Year of Security For Java

Series Written 2012

John Melton

@_jtmelton

Last Updated: 1/7/2013

Table of Contents

Introduction

Year Of Security for Java

This will serve as the introduction for a new series that will have roughly 1 article per week for a year. This series will be different from my last series (OWASP Top Ten – Java) in that each article will be pretty short and focused.

There are several motivations for this series:
1. Get some old topics written down
2. Research some new technologies
3. Write
4. Learn
5. Answer questions from Java friends

It's the last one that serves as the primary motivation. In my work with Java developers, I understandably get asked a lot of questions about security. These are pretty far-ranging from the "what is this thing I heard about" to "how do I do this very specific thing correctly". Developers are sharp, but have a lot they are held accountable for, of which security is only one. In this series, I hope to highlight some of the common issues the developers I work with ask me about frequently, as well as point out a few things they probably should be asking about, but don't know to just yet.

Hope you enjoy.

Below, I've broken down the articles from the series into 3 basic buckets: people, process and technology.

People

Week 40 - Get A Security Person (Or Some People) If You Can

What is it and why should I care?
I spend a good bit of time talking about both development and security. I spend a lot of time working with other developers and other security people. There are a precious few that I know of that excel at both development and security. This is a sentiment echoed by many, so I won't spend time belaboring the point. If you can't have everyone be an expert in both, how should you structure your team so you have the optimal blend of both? There is some usefulness in discussing the make-up of teams with regards to development and security, as it can heavily affect your security posture long-term.

What should I do about it?
Let's consider a few different options when it comes to team make-up:

No security people
I thought about leaving this group out, but it's so prevalent that I just couldn't. Many small and medium sized organizations haven't yet added security to their SDLC (another post in the coming weeks on this topic). This is tough. This will take a long time to resolve, and will require changes to developer education and training programs as well as general industry awareness. There's a lot of work being done to get the information out there, but this will just take time.

Developers with some security training
This is a popular option. Ok, we need to do better on security – send one of the team members to a week's training! This is better than nothing, but a pretty weak option. Unless the person is passionate about security and spends time coming up to speed on his/her own, you're going to get little benefit. You may pick up a few of the obvious things, which is certainly helpful, but it does not usually improve your overall security stance. Additionally, this person is not going to have a mentor of any type for the security work they are doing, which can be important in the security field particularly.

Security people at the enterprise level
I think this is a great option for a lot of things, and should certainly be considered depending on the size of the enterprise. Security people at the enterprise level can do things that security people embedded in development organizations just can't do. They can set high-level standards and policy. They can also build security strategies and architectures for development.

As an organization grows, it becomes more and more important to have consistency (assuming the standard is good) across the enterprise. There's a lot of time and money being spent in just trying to figure out what organizations have deployed. It quickly becomes a nightmarish problem, particularly for organizations that have lots of legacy software.

Security people on the development team
Having security embedded in the development organization is also a great option for making impactful changes on application architecture, design and implementation. Producing standards at the enterprise is great, but useless if no one follows them. Also, having security folks deployed in the team helps tremendously with training as your "non-security-trained" developers get direct on the job training tailored to your organization. In addition, you have a built-in mentor to ask questions of if something comes up that's security related. You can also catch issues earlier in the development cycle, since the security person can help do things like code review or design review with an eye for security.

There are a couple of models for this. One options is to have a security minded person doing actual development, but also security stuff, essentially splitting time and focus. Alternatively, you have a security person that round-robins for a few dev teams and functions as a kind of internal consultant. I've seen both of these models work, and it often comes down to organizational culture as to which one is a better fit.

A good article related to this (and with REAL data!) is from David Rook (@securityninja) and is found here. In it, David says that their company embeds security people in with the development team. They do code reviews as well as other security related activities. He has tracked their data over time, and has found that 1 security person to 10 developers is a ratio that works well in their organization. Compared to current standards, that's a LOT. According to BSIMM, there's a ration of 1.95%. That means on average (for the companies that participated in BSIMM), there are 2 security folks for every 100 developers. That includes people who sit at the enterprise level as well as those are directly related to security in development and architecture teams.

Security people outside the organization
A final option for consideration is the "security consultant". This can come in lots of forms. It could be paying people to come in and build your code for you in a secure way. It could be someone coming in and reviewing/testing the code you wrote for security. It could be purchasing or using tools/services.

Using outside consultants is often a business decision in many fields. Is it cheaper for us to develop this talent internally or outsource it? However, that's often not an option in security, though it's getting closer. At the current moment, a lot of the "security" people are at outside consultancies. There are clearly domains (financials, government, etc.) where there is a lot of security knowledge, but many verticals just don't have the internal knowledge.

Using and consuming external security knowledge can be a great idea, but IMHO, shouldn't come at the cost of building at least some of that talent internally. By creating that skill-set internal to your organization, you can tailor your strategy to your organization, a powerful concept.

In conclusion, if you're developing or deploying software, you should be building security into your process, and that means getting good security people on board. Security talent can come from internal and/or external resources. Considering your organizational model and embedding security in the appropriate places can greatly improve your overall security posture.

References
———–
http://www.securityninja.co.uk/application-security/application-security-data/
http://bsimm.com/facts/

Week 41 - Spend (Wisely) On Developer Security Training

What is it and why should I care?
In the last post, I gave some justifications for getting security people into your organization, as well as reasons to have them closely knitted into your team. In this post, I'd like to move the attention to the developers already on your team.

Let's say you've got a model where you at least have some security representation on your team, whether that be an enterprise group that consults periodically, or someone that spends 2 days a week helping write code, or anything in between. (I discussed various models of execution in the last post, and mentioned that your organizational structure will likely dictate the best model.) You have a security expert on your team, but that person probably won't be full time on your project, unless it's huge, and many times, won't be contributing to your codebase at all. That means we need additional security knowledge embedded into the team.

What should I do about it?
You could go about this in lots of ways, but I think Jim Bird's approach is fantastic. It argues for a scaled model where you have different developers with varying skill-levels in security, but all with at least a minimal understanding (basic training – think OWASP Top 10 / SANS Top 25 with a 1-2 times yearly refresh). This is exactly what we do with other concerns in coding, such as performance or scalability. We often have 1 or 2 experts, then a smattering of capabilities among the remainder of the team.

Everyone on the team should know that you shouldn't concatenate request parameters from the user into a sql query string, but maybe not everyone will understand the intricacies of DOM XSS encoding or the ins-and-outs of Content Security Policy (CSP). By ensuring the most common vulnerabilities and associated controls are well-understood by everyone on the team, you create an environment that generally produces more secure code. By having an expert or two on the team, you have resources that know about the latest and greatest protections and who also understand implementation caveats for the basic protections.

The scaled training approach has several benefits:

(More) Secure Code
You can produce rather secure code if the general team has basic training, and there are 1 or 2 experts helping out with the difficult problems.

