The Secure Development Handbook

Step by Step to Software Security

Charles Weir

Security Lancaster
Lancaster University, UK

Copyright

Title book: The Secure Development Handbook

Author book: Charles Weir
Illustrator: Noel Ford

Contact: c.weir1 @ lancaster.ac.uk

Version: 1.0 September 2018

Contents

Chapter 1: Introduction

As someone working with developers and concerned about software security, you want the 'how to do it'. Not the technical detail, but how you can change your professional approach and that of your colleagues to deliver adequate software security at a minimum cost. You reach for the dummies guide, and there's nothing like that available. This book addresses that gap.

Not so long ago, software developers mostly lived in a world of safety, in which crime and accidental privacy loss were unknown, prevented because the computers stood alone and rarely communicated with each other. Nowadays every computer and every application is essentially part of a world computing network, widely accessible to other people; and criminals and accidental disclosure can hurt large numbers of people very badly.

We all know the stories – grandmothers losing their life savings to online fraudsters; cars potentially crashing at speed because their control systems are compromised; hospital patients dying because the hospital computers are encrypted by an 'Internet Worm'. In 1990 such things were hardly considered; in 2000 only science fiction writers worried about them; in 2010 we started to see real problems; and by 2018 as I'm writing this, most people have encountered the risks and dangers of software security problems. And they're right to be worried.

Clearly software developers can have a part to play to keep software users from harm. In this book we'll explore how we can do this, and what helps a development team – programmers, testers, managers and product managers – to achieve security without compromising all the other demands on ourselves.

How This Book Came to Be Written

Six years ago, the lead author, Charles, was running Penrillian, a 20-strong company devoted to creating leading edge software for mobile phones. Our customers were mainly mobile operators ('carriers'), and we were delighted to receive the commission to produce the first commercial Android mobile money application. Our knowledge of software security was sketchy, so naturally we went to the internet to learn how to tackle secure software development. We found a good deal of information on how to use low-level APIs correctly; instructions how to sign apps; and a lot of horribly-detailed descriptions of 'All The Things That You Might Do Wrong'. Nowhere could we find a friendly step-by-step introduction to creating and verifying an application design that would satisfy a given set of security needs. Charles was horrified at the omission, and when he had the opportunity to return to the academic life, he joined Security Lancaster and chose this as an area to study.

Two years later, NCSC, the government agency tasked with improving Britain's cybersecurity, challenged the Developer-centred Security team at Security Lancaster to research interventions for software developers. They asked what would make a good intervention to help a software

development team achieve better security; how would such an intervention work with different types of team and culture?

To find the answers, we first asked a range of some of the most successful people working to help software developers produce secure code, and analysed how they did it. We looked for positive approaches we found – most security experts love discussing attacks and failures, so this was harder than you might expect!

Based on that analysis, we put together a package and have been trialling it on a variety of development teams; we call this package 'Developer Essentials'. We've conducted highly-structured trials with several companies and improved the package from the results. This book describes both the Developer Essentials package, and explains why each step is important, and how you might want to take it further.

Our mission with this book is to help you to get your team good enough at software security, with a minimum of effort and even a certain amount of fun.

Chapter 2: How to Use This Book

This book is designed to be easy to read a section at a time. Each section takes less than five minutes to read and covers one step on improving the security of the code delivered by a development team. You can get to each one individually from the contents section at the start of the book. Each has section has a brief summary, some introductory explanation, instructions where appropriate how to incorporate it into Developer Essentials, and then discussion of alternatives and different ways to implement it. Most sections also have amusing illustrations by cartoonist Noel Ford.

Leading Developer Essentials

If you're planning to run the Developer Essential package yourself, we recommend reading all the 'Developer Essentials' sections before you start, since some are introduced within other steps. For example, Component Choice is something for the Developers to consider themselves in the Risk and Cost workshop. If you're interested in a wider view, then read the whole of each step. There's a button at the bottom of each page to take you to the next step.

A Note about the Text

Secure development involves a range of people: programmers, testers, project managers, product owners and perhaps other roles. Throughout this handbook, we'll refer to all of them as 'developers'. There are many aspects to software security. Some authors like to distinguish 'security' (against malicious attack), and 'privacy' (from legitimate users); in this handbook we call both 'security'.

We'll also include quotations from experts. And much as we'd like to give full credit, the quotations come from interviews done on condition of anonymity – so instead we have cited the role of the speaker.

Chapter 3: Running an Incentivisation Event

Carry out a course, workshop or discussion to persuade your developers to take security seriously.

"Why should I care?" It's there in your colleagues' eyes, in their attitude. Software security is not a part of most software development. They don't teach it much – yet – in university. Many developers and testers won't have encountered it at all, or consider it something for other groups to handle.

Or maybe they have encountered it, and consider it too expensive, or too painful, to countenance. How can we answer that?

If we want our team committed to improving software security, we need to address that question. Specifically, we need to address it in ways that are meaningful for this team, in the context of the work they're doing. But how do we convince them that security isn't 'somebody else's problem'? How do we motivate them to start taking it seriously?

The Developer Essentials Incentivisation Event

For the Developer Essentials package, the Incentivisation Event takes the format of the Agile Security Game. It doesn't need security expertise to run, making it suitable for teams who don't have security experts available.

The Agile Security Game involves groups of up to eight people all taking the role of Product Manager for a mobile application development. The facilitator has the role of 'games master', following instructions provided in the game. The players chose 'security story' cards for each development cycle, and then discover, based on the outcomes described by the facilitator, how successful they have been in deterring security threats.

