, to internal applications, to networked resources, to printers, to cars, etc…).

We will be creating an internal reward system and make sure there are some good professional perks for finding and reporting issues.

The word 'responsibly' means that if you find a way to blow up one of our website or access confidential data (which you shouldn't have access to), we expect you to create a 'non destructive' PoC and use test accounts (i.e. not real customer data). Of course that accidents will happen and we will use common sense.

Eventually we will create a public 'bug bounty' program (after we've done a couple of internal rounds), so if you feel that your app will struggle with a public call for *'..please hack XYZ…'*, then you better start finding those issues :)

…in conclusion

Times are changing and XYZ is changing, when faced with a scenario where security will be affected by a feature, we need to chose security (or have a clean understand of the trade-offs).

Security is now a board-level issue and if you feel that a particular area of our coding/technological world is not receiving the focus it requires, then your duty is to escalate it and fight for it (after all, XYZ is your company too)

These changes represent a great opportunity to make our technology stack and code even better and I hope that you are as excited as me in taking XYZ into the next level

2.8.2 Becoming A Security Champion

To become a security champion, the most important property is that you want to be one.

You need a mandate from the business that will give you at least half a day, if not one a full day per week to pick up this role. The business should also provide the means to educate and train yourself and other. Increasing and spreading knowledge will increase awareness and control.

You basically need to be a programmer and understand code, because your job is to actually start looking at your application and understand the security properties of it. You should also know 'the tools of the trade' and how to implement them in the most efficient way. Lastly you must be able to identify useful metrics and instruct how to obtain them.

2.8.3 Collaboration Technologies

Here are some of the technologies that are key in making the workflow between the Security Champions and JIRA workflows to work

- **Email** - good when used in moderation, specially when emails contain links to online resources
- **Mailing lists** - still the best one, scales a lot, easy to filter, easy to reach a wide audience, great to motivate new Security Champions when they see their name on the list, good to allow interest parties (and older non-active Security Champions) to just hang around and stay connected with what is going on
- **JIRA issues** - discussion threads provide a lot of information and details about that specific topic
- **Wiki** - key to capture knowledge in a more structured and long term environment. Be aware the wikis require maintenance and should be curated (so that they keep being relevant and don't fall for the 'tragedy of the commons' problem). Wiki's should use the JIRA issues has evidence for that is being said.
  - **Confluence** - when integrated with JIRA, it creates a very powerful way to create dashboards to present the data stored in the JIRA tickets
- **Video conferences** - tools like Join.me, BlueJeans, Google Hangouts, Skype are great ways to allow remote working and participation possible
- **Slack** - realtime collaboration tools are key to allow questions to be easily asked, and to allow for asynchronous collaboration (and catching up on specific topics)
- **Slack integrations** - very powerful workflows can occur when SDL tools (and CI pipelines) can feed data into specific channels (not only a good way to get a sense of what is going on, a good way to alert for possible issues or attacks). It gets even better when these integrations are interactive:
  - **Hubot(s)** - is a really great example of this (where it can listen to messages posted and respond to them)
- **Log visualisation** - tools like Splunk, ELK or Graphite, when supported by strong dashboards and visualisations, are one of the best ways to present information and collaborate
- *Diagram technologies* Visio has been the gold standard for ages (draw.io is a recent new player), but the problem is their lack of non-human readable data storage format. In order to promote collaboration and to allow for 'revision of what changed since last analysis' (i.e. Diffs), Diagraming technologies that are created from textual descriptions, are much more powerful and useful:
  - for example PlantUML or DOT (Graphwiz) are easier to read (in source format) and can be stored in git (i.e. versioned controlled)

2.8.4 Conference for Security Champions

Every 6 to 12 months, it is good to have a conference exclusively dedicated to Security Champions (specially for companies that multiple locations and where the Security Champions don't meet regularly in person)

- Bring external speakers to present on specific topics
  - if there are already a number of external AppSec consulting companies under contract, the consultants involved in existing projects are perfect candidates to present
    * they can use their own examples and stories
    * make it easier to present internal materials if all participants are under NDA
- never underestimate the power of team collaboration and on getting team members to know each other
- social events are quite important
- the model of OWASP Summit's is also a good one
- Microsoft Blue Hat is a good example of what this could look like (find other examples)

2.8.5 Do you have an heartbeat, you qualify!

- important to understand that AppSec skills are not a key requirement to become an SC
- what is key is to want to do it
- I can make a good developer that is interested and is dedicated, into a good AppSec specialist in 6 months
- if the dev is an expert in AppSec, then he should join the central AppSec team

2.8.6 How to review Applications as an Security Champion

You need to start looking at the application from the point of view of an attacker, because in the beginning thatâ€™s the best way to get your head around it.

You need to start thinking about data inputs, everything that goes into the database, into the application, all the entry points of the application, everything that the attacker could control, and that could be anything from headers to cookies to sockets to anything that actually enters the application.

Authorization is also a great way to take a look. So just looking at how you handle data and how you authorise things is a great way to understand how the application works

- add info about how this is best done using Threat Models and asking the STRIDE questions

2.8.7 Involvement in senior technical discussions

- great opportunity for SCs (and perk)
- once SC program is established it is a problem if the SC are not involved (since it means that security is not being taken into consideration)

2.8.8 Learning-resources.md

Books to learn from

Hacking Exposed, both the normal version and the Hacking Exposed: Web Applications.

The Shellcoder's Handbook is also a great book from an application security point of view, and they will basically walk you through the vulnerabilities.

The Web Application Hackers Handbook provides a good and solid overview of problems in web applications and how to identify them

The OWASP testing guide is also a good resource to start you AppSec skills.

- add more books (with a small description each)

Vulnerable by design applications

Another great resource is the OWASP WebGoat Project which actually just released a new version, and it has a whole bunch of exercises in vulnerabilities so that you can learn how they work and they give you clues for if you get stuck. The first thing is to actually do those exercises and essentially hack into these applications.

- Add more examples of apps, split by technology
  - .Net - WebGoat.Net, HackmeBank
  - Java - WebGoat
  - Node - NodeGoat, JuiceShop
  - Ruby - Railsgoat
  - php - Damn Vulnerable Web Application
  - Android - GoatDroid
  - iOS - iGoat
- OWASP Broken Web Applications Project[21]
- [OWASP Vulnerable Application Directory] (https://www.owasp.org/index.php/OWASP_-Vulnerable_Web_Applications_Directory_Project)

Hack your apps

You can hack anything that moves in your company, because you have an implicit mandate to protect your own company.

Go for your own application, or go for your colleague's application. Sometimes that's a bit easier to digest :)

BugBounties

Also go for things like bug bounties, which are basically companies that give you permission to hack them and find security issues.

- mention programs like https://hackerone.com/

[21] https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project

2.8.9 Make sure your Security Champions are given time

It is very important that security champions are given the time, the focus, the mandate and the information required to do their jobs.

The good news is that now that you have security champions (at least one per team), their work will allow you to see the difference between the multiple teams and the parts of a company who are able to make it work, and those who are struggling make it happen.

The key activity of the security champions is to participate in the security of his project: * review code, * fixing code, * writing tests, * knowing what is going on, * maintaining the JIRA tickets, * creating Threat Models, * basically being involved in the security practices of the teams.

This ultimately leads to better code, better project briefs, up-to-date documentation and tests for the application

Security champions should be able to spend at least one day a week on those activities which, although easy for management to accept, are in fact much harder to put into practice.

In the beginning, Security Champions will barely be able to spend a couple hours a week.

One of the things you want to make sure you look at from a central point of view is exactly who is doing this kind of work, and who is actually able to spend the time doing it.

The good news is these things can be measured and tracked from the point of view of all the teams (using dashboards and graphs from the JIRA Risk Project).

2.8.10 Making Security Champions AppSec Specialists

- once you have SCs you need to grown them
- training is very important

2.8.11 Regular Hackathons

- to be organised by Security Champions
- these are the next level of SCs applications
- used to find issues and look at new features
- every Friday or Monday afternoon (ideally with some beers and pizza)
  - ideally this would happen every week, if not every 2,3 or 4 weeks
  - what is important is to have movement
- fits well into 10% of research time that developers should have
- inviting external persons (like e.g. pen-testers that regularly test the applications) have the benefit of providing new insights
  - this is also one of the best ways to learn

        * try to do something and then see a more experienced and knowledgeable person being able to do it
        * see that is the workflow and strategies used by them (to discover and exploit the issue)
- organising red and blue teams (attacker and defenders) can also be very effective
    - these are much are harder to organise, with a lot more preparation needed

2.8.12 Rewarding Security Champions

- budget to sponsor the best one in last month (or week, if there is a lot of activity)
    - Participation in conferences (ticket + travel expenses)
    - Books
    - Bonus (some companies prefer old fashioned cash prices)
- How to measure who is the best of the month
    - Number of JIRA tickets: opened, fixed, tested
    - Number of Threat Models
    - Highest improvement
    - Innovative research
    - Shipped code (of an module used by multiple teams)
    - Above average documentation (Secure coding standards)
- provide presentation opportunities (for example at the company wide Security Champions conference, or to senior directors (a couple levels above the current developer's position in the org chart))
- basically anything that you can do to a developer that he/she can put on the CV is a good reward
- It is very important to create explicit and open rules about these rewards, since the worse situation is when a particular Security Champions receives an award not because of his achievements but due to other (political or friendship based) decision.

2.8.13 Security Champions activities

What do Security Champions do in they allocated day-to-day time (4h to 8th)

- it is very important that JIRA tickets are use to map, track and allocate these tasks
    - the number of these tickets is what justifies the work the SC does
    - it also provides movement and management information about what is being done
    - it is important to have a good number of actionable (in the 'Allocated for Fix') stage (30 to 50 are a good number, as long as they are realistic and real)
    - you know the model is working when the managers start asking: "Do you have enough time of those SC activities?", "Do you need help from other team members (i.e. more resources)"

    * eventually the SC should also be managing the AppSec tasks that are performed by other team members
- the time allocation is better if done in blocks of 4h (over two days), or 2h every day (at the same time)

2.8.13.1 Other AppSec projects to be involved in

- Map Attack Surface tool
- Web Services Visualisation tool
- Standard Schemas across the company

2.8.14 Security Champions Don't Take it Personally

If you are a Security Champion, don't take it personally if teams aren't listening to you. Don't think that you are the problem, that you aren't good enough or that you are failing to communicate in some way.

In most cases, the problem isn't you. The problem is actually the system; the company isn't structured in a way that allows the security champion's questions to be prioritised and answered. In other cases the Security Champion is not included in security-relevant architectural meetings and decisions.

So, if you find that you are struggling to get traction from a team, the team isn't responding, or it fights you, then drop those requests (as long as the Risk as been accepted). If they treat you as a TAX, as somebody who is giving them work they don't want to do, then also drop it.

In the Risk ticket, explain that you tried to persuade the team to accept the risks of not doing security, and that they are now responsible for their security, because you cannot help them.

In such cases, the problem lies not with the Security Champion, but with the company and the organisation, maybe even sometimes with the team itself. This is why it is important to have success stories you can point to and say, *"Hey! It worked with that team, and it worked with that team. If it doesn't work with this team, then I am not the problem"*.

2.8.15 Supported by central AppSec

They need to be supported by a central AppSec team, but it is key that each team has one. If there are not enough champions they can create a guild or chapter and rotate over teams. It is also important that they are backed in their decisions by corporate security like e.g. the CISO.

Tasks (expand) * Code Reviews and Pen-tests (internally or via external managed services) * AppSec Automation (SAST and other tools) * Secure coding standards * Incident Response * Recruitment and Training

2.8.16 The Security Champions Concept

> *"If everyone is responsible for security, then nobody is"* [22]

[22] a variation of the quote:

- What are Security Champions?
- Why Security Champions
  - challenges to scale and propagate AppSec Knowledge
  - Keep AppSec focus and energy
  - have somebody responsible for AppSec
- What is the target audience of this section
  - *... you want to become an SC ...*
  - *... you want to set you a SC network ...*
- There is an heavy AppSec focus, but this is applicable to all of IT, Dev and Risk management practices
- explain how JIRA Risk Workflow is connected to the Security champions

â€œIf everything is important, then nothing is.â€ from Patrick Lencioni

2.8.17 Threat Modeling workshops

- to happen every week
  - or every other week, alternating with a Security Champions meeting (this way there is always an SC activity happening every week at the same time and place)
- good place to ask questions and to present Threat Models created during the previous week
  - If there are not a lot of materials to present or show, that indicates that the quantify of Threat models being created is quite low (note that the objective is to create Threat Models for new features, which are happening all the time)
- another great learning resource, specially for new SCs (who are still getting their head around the workflow)
- see Threat Models section for more details

2.8.18 Training Security Champions

- Training is key to improve SC skills
- Best training is done on top of languages and frameworks they use
- Wiki pages with links to actual issues and relevant resources make a massive difference
- Using vulnerable by design apps (or older versions of current main apps with known vulnerabilities) are a great way to learn (by exploiting them)
- Writing exploits and finding vulnerabilities is a key step of the required accelerated learning curve

2.8.19 Weekly meetings

- explain why these meetings are so important
  - what happens at one of these meetings
  - what is the normal agenda
  - who turns up
- should happen every week, but a good compromise is for them to occur every other week
- everything shown and discussed at the Security Champions meeting needs to be done via one or more slides (to be added to that week's slide deck)
- these slides are VERY important when creating learning materials
- create wiki pages with a full list of all past SC meetings (each entry is categorised by labels)
  - there are a great way to teach Security Champions and to call their attention to areas to research in their own apps
    * I've seen many cases where Security Champions will see a presentation on a topic/technology relevant to their current domain and think/say "Humm... I think we might be vulnerable to THAT vulnerability"

Initially, it will take considerable effort to generate content for these meetings; to find presenters; and to keep it engaging / interesting.