Cheaper than Training Everyone
As Jim points out, this model produces secure code in a much more scalable way than trying to make everyone an expert. Training developers and turning them into security experts is not a cheap proposition. The basics are usually pretty easy, but there are lots of gotchas to be found, and that takes lots of time and money. By focusing the majority of your money on fewer resources, you're able to make it count for more.

Easier Hiring
Training developers to be security-aware is a requirement, but many will try to hire in that talent. It's well-known that it's quite tough to find security-knowledgable developers. Needing fewer means you have a bigger talent pool to pull from, and most of your needs will come from the standard developer bucket, which is much more abundant.

Natural Path to Secure Frameworks
By setting up a model whereby a few people are experts in a single area (and others aren't), it creates a natural environment to encode that knowledge into some system for the benefit of the larger group. In this case, that encoding is likely to involve creating a reusable security framework. Now, you get the benefits of that knowledge and have it codified in an executable form. In addition, it's simpler to "update" the knowledge store by adding features to the framework and fixing bugs over time. By hiding the gory details in a framework, you give the standard developer security capabilities that they wouldn't have otherwise had, and at the same time increased your security posture.

In conclusion, developers in your organization should get trained on security. The majority should be familiar with the basics, while a few should be experts. By scaling your investment, you're able to efficiently build more secure code and create an environment that fosters institutionalizing the security knowledge of your experts for the benefit of your full team.

References
————
http://swreflections.blogspot.com/2012/05/building-security-into-development-team.html

Week 50 - Think

What is it and why should I care?
With the current series coming to a close (wow, finally :>), I'm going to do a bit of wrap-up.
While all the posts in the series hopefully have something to offer, I've saved my 2 most oft-repeated pieces of advice for last. Actually, neither is specific to Java, security, or even technology. However, they are the two concepts I find myself sharing the most with others. I'm sure you will relate.

The first concept and focus of this post is … "Think".

Sounds fairly simple and clear, but not done nearly enough. It's not that people don't assimilate information and spit out an answer – they do. The problem is that a) they often don't consider all the variables and b) they don't second guess the existing solution or consider if a better solution is even possible. I'm not suggesting that you have to re-think every decision you make, but if you're making an impactful (time, money, people, etc.) decision, it's worth your effort to consider the implications and make the best decision possible.

What should I do about it?

Examples

There's a whole host of decisions that this can apply to. As a for instance, here are a few examples using other posts in this series:

- You need to build an application securely, so you create a threat model. Certainly feel free to reuse, but think about why those threat actors and attack vectors are there in the first place. Do they even apply to your application? Are there others that do apply that you're missing?

- You are storing user passwords, and need to make sure you're doing it securely. Well, don't just grab the library someone used on another project (without a good reason). Think about the actual requirements you are trying to fulfill, and evaluate solutions based on that.

- You are tasked with architecting a reusable authentication and access control solution for your organization. Consider the requirements that you know of right now as well as those that are coming down the road (as best you can). Consider the needs of various stakeholders. Another organization's solution may or may not be right for you.

- You perform a code review on an application that is using a new-fangled data storage and query solution. You've never seen it before, but it looks a lot like nosql. Consider the security issues related to nosql (authentication, access control, injection, data management, etc.) and extrapolate that these issues likely apply to the new solution. Also look for differences, and consider what issues might crop up because of the deltas.

I could make a really long list here, but I think the basic idea is clear. In order to be effective, you must give thought to why you are doing what you are doing. There's a lot of security advice floating around (I gave a lot this year), some good and some bad. However, even the good advice must be tailored to your specific needs. Maybe a solution might apply for 99.5% of people out there, but your organization may not be one of them. That's ok, but solving the problem for you means a) you don't blindly accept the advice as law, and b) you know enough about your environment to decide what the correct solution is.

Creativity

There are many problems that exist (and most that get solved) in security and elsewhere that are generally resolved by considering the problem and choosing a well-known solution from your toolbox. That's logical because many of us encounter similar problems and come up with similar solutions, contributing to the communal mind (think OWASP documentation and projects).

However, there are some issues and problems that require a bit more work to solve. You start hearing things like "think outside the box", be a "creative thinker", and a whole host of other catchy phrases. The point is that the answer hasn't been easy enough or popular enough to be solved (or solved properly/completely) yet, so you must put some real thought into developing a solution.

I relatively recently found an old talk on creativity by John Cleese (brilliant). One of the ideas he puts forth is that in order to be creative, one must have time set aside on a regular basis to dedicate to being creative. You have to give yourself the window of time. You can't expect to be in meetings all day, and suddenly catch an epiphany in the break room. It just doesn't work that way for most people.

Unfortunately, after working as a developer and security person for the last 10 years, I've seen very few people have any sort of dedicated time to "think". The only common example I've seen are those who are going back to school after work, and therefore, have a reason outside of work to study and think hard about certain things. Apart from that, though, I've not seen much in the way of thinking hard thoughts. Development is bad (the exception would be design work), but security is worse. To an extent, the industry is currently so bad at security that we're still dealing with low hanging fruit. When we don't have to try that hard, often we won't.

People

I have a theory I've built over time that places people in 4 rough buckets with regards to their workplace contributions and abilities. I'm specifically talking about dev/security folks, but the same is probably true elsewhere. I've tagged them with musical monikers for clarity (or confusion :>).

1. *Tone-deaf* – These are people that don't try or are in the wrong job. It would be better both for them and you if they were not on your team (and if you didn't have to listen to them).

2. *Knows Basic Chords* – These are people that can learn a task and repeat it properly. In order to move to a new task, they need to be trained on that task, but they will then perform it well. They can be solid contributors, but are not your "idea guys/gals"

3. *Knows Scales* – These folks can learn concepts, and apply them in various situations. They'll often put out ideas that start with "what if we tried …?". They have the ability to learn new things on their own and apply previous lessons learned to new situations. Folks like this often thrive in multi-disciplinary environments. They can take the abilities they have as a software security person or a plumber or a pilot and apply them to solving problems in other fields. This is the highest level most people will achieve.

4. *Virtuoso* – This is what a "thought leader" should mean (heavily abused term in security given those that are called that). These folks are rare … very rare. They come up with entirely new concepts. They develop ideas from scratch. In practice, and in retrospect, it's usually a combination of a couple things that I've seen when working with these folks. They are a) knowledgeable about a lot of

things (places to draw ideas), b) their solutions are elegantly simple (frustratingly so), and c) their solutions are further out (bigger) than everyone else’s. If you find one or more of these people in your organization, do what you can to keep them happy and learn from them.

Note 1: These buckets have nothing to do with formal education. There’s no implication that a higher level of education correlates to a higher level of capacity. However, more education might mean a broader and/or deeper set of specific skills.

Note 2: These buckets are not meant to imply that people can’t move from one bucket to another – they can and do. They often move buckets when they become more or less interested in the work they’re doing.