It's good fun, and participants generally enjoy playing. It introduces very effectively the principle that the Product Manager is the one to make decisions about security cost/benefit trade-offs. Its limitations are that it doesn't consider privacy aspects, and concentrates on relatively anonymous threats, ignoring others (such as insider fraud and customer repudiation of transactions) that might be equally valid. It's worth the facilitator saying as much at the end of the game.

Full instructions and materials for the Agile Security Game are here on this site.

Other Forms of Incentivisation Event

There are several other ways to use the power of group discussion and learning, using a variety of team activities or even one-to-one discussion. All three of the following are widely used by more security-expert teams:

- Pen test workshop,
- A security lecture, or
- The Talk

In the next sections, we'll consider each of these in turn:

Penetration Test Workshop

Penetration testers are software security specialists; they 'wear the hat' of a possible attacker and try to break into the software you have produced. A good 'pen tester' may be able to work with the developers and show them the kinds of things an attacker can do. By doing so, they can convince the developers and testers that their software already has a problem.

> *With these companies, we would do an initial project where, for example, we pen test one of their existing products, and show them "this is how you designed this product, and this is how you went through this process, and this is what we found (Professor, German research organisation)*

This is very much the 'traditional' incentivisation workshop, since many security specialists are expert pen testers. It's excellent in one way – it is very immediate and links in immediately to the context of the development team. It does have disadvantages as an approach, though. First, it sets the specialist up as an 'adversary' to the development team, which makes it more difficult for that specialist to be seen as a helper later on. Second, it tends to give the impression that software security is about low-level fixes to existing code, whereas often the biggest security issues are related to design and usability. Third, it's no use at the start of a project when there's no code to test.

Security Lecture

Some bigger companies, with thousands of developers to bring up to speed with security, use the traditional teaching approach of couple of days of lectures. As an approach it works only when the teachers are really expert and know a great deal that is relevant to the company.

So we run a very large scale education programme … where we … tell developers exactly what happens in the real world, how TalkTalk was hacked, how Sony was hacked. And then we go in detail how we have been attacked, and whether they were successful and how they were detected. Then we also show them all the stuff that our red teams do – our internal hackers – which really scares them! (Security team lead, Operating system supplier)

The Talk

When there's only one developer involved – for example a new joiner to the team – the approaches above don't really work. Many companies, instead, have a discussion between someone who knows the issues and the new joiners.

The conversation can take anywhere between 40 mins to several hours depending on who the person is, and you won't know until you've had the conversation. And the conversation is, you explain how to break into, how an attacker would attack the systems, and what the various things you need to be aware of, are. (Security consultant and pen tester)

This is not easy; the trick is to link the issues to something that has meaning for the developer in question. One expert aims to find out situations with the developer's own family and friends where software security is an issue, and then uses that as a hook to relate that to the software they themselves will be developing.

The Cardinal Rule

Don't leave the developers scared! Conventional software security wisdom used to be to 'scare developers and leave them scared'. Our

experts don't agree – instead they stress that the event should leave developers aware of security issues, but confident in their ability to address them.

> *It needs to be positive. That is why the day is balanced. For every one of these problems we show you in the commercial world and internally, we absolutely have a way of mitigating it. And even if we know we can't stop it, we can certainly detect it, contain it and then exfiltrate those people. (Security team lead, Operating system supplier)*

Chapter 4: Threat Assessment

Find out as a team what are really the threats for your project.

Know Your Enemy...

Your development team has an understanding that software security is important for your organisation, and that it is something they can affect.

But how are they to do that? To answer that, we recommend you first lead your developers to a different question: not how but what? What should we be worried about? What is likely to hurt us, and how might it happen? If it involves external attackers, what do they want, how might they go about getting it and how much do we care?

Finding the answer to those questions is called ‘Threat Assessment’, or 'Threat Modelling'. It is essential for any project involving security, but is not yet, alas, common practice:

"The first conversation is with the [development team] manager to ask "what is the threat model". There is a lot of things you can do in your head to guessimate how bad things are going to be and none of them are fool proof. But when you ask that question and the person in charge of the project says "that is an interesting question, I'll have to think about that," you think, this is going to be interesting. But that is not at all uncommon." (Secure Development Consultant)

Developer Essentials Threat Assessment

For the Developer Essentials package we use a brainstorming approach, as follows. To run it requires only the ability to facilitate a group of people working together – a skill many people can claim.

Brainstorming requires one member of the team to act as facilitator. The team sits in chairs in a circle facing each other and a flipchart (or whiteboard). The facilitator writes a question to focus the discussion, along the lines of 'what threats do we face'. Then everyone suggests possible threats – without analysing each or attempting to filter out any of them. As they do, the facilitator writes down each threat (whether sensible or not) on the flipchart. There's more about brainstorming on the web – for example here.

As ways of generating idea, it's helpful to consider also who might be the attackers – what would they want and how would they go about getting it. That may generate a different set of possible threats. Similarly looking at the architecture of the system in detail, concentrating particularly on interfaces between systems and components is a further excellent way to find threats.

Create a document listing each of the threats: the attacker, what they might get from it, how they might get it.

Alternative Approaches to Threat Assessment

If you have a little more security expertise in your team, then we recommend considering the 'games' Protection Poker and Elevation of Privilege. These both use rather more security jargon, and will need someone with the knowledge to explain them.

Or consult Adam Shostack's book 'Threat Modelling' (see section Chapter 13:) for a more in-depth discussion of all the possible approaches.

Some Other Thoughts

The Threat Assessment session is worth repeating every year or so in a project. Whether or not it finds new threats, the discussion can be very helpful to show team members who knows what about security aspects:

> *I think it got everyone talking about security a bit more, especially within our team... There were a lot of security things going on that I didn't know about. But other people knew that they were happening (Tester, Large company team)*

It's always difficult to think of everything, and you may be able to find useful prior work. Larger organisations may have existing generic threat models for applications of your kind in your industry; some have used the ISO/IEC 27005 standard (or former UK IS1). Or you may find appropriate discussions for your specific architecture or industry, such as CESG's 'Architectural Pattern' work.