- think of the money that it costs the company to have all those resources in there. Make it count and don't waste participants time
  - the developers are very busy and if the meetings are not relevant or not interesting they will just don't turn up
- central AppSec team (to help kickstarting the meetings) and to keep the energy level up) must always be looking for topics to present at the next SC meeting. For example:
  - Threat Models
  - AppSec Questions
  - AppSec ideas
  - events from a point of view on an attacker
    * attacks
    * AppSec news,
    * basically any AppSec related topic that has not been presented recently)

2.8.19.1 Contents for weekly meetings

Here are some examples of what to present at these meetings:

- Latest news on AppSec (DDos, exploits, etc..)
- Latest bug-bounty findings and payments (a really good source of real-world examples)
- Issues found and issues fixed (on the SC's application or service)

- Secure coding techniques
- Security tools and technologies (e.g. OWASP ZAP Project, OWASP dependency checker)
- Tools/techniques to improve the Developer's Productivity (e.g. WallabyJS, NCrunch)
- Hack challenges
- Security reviews current in place
- Other OWASP tools and documents (ASVS, OwaspSAMM, AppSensor, Testing Guide)
- Testing techniques, workflows and technologies used (which might be different from the current development stack)

2.8.20 What is an Security Champion and what do they do?

Security Champions are a key element of an AppSec team, since they create an cross-functional team focused on Application Security

What is an Security Champion?

- Security Champions are active members of a team that may help to make decisions about when to engage the Security Team
- Act as the "voice" of security for the given product or team
- Assist in the triage of security bugs for their team or area

What do they do?

- Actively participate in the AppSec JIRA and WIKI
- Collaborate with other security champions
  - Review impact of 'breaking changes' made in other projects
- Attend weekly meetings
- Are the single point of contact for their assigned team
- Ensure that security is not a blocker on active development or reviews
- Assist in making security decisions for their team
  - Low-Moderate security impact
    * Empowered to make decisions
    * Document decisions made in bugs or wiki
  - High-Critical security impact
    * Work with AppSec team on mitigations strategies
- Help with QA and Testing * Write Tests (from Unit Tests to Integration tests) * Help with development of CI (Continuous Integration) environments

2.8.21 To Add

add references to

- Creating a Security Champions Network[23]

2.8.22 Security Champions

- So you want to be an SC?
- … here is how you do it…
- Need to be a developer

2.8.23 Becoming a Security Champion

To become a security champion, the most important property is that you want to be one.

You need a mandate from the business that will give you at least half a day, if not one day.

Basically you need to be a developer, who understands code (it is really hard to be an AppSec SC and not being able to code).

Your job is to actually start looking at your application and understand the security property of it.

You need to start looking at the application from the point of view of an attacker. In the beginning the best way to get your head around it, is basically to try to think about all the entry points of the application (i.e. inputs).

This is everything that the attacker can control: from GET/POST parameters, to WebServices calls, to headers, to cookies, to websockets, to anything that actually enters the application.

After that, authorization is also a great way to take a look.

Basically, start with doing Threat Models :)

Just looking at how you handle data and how you authorize things, is a great way to actually understand how the application works.

2.8.24 Security Champions's Books

Hacking Exposed series. The normal version and the Hacking Exposed Web Applications variation are great books.

The Shellcoderâ€™s Handbook is also a great book from an application security point of view, which basically they will walk you through the vulnerabilities.

[23] https://securingthehuman.sans.org/blog/2015/01/19/creating-a-security-champions-network

2.8.25 OWASP WebGoat

Another great resource is the OWASP WebGoat Project which actually just released a new version, and it basically has a whole bunch of exercises in vulnerabilities so that you can learn how they work (you can get you clues if you are stuck).

The first thing is to actually do those exercises, but also hack in your own applications (it's better if the issues are discovered by you, vs an attacker).

2.8.26 Hack anything that moves

You should hack anything that moves in your company, because you have a mandate almost to protect your own company (in some companies it might be better to get this 'hack everything' authorization via official channels).

In terms of targets, you can go for your own company, for your own application or for your colleagueâ€™s application (which sometimes it's a bit easier to digest).

2.8.27 BugBounties

Also go for bug bounties, which is basically a nice list of interface of companies that give you permission to 'hack/attack' them and find security stuff.

2.8.28 Developers we need YOU in AppSec

- big oportunity for existing developers to move into appsec

2.8.29 Big market demand for AppSec Professionals

At the moment there is a massive demand for AppSec devs. Historically the path info InfoSec via Network Security, which means that the majority of InfoSec professionals cannot move to AppSec, because they can't code professionally (i.e. no real knowledge on how to build, test and ship software/apps)

It is also possible to learn on the job since there is such a shortage, which makes it easier to hire Devs who like AppSec and then train them up

- add stats on salaries for: AppSec and projects that mention OWASP
  - http://blog.diniscruz.com/2009/09/owasp-driven-jobs.html
  - http://www.indeed.com/salary?q1=appsec&l1=

2.8.30 If you don't have an Security Champion get a mug

If your developer team doesn't have an assigned security team champion, get one of these Mugs :)

That 'Security Expert' Mug represents the fact that at the moment when a developer has an Application Security question, he might as well ask the dude on that Mug for help :)

I also like that it re-enforces the idea, that for most developer teams, **just having somebody assigned to application security, is already a massive step forward!!**

Basically we have such a skill shortage in our industry for application security developers that '**if you have a heart-beat you qualify**'

2.8.30.1 How to create the SC Mug

- Get a mug with lots of white space in the front and back
- write ***Security Champion*** at the front in large letters (but not so big that the text can't be read from a distance)
- optionally at the back write: ***It's me or Google or Stack Overflow***

- optionally if you have a company little stuff animal or object, get one of those and put them inside the Mug

Put the mug in the central and visible place of the team. It is important that the Mug is an isolated place and not really 'assigned' to anybody.

In some teams I've seen the ritual that when a Security Champion is appointed, he/she gets the Mug to put on his desk.

2.8.31 Public references to Security Champions teams

Microsoft

The Microsoft Agile SDL describes them as Team Champions

In Simplified Implementation of the Microsoft SDL[24]

> ***Team Champions***. *The team champion roles should be filled by SMEs from the project team. These roles are responsible for the negotiation, acceptance, and tracking of minimum security and privacy requirements and maintaining clear lines of communication with advisors and decision makers during a software development project.* * ***Security Champion/Privacy Champion***. *This individual (or group of individuals) does not have sole responsibility for ensuring that a software release has addressed all security issues, but is responsible for coordinating and tracking security issues for the project. This role also is responsible for reporting status to the security advisor and to other relevant parties (for example, development and test leads) on the project team.* * ***Combination of Roles***. *As with the security and privacy advisor role, the responsibilities vested in the champion role may be combined if an individual with the appropriate skills and experience can be identified.*

In "The Microsoft SDL Process Template – Making Secure Code Easier"[25] Brian Harry blog entry says this about Security Champions

> *With the SDL Process Template, a security owner can easily tackle that initial question of "where do I start"? The Process Guidance page provides a security owner (and the entire team) with a brief overview of the SDL, five steps for Getting Started on an SDL project, and details on customizing the template and extending it for third party security tools. There is even more material supporting SDL implementation and customizing the SDL Process Template in the SharePoint library.*

and

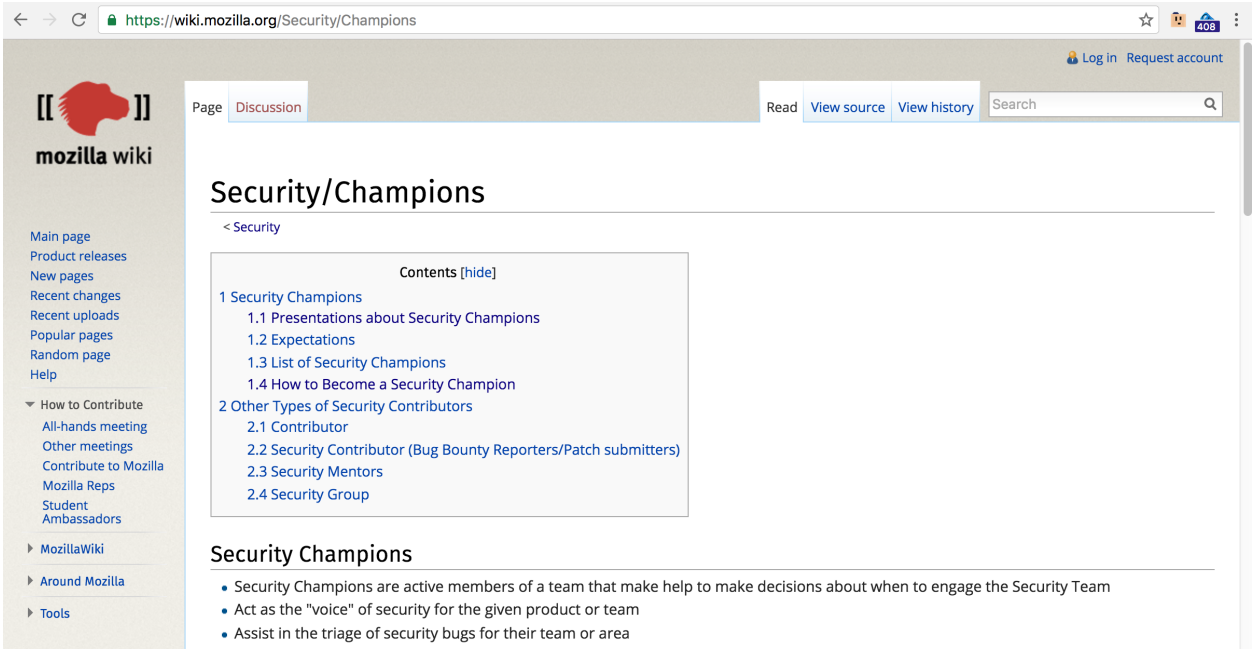
[24] https://www.microsoft.com/en-us/download/details.aspx?id=12379

[25] https://blogs.msdn.microsoft.com/bharry/2009/05/19/the-microsoft-sdl-process-template-making-secure-code-easier/

> *A security owner can accelerate the task of defining security requirements by opening up a query that includes all of the default SDL requirements – ready to triage and assign! There is also a custom work item to add your own requirements or recommendations*

Mozilla

Mozilla has a good pages at Security[26] and Security/Champions[27].



In the SC page they mention other types of Security Contributors:

- Contributor
- Security Contributor (Bug Bounty Reporters/Patch submitters)
- Security Mentors
- Security Group

Unfortunately this program has ended in 2012 following an internal reorganisation[28]

OwaspSAMM

Owasp SAMM[29] (Software Assurance Maturity Model) uses the term **Team Champions**

From Secure_SDLC_Cheat_Sheet[30]

[26] https://wiki.mozilla.org/Security
[27] https://wiki.mozilla.org/Security/Champions
[28] https://wiki.mozilla.org/Security/Meetings/2012-01-25
[29] https://www.owasp.org/index.php/OWASP_SAMM_Project
[30] https://www.owasp.org/index.php/Secure_SDLC_Cheat_Sheet

Step 6 - **Roll out**	Ensure that improvements are available and effectively used within the organization	**Evangelize Improvements** Make the steps and improvements visible for everyone involved by organizing training and communicating. **Measure effectiveness** Measure the adoption and effectiveness of implemented improvements by analyzing usage and impact.		Categorize applications according to their impact on the organization. Focus on high-impact applications. Use team champions to spread new activities throughout the organization

BSIMM

In BSIMM security champions are named ‘satellites’ and described in section 2.3 > The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill. Identifying this group is a step towards creating a social network that speeds the adoption of security into software development.

…**others?**

… add more

2.9 Threat Models

- generates risks
- good way to create highly accurate technical documentation
  - source of truth
- easier to get management to support

2.9.1 Capture the success stories of your threat models

One of the key elements of threat modeling is it's ability to highlight a variety of interesting issues and blind spots, in particular within the architecture of the threat model. One of my favorite moments occurs when the developers and the architects working on a threat model realize something that they hadn't noticed before.

In such cases, sometimes it is the developer who says, "Oh, I never realized that is how it worked!". Other times when the architect says, "Well, this is how app was designed", and the developer responds "Yeah, but that didn't work, so we did it like this."

What is actually happening when such exchanges take place is the mapping of reality, and the creation of a much better understanding of what that reality actually means within the company. Truth is being mapped, and the threat model becomes a single source of truth for the company.

It is very important not only to capture these success stories, but also to advertise and promote them. Promoting them allows you to explain one of the reasons why you want to work in threat modeling; because you want to understand what is going on, and you want to make sure that everybody working on a threat model is on the same page in terms of development, QA, testing, and so on.