Note 3: There’s immense value to be found with people in each bucket – it’s just a matter of applying their capabilities properly.

In conclusion, one of the things I find myself recommending most often is “think”. Don’t just do something, think about why you are doing it and if that’s what you should be doing. This concept is universal and is not restricted to security or even technology in general. Part of thinking is being creative and doing the hard work of problem solving. There are ways to work towards bringing out your creativity and you can exercise them and get better at them. Lastly, people generally fall into a few buckets when it comes to their “thinking” tendencies at work. Recognizing these traits and using people to their strengths can help you get the most out of your people.

References

———–
http://www.ted.com/
John Cleese talk on Creativity

Week 51 - Document Everything

What is it and why should I care?
As I mentioned last week, this series is comming to a close. I also said that I have two concepts that I find myself sharing more than any others. The first I shared last week was to – Think.

This week I'll briefly cover the second topic … "Document Everything".

Again, this is a simple thing, but it is usually not done, especially by programmers. There are exceptions to the rule, but programmers are stereotypically bad at documentation. We generally either a) don't do it or b) do it once and never update it. The example I usually go to is – think about some of the better-documented open source projects you've used (spring, grails, bootstrap), and then think of some others that have poor documentation (won't name them :>). Consider which ones you generally enjoy working with more and how much of a role documentation has to play in that. Also, think about how long it took you to come up to speed on those frameworks and how long it might take you to refresh version changes, etc. Documentation can make all the difference in usability and desirability of a product.

What should I do about it?
Documentation has a reputation for requiring huge amounts of time and effort. Depending on what you are doing, that might be true. However, in practice, you can ease those burdens quite a bit if you follow a few simple steps:

1. Document as you go.
If you are writing comments for code, add them before coding or inline while you're writing the code. Don't commit the code without comments. If you're writing to produce end-user or other more formal documentation, write as you do the design, or while working on the component you need to capture documentation for. The process of writing will help you clarify issues in your mind as well as root out areas of uncertainty that you might have not considered before.

2. Write as simply and clearly as possible.
Don't add fluff. Write down the steps that are necessary and stop. Over time, you may expand your documentation, but don't start with lots of documentation that is unnecessary, especially since it's more to maintain over time.

3. Get a good proofreader.
A good proofreader is invaluable for documentation. Your editor will help you with some dumb mistakes, but your proofreader will catch others. Much more than that, your proofreader will likely think a little bit differently from you and help you find areas of your documentation where you might have implicit assumptions or unclear statements and help point you to areas that need work. Try to team up with someone else who is writing and trade proofreading tasks – reading for others will also make you a better writer.

4. Update your documentation on a regular basis.
Stale documentation is often worse than no documentation, because it's misleading to those who are trying to use it. You may be actively giving out false information. It's a good idea to consider what level of freshness your documentation needs, and then set calendar reminders to review and update the documentation on a periodic basis.

As long as you follow these basic steps, you’ll have a good start on producing solid documentation, and you will improve over time.

Documentation is a powerful tool. The reason I started this blog was actually because I needed to document programming concepts that took me some time to figure out and I wanted to remember them and not have to re-learn them. Over time, folks started finding the site and letting me know that they found them helpful as well and requested I write on different topics. Some of these requests came from co-workers and others came from strangers on the Internet. That was the genesis of this series, in fact – writing down what is essentially an FAQ from folks over the years about various topics related to Java security.

In conclusion, there are two mantras I repeat to folks constantly – think and document everything. Thinking means you should be coming up with the right (or at least better considered) solutions and documenting them means you are preserving both the solution as well as your discrete thoughts for posterity. By using these in tandem, you should be able to perform at a higher level, and share your work in a more meaningful way.

Week 52 - Never Stop Improving

What is it and why should I care?
Information security is a quickly growing field that is changing rapidly in many ways. We are tasked with securing all sorts of technologies and those technologies are moving quickly.

The implication here is that even to maintain the status quo requires significant work. However, we don't want to just maintain the status quo – we actually want to improve. How, then, should we proceed?

What should I do about it?
We have to work hard to consistently improve. Technology is a field that requires a lot of effort to stay current, but that effort pays dividends with experience. I recently had a conversation with a colleague where we discussed how quickly things were moving, but how similar the technologies were, particularly when you look across platforms or deployment models. There are certainly differences about mobile from the web, but there's a lot that's the same. If you account for web and desktop, there are even more similarities. Similar ideas have been bandied about when discussing the cloud and mainframes.

I certainly understand that there are nuances to most technologies that make them unique or valuable in some way (else they don't gain traction). However, it remains that exposure to different situations (experience) gives you a significant upper hand in this space. Below are a few thoughts about what I've seen to be successful with this approach.

1. Learn the fundamentals
This is the core of every good security person I know. Their quality is often directly reflective of their understanding of core principles. This is logical when you consider that we repeat technological decisions repeatedly. A good understanding of what we do and why we do it is essential to being a good security person over the long haul.

2. Work with lots of things
Try to get exposure to different technologies, platforms, toolsets, development methodologies, risk analysis techniques, etc. The more you see, the more you can build a mental framework around which to base your decision making. You see the components for what they are and how they fit together – a powerful piece of information.

3. Build a nice toolset
A natural extension to having a good grasp of the fundamentals and getting exposure to different things is that you build a solid toolset. You may be a specialist (that's great), but work with others and try to understand what they do. That knowledge lets you further understand your role in the process and gives you a way to add even greater value.

4. Look for novel solutions
Even though it's rare, there are good new ideas that come along. Many of the best ones in technology were generated in the 50's and 60's, but there are still good new things that come up all the time. Be on the watch for good ideas that can fundamentally improve how we secure systems. As one friend put it, look for things that make it cheaper for us to secure things than it is for the bad guys to break things.

In conclusion, security (along with technology) is quickly evolving in the particulars, but pretty steady in the fundamentals. In order to improve the overall security of our systems, we need to stay ahead of that curve. We can do that by having a solid understanding of the basics, getting exposure to different tried-and-true techniques and solutions, and then finding those new solutions that move us forward. Following these steps we can make sure that we are moving the field forward and that we never stop improving the security of our systems.

Process

Week 13 - Know Your Frameworks

What is it and why should I care?
Libraries and frameworks are a reality for every J2EE developer (pretty much any developer, actually) out there. We use them for MVC, DB, logging, web services, security, XML processing, as well as a host of other features. We rely on them in our production apps every single day. All this code that's been written by someone else. Code that likely hasn't been internally vetted. Code that likely hasn't even been looked at. Yet, we still use these masses of code (generally MUCH larger than the custom code written for the app itself) to add functionality to our applications.

Knowing your frameworks means you don't accept the code blindly. When you include a piece of software in your application, you've inherited and are now responsible for it. From a functionality perspective, you fix it when it breaks. From a security perspective, you are now responsible for dealing with it's vulnerabilities. This is the crux of the problem: we manage a LOT of code now (code we didn't write) and are responsible for making sure it is functional and secure: no easy task.