Chapter 5: Risk and Cost Assessment

Assess your threats for both impact, and cost to mitigate.

Once you have a list of credible threats to your system, you will have a better idea what are likely to be the security requirements.

A common misconception is that it is the programmers' role to make security decisions. This is wrong. Implementing security improvements of any kind costs effort and usually money, and decisions about resources are very rarely for the developers to make. Typically, all but the most trivial security decisions will need to be made by the Product Manager – the person who decides how to allocate resources to development. The task of addressing each security threat then enters the project workflow as other tasks, such as functional enhancements and functionality bug fixes.

> *Developers don't fix security; issues, tasks, epics and stories do! (Johanna Curie, speaking at AppSec Europe 2018)*

Fortunately, balancing risks against costs is a normal part of a Product Manager's role. To make informed decisions, though, they will need more information:

- The likelihood of each threat,
- The impact on the organisation if an incident occurs and
- The cost of addressing (mitigating) the threat

Developer Essentials Risk and Cost Workshop

In the Developer Essentials package this workshop follows the Threat Assessment. We recommend doing it in a separate session (or at least, after a break), since the analytic type of thinking involved is different from the brainstorming in the previous section.

In this workshop, the threats are written up on a list – usually on a display screen. The participants address each in turn, perhaps by voting on which ones look most important to address first. For each, they estimate:

- Likelihood: Low, Medium or High
- Impact: Low, Medium or High.

Obviously, these will be relatively inaccurate assessments: the aim will only be for finger-in-the-air accuracy. If the workshop can involve a Security Specialist, they may have helpful knowledge about likelihoods of different threats, and possibly even typical impacts.

And then, taking those with high impact or likelihood first, the team identifies possible mitigations – possible things we can do to deal with the threat. Some mitigations may be in code, or changes to functionality; others might be processes, discussions with other teams, or even preparing a plan for dealing with a successful attack. They then estimate development costs for each using the same process as estimates for any other piece of development (story points, for example).

During this workshop, it's important to consider some of the other techniques described in this book. These two are particularly important:

- Component Choice, and
- Automated Static Analysis

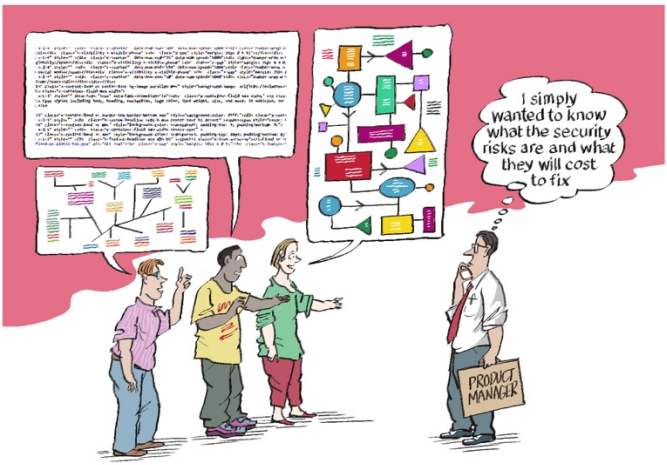
Both are good ways to mitigate some kinds of threat – though they may or may not be appropriate in any given case.

Other Approaches

Some organisations may even have professional Risk Assessors, who can support this even better. Or you might choose to have a smaller group of people making these assessments.

Chapter 6: Security Negotiation

It's rarely the programmers' decision which security enhancements to implement. So help your product manager by expressing problems, risks and costs in helpful terms!



When you have identified threats, done a risk assessment (likelihood and impact of each), and thought up some mitigations for each, what should be your next step?

In many cases, simply knowing about each threat may be sufficient: the developers can make choices that avoid the security issues. They may require care in coding, or influence the architecture and design choices. However, for many of the threats the mitigations will require support: either because they involve significant cost or effort; or because they are outside the scope of the development team. What do we do about those?

In most projects, it won't be commercially sensible to address all the security issues identified. There will always be a trade-off between the effort and cost required for security improvements, and other practical demands such as the need for new functionality or the need to save and

money. The best commercial decision may also be to defer some mitigations to a later stage in the project.

> *And it has to be a bit of a trade off as well in terms of business. You've got to make the trade off as to what's good for getting a solution available now, and having one available in a year's time, which no one will buy, because everyone's gone with one which doesn't even consider security at all. (Business App Developer)*

This means that perfect security is not only very difficult; it is not in fact what we want. It is sensible and good business practice for an organisation to accept a level of insecurity.

> *And actually the way this works, in practice is you have to do less than a perfect job, in order to have a measureable degree of failure or fraud or whatever, so that you can adjust your investment and say 'I am managing this to an economically viable level' because if it is zero, you have invested too much. (IoT Security Consultant)*

Supporting Security Choices

How are the development team to support the decision which security mitigations are carried out?

Software development involves many choices: what functionality to include; what non-functional requirements to satisfy; how best to implement it. Software security adds to these choices: how do the benefits of mitigating each security issue weigh up against other possible uses of the resources required?

Let's call the role that makes those choices 'Product Manager'. In practice, the people concerned may have a variety of titles: 'CEO',

‘customer’, ‘product committee’, ‘project manager’, amongst others – or it may even be the developers themselves.