2.9.2 Chained threat models

When you create threat models per feature or per component, a key element is to start to chain them, i.e. create the connections between them. If you chain them in a sequence you will get a much better understanding of reality. You will be able to identify uber-vulnerabilities, or uber-threats, that are created by paths that exist from threat model, A to threat model B, to threat model C.

For example, I have seen threat models where one will say, *"Oh, we get data from that over there. We trust their system, and they are supposed to have DOS protections, and they rate limit their requests"*.

However, after doing a threat model of that system, we find that it does not have any DOS protections, even worse, it doesn't do any data validation/sanitisation. This means that the upstream service (which is 'trusted') is just glorified proxy, meaning that for all practices purposes, the 'internal' APIs and endpoints are directly connected to the upstream service callers (which is usually the internet, or other 'glorified proxies' services).

This example illustrates how, when you start chaining threat models, you can identify data that shouldn't be trusted, or data that is controlled by the attacker. Usually the reverse also applies,

where when you go downstream and check their threat models, you will find that they also trust your data and actions far too much.

Of course, the opposite of this scenario could also be true. One of the threat models might say, *"...we have a huge number of issues at this layer"*. However, when you look at the layers above, you find they are doing a good job at validating, authorising and queuing the requests; they are all working to protect the more vulnerable layer, so the risk is low overall.

When you chain a number of threat models, you track them, document them, and you greatly increase your understanding of the threats. You can use this new knowledge in the future to ensure that you don't expose that particular threat into new systems or new features.

2.9.3 Developers need data classification

Every type of data that exists in an organisation, especially the data that is consumed by applications, needs to have a Data Classification mapping.

Developers need to know if a particular piece of data is sensitive, and what value it holds for the business.

A good way to determine the expected level of confidentiality and integrity, is to ask what would happen *'If a particular set of data were to be fully disclosed?'* (for example uploaded to PasteBin) or *'If some of the data was being maliciously modified over a period of months?'*.

These are really hard questions, and only by answering them, the developers (and business owners) can start to understand the value of an particular data set (given-to or generated-by their application).

Developers need to understand what they are dealing with, what is valuable to the business, and what needs to be protected.

See Microsoft's Data Classification Wizard[31] for a good list of data types that exist on large organisations (this will need to be tweaked per application)

> todo: add references to Threat Modeling

2.9.4 Threat Models as better briefs

- diagrams created (DfDs for example) will represent reality much better than existing documentation
- It is key that Threat Models are used as 'sources of truth' (which are then maintained as code/architecture changes)

[31] https://www.microsoft.com/security/data

2.9.5 Threat Model Case Studies

- for each case study, add list of Risks that need to be added
- Examples of Case Studies to add:
  - Source code, Keys and passwords stored in Developer's laptop
  - HTTP to HTTPS transition, lack of HSTS header, insecure cookies
  - Homeopathic Apigee127 web service
  - Smart-carts that control door access in buildings
  - Login brute of accounts
  - 4 version of HTML editing and Rendering
    - raw HTML
    - raw HTML with CSP
    - using Markdown
    - using Markdown with CSP
  - File upload solution vs GitHub fork
  - QA team with bad test environment
  - Insecure APIs with vulnerable by design methods

2.9.6 Threat Model Community

There is currently (late 2016) space within the application security world to develop a community focused on Threat Modeling. Such community would allow the many parties working on Threat Modeling to share information and provide a voice to all different stakeholders.

Questions to be considered by this community include:

- What are common patterns and threats across projects?
- What do developers understand about it?
- How are Threat Models consumed by managers?
- What do we name an issue/threat/risk?
- What schema can be used to store the data?
- How to version control the artifacts created?

These questions are important because they are the ones that really allow us to plan and understand the best way to structure Threat Models.

Open up the models

At the moment, 99.9% of Threat Models exist within companies in proprietary/closed environments. This doesn't mean that these companies don't want to share their models. It may just be that the information isn't in a format that is easy to share.

One of the advantages of approaching this as a community, in an open way, with clear licenses and clear open standards on how to communicate, is that it forces us to solve the problem of separating confidential data from generic public data.

This community effort will also help to resolve the issue of data versioning, which is a very complex problem.

Version the models

Today, versioning (of Threat Models) is either done using the file system (for example appending v1.x to the file name), or even worse, not done at all (note: existing Threat Models applications, desktop or online based, don't have an historical view of data).

This is not an effective way to work, doesn't promote collaboration and doesn't scale.

More importantly, this way of (quasi manual) versioning of Threat Models, prevents us from understanding the evolution of a particular Threat Model.

For example, imagine a Threat Model that starts is small, then grows bigger and then shrinks again, all depending on the desired (or implemented) features. To understand the present and future threats it is important to know the past.

Let's say that you have a particular Threat Model of a particular feature of an application that is reasonably self-contained, or in a fairly good state. However, the addition of a new feature will cause the whole thing to explode. Essentially, what you are now dealing with is a situation where the new feature has either created a number of issues, or it hosts a number of vulnerabilities. These are much easier to visualise in a state where you can actually see the new connections and the impact of the change/feature request (by the business owner).

Reviewing Threat Models is much easier when only looking at what changed since the last version.

Existing efforts

Note: OWASP currently has an active Slack channel[32] and an inactive project[33] on Threat Modeling

2.9.7 Threat Models mapped to Risks

- every risk identified in the Threat Model needs to be opened and tracked in the JIRA Risk project
- using Confluence to host the Threat Model content and have 'live' queries with the relevant risks (makes a massive difference in the maintainability of these Threat Models)
  - When special views are needed, Jira's JQL Queries can be used to create some of the queries

[32] https://owasp.slack.com/archives/threat-modeling

[33] https://www.owasp.org/index.php/OWASP_Threat_Modelling_Project

2.10 Threat Models

- generates risks
- good way to create highly accurate technical documentation
  - source of truth
- easier to get management to support

2.10.1 When to do a threat Model

normal development flow

```
digraph G {
  size= "3.0"
  node [shape=box]
  Idea -> "Project brief"
       -> "Scheduling"
       -> "Development"
       -> "QA"
       -> "Release"
}
```

Proposed development flow

```
digraph G {
  size= "4.5"
  node [shape=box]
  Idea -> "Project brief"
       -> "Scheduling"
       -> "Development"
       -> "QA"
       -> "Release"

"Project brief" -> "Threat Model"

"Threat Model" -> "Option A" -> "Risks"
"Threat Model" -> "Option B" -> "Risks"
"Threat Model" -> "Option C" -> "Risks"
"Risks" -> "To be accepted"
        -> "Scheduling"
"Risks" -> "To check implementation"
        -> "QA"
```

```
19
20  "To check implementation" -> "Pen-test"
21                            -> "Release"
22
23  }
```

3. Appendix

- Appendix A: Creating Workflow in JIRA
- Appendix B: GitHub book workflow

3.1 Appendix A: Creating Workflow in JIRA

This section shows how to create the JIRA workflows without using any JIRA plugins

Key concepts of this workflow

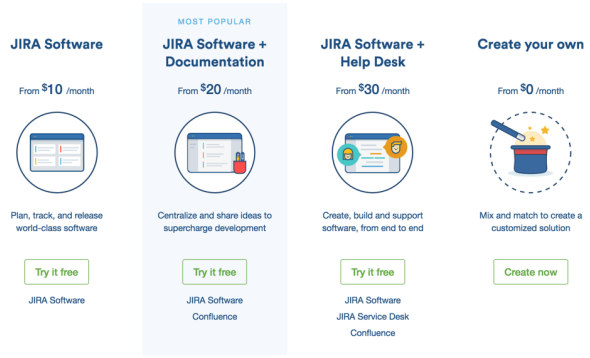
- All tests should pass all the time
- Tests that check/confirm vulnerabilities should also pass
- The key to make this work is to:
  - Make business owners understand the risks of their decisions (and click on the 'accept risk' button)

3.1.1 Creating-a-Jira-project

For these examples we will use the version hosted JIRA cloud called (in Oct 2016) JIRA Software.

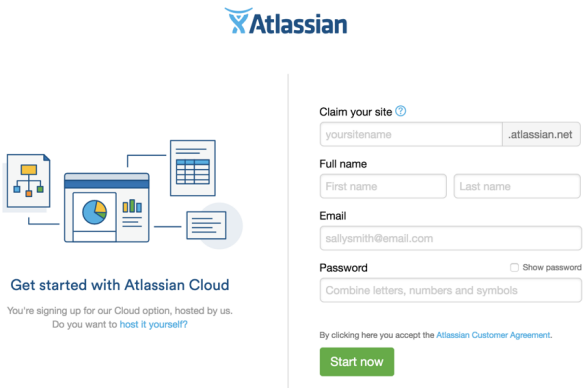
Note that the same workflow can be created on the on-premise versions of JIRA (including the older versions)

If you don't have a JIRA server you can use, then use can create on using the Jira evaluation page[1] and choosing the *JIRA Software* option. I would also add in the *Documentation* (aka Confluence) module since it is a very powerful wiki (which is called *Confluence*)

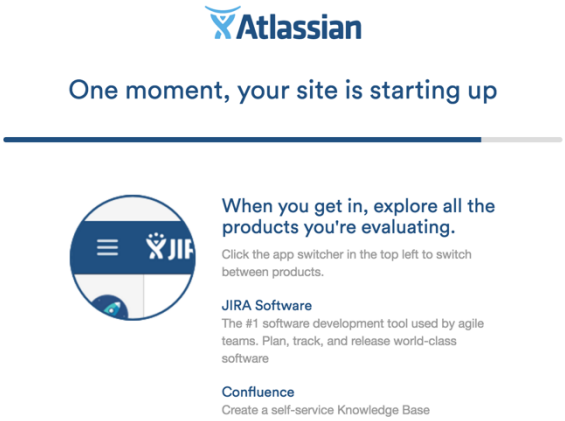


If you don't have an account you will need to create one.

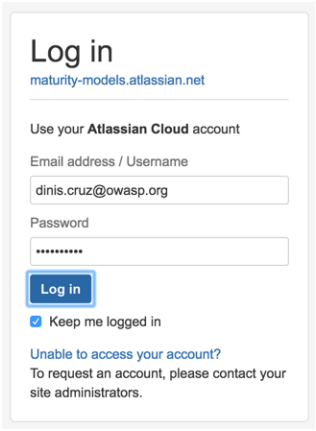
[1] https://www.atlassian.com/software/jira/try



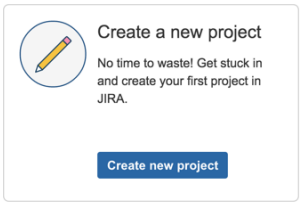
After clicking on *Start now* your cloud instance of JIRA will be created (my bet is that this is a docker container with a dedicated data store for each customer/trial)