What should I do about it?
There are many things you should do when dealing with frameworks. I'll cover the two I think are most important.

First, you should patch your frameworks when new vulnerabilities are found. This is a significant effort because it requires much testing and coordination to upgrade frameworks within applications. However, there have been significant vulnerabilities found in extremely popular libraries, and that necessitates patching. Sometimes, patching can be done without upgrading the library. It could be moved off to a WAF or some such product. The point is you need to prevent the vulnerability that's been exposed.

Second, you should really know and understand how your framework functions. While most frameworks patch vulnerabilities reasonably quickly (especially if the vuln public knowledge), they will often not patch their "design decisions". These are often architectural patterns that benefit functionality, but not security. One popular pattern that comes to mind is auto-binding / mass assignment. The technique of populating the model using request data is not new, and is very powerful. It can make code much easier and cleaner to write. However, it's often implemented with no security at all. The best you'll usually get is an opt-in mechanism for securing it. However, most people are not going to opt-in, so it will be used insecurely in many cases. Patterns like this are frequently seen in modern frameworks, and developers really need to be aware of what's going on internally in the framework to understand how the security and functionality of their application is going to be affected.

Frameworks are a necessary piece to most any development work going on today, but blindly trusting them is not. Be aware of what the frameworks you're using do and how they do it. Keep an eye on them and patch them as necessary. This will help manage the risk of using them in your applications.

This post turned out to be very timely. Aspect Security just put out a nice paper (sorry, behind registration wall) on some analysis they did regarding the usage of java libraries through the maven central repo. They analyzed 113 million downloads and found that 26% of those downloads have known vulnerabilities! That's a significant number. Their analysis doesn't say whether or not those downloads were followed by requests for the patched versions, but I would bet not.

References

————

https://www.aspectsecurity.com/blog/the-unfortunate-reality-of-insecure-libraries/

Week 19 - Reduce The Attack Surface

What is it and why should I care?
Reducing the attack surface of an application or system means reducing the ways that you can interact with the application, and may involve reducing the functionality the application provides.

To most business folks, this sounds very, very bad. However, at its' core, it's really just a matter of simplifying the system. This is a really *good* thing to the business. Rarely do you find anything that people genuinely enjoy using that is complex. The best designs are simple, and that benefits us in this case from a security perspective.

What does this simplification look like? My favorite example is the difference between Google's standard and advanced searches. It's likely that 99.9% of people don't need the advanced search features. Imagine if google removed that page – that greatly simplifies the "search" application they build. It reduces the application footprint, saves them money (dev, support, etc) and gives their customers a better experience – what could be better? (Note: I'm simplifying this case as google's standard search does allow advanced operators, but you get the idea.)

Most developers I've worked with (myself included) have the tendency to a) want to build lots of cool stuff, and b) be poor designers. This results in designs that are larger than necessary in that they encompass more code than planned (feature creep). It also results in an often unpleasant user experience. By being ruthless in removing non-required functionality, and simplifying what is required, the user experience is enhanced along with security, not to mention the bottom line – time and money.

What should I do about it?
Saying you should remove features/functionality and simplify is a bit vague, I realize. I'd like to offer a few examples of common situations where you might be able to have some impact on your applications for the better.

1. Dead Code
Every modern IDE has a "dead code" detector. If you don't use an IDE, tons of open source "code quality" tools have this feature as well. Use it. If you're not using code, remove it. If you comment out code, but keep it in the code-base, stop. Remove it. Heck, you can get it back through your version control if you ever really need it.

As much as you can you should also remove code that is "dead" because it's not enabled via configuration. This may not always hold depending on the specific circumstance, but if you don't have a need for a feature, don't have it in your code base.

Dead code doesn't get looked at or dealt with as closely as "live" code, so that makes it even worse from a security perspective, as there are likely to be lingering issues that aren't dealt with because "no one is using that".

2. Copied code
Everybody's done it. You've taken code from an old project and used it in a new one. You've taken an example from the web and plugged it into your app. It may have done more than you needed, even way more. What have you done? You've added extra code that has be maintained, debugged, supported, tuned, secured, etc. This is a bad idea. It's fine to use others' (assuming they're ok with it) code, but don't add a bunch of stuff you don't need.

3. Extra features
It’s undoubtedly great to wow your customer. In my opinion, adding unplanned features is usually not the best way to do that. Usually, giving them the absolute best version of what they need is much better for both you and them. It’s the idea of doing a few things well as opposed to lots of things just OK. Adding in extra features is a common thing for developers to do, often because they saw some cool thing somewhere and thought “hey – that’d be cool here”. Again, adding extra features means extra code, and that’s more to do, and takes away from the quality of what you actually need to do.

4. Extra code – 3rd party libraries
3rd party libraries are great. They are core to most any development done today. They enable us to create more functional apps quicker. However, they also put into your application TONS of functionality and features you may not have planned on being there, and that you probably don’t know exist. I would venture a guess that most J2EE apps I see probably include hundreds, if not thousands, of times more code in 3rd party libraries than in the code written for the application. That’s great from the perspective of “I didn’t have to write this”, but could mean danger when it comes to securing your application with those frameworks. From a security perspective, it doesn’t help that most of these libraries are there to have things just work instead of having secure defaults. I’m not saying frameworks are bad; I’m saying you need to know their capabilities well, and have a plan for dealing with them from the security perspective.

5. Extra services enabled
This is particularly common with 3rd party applications, but can be true for custom apps as well. What happens is an application is built in a generic way, and then sold/used by several groups or companies to solve different problems or similar problems for different users, etc. The functionality in the app is the sum total of what all the customers need. You as an individual customer might only need 30% of the overall functionality, but you have 100% enabled. That’s a problem. The better apps give you a simple way to disable features you’re not using, and a simple way to verify it’s actually turned off. Use these features. It’s always better to have to do an update to enable a feature than to have to tell your boss you were hacked using a feature that wasn’t even needed.

The above represents just a handful of ideas on how to reduce attack surface in your application. They all really boil down to simplify, simplify, simplify. It helps your application be better, and thankfully helps your security be better as well. Next time you have a bug-hunting session, try some of these ideas out. Also add comments if you have more/better ideas.

References
————
http://www.jtmelton.com/2012/03/29/year-of-security-for-java-week-13-know-your-frameworks/

Week 20 - Trust Nothing

What is it and why should I care?
While trust spawns interesting philosophical discussions, here I want to discuss the implications of trust within the applications we build. Trust is a funny thing in that we implicitly give it frequently without considering what we're trusting. A simple example:

```
//bad bad do not use
executeDbQuery("select * from my_table where id = " +
request.getParameter("my_id"));
//bad bad do not use
```

Here we've said that we trust that the user of the application has not tampered with the *my_id* request parameter in any way that may cause problems for our application. Obviously this is a poor assumption. We can do better by moving the above query to a prepared statement with parameter binding to prevent SQL injection and we can also validate the *my_id* parameter for appropriate input, but why do we do that?