What is apparently new for a Product Manager is that security decisions involve understanding risk: rather than ‘if we choose this, that will happen’, it is usually ‘if we don’t choose this, that may happen’. But is it really new? Even without considering security, Product Managers already make decisions based on uncertainty: development timescales usually involve risk of overrun; choices between functionality involve uncertainty as to which will pay off; many other factors make the outcomes uncertain. Product Management already deals with risk.

In order to make good decisions, though, a Product Manager needs to *understand* the risks involved, the potential costs of not having each mitigation, and the actual costs of implementing it. It is the role of the development team to provide that information. For each threat that needs consideration, the Project Manager will need:

Cost estimate: An estimate of the effort required to implement the changes required – in time, and in some cases money.

Impact description: A non-technical description of what the threat could mean in practice for the organisation and the software users. E.g. ‘A public leak of our customer data, reported in all the news services; and a corresponding very large fine.’

Risk description: An estimate of the likelihood of the threat being realised over a given timeframe if nothing’s done. E.g. ‘very likely within the next two years’.

This may be quite informal. One developer describes the negotiation as a conversation about security:

[When I started] a project I'd go back and ask [my customer]…'You do realize this [information] can be seen'. It goes from there: 'how secure do you want it to be?' (Secure App Developer)

In an agile development environment, these mitigations typically appear as tasks, for the product manager to organise and prioritise along with the rest.

Continuous Reminder

Incentivising developers to know the importance of security is great; but faced with normal life people forget quite quickly. So, make sure you have a trickle of regular events and nudges to remind the team.

While your initial Incentivization Session may be effective, its impact is relatively short-term. After a few weeks or months we usually find that the drive of the development team to do security tends to be lost. To counter this, it's important to keep reminding the team of the need for security. Typically this needs to be at least every month. You can think

of the technique as kinds of 'nudge': small reminders of the importance of security issues.

Developer Essentials Continuous Reminder

The Developer Essentials package approaches Continuous Reminder as a monthly follow up session devoted to security. Typically, the session will look back on what has been achieved over the previous month, what problems there have been and how to address them, and what security improvement opportunities there are for the team in the next session.

During these sessions, is the right time to consider some of the other possible steps discussed in this book, specifically:

- Security Negotiation – How effective is this being, and how to improve it.
- Security Champion – could someone take on this role, and how to support them?
- Code Review – is this workable?
- Penetration Testing – Is it a financial and practical possibility?

Other Approaches to Continuous Reminder

The delightful thing about Continuous Reminders is that there are many ways of doing them, limited only by the team's ingenuity. The following are some example of 'nudges' we've encountered.

An excellent approach is positive feedback built into the development process. It's very effective to give developers small encouragements for their successes related to security, even quite small ones.

> *The other thing I have to mention is the feedback loop. A pen test report or even just a static code analysis coming back and saying you are doing well – having a desk report or the executive summary that says "wow that is a really*

secure application, well done!" (Security trainer & consultant)

Some companies organise a monthly security competition, defining teams and awarding points based on the security success of each.

> *We tally the results, and there is a league table, and the team at the top get a prize and a little trophy, and we have these 'town halls' and all that. (Security expert, Internet infrastructure provider)*

Others work through security checklists, such as the OWASP Top Ten, concentrating on a new item each month.

This kind of positive feedback may even be built into automated tools. For example Detectify, an online web security review tool, produces encouraging feedback to its users in the manner of fitness apps, before making suggestions for improvement.

> *Now you have 8 out of 10. Nice work! (Detectify)*

We strongly recommend using a similar 'nudge' approach with other stakeholders. Use incidents in the news to remind product management and senior management of the importance of security.

> *Never waste a public disaster. If it is in the news, everyone here has read about it. Like Sony: we went straight to our VPs and said "right, what if that was [us]? What would you do, how would we cope, how would we react, what would it do to our share price?" (Security team lead, Operating system supplier)*

On-the-Job Training

Somewhat more heavyweight, but often very effective, are training sessions. A commercial training course might be useful. But even more

helpful are *regular* events of different kinds. They too act as regular 'nudges', reminding the team of the importance of software security.

Some teams have found 'Brown Bag' Sessions very helpful. These are lunchtime sessions, where food is provided or people bring their sandwiches to eat. For security, these sessions generally use informal presentations and discussions involving the whole team. Because they are valuable as nudges, it is more important that they happen than that they are good! A presentation that leads to an animated discussion between team members is excellent. The people to lead them might be a security champion, or an external security expert. Typical topics might be sharing recent security experience from one's own and related projects; or working through and discussing different kinds of security issues, such as the OWASP Top Ten.

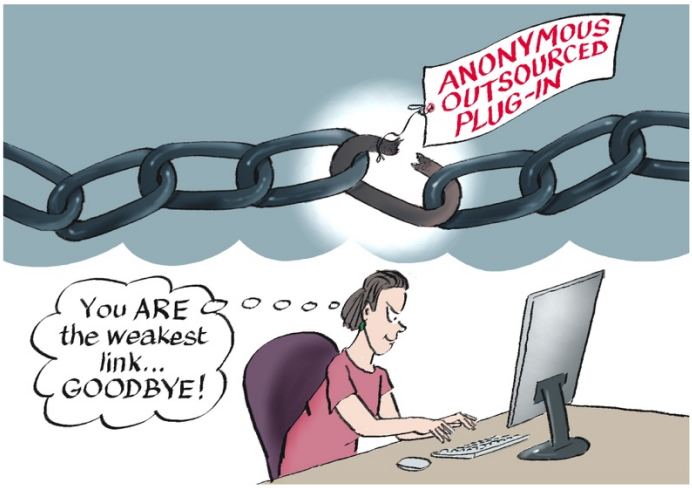
> *[Our security specialist] will take the most interesting or most relevant findings for the team out, and those go into a slide deck that we keep, and that deck is used as part of a show and tell. That happens… a few times a year. (CEO, outsourced secure web developer)*

Another possibility is for developers to work together through some of the 'ethical hacking' training materials to learn how an attacker might work. There is a variety of courses available; however almost all concentrate on purely technical attacks, so we recommend having other forms of training as well. A discussion on Quora here gives some idea of the range of hacking training available.