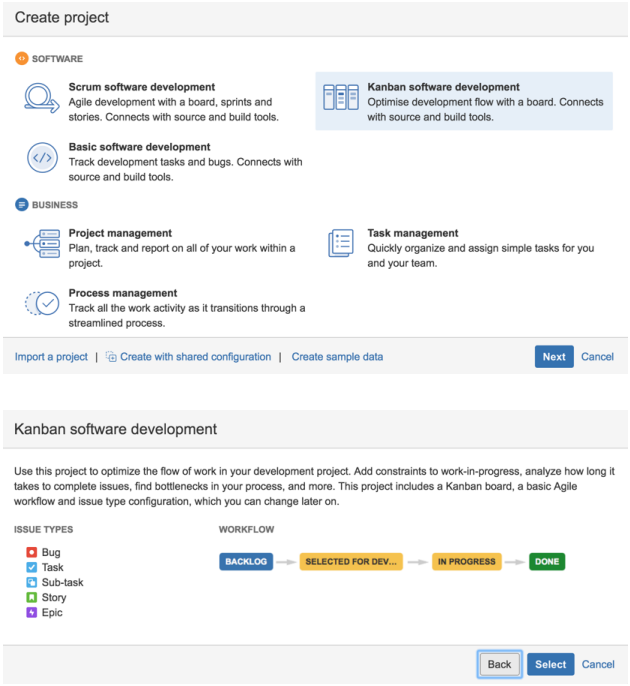
3) login



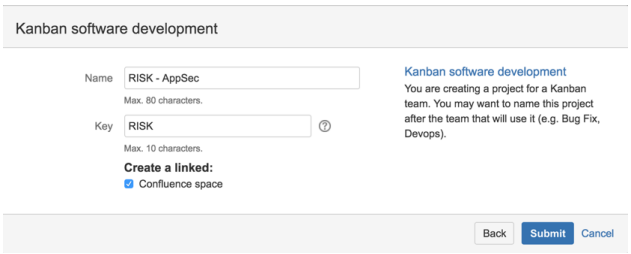
4) create new project



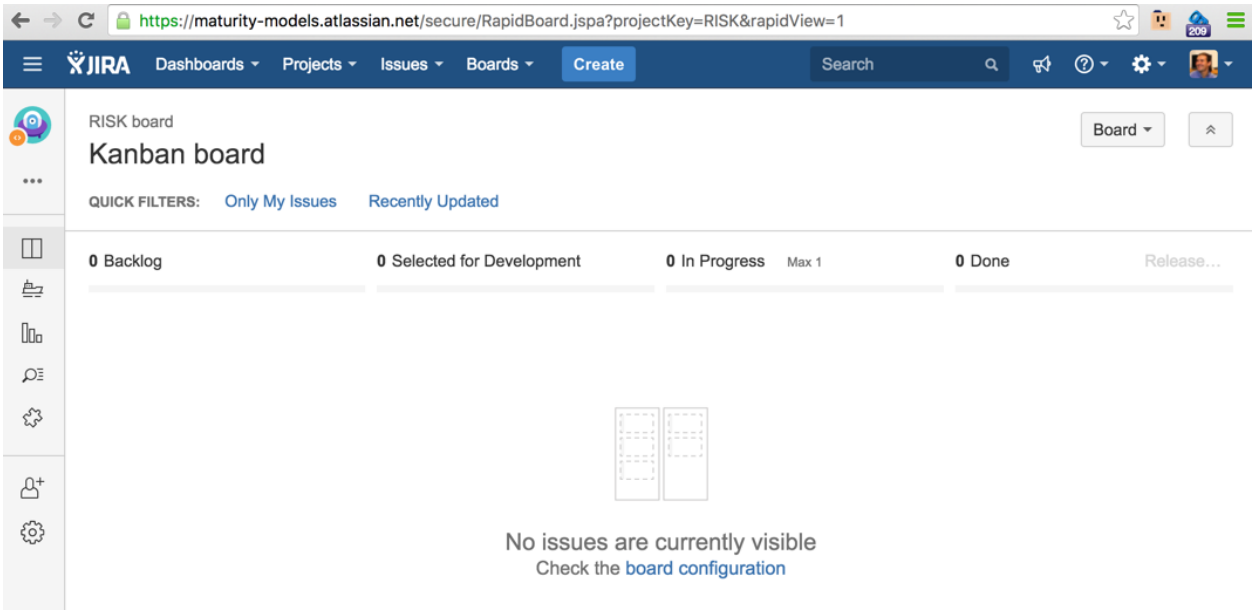
5) choose Kanban Software Development



6) Name it ‘RISK - AppSec’ with the key ‘RISK’,



7) Your new JIRA Project dashboard should open and look something like this

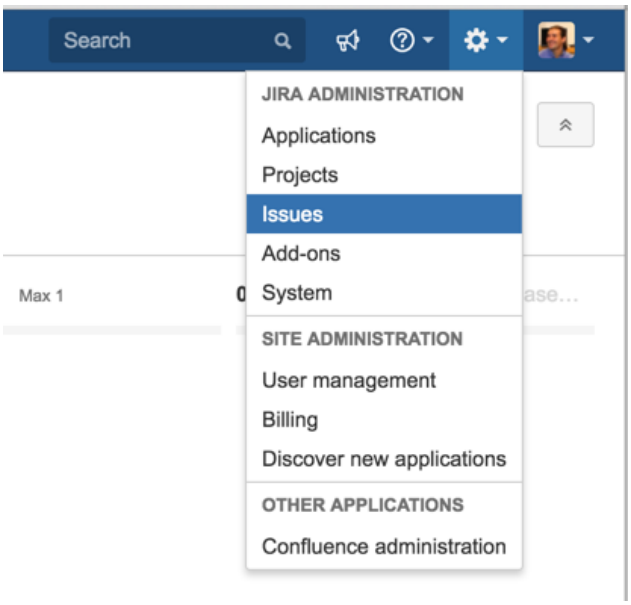


3.1.2 Step-by-step instructions

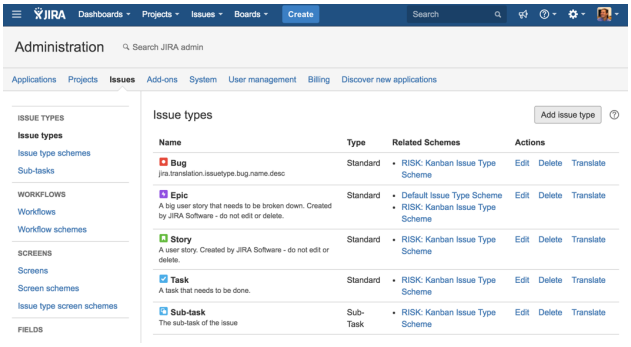
Creating RISK workflow

as seen here http://blog.diniscruz.com/2016/03/updated-jira-risk-workflow-now-with.html

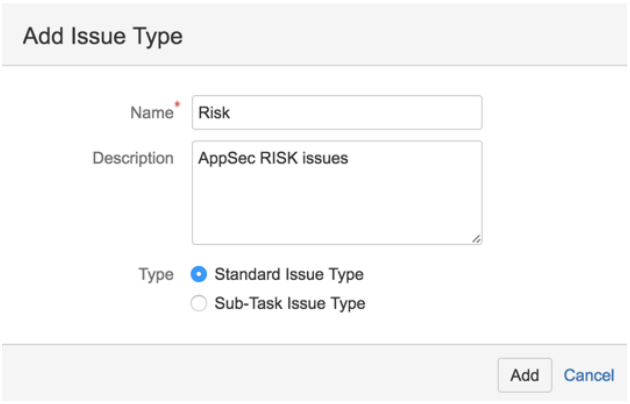
7) Go to JIRA Administration , Issues



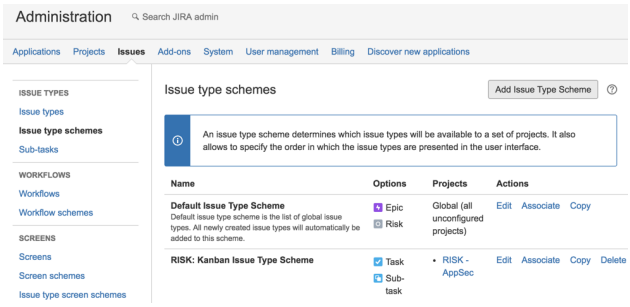
8) Add an issue type



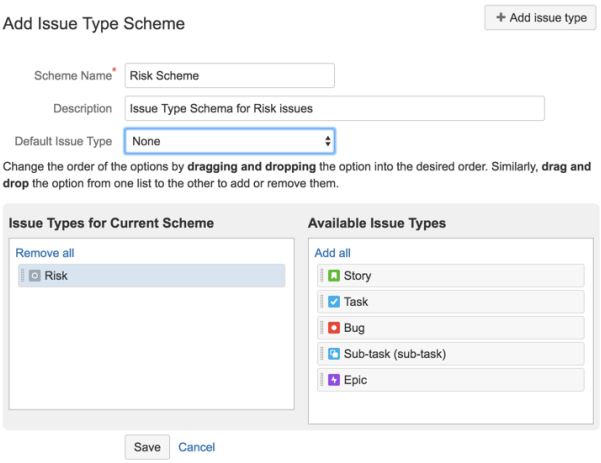
9) call it Risk



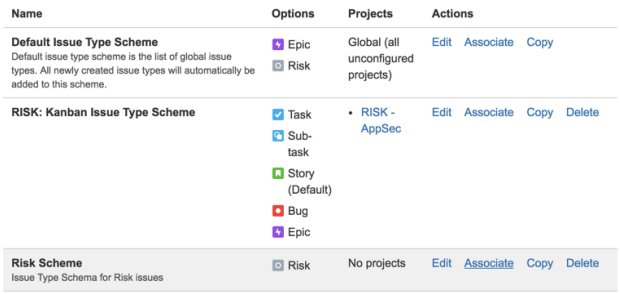
10) Go to Issue type schemes and click on ‘Add Issue Type Scheme’



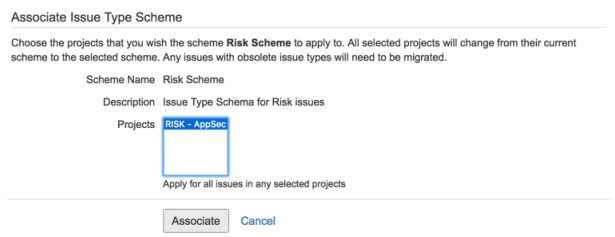
11) Call it Risk Scheme and add the Risk Issue type into to (click Save to continue)

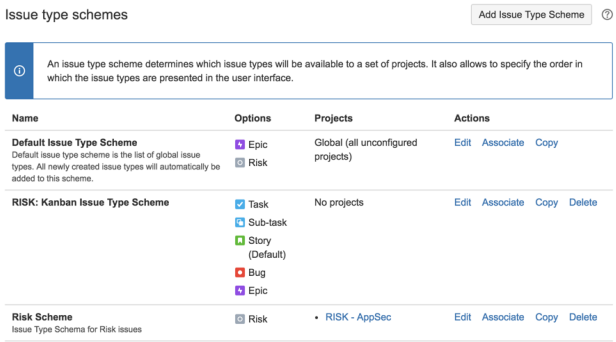


12) Associate that Risk Scheme

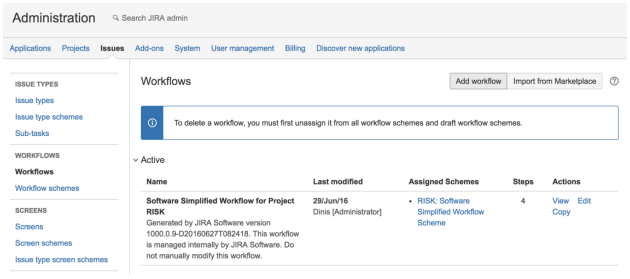


13) To the ‘RISK - AppSec’ project

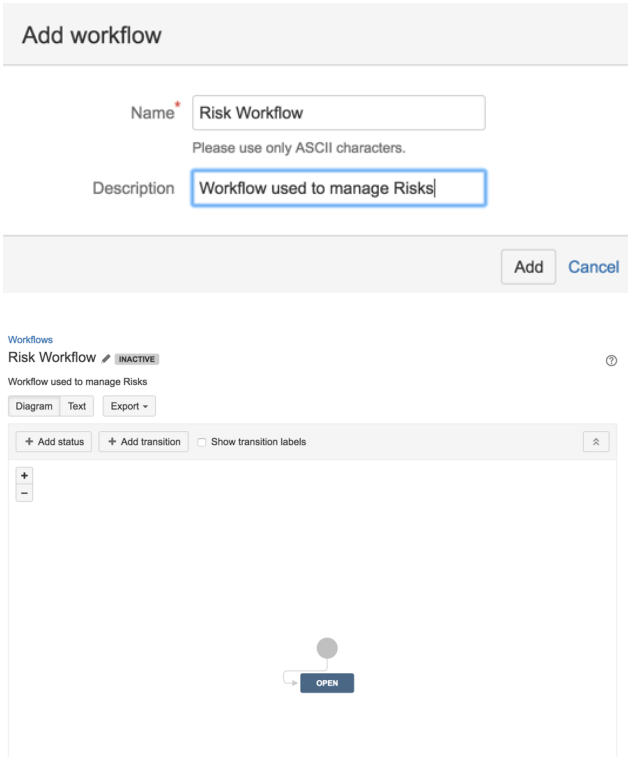




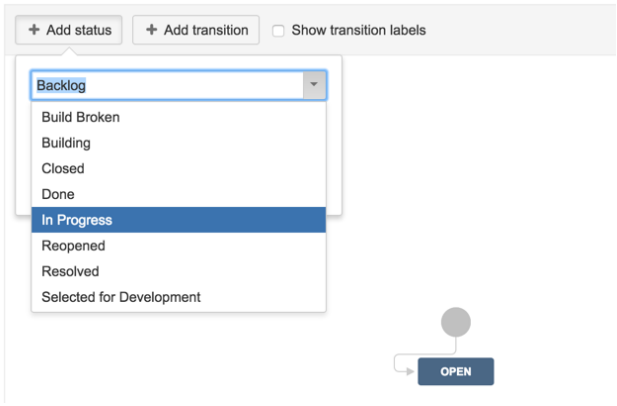
14) Go to Workflows and add new one

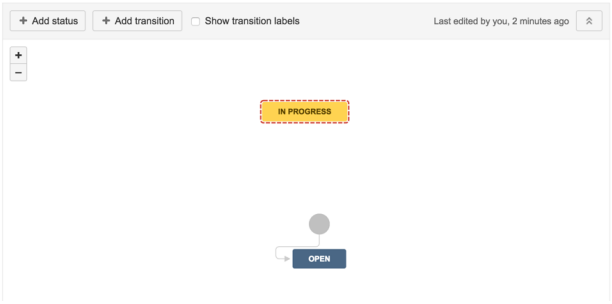


15) call it 'Risk Workflow'