It's because we don't trust the input to our system. We don't (and shouldn't) trust that a user or system is going to use our application in the way we would expect, or even the ways we've thought of necessarily (a good reason against blacklisting for security). We must build systems that not only are functional (use) but stand up under attack (abuse) or ignorant usage. Our systems must be robust or as some have called it,rugged. Whatever your term, the idea of trust is either explicitly or implicitly central to the idea. We can't trust the environment.

If we can't trust the environment, what does that mean? Does that mean we deal with XSS and SQLi? Yes, but much more than that, it's a different way of thinking about the application. It becomes that simple picture of input-processing-output at varying levels of scope. A single request has inputs (request parameters, headers, database input, etc.), processing (authn/z, logic, etc.) and outputs (DB, screen, file, etc.). The application as a whole has inputs, processing and outputs that are essentially the combination of all the individual components of the application, and then you can scale on up to systems and organizations.

The "environment" I'm referring to changes depending on your specific situation, and it's difficult to say that you simply can't trust anything, because that's usually a non-starter. You may have to trust your configuration files or your external SSO system, or any number of other entities. The idea is that you specifically label those things as trusted (an assumption) and treat everything else as being tainted.

These types of issues are considered in threat modelling, which is another planned topic in this series. For now, it's sufficient to simply note that you should be thinking in terms of what data am I taking in, processing and sending out?

What should I do about it?
Now that we've established the environment can't be trusted, the next logical question is what constitutes the environment?

This could be a long answer depending on your setup, but a decent starting list for web applications in particular might look like the following:

- web request data (parameters, headers, body, cookies)

- database data
- directory data (ldap)
- filesystem data
- web service data (any data in headers or body)
- external system data (any data you receive from another system – software you're integrating with)
- network connection data (any data you receive while acting as the "server" – generally socket-based communication)
- user input (command line input)
- system environment variables
- third party software (libraries that you call that provide you data)

This list is incomplete I'm sure, but the idea is there. Any data you receive from any of these users or systems is generally untrusted, possibly with certain organization/application-specific well defined exceptions. When you start to view your applications in this way, you start to build better protections around them. You build better defences, and better logging/auditing so that you can detect when something actually does break (it will, I promise). However, thinking in this way can go a long way to helping you build safer and more secure systems.

References
———–
http://www.ruggedsoftware.org/
http://www.schneier.com/book-lo.html
http://en.wikipedia.org/wiki/Robustness_principle
http://www.jtmelton.com/2012/05/01/year-of-security-for-java-week-18-perform-application-layer-intrusion-detection/
http://www.jtmelton.com/2012/04/10/year-of-security-for-java-week-15-audit-security-related-events/

Week 24 - Use Static Analysis

What is it and why should I care?
Static analysis is the analysis of software that is performed without actually executing programs built from that software. Essentially, it's automated inspection of source code. There are varying levels of complexity achieved by the different static analysis tools available. I will roughly group them into a couple of buckets: grep+ and data/control flow analysis.

Grep+
Grep is a great tool and you can do a lot with it, but it's not really meant for serious static analysis. The earliest tools started here, but found out it wasn't the best idea. You can certainly do simple things like "calls to this function (strcpy) are bad". You can add in regular expressions and get a little better, but it usually gets unwieldy pretty quickly. There is some marginal value to be found here, but not a lot. While there are still some tools available today that work this way, most of the useful ones have moved beyond these techniques.

Data/Control Flow Analysis
Data Flow Analysis and Control Flow Analysis are (generally) the current standard techniques for the more advanced static analysis tools. These concepts are not new, but rather originated in compiler theory, specifically related to optimization techniques (Dragon book, anyone?). While the concept is not new, it took quite a while for it to be used heavily outside of compilers.

In general, these tools build up a data structure model (referred to as the AST – Abstract Syntax Tree) of an application, and then traverse the AST in various ways using different types of analysis to look for issues. There are lots of different types of analysis that can occur at this stage, but these tools generally use this as the model on which to base their analysis.

We as developers actually use these concepts constantly in the form of our IDE's. These tools load up an AST and constantly do checks and will give you warnings and errors that are based on static analysis. Some IDE's even include AST-driven security-related checks out of the box.

Note: While static analysis has no actual ties to security per-se, I'll be referencing it's use with respect to security since that's the topic here. However, just note that these techniques are useful to solve general analysis problems, not only security. Also note static analysis specifically used for security is often referred to as Static Application Security Testing (or SAST) in the industry.

What should I do about it?
Beyond what you get in your IDE, you should use security focused static analysis as and embed it as part of your development process. I usually see tools deployed either in a single execution mode (collect your code and push "run scan now") or in a continuous integration mode (scan automatically kicks off every time code is committed). Your environment may dictate that to an extent, but if your tool is good (and/or you've tuned it properly), then using it in continuous integration mode is a big win since you'll find the issues earlier in the cycle and can often address them much easier.

I won't venture into the debate about which product is better than another (especially given I currently work for a vendor), but I will say that all of them have trade-offs (like any tool), and that you should consider the tools carefully before including them in your environment. If you're a Java shop, and want to get started (for free !!!), then I'd suggest taking a look at the excellent FindBugs from the folks at UMD. It's a great tool to use and to learn how static analysis works if you're interested.

A couple of caveats related to static analysis, specifically those tools that use it for security:

1. Static analysis tools do get the (sometimes well-deserved) bad rap that they produce too many false-positives. My experience with these tools is that you usually need to tune them (either yourself, or pay for some help) to get good results for your environment. This is an additional investment to consider, but can drastically improve your experience with the product. I mention it because I think it's helpful to know going into the process.

2. While SAST tools are good at finding issues, they don't find them all, and they're not a replacement for testing your code.

Finally While static analysis doesn't solve the security problem, I hope I've shown it is a good tool to have in your tool-belt when it comes to securing your applications.

[Full Disclosure] I currently am employed by a company that just released a static analysis product, which I work on. However, I can certainly say I recommended the use of static analysis tools before joining and will continue to suggest their use in the future irrespective of my employer.

References

————

http://en.wikipedia.org/wiki/Static_program_analysis
http://en.wikipedia.org/wiki/Data_flow_analysis
http://en.wikipedia.org/wiki/Control_flow_analysis
https://blog.whitehatsec.com/mythbusting-static-analysis-software-testing-100-code-coverage/
http://swreflections.blogspot.com/2012/01/static-analysis-isnt-development.html

Week 25 - Use Dynamic Analysis

What is it and why should I care?
Dynamic analysis is the analysis of computer software that is performed by executing programs built from that software system on a real or virtual processor. Essentially, it's automated execution of an application.