> *I think training is obviously very effective, and we sometimes do specialized training. … So we had pen testers coming in, and I have got it so that we can now do it ourselves, where we have got a VMWare image with all the hacking tools on it, and vulnerable webpages, so they can play and see how easy it is – and what issues they need to look for. (Security expert, Web infrastructure provider)*

Chapter 7: Component Choice

Programmers use lots of components – chunks of software written by other people. These components frequently have security issues. So make sure you choose secure components, and keep all your components are up to date.



Components, plug-ins, development frameworks, we cannot avoid them. It is components that give simple lines of code, in languages that are relatively little changed since the early days of computing, the power to create the complex and sophisticated software we rely on today.

However, components can bring with them enormous baggage: thousands of lines of code, references to other components we know nothing about, unknown quality and test status. A component from a reputable open-source library might be the creation of brilliant developers, tested by thousands of experts; or it might be a first creation by a novice programmer.

WordPress plug-ins are an enormous liability. Anyone can write one, most of them are rubbish. Anyone can get them put up on the plug-in directory, which gives them this air

of authenticity, and quality that they don't deserve, frankly. (CEO, outsourced secure web developer)

This means, unfortunately, that components are the first and simplest way for an attacker to break software.

So from an attackers point of view, you look at whatever the system is, you don't need to look at the code at all, what [components] would they have used to produce this... And you'll find code exploits! You'll find the OWASP 10 in one [component] alone! (Security consultant and Pen tester)

So as a developer, one of the first and most powerful ways to ensure the security of our software is to choose only appropriate frameworks and components. If you are helping developers to improve their security, this is one of the first things you will need to address.

So that is one of the things we end up sitting down with ... developers going 'I'm sorry, but I know this is actually going to slow you down'. And we are desperately normally trying to avoid that, I'm trying to make your lives as easy as possible. ... But [I] have to say "well, no, you can't just add components – you have to review them, you don't have to do the most detailed review in the world, but if you think it looks worrying, then don't put it in your code". (Security consultant and Pen tester)

There are two aspects to this. We choose components that are secure in the first place; and we update components when the component developers find and fix security problems.

Finding Secure Components

Use only secure components! It sounds simple, if somewhat limiting. But how are we to know which components are secure? If the components are open-source we could review the code in detail; that would take a

great deal of effort, and we might easily miss issues. What else can we do?

Fortunately, this problem affects a large and increasing number of developers, and so there are both open and commercial solutions. The usual approach is to incorporate them into the build process: the build process 'knows' which components are incorporated into the system, so it's usually straightforward to add a check whether they are acceptable. The check usually involves interrogating a web-based 'vulnerability database'. Some of these are free like Wpvulndb for WordPress; others paid like Snyk.

> *[Our tool chain] also queries Wpvulndb for the plug-in that you are expecting, and tells you if there have been any published vulnerabilities in it. (CEO, outsourced secure web developer)*

For components not in the vulnerability database – such as components internal to your organisation – your options are two: either you review the code yourself, or you make a judgement based on the security credibility of the supplier. Components from big, security-aware companies like Microsoft, Google or Twitter, for example, are unlikely to present a risk. Components that regularly receive security updates are likely to be OK; those that do not update are likely to be a problem.

Keeping Components up to Date

There is a second problem with components. Since they tend to be widely shared, any weakness in a component will become known to attackers, and therefore it is important to keep components upgraded to the versions in which defects have been corrected.

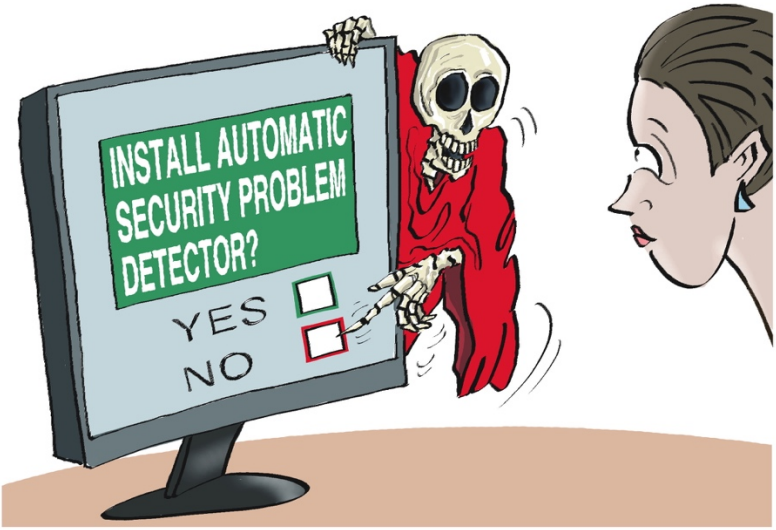
> *We keep track of all the patches and everything for all our systems. (Developer, Security-oriented phone manufacturer)*

This is a second function of the vulnerability database: to keep track of which versions of components contain which weaknesses. We may not need the latest version of each component; after all, for this purpose we are only interested in updates that correct security defects.

Keeping components up to date has a cost: some updates require changes to code; others may introduce defects that were not present before. Sometimes the cost of upgrading to the latest secure versions of components may be enough to need business justification – see Security Negotiation [Link]. However at least, with the help of the vulnerability databases, there can be a good cost-risk-benefit-based decision about when to do the upgrade.