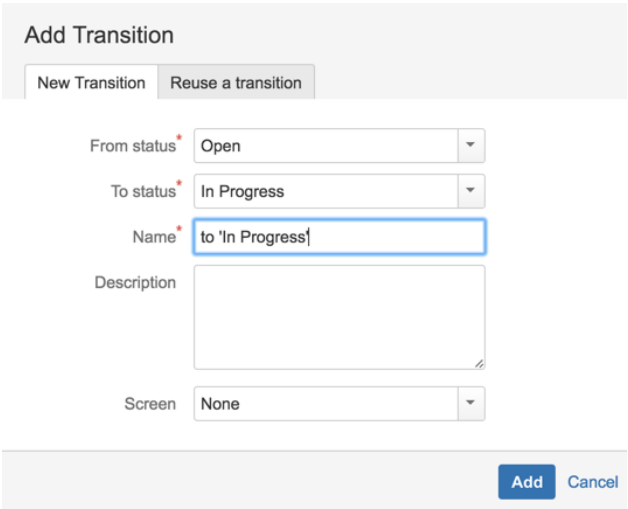


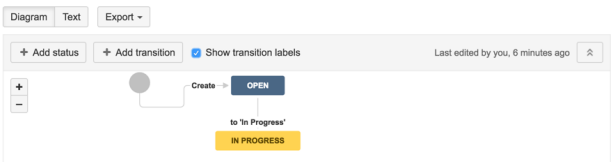
16) Add status 'In Progress



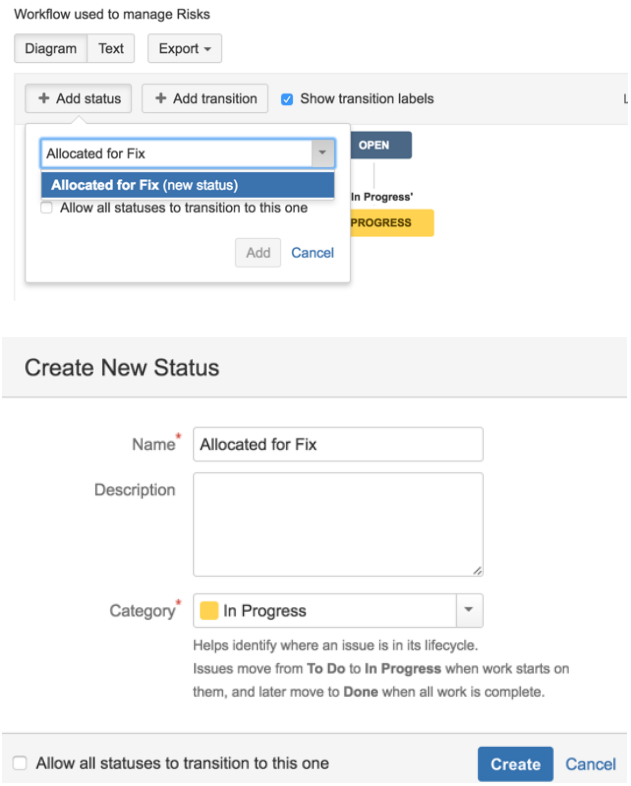


17) Create transition from Open to In Progress

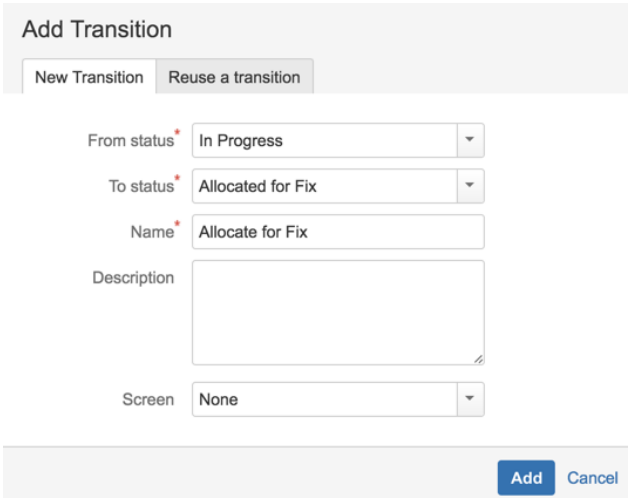




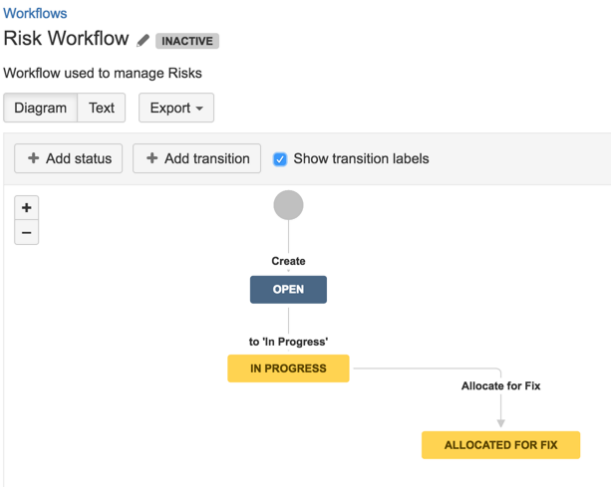
18) Create a new Status called ‘Allocated for Fix’



19) add a transition to 'Allocated for Fix' state

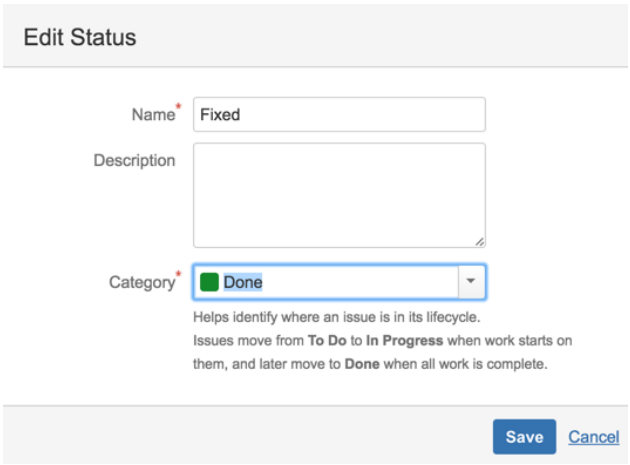


20) how workflow looks like at the moment

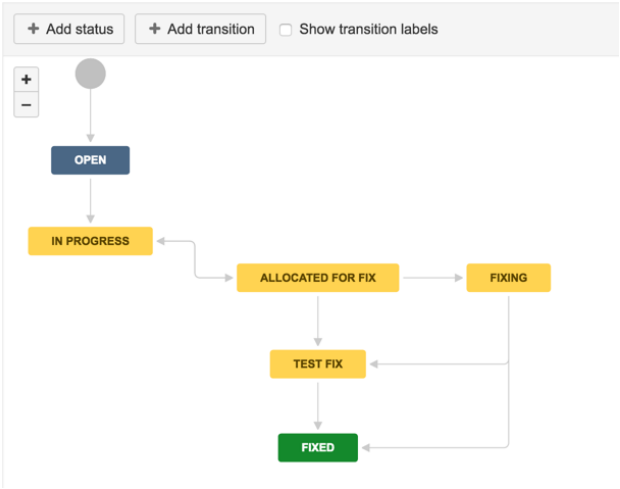


21) Add status: Fixing, Test Fix and Fixed

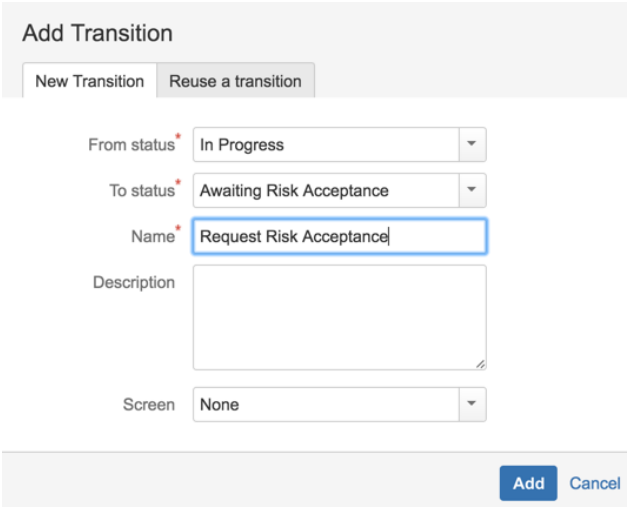
with fixed set to the ‘Done’ Category

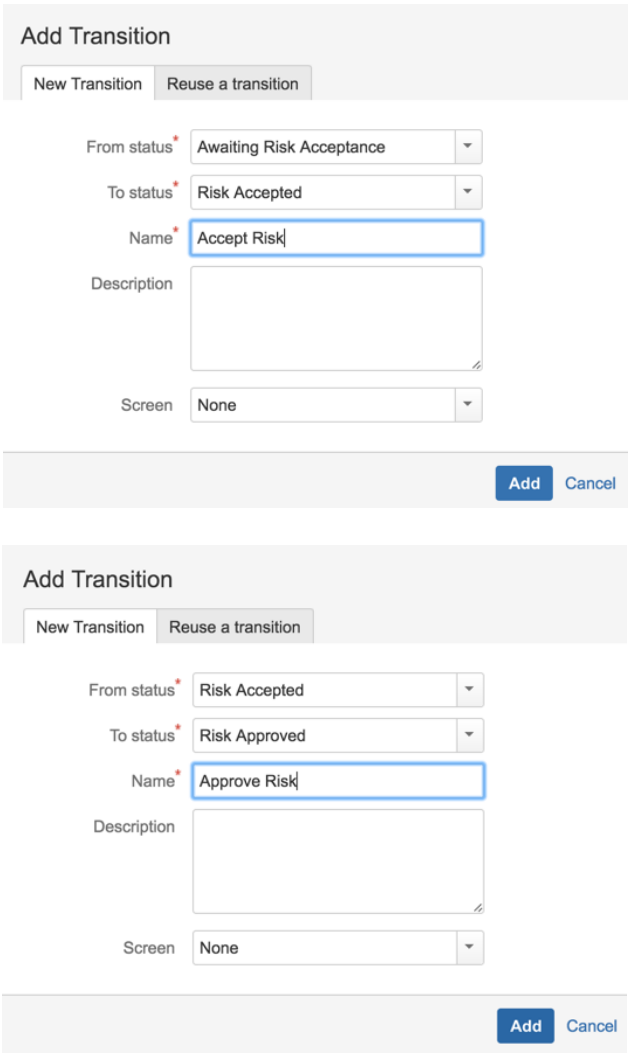


21) add transitions to those status

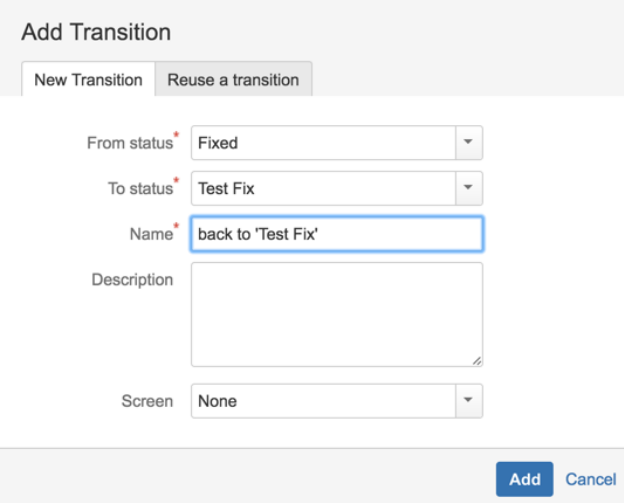


22) Add Status: Closed, ‘Awaiting Risk Acceptance’ , ‘Risk Accepted’, ‘Risk Approved’, ‘Risk Not Approved’, ‘Risk Approval Expired’

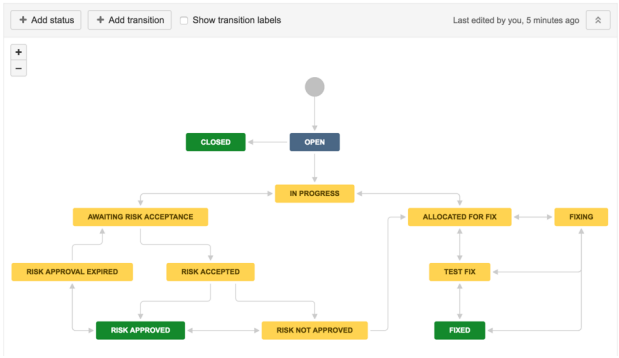




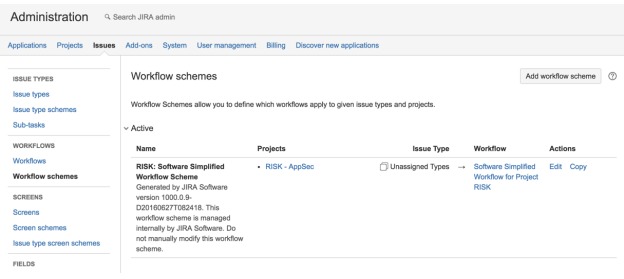
23) Add transitions (including a couple to reverse some of the steps)