Note: While dynamic analysis has no actual ties to security per-se, I'll be referencing it's use with respect to security since that's the topic here. However, just note that these techniques are useful to solve general analysis problems, not only security. Also note dynamic analysis specifically used for security is often referred to as Dynamic Application Security Testing (or DAST) in the industry.

So, how do dynamic analysis tools do what they do? In the world of web application security (admittedly a constrained subset, but the topic of focus here), it's building something akin to a special-purpose web browser that attempts to attack the running application by probing for vulnerabilities and detecting based on some output heuristic whether or not the attack was successful. For example, with XSS this is logically as simple as:

```
Step 1: Go to page with form.
Step 2: Fill in field of form with javascript alert.
Step 3: Submit form.
Step 4: If response has a javascript alert, we have an XSS vuln.
```

This is certainly a simple example, and real scanners are quite complex in what they can do. However, logically this is the basic concept.

Dynamic analysis, as opposed to static analysis, has the added benefit of proof of exploitability. Many times the results of static analysis are either wrong or questioned because "well, the live system has security control X that prevents that". In dynamic analysis, you're generally testing the live environment, or at least the testing environment which is meant to look like the live environment. When you show someone a vulnerability found by actually exercising the deployed site, it's hard for them to argue that it's not exploitable.

What should I do about it?
You should use dynamic analysis as part of your development process. These types of tools are often executed in the QA and/or user/business testing environment. You should also get these going in whatever other environments you can, such as the continuous integration environment (have a task to build/deploy the site, then scan it), the integration test environment, QA, etc. The earlier you get these tools executed, the cheaper it is to resolve the issues they find.

I won't venture into the debate about which product is better than another (especially given I currently work for a vendor), but I will say that all of them have tradeoffs (like any tool), and that you should consider the tools carefully before including them in your environment. If you want to get started (for free !!!), then I'd suggest taking a look at skipfish from some folks at Google. I've been told it's pretty good, and the codebase is relatively small so you could learn about how it works pretty easily (It's a C project by the way).

Finally, while dynamic analysis doesn't solve the security problem, I hope I've shown it is a good tool to have in your tool-belt when it comes to securing your applications.

[*Full Disclosure*] I currently am employed by a company that provides a service related to dynamic analysis. However, I can certainly say I recommended the use of dynamic analysis before joining and will continue to in the future irrespective of my employer.

References
————

https://www.owasp.org/index.php/Dynamic_Analysis
http://en.wikipedia.org/wiki/Dynamic_program_analysis

Week 26 - Do Code Reviews

What is it and why should I care?
Code reviews are an important process whereby developers have their code systematically examined by another set(s) of eyes in order to find defects. It's a simple concept (double-check my work), but surprisingly effective. Studies show that you can detect 20-75% of defects with code review (range varies widely depending upon level of rigor applied). That is incredibly powerful, especially once you combine it with other quality processes.

Note: Since this series is specifically about security, I'm going to consider security focused code reviews, but the term is used commonly to refer to the generic defect finding reviews performed by developers.

The only thing that differentiates security code reviews from standard code reviews is that we're looking for a specific set of issues. The same would be true of a "performance" code review where you only look for performance issues. This means that while a general code review might turn up plenty of poor programming practices that have no bearing on security, a security code review would usually ignore those. It's just a matter of focus, not of different methods.

What should I do about it?
If we believe the studies (and I do based on my own experience) that tout the effectiveness of code review, what do we do now? We follow their advice and do code reviews! Specifically, here are the steps I think have a place in the security code review eco-system.

Define the Plan
As with any process, you need to set goals. These are obviously specific to an organization, but a few examples might be:
- Reducing vulns found in QA/Prod (basically find issues sooner)
- Ensure compliance with corporate standards or legal requirements
- Train developers on security (can be done with the process of review by close interaction with development team)
- Enhance overall security posture

Do It
Quite simple, but not always easy. As with all other code review types, it's very common for people to either a) not do reviews, or b) do "rubber stamp" reviews. Neither are helpful, and both are common. Code reviews must be setup and executed on a regular basis to provide the desired value.

Don't Over-Do It
There's certainly a point of diminishing returns here. Reviewing low-value code is often not worth the effort, particularly from a security perspective (hint: reviewing java beans is usually a waste of time). It's often fairly obvious (assuming you understand the app) what parts of the application are security-sensitive, so focus on those areas. In addition, you can also add in processes to detect changes to the security-relevant portions of the code and trigger automatic code reviews.

Automate What You Can
There are tools like static and dynamic analysis that can help you find some classes of issues. Use them. Automation should be preferred whenever it's accurate and available. However, automation does NOT find everything. You also need to have manual reviews, especially of certain classes of vulnerabilities. This means you have to know the capabilities of your tools. If there's a coverage

gap, you'll need manual review to cover it. Balancing this manual/automated approach is the key to security with scalability. Different organizations are going to slide along that scale a bit, but it's important to know that the decision must be made, and to be conscious of the choices you're making and why you're making them.

In addition to using tools to help you find issues, there are tools that help you do manual reviews. You don't want to just read all the code. You'll want to look at diffs, view versioning comments, make notes for the developer, etc. and there are tools that help you do that, so use them.

Iterate and improve
Any good process has a feedback loop. Implement your first version, then evaluate what worked and what didn't. Then remove things that are broken, and try new things. The point is that you want a holistic process to catch security issues. I usually view it similar to unit testing. I try to think of every valid test I can, and add them to my test suite. This makes my code better than if I had no tests. Invariably though, I find a bug that covers something I didn't consider, so I add a test to cover that scenario, and now I've improved my code and test suite. That's the way it should be with the code review process. Honestly evaluate the value of the steps in the process, then a) keep those that work, b) throw away those that don't, and c) add to the process when you need to cover new issues.

I personally find that it's helpful to do an evaluation/review of the process every 6 months to determine if everything's still useful, and if anything needs to be added.

In conclusion, code review is a necessary process if you really want to improve your code. Security focused code review looks specifically at security related issues in the code, and gives you a double-check from another set of eyes. This process has a high likelihood of success given you perform it regularly.

References
———–
http://swreflections.blogspot.com/2011/05/not-doing-code-reviews-whats-your.html
http://www.aleax.it/osc08_crev.pdf
http://software-security.sans.org/blog/2012/06/26/different-ways-of-looking-at-security-bugs/
http://en.wikipedia.org/wiki/Fagan_inspection
http://www.slideshare.net/zanelackey/effective-approaches-to-web-application-security
http://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/
http://www.cc2e.com
http://www.codinghorror.com/blog/2006/01/code-reviews-just-do-it.html
http://en.wikipedia.org/wiki/Code_review

Week 27 - Penetration Testing

What is it and why should I care?

Penetration testing is a process of evaluating the security of a computer system or network by simulating an attack. The process involves an active analysis of the system for any potential vulnerabilities, is carried out from the position of a potential attacker and can involve active exploitation of security vulnerabilities. This is sometimes referred to as pen-testing or ethical hacking.