Chapter 8: Automated Static Analysis

Some kinds of security problems are mechanical: a computer program can spot them in your code. So use one...



Once you've made sure of your components, what can you do to improve the team's own code? In these days of automation, we want to do as much as possible to help the team get things right.

So, consider using an automated code checker! These do not provide code security – far from it. However, they do help removing some types of security error.

Automated code checkers review the code to look for possible security flaws. Such tools are sometimes called 'lint' checkers, after a UNIX tool that does extra checking for C code. There are now many such tools, some produced by commercial companies, supporting different languages and purposes:

> *One of the most common things… for anything using C or C++ is to look for potential buffer overruns. And anything that has SQL Injections that do the same sorts of things: anything that can go outside of the expected bounds, that aren't being checked. And there are a number of Lint checkers that will pick up on that sort of thing. Use them! (Developer, Security-oriented phone manufacturer)*

Probably the most important feature you want from an automated code checker is the ability to suppress errors on subsequent runs. Most will provide a host of warnings: wading through them takes a while, and no programmer would want to do that twice. Yet you will want to run the tool regularly on the code as it changes.

It may be possible to incorporate the tool into the project toolchain. This is worth doing only if the team regularly look at the errors generated during the compilation process; many do not.

Most tools support configuration to suit specific systems; this too is worth considering, though it is expensive in time.

> *Certain tools are very good with custom rules; they are very easy to tweak to try the custom rules. If someone is doing something different from a web application, I contend that they need to find an Automated Static Analysis tool where they can write the rules easily, because they are not going to have cross site scripting, they are not going to have the standard thing that the tool looks for. (Consultant trainer, Security specialist)*

Please ensure, however, that developers do not get the idea that using a tool on its own will achieve security.

> *Are they useful? To a degree, yes, they are useful. ... Obviously finding potential issues in your code, that you have to then look at and say, "actually I meant to do that*

and that is okay", or "oh dear, I've missed that, I'll …", that's great. But the danger with them is that you think that is making your code secure, and it is not. It is just finding a certain class of problems in it. (Security consultant)

Chapter 9: Code Review

A great way to spot security problems is to have another developer or security expert review the code. Do it if you can.

We've looked at two types of tool to improve software security: checking the status of components, and analysing the code for basic security issues. But what about using teamwork – the 'second mind on the problem'? How might we do that?

The classic, and very effective, answer is: code reviews. Our experts distinguished three kinds of code review, for different purposes:

Expert code review: A security expert reviews important sections of code for vulnerabilities.

Commercial review: A professional security company reviews your code against their vulnerability list.

Peer code review: One or more developers review another developer's code, either in a special session or simply using pair programming.

The next sections discuss how we might incorporate each into a development team.

Expert Code Review

If you have access to experts on secure coding, then code reviews are an effective way to use and transfer their expertise. The expert involved need not be a programmer; their skill lies in understanding where issues in code may cause problems, and in questioning the developers about such issues.

> *[We have] 'software pen testing', where you have some Subject Matter Experts who take a piece of code and review that in detail, and work with the developers in tandem, looking at the code and saying "we think this is really high risk, we really want to look at this." And then somebody goes through that code with the mindset of "how do I exploit the code" (Security team lead, Operating system supplier)*

Such reviews serve other purposes as well as improving the section of code reviewed: they are a good way of teaching secure coding to the developers; and they can act as a 'nudge' to remind programmers of the importance of security.[Link]

Getting time from external experts of this kind tends to be difficult, and can be expensive for your organization so do triage code according to its security risk and use this kind of review only for the most important.

> *If there is something in the application that is a particularly sensitive area ... you probably want a security guy to come and sit for a few days and have a look at it,*

someone who can work with the develop team but who hasn't been part of that development team, in a way. So somebody who can ask the stupid question - I think that is really important, to bring in an external person. (Consultant trainer, Security specialist)

Commercial Code Review

The widely used equivalent to Penetration Testing for a mobile app is an external security code review.

Many of the companies who perform Penetration Testing also do this kind of code review; they gather lists of known security issues found in apps, with mitigations for each, and then review the provided code to look for those security issues.

Peer Code Review

For most teams it's much more practical to do internal code reviews – to have one or more other team members as the reviewers. Some teams even have a formal review process, with separate review meetings delivering lists of defects for a developer to fix.

It is in the culture. We do reviews; we always have to do reviews. We set things up – and it is not regarded as a second class. (Security expert, Security and military supplier)

However, formal code review is regarded as relatively expensive http://eprints.lancs.ac.uk/78969/1/information_assurance_techniques.pdf, and others use more informal peer review, using pair programming.

We do a lot of peer review. So we will do 'buddy builds' where people will work together for a week, and work on their own, and work together for a week. (Security team lead, Operating system supplier)

Research suggests that pair programming and code reviews have a productivity benefit rather than a cost: when evaluated over a long period, teams that do code reviews are more productive than those who do not.

But programmers tend not to like code reviews; it is uncomfortable criticising other people's code, and even more uncomfortable when it's your own code being criticised. It is surprisingly difficult to set up Code Reviews and keep them going if you do not already have the kind of culture that supports reviewing. And if you either work alone, or effectively alone in that you have no colleagues working on the same project, peer Code Review may not be an option for you.

Recommendation

Thus, our recommendation is to get security experts to code review key parts of the system if you can – that is, if they have the expertise and willingness to do it. If your budget allows it, consider paying for a commercial review.

Considering peer code reviews, though, our recommendation is not to set yourself up to fail – it's demotivating for everyone – so unless the development team are really keen or your process already supports it, avoid trying to set up a system of code reviews. Do consider pair programming, though, especially for sensitive parts of the code.