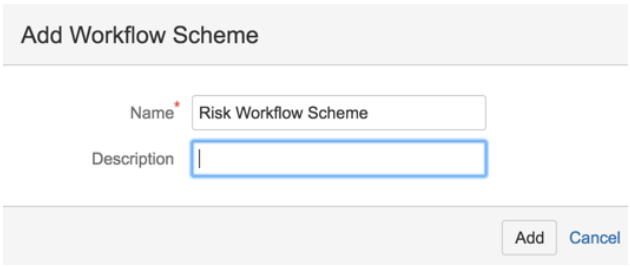


24) Completed workflow should look like this

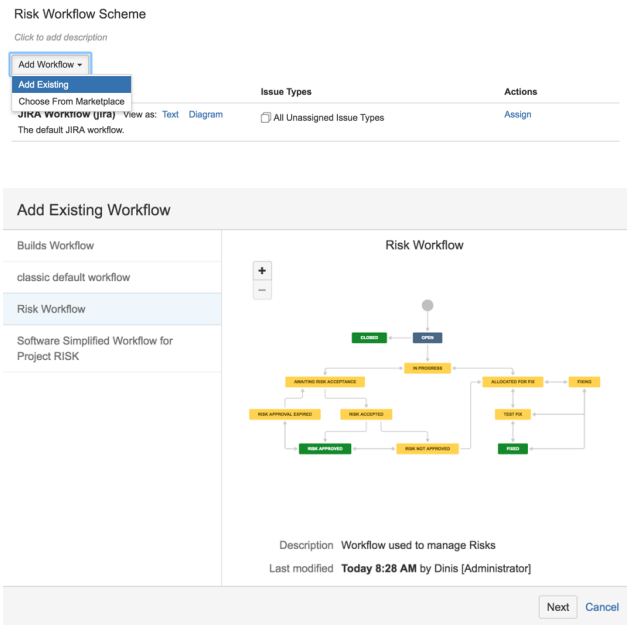


25) go to Workflow Scheme and chose to ‘Add workflow scheme’

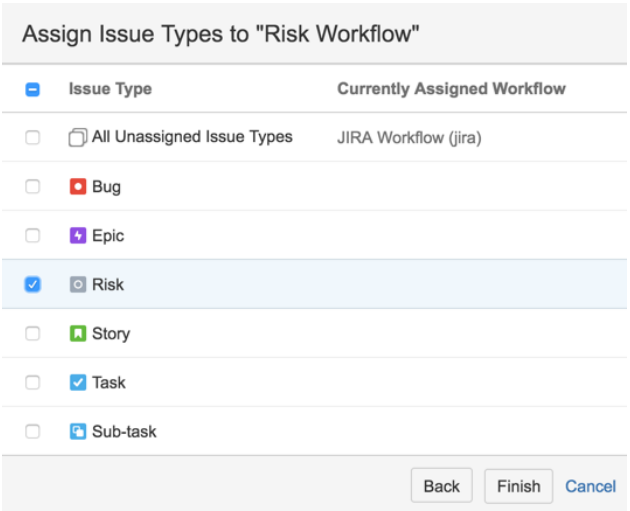


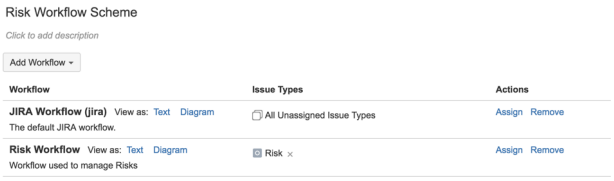


26) Add Existing Workflow

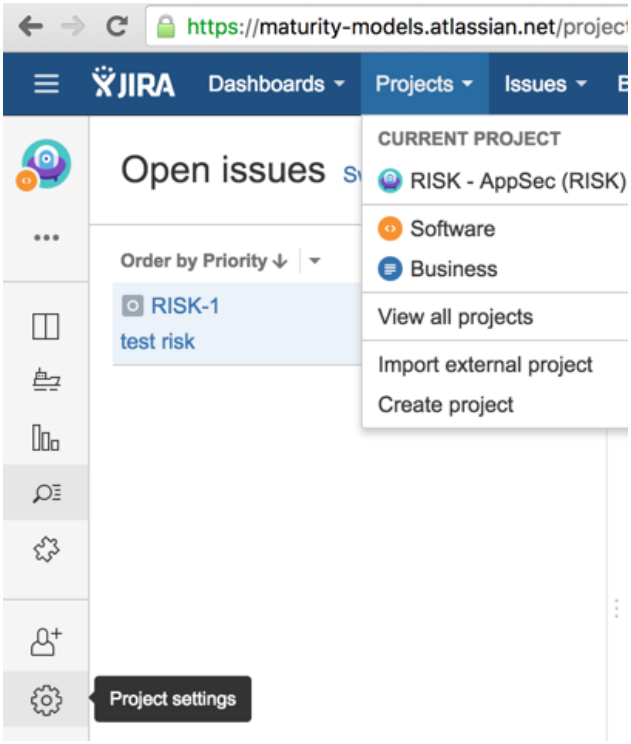


27) Assign Risk Issue type to it

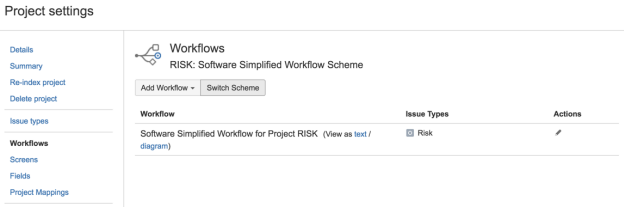




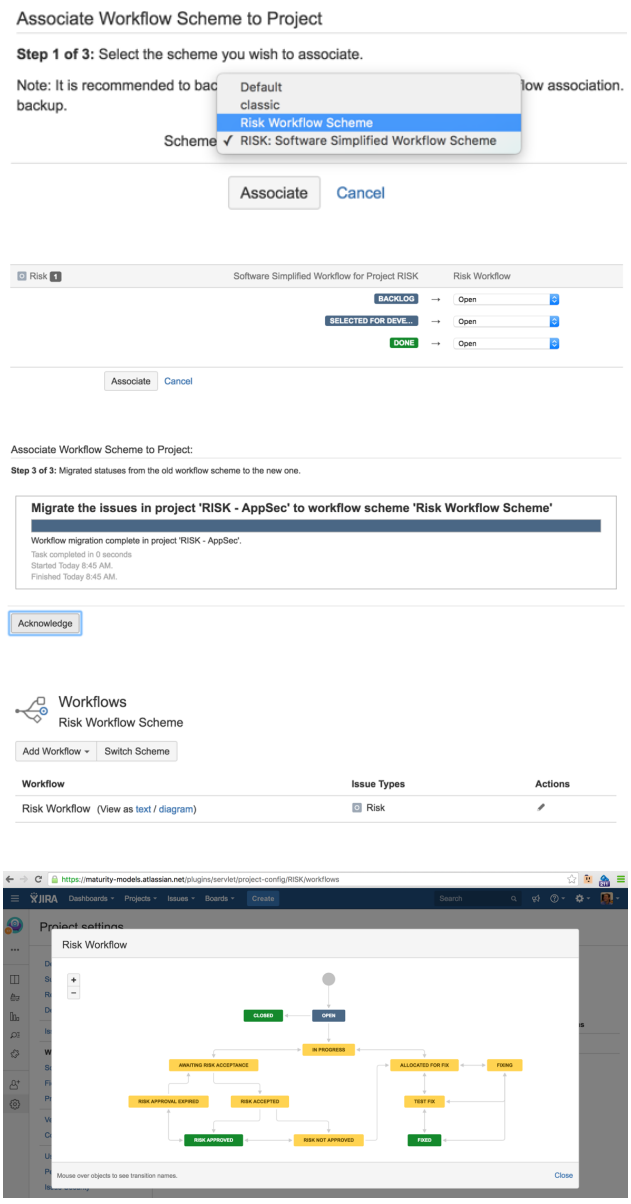
28) exit admin and go to the RISK project's settings



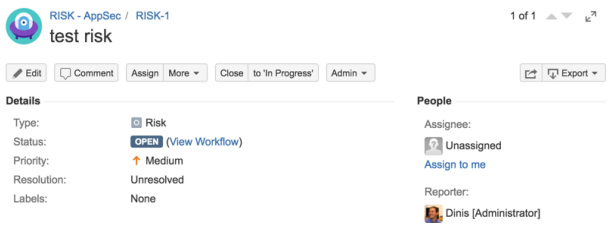
29) in the Workflow page chose to Switch Scheme

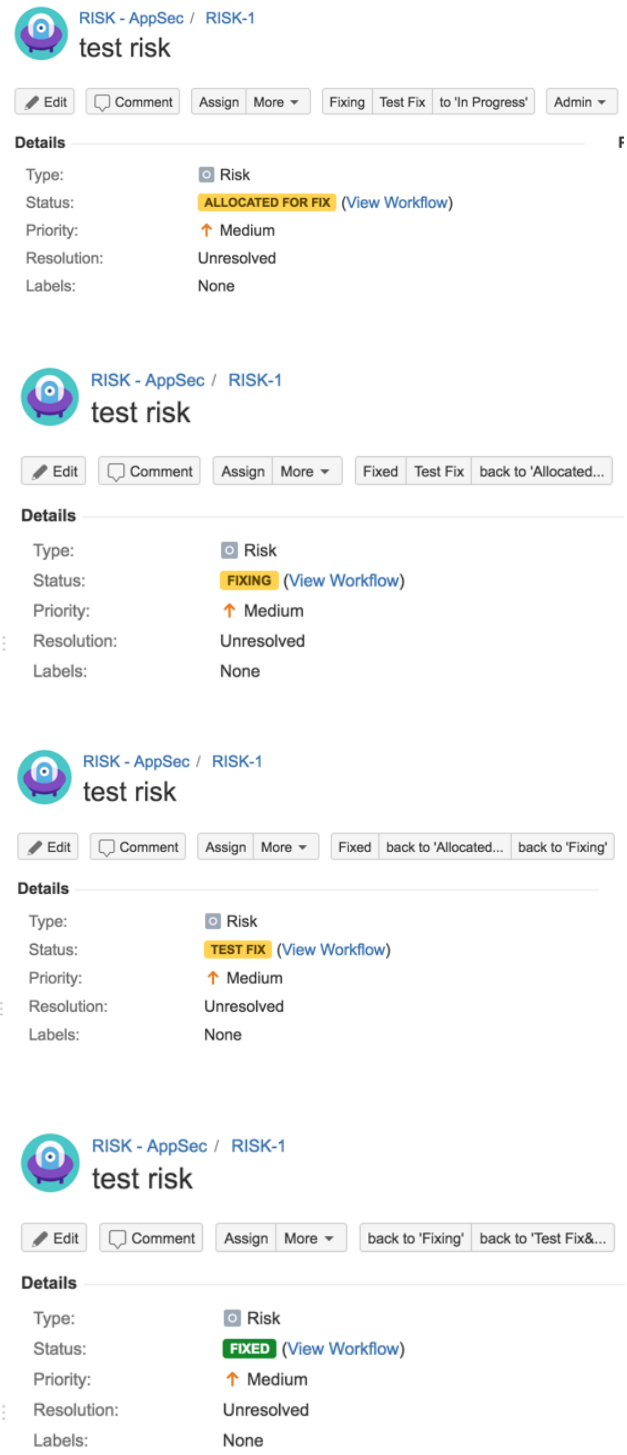


30) Pick the 'Risk Workflow Scheme'

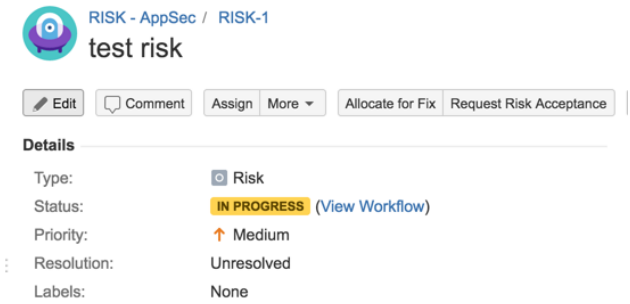


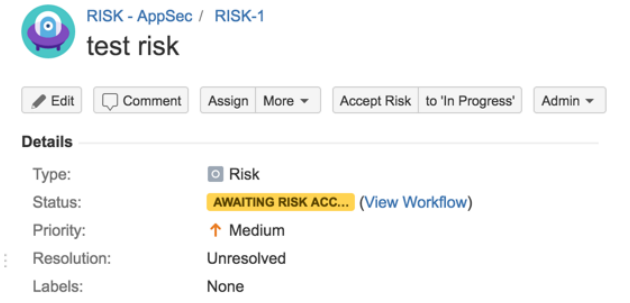
31) Test workflow (fixing path)

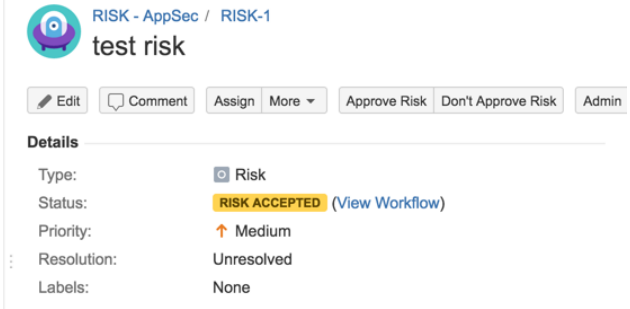


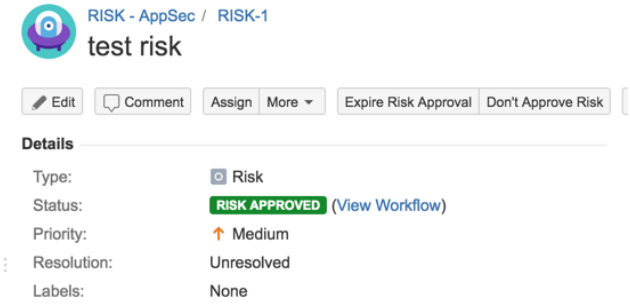


31) Test workflow (Accept Risk)