This means that you actually are probing the live system (usually in Dev, QA, or UAT) and trying to find (and sometimes exploit) actual vulnerabilities. There is tremendous value and power in being able to show not only that a vulnerability exists, but that it is directly exploitable. In my experience, it also opens up an honest dialog between development and security if you can show that something actually is exploitable, and approach it with the goal of getting it resolved together. There's no more conversation about false positives at that point :>.

What should I do about it?

Just as you should do code reviews in addition to using static analysis, you should perform penetration testing in addition to dynamic analysis. Static and dynamic analysis give you the ability to point software at your applications and get back results (often good). However, there is a limit to the amount of analysis that current products (or even theory, for that matter) can provide.

So, that leaves us with supplementing our tools with humans (the common refrain in most security efforts). By adding code review to supplement static analysis, we're able to find specific instances of vulnerabilities, and even whole classes of vulnerabilities we wouldn't have found before. The same is true with penetration testing. By supplementing dynamic analysis, we find issues that the base tools wouldn't have found. The tools generally continue to improve, but it's debatable (and actually is heavily debated) whether or not the tools are even keeping up with the pace advance in software. Whichever side of the fence you sit on, the current situation is that we need to add humans to the mix to get better coverage.

There are a couple of very helpful resources that I would be remiss if I didn't point out with respect to the penetration testing process. They are the Open Source Security Testing Methodology Manual and the OWASP Testing Guide. Both of these resoures are full of good (and thorough) information about both process implementation and integration within organizations.

In considering what to recommend with respect to the process of pen-testing, I came up with a similar process to my code-review list (so either the other list was good enough to work for both, or they're both equally bad).

Define the Plan
Processes should have a set of goals. Define these within your organization based on your needs.

Do It
Again, simple, but not easy. This is a human-driven process, so it's very common for people to either a) not do the tests, or b) to "rubber stamp" the tests. Neither are helpful, and both are common. This process must be setup and executed on a regular basis to provide the desired value.

Don't Over-do It
Again, we have the idea that you have the law of diminishing returns. Showing that something is an issue in a couple places and thoughtfully working with the developers to come up with a resolution

plan may very well be good enough. The developers can probably track down other instances of that pattern fairly easily.

Automate What You Can
You have to understand the capabilities and limits of your static and dynamic analysis tools. If there's a coverage gap, you'll need manual reviews or tests to cover it. Balancing this manual/automated approach is the key to security with scalability. Different organizations are going to slide along that scale a bit, but it's important to know that the decision must be made, and to be conscious of the choices you're making and why you're making them.

Just like a code reviewer is going to use tools to assist in the review process (like a good IDE), a pen-tester uses special-purpose tools to have more control over the testing process. While I'm not a pen-tester by trade, I know some folks that are, and the tools I hear referenced most often as being good are OWASP ZAP and Burp. Again, I can't personally vouch for either, but I really like the concept explained here regarding using ZAP for security regression tests, as this aligns nicely with my future article on testing.

Iterate and improve
This is, again, the idea of iterative and continual improvement. Come up with an initial process, try it out, then keep the things that work, and remove the things that don't. I personally find that it's helpful to do an evaluation/review of the process every 6 months to determine if everything's still useful, and if anything needs to be added.

In conclusion, penetration testing is a necessary process if you really want to improve the security of your applications. It allows you to supplement the analysis from the tools and perform in-depth human-driven testing. This process has a high likelihood of success given you perform it regularly.

References
————
https://www.owasp.org/index.php/Testing:_Introduction_and_objectives
http://en.wikipedia.org/wiki/Penetration_test
https://www.owasp.org/index.php/OWASP_Testing_Project
https://www.owasp.org/index.php/Category:Penetration_Testing_Tools
http://www.osstmm.org/

Week 35 - Solve Security Problems One At A Time

What is it and why should I care?
This article (and several of those remaining in the series) is not so much technical in nature, but rather deals more with processes related to security problem solving.

It's a fact of life in most development and/or security shops that there are those fire-drill days, and that is the case for security practitioners many times due to the "we have a security problem and the sky is falling … fix it" mentality. This course of action, however, doesn't lend itself to fixing things properly (root-cause analysis), and certainly doesn't allow for the methodical eradication of entire classes of vulnerabilities.

That is a problem.

In order to make a dent in the security problems plaguing the Internet, we can not solve problems as they come up (referring only to known attack vectors, certainly we don't know what we don't know). We have to get ahead of them. We can't fix problems when they get here – at that point it's too late. This brings to mind several ideas, like building security in from the start and looking at what others have done to solve their problems and using their good ideas in our own processes.

However, one issue I don't see addressed much in the security realm (though it does come up) is the idea that we're trying to tackle too many problems at once. Make no mistake, there's a lot of issues, and they all seem important, but there's generally a prioritized order for most situations – some type of risk ranking. If you get the appropriate stakeholders in the room, and make the possible security issues clear, there will be some issues that are clearly more important or impactful than others.

If we work from the assumption that we have ranked our problems in priority order, why should we be haphazard in our approach to their resolution? We absolutely shouldn't. Security tooling has approached this with the familiar red-yellow-green solution in many cases, which is fairly helpful if you can inform the tool what constitutes red-yellow-green in your environment. However, this otherwise helpful approach missed the point that solving many security problems requires an architectural solution.

Let's consider 2 examples from a different problem set – performance.

1. One problem might be a particular method that's reasonably slow and gets executed many, many times. In this case, you'd probably just go in, rewrite the method using some optimizations specific to the method, and be done with it. Instant performance increase, and very little fuss.

2. Another problem might be queries taking too long across the application. If we assume a relational data access layer, there could be lots of solutions. You might scale the database hardware somehow, swap out the database vendor, add caching either internal or external to the application, tune queries, or a handful of other things. The point is that many of these "fixes" involve significant software and/or hardware architectural changes, and you wouldn't think of making a decision on those nearly so quickly.

Some security issues (eg. session fixation) are pretty simple fixes, and you make them in one place and you're done. Others (sqli, xss, etc.) are certainly more complex and generally are best solved with architectural changes.

What should I do about it?
Hopefully I've convinced you that solving security problems fire-drill style is a bad idea and that many require a more rigorous approach, so how do we solve them correctly?

My recommendation to developers is that you approach them individually (I caveat this with "you need to fix the easy/terrible ones first" to knock out the true fire-drills). This means that you pick your biggest problem (calculated by some risk rating methodology) and try to a)eradicate the issue from your codebase(s) and b)make it as impossible as you can for it to ever happen again.

That can be daunting, but here are a few recommendations to get that process started.

1. Understand the problem.
Don't ever try to tell anyone how to solve a problem you don't understand yourself. You usually don't actually improve anything and you look foolish. This is a common problem in security, so enough said here.

2. Consider all use cases where the issue can occur.
Figure out the ways that developers can cause the issue, as well as any they might not be using yet, but will be soon. This gives you the breadth of functionality that a possible solution has to at least consider, if not account for. The goal is that you don't give developers an excuse to go around your solutions because "we need this feature".