Code Review Resources

A good resource on code reviews is the OWASP online handbook. https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project

Surprisingly few books tackle the topic. The original was Software Inspections by Tom Gilb and Dorothy Graham https://books.google.co.uk/books?id=GO11btO0-vIC&q=software+inspection&dq=software+inspection&hl=en&sa=X&ved=0ahUKEwj55Jjky_DYAhWMUlAKHXK0AIcQ6AEIJzAA .

Pair programming was originally introduced on the C2 Wiki http://wiki.c2.com/?PairProgramming . Any books on Agile Development will have some discussion on the subject. There's a friendly introduction on WikiHow https://www.wikihow.com/Pair-Program

Many of the companies who do Penetration Testing (see Penetration Testing Resources) https://support.jimdo.com/faq/how-to-create-anchor-links/) will also do code inspections.

Chapter 10: Continuous Reminder

Incentivising developers to know the importance of security is great; but faced with normal life people forget quite quickly. So, make sure you have a trickle of regular events and nudges to remind the team.

While your initial Incentivization Session may be effective, its impact is relatively short-term. After a few weeks or months we usually find that the drive of the development team to do security tends to be lost. To counter this, it's important to keep reminding the team of the need for security. Typically this needs to be at least every month. You can think of the technique as kinds of 'nudge': small reminders of the importance of security issues.

Developer Essentials Continuous Reminder

The Developer Essentials package approaches Continuous Reminder as a monthly follow up session devoted to security. Typically, the session will look back on what has been achieved over the previous month, what

problems there have been and how to address them, and what security improvement opportunities there are for the team in the next session.

During these sessions, is the right time to consider some of the other possible steps discussed in this book, specifically:

- Security Negotiation – How effective is this being, and how to improve it.
- Security Champion – could someone take on this role, and how to support them?
- Code Review – is this workable?
- Penetration Testing – Is it a financial and practical possibility?

Other Approaches to Continuous Reminder

The delightful thing about Continuous Reminders is that there are many ways of doing them, limited only by the team's ingenuity. The following are some example of 'nudges' we've encountered.

An excellent approach is positive feedback built into the development process. It's very effective to give developers small encouragements for their successes related to security, even quite small ones.

> *The other thing I have to mention is the feedback loop. A pen test report or even just a static code analysis coming back and saying you are doing well – having a desk report or the executive summary that says "wow that is a really secure application, well done!" (Security trainer & consultant)*

Some companies organise a monthly security competition, defining teams and awarding points based on the security success of each.

> *We tally the results, and there is a league table, and the team at the top get a prize and a little trophy, and we have these 'town halls' and all that. (Security expert, Internet infrastructure provider)*

Others work through security checklists, such as the OWASP Top Ten, concentrating on a new item each month.

This kind of positive feedback may even be built into automated tools. For example Detectify, an online web security review tool, produces encouraging feedback to its users in the manner of fitness apps, before making suggestions for improvement.

> *Now you have 8 out of 10. Nice work! (Detectify)*

We strongly recommend using a similar 'nudge' approach with other stakeholders. Use incidents in the news to remind product management and senior management of the importance of security.

> *Never waste a public disaster. If it is in the news, everyone here has read about it. Like Sony: we went straight to our VPs and said "right, what if that was [us]? What would you do, how would we cope, how would we react, what would it do to our share price?" (Security team lead, Operating system supplier)*

On-the-Job Training

Somewhat more heavyweight, but often very effective, are training sessions. A commercial training course might be useful. But even more helpful are *regular* events of different kinds. They too act as regular 'nudges', reminding the team of the importance of software security.

Some teams have found 'Brown Bag' Sessions very helpful. These are lunchtime sessions, where food is provided or people bring their sandwiches to eat. For security, these sessions generally use informal presentations and discussions involving the whole team. Because they are valuable as nudges, it is more important that they happen than that they are good! A presentation that leads to an animated discussion between team members is excellent. The people to lead them might be a security champion, or an external security expert. Typical topics might be sharing recent security experience from one's own and related projects; or working through and discussing different kinds of security issues, such as the OWASP Top Ten.

> *[Our security specialist] will take the most interesting or most relevant findings for the team out, and those go into a slide deck that we keep, and that deck is used as part of a show and tell. That happens… a few times a year. (CEO, outsourced secure web developer)*

Another possibility is for developers to work together through some of the 'ethical hacking' training materials to learn how an attacker might work. There is a variety of courses available; however almost all concentrate on purely technical attacks, so we recommend having other forms of training as well. A discussion on Quora here gives some idea of the range of hacking training available.

> *I think training is obviously very effective, and we sometimes do specialized training. ... So we had pen testers coming in, and I have got it so that we can now do it ourselves, where we have got a VMWare image with all the hacking tools on it, and vulnerable webpages, so they can play and see how easy it is – and what issues they need to look for. (Security expert, Web infrastructure provider)*

.

Chapter 11: Security Champion

Very few teams have a security expert readily available, so instead, have one member of a development team be the 'security champion', and support them learning and helping others in the team.

You may be able to support developers with software security by introducing a 'security champion'. There are several ways to approach this: specialist secondment, expert secondment, and home grown – as follows.

Specialist or Expert Secondment

If you have available security experts who are not themselves developers, they may be able to work directly with the developers to sensitize them to security issues and address those issues in practice.

Our advisors... we actually plug into different parts of the service teams, the engineering teams, and they work as security subject matter experts. And they are highly skilled

> *security people. They are people who couldn't write a product but they certainly know everything about [specific security features], and how would you do that at scale, how would you do the threat modelling. (Security team leader, SAAS multinational)*

Another effective approach is to get someone from a different part of the organisation, or outside, who has worked on a successful secure project and understands how that was achieved: expert secondment. One company that specialises in secure development even offers this as a service for their clients.