RISK - AppSec / RISK-1
test risk
Edit
Comment
Assign
More
Allocate for Fix
Request Risk Acceptance
Details
Type:
Risk
Status:
IN PROGRESS
(View Workflow)
Priority:
Medium
Resolution:
Unresolved
Labels:
None


RISK - AppSec / RISK-1
test risk
Edit
Comment
Assign
More
Accept Risk
to 'In Progress'
Admin
Details
Type:
Risk
Status:
AWAITING RISK ACC...
(View Workflow)
Priority:
Medium
Resolution:
Unresolved
Labels:
None


RISK - AppSec / RISK-1
test risk
Edit
Comment
Assign
More
Approve Risk
Don't Approve Risk
Admin
Details
Type:
Risk
Status:
RISK ACCEPTED
(View Workflow)
Priority:
Medium
Resolution:
Unresolved
Labels:
None


RISK - AppSec / RISK-1
test risk
Edit
Comment
Assign
More
Expire Risk Approval
Don't Approve Risk
Details
Type:
Risk
Status:
RISK APPROVED
(View Workflow)
Priority:
Medium
Resolution:
Unresolved
Labels:
None

3.2 Appendix B: GitHub book workflow

- explain the workflow used to create this book
  - audio transcripts and copy editing (and upwork)
  - Pull requests for copy editing
  - Labels for managing tasks and issues
- show how to report a problem with the book or suggest ideas

3.2.1 Book creation workflow

- what are all the actions that occur, from making a code change to having a preview done
- explain two modes (Github online editing and offline editing using Atom editor)

3.2.2 GitHub Leanpub hook

- how it works
- what it does

3.2.3 GitHub online editing

- explain how it works and the workflow used
- mention leanpub service hook and how every content change will eventually result in a new preview

3.2.4 GitHub repos used

- https://github.com/DinisCruz/Book_Jira_Risk_Workflow
  - hold content and raw files
  - better searching since the manuscript files are not there
  - used to create the stand-alone version of the book
- https://github.com/DinisCruz/Book_Jira_Risk_Workflow_Build
  - holds files in Leanpub friendly format
  - hooks into leanpub via GitHub service
    * every commit to this repo will trigger a build

3.2.5 Tool leanpub-book-site

- explain what it is and how it works
  - rules of engagement
  - folders and file structure

    * auto-generation of book.txt file
    * consolidation of images
- reason for doing it was : 1) solve problem of massive image folder (now each chapter is directly mapped to it's images, which is ok as long as the image's name are not repeated) 2) solve problem of having to maintain the Book.txt file 3) allow splitting of manuscript folder into separate repo

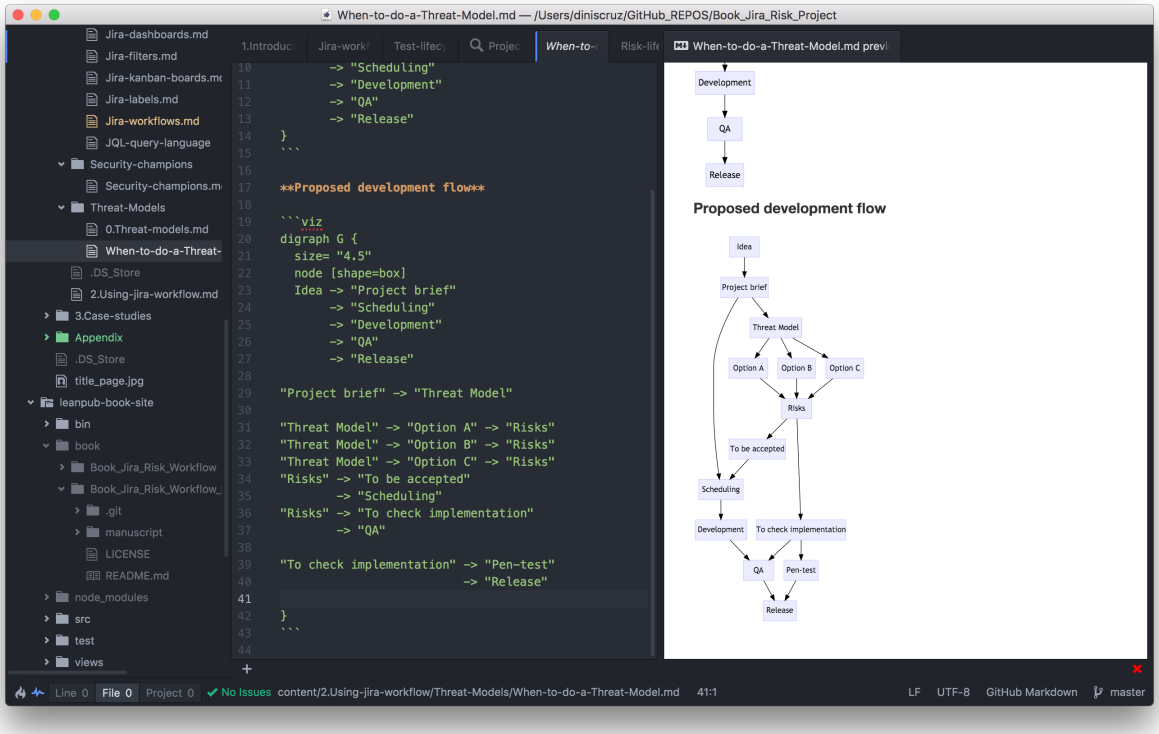
3.2.6 Atom, Markdown, Graphiz, DOT

Editing and diagram creation was done on Atom editor[2] with the markdown-preview-enhanced[3] plugin

Text was written in markdown[4]

Diagrams where created using DOT Language[5] , rendered by GraphWiz[6] and Viz.js[7]

This is what the IDE looks like:



[2] https://atom.io/
[3] https://atom.io/packages/markdown-preview-enhanced
[4] https://leanpub.com/help/manual
[5] http://www.graphviz.org/doc/info/lang.html
[6] http://www.graphviz.org/
[7] https://github.com/mdaines/viz.js

References:

- GraphWiz and Dot:
  - Polygon-based Nodes[8]
  - Node, Edge and Graph Attributes[9]
  - Viz.js online demo[10]

[8] http://www.graphviz.org/doc/info/shapes.html
[9] http://www.graphviz.org/doc/info/attrs.html
[10] http://mdaines.github.io/viz.js/

3.3 Appendix C: Security Tests Examples

- add multiple examples of security tests
  - in node/coffee script
    - HSTS header check
    - detecting attack surface changes
    - performance tests
  - in Javascript
    - emberjs safehtml issue
  - in java
    - random() lack of randomness
    - detecting methods calls
  - in .net
    - email regex issue
    - using reflection to check api usage
    - testing XSS on HTML Elements
  - android
    - query SQL Injection

3.4 Appendix D: Case Studies

3.4.1 File Upload

- public competition where external users where supposed to upload their work (this was aimed at University grads)
- lots of moving parts in original design
- better solution was to use GitHub for file submissions
- massive difference in the risk and complexity of each solution

3.4.2 HTML editing

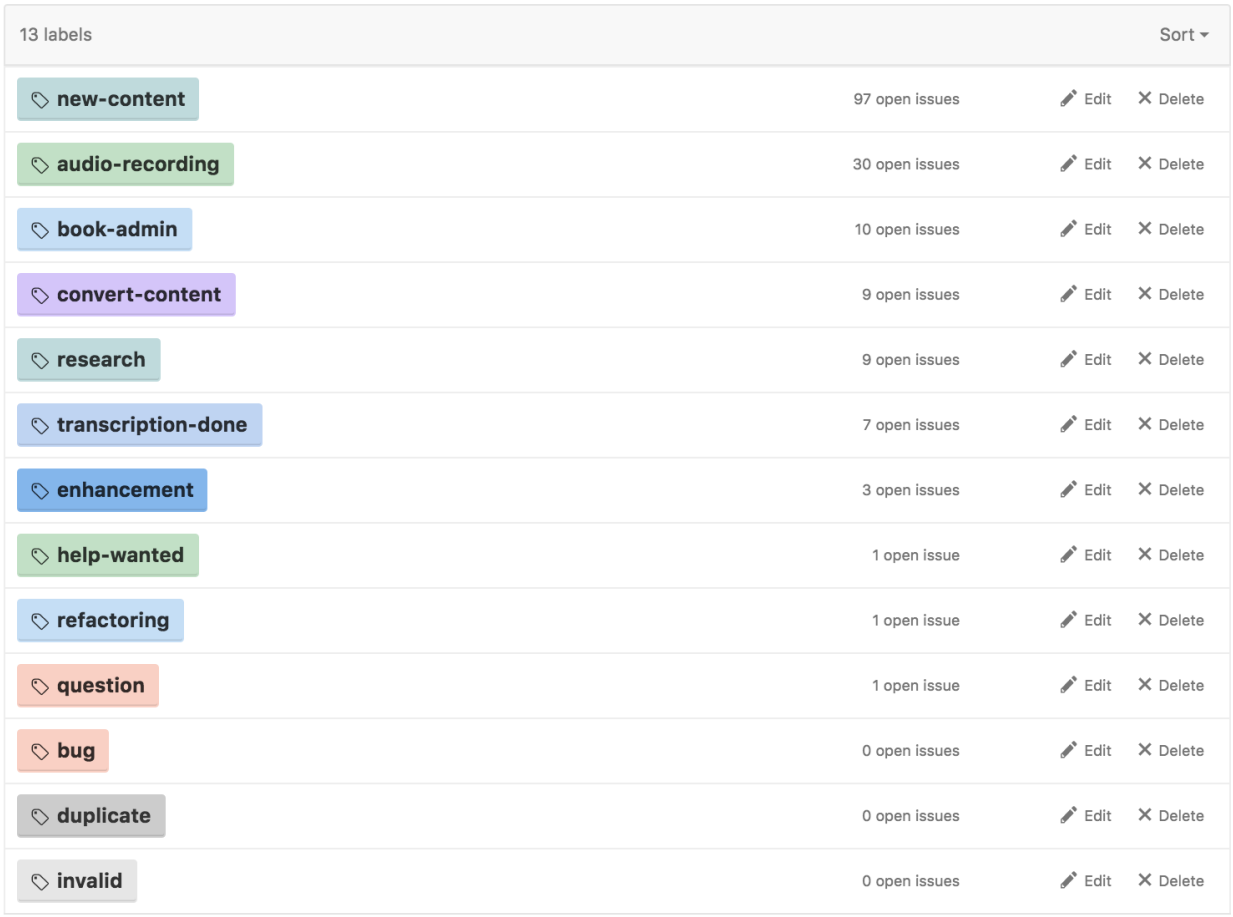
- common request/feature in web-apps
- massive attack surface and security issues (equivalent to XSS) i
- prevents clients to protecting themselves (unless they can use CSP)
- good example of not answering the real business need
  - which tends to be *'edit text, with images, some formatting (bold, italics), links and tables'*
  - all these can be meet if using Markdown (which can be even better for the user, due to its easy of use, ease of diff and readability)
- lots of un-intended side-effects, for example with copy-and-paste
- trying to create 'safe html' is very dangerous due to the crazy stuff that HTML allows and the 'cleverness of some browsers' (which are able to fix broken HTML and Javascript)

3.5 Appendix E: GitHub Issue worklfow

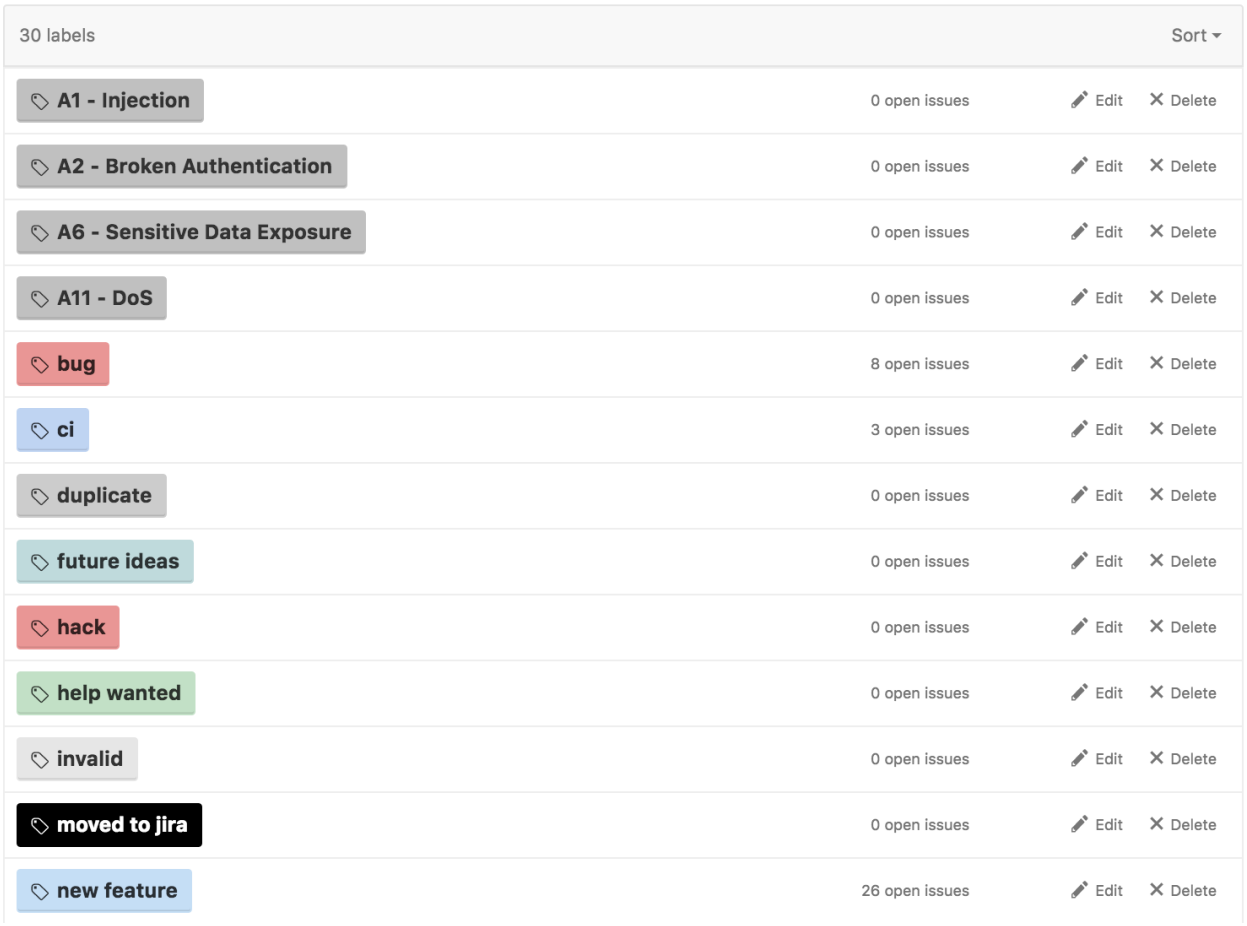
3.5.1 GitHub Labels

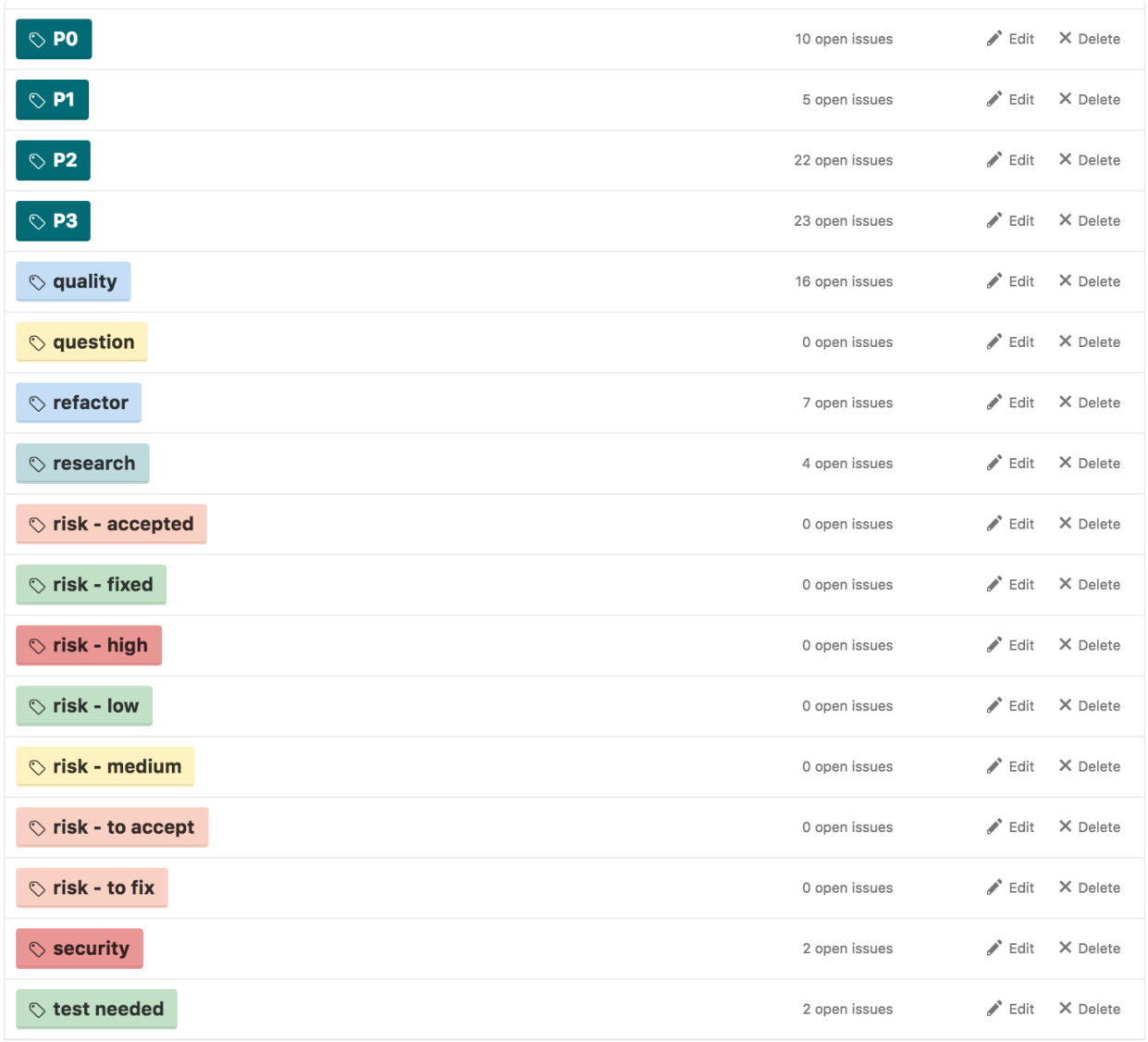
- Below are some examples of the use Labels

Labels on book generation



Labels on complex software development

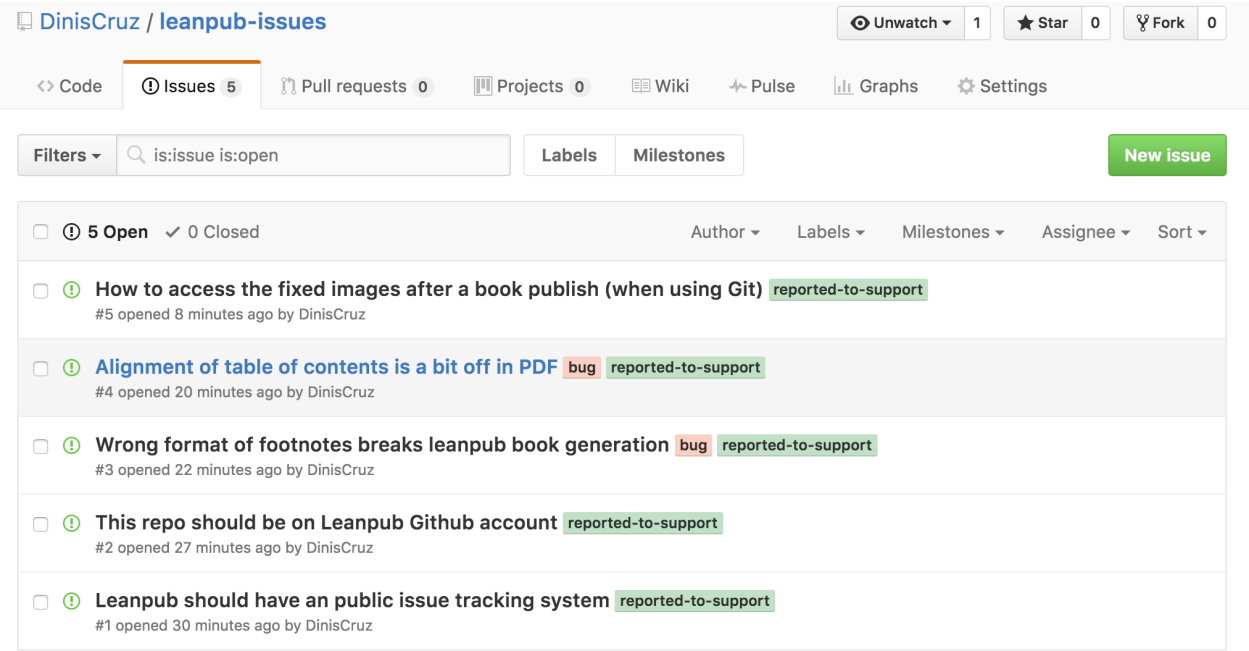

30 labels
Sort
A1 - Injection
0 open issues
Edit
Delete
A2 - Broken Authentication
0 open issues
Edit
Delete
A6 - Sensitive Data Exposure
0 open issues
Edit
Delete
A11 - DoS
0 open issues
Edit
Delete
bug
8 open issues
Edit
Delete
ci
3 open issues
Edit
Delete
duplicate
0 open issues
Edit
Delete
future ideas
0 open issues
Edit
Delete
hack
0 open issues
Edit
Delete
help wanted
0 open issues
Edit
Delete
invalid
0 open issues
Edit
Delete
moved to jira
0 open issues
Edit
Delete
new feature
26 open issues
Edit
Delete



3.5.2 Reporting issues

- GitHub can also be used to create similar risk (and other) workflows
- Leanpub Issues[11]
- Veracode Issues[12]

[11] https://github.com/DinisCruz/leanpub-issues/issues
[12] https://github.com/DinisCruz/veracode-api/issues



Security Makes You A Better Developer

When you look at the dev world from a security point of view, you learn very quickly that you need to go deeper than normally a developer would go. You need to understand how things occur, you need to understand how the black magic,how the stuff actually happens under the hood.

And what that gives you, it gives you a spectacular accelerated way of learning something. In a way your brain is not very good at learning when you only see the behavior. It is like when you have instructions, you've been given instructions from a GPS system after a while you don't learn them because your brain doesn't capture it.

So if you are only dealing with behavior, you don't learn very well you don't really know why something is happening or what is in the hood and what is the real root causes of those choices that were made in the app or the framework where it is.

So, security promotes you to go deeper, promotes you to find ways to learn technology, to understand how it works in a way how to test it that is why I have such a strong testing background because I spend so much time trying to replicate things, trying to make things work, trying to connect A to B, try to manipulate data in between A to B.

But what is interesting is that when I go back to development I realize that I have a bigger bag of tricks. I always find that my understanding of technology tends to be much deeper and in a way wider than a lot of developers.

I might not be the best developer at some algorithms but I do have sometimes a much better toolkit and a much more creative way of solving problems that much more intelligent and better and sometimes creative individuals or developers are not able to do because again their frames of mind are smaller, they don't have a lot of those references.

And referencing is important in programming because a lot of the times once you know that something is possible, you could do it much easily. But when you don't know something is possible, you have this question; well should I go that way that might be you know two hours or ten hours into it and have to give up. Well if you know something is possible you know that it is X hours away and you can see the evolution or if it is a total lost cause or not.

So those things make a big difference because when you look at the problem it is important to know which rabbit holes you can go and which ways will create good and sound solutions and which ones don't especially when we are talking about testing the app where there tends to be a lot of interesting lateral thinking and creative thinking required and even innovative solutions in order to test specific things.

So ultimately, my argument is that when you do application security, you actually become a better developer, your toolkit expands, your mind opens, and you learn a lot. Just in the last couple of months I had to learn Spark, Go, Camel, Objective C, Spring, Open IM, all these kind of languages and frameworks that I am reviewing and looking at.

And after a while you get very good at learning new systems, making the connections and that again makes you better because you certainly are much able to observe concepts and technologies.

When Failed Tests Are Good

So when you make a code change,it is fundamental that every change you make will break a test or break something. Because what that means it means that you are testing for that particular behavior, you are testing for that particular action that you actually are changing.

So what this means it means that you are happy when you make a change and the test fails because you get this confidence you are being covered, you get this confidence that you know the side effects. So if you make a test change here and a couple of test break and you make another test change there and earn fifty test break you get a much bigger sense on the impact of the changes that you are making.

Now what is scary is when that doesn't happen, what is scary is when you make code changes and you don't see any test failing, and you don't have anything that breaks which in essence means you don't understand the side effects of the change that you just made. And that is what tests should be giving.

So broken tests are great when the test that you expect to break is the one that fails. The changes that you were expecting to make actually are the ones that happens or that it makes sense.

So when you review that code you go, "yes I get it why this is breaking I understand now what is the side effect". And as long as the test fix is fast you have a very quick loop, you have a very effective loop of TDD because that is what it is.

And sometimes I will actually do cases where I will write something and it passes so in a way it is already working the way it is supposed to work but I will have other couple more cases where it just fails because I am just confirming that it is failing the way I expect it to fail.

And sometimes I will codify those failures in tests so for now it is certain that it will fail or I will revert back to the previous one but you just get that assurance, yes I touched here and that guy

broke so yes this is the place that I should be changing or this is the fix that happened in a way that I expected it to happen.