> *We send people on site, and we embed them into other teams. The normal outcome, and I can't think of a situation where this hasn't happened, is for them to export our processes like it was the obvious thing to do – and I think it is! And for that then to be taken up by customer [developer] teams. (Team leader, security and military supplier)*

Programmer Champion

More practically in most cases, you may be able to identify one team member with an interest in security, and to encourage them to be a 'security champion'. This developer learns as much as possible about the subject, and then provides support to others in the team on security matters.

> *One thing that we find works with software development teams is ... Security Champions – the idea is that one person in the team is more interested in security – not responsible – but who is the 'go to' person in the team if there is an issue in the team before they go to an external consultant. ... You need that person in a team, you actually do. (Security consultant & trainer)*

Chapter 12: Penetration Testing

If you can afford it, get a security expert to attack your system and tell you its weaknesses.

There's another, widely used, approach to using external people to improve your software security. It's called 'Penetration Testing'.

An external 'white hat' security team simulates what an attacker would do to attempt to gain access or disable the service. They then feedback any 'successful' exploits they have found to the development and operations teams.

[Ensuring software security] tends to get handed off, in most companies I've worked with, to a white-hat hacking team. [They] don't do it a code level. (Developer, Security-oriented phone manufacturer)

Would it make sense for you? On the one hand, Penetration Testing requires specialist skills, which are in short supply: if is internal to the organisation, it requires expensive staff; if external, the cost is significant.

[The problem about recruiting a pen tester is that that knowledge is really quite rare, and the attitude ... is also quite rare] And finding them together is difficult. The people who do have both of those things are always in high

> *demand. Most of them are contractors, because they can make a lot more money that way, and why wouldn't you be. (CEO, outsourced secure web developer)*

On the other hand, Penetration Testing can find a range of possible security problems, including various types of misconfiguration, and vulnerability to injection attacks.

Penetration Testing cannot prove that a system is secure; merely that it lacks some common security faults. Some experts, therefore, prefer not to use them.

> *I don't believe in [penetration] test teams, because I believe that takes away responsibility from people to do the right thing. (Security expert, Security and military supplier)*

The decision whether to use Penetration Testing, as with other security decisions, is a business decision. It requires the approach discussed in section **Error! Reference source not found.**.

A variation, requiring even more skill from the practitioners, is 'Red Teaming'. A Red Team may use more sophisticated tactics, such as social engineering (persuading an employee to do something that assists them) or physical access to the systems involved, to achieve their goal of 'breaking' the system. Currently, unfortunately, setting up or employing a red team is impractical for most development groups.

Pen Testing Resources

The UK Industry body for penetration testers is CREST, which has a list of its UK member companies who've satisfied appropriate criteria here, and some international (USA, Australia, New Zealand) members here.

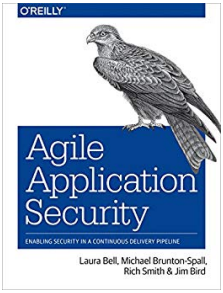
For the USA, there's a huge list of security-based companies, including some pen testers, available from website tool vendor

vARMOUR through this signup page, or for the 2017 version, directly here.

Red teaming will be done by a subset of the above organisations – googling ‘Red teaming specialists uk’ provides a start.

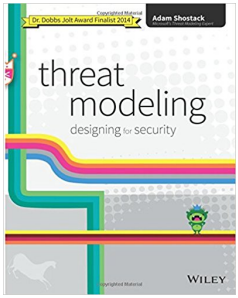
Chapter 13: Further Reading

In this, the final section of the book, let us consider some resources for software developers to learn more about security. Here are six of the best.



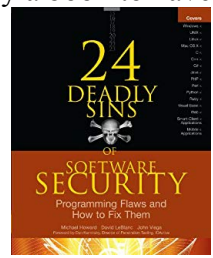
Agile Application Security: Enabling Security in a Continuous Delivery Pipeline, by Laura Bell, Michael Brunton-Spall, Rich Smith, and Jim Bird. O'Reilly Press 2017.

This is the book we wish had been there when we first looked for software security advice. It provides a good introduction on software security for application developers, and to agile software development for security experts. It then explores a range of issues in much more detail than we have in this book. Though it assumes that there are security experts available to work with each development team, is easy to read, and contains invaluable practical advice.



Threat Modeling: Designing for Security, by Adam Shostack. Wiley 2014.

This book sets out and achieves to be the definitive guide to threat modelling. Based on the author's extensive experience at Microsoft, it's targeted at security experts, and assumes more technical knowledge than many software developers will have; but the writing is approachable to anyone, and this is definitely a book to have on your shelf.

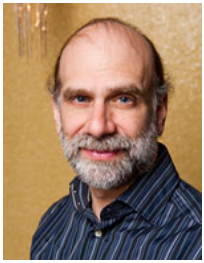


24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them, by Michael Howard. McGraw Hill 2009.

This book describes a set of 'anti-patterns' for software developers. It explores each kind of security flaw, with examples, explanations of why they're problematic, and references to the security literature around them. This is a book to dip into rather than read, but it's worth having on your shelf.

OWASP
Open Web Application
Security Project

Effectively this is the online and updated version of the above.



Though not specific to software development, this monthly email of links to security-related news stories is one of the most widely-used resources for software developers who want to keep up to date with security issues.

Finally, we should add some academic papers. While they are not written for software developers, they provide a foundation for the advice in this book.

Supported by all of these resources, you can have a state-of-the-art knowledge of the best ways to achieve software development security.

May success attend your efforts! And please let us know how you get on, and what might help us improve this work for other readers.

- Charles and the Security Lancaster team.