3. Evaluate solutions.
This is certainly a broad topic with lots of possible tasks, but there are a few obvious ones.
- Distill the known "secure" approaches and their associated tradeoffs
- Look for known attacks against those approaches
- Decide on a single or hybrid solution (most of the time, building your own is the wrong idea)
- Try to find a good implementation that matches your chosen solution
- Follow the guidance to implement the solution properly

4. Institutionalize the chosen solution.
Once you have a chosen solution for your problem and a working implementation, you now need to make sure that is the solution that actually gets used. One approach that seems to work pretty well is the ESAPI model. Here, you build a set of controls specific to your organization that function as the "approved" solution for a given problem area. You also build appropriate documentation showing developers how to use it properly. This brings in all the benefits of code reuse, as well as the consistent application of security controls.

5. Add technology and processes for verification.
This is an important step that is often not done. After you've considered the problem, come up with a solution, and got people to use it, you need to make sure they keep using it. Again, this could mean a lot of things, but here are a few ideas to get you going:
- Get (or build) a tool that not only allows you to check if you're not doing something wrong, but that you are doing something right. This is probably going to be custom, but it's very cool to be able to see everywhere you're sending queries to a database that DON'T go through these 10 "approved" methods. That's a much more manageable problem.
- Add people and processes to cover areas where tools don't work. At the moment, software can't catch all of these things, but humans can if they have the time. By only requiring humans to step in and evaluate those areas that technology can't deal with, you cut down on the time requirement, and give folks a chance to focus on those human-only tasks where they're actually needed.

In conclusion, there are lots of security problems to be solved, and not enough time or people to solve them. However if we prioritize our problems, and then deal with each one thoroughly, we can create significantly more secure applications consistently.

References

————

https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
https://www.owasp.org/index.php/OWASP_Security_Blitz

Week 39 - Don’t Reinvent The Wheel (Unless It’s Square)

What is it and why should I care?
This is a bit of a follow-up to my last post with a bit of a different viewpoint. In that post, I specifically looked at code reuse from the perspective of creating an internal framework to centralize code related to security functionality.

This week, I want to consider security a little more generally. Code is not the only thing produced by “security” mechanisms. There are processes, documentation, code, people, etc. Re-inventing each of those per-application or even per-organization is a huge waste of time and money. In addition, for the same reasons that I mentioned regarding code reuse, each individual incarnation (say, by a given company) of a given mechanism is likely to be of a lower quality than if multiple organizations pooled their knowledge and/or resources to produce a common template.

By looking for areas of commonality and building tools, processes and documentation to meet those needs, everyone benefits.

What should I do about it?
Essentially, don’t reinvent the wheel.

- If you’re looking to teach your developers (and/or yourself) about how to prevent XSS, don’t start writing your own doc, look at resources that already exist like the OWASP cheat sheets. The OWASP material is really good on XSS. In some other areas, OWASP’s documentation is weak, but you can often find other good resources, and take a best-of-breed approach.

- If you’re building out a plan for how to add security to the SDLC in your organization, don’t try to roll your own. At least evaluate existing models and see if they fit your needs. Chances are they cover most if not all of what you need. Additionally, they probably have thought of things you haven’t considered yet, particularly if it’s an established methodology.

- If you’re considering which, of all the important security issues, you should cover first in your application, look at existing analysis. Consider breach reports and what’s being exploited. Use the data that’s available, and don’t just start with what you *think* is true. Measurement of the data will often quickly change your opinion.

Here are a few concrete steps to prepare you for working in this way:

1. Assume you’re not the only smart person around
Let’s go out on a limb and assume there may have been others that have come before you that have had an intelligent thought or two. It’s OK (excepting certain license/copyright issues) to reuse what others have done. It’s OK if it’s not all done in-house. Reuse what you can from others, and then use your time to make improvements instead of starting from scratch.

2. Read, read, read
In order to know what others have done, it’s really beneficial to be knowledgeable about what’s current in your field. This obviously applies to many fields, but technology and security change quickly, so you must stay current.

3. When you have a need, do the appropriate research

When you have a specific need, you should have a general idea of what's available (if you followed step 2 above) to meet that need. However, you'll probably still need to do a bit of directed research to get the detailed information and see what will fill the current requirement you have.

4. Tailor the mechanism to your environment
While it's great that there is a lot available for reuse if you look for it, there are often good reasons to tailor the solution to your specific environment. Often times, you may need to change terminology, or you might have a simpler or more complex model to work from. You might have different threats, or you may have solutions that prevent entire classes of issues across the board. Tailoring a solution makes it specific to your organization and can add a lot of value.

5. Fix things that are broken – sometimes the wheel IS actually square
Sometimes what's out there is bad … sometimes it's really bad. When that's the case, you'll have to start from scratch. You may be able to glean some things from what's out there, but you may end up needing to do some or all of the work. This is obviously not the greatest option for most situations, but is sometimes necessary. If this happens, try to build something that is as reusable as possible – you never know how others might use your work.

6. Contribute back to the community
Whether you have something you tailored, or if you had to build something from scratch, you can often help others by sharing your work with the community. Obviously, this is not always possible, but when it is, it can be a boon to you and the community. If you put something out that gets used, then you could be a) boosting your career prospects, b) creating a standard tool/process/document, and c) helping others by sharing your insights.

7. Build a custom holistic plan
The real benefit to an organization of the "thinking" of its' security people is that they can evaluate the environment that the organization operates in, and build a working model that custom-fits the organization using best-of-breed individual solutions. If you don't need something, throw it out. If you have a gap, fill it. That's tremendous value-add to the organization. By knowing both the general landscape as well as your specific environment, you can build a tailor-made plan that evolves over time and is best suited to secure your organization. This plan should be holistic in nature, and will certainly fill out over time, but you should begin your planning with your end-goal in mind.

In conclusion, there are lots of existing resources at the disposal of the security practitioner. If there's no resource to meet your specific need, you can often tailor something that already exists, or in some cases, you may have to build one from scratch. Either way, sharing your work with the community can help us all move forward.

Week 42 - Break Something

What is it and why should I care?
Breaking something (legally, of course) is one of the best ways to learn how it works. Software is no different. Breaking software is sometimes trivial and sometimes extremely complex, but either way is a great exercise. In particular for developers, it forces you out of the mindset of building, and gets you to think about how your software might break. *It also brings you to a harsh reality of securing anything: the protector has to secure every avenue of attack, whereas an attacker only has to find a single path unprotected*. The scales are a bit unfair in that respect.

In software testing, we generally look at blackbox or whitebox testing. Blackbox testing is testing externally, essentially focused on the inputs and outputs. Whitebox testing is testing internally where you know the internal properties of the software. Both are extremely useful, and I've discussed both in previous posts. However, for the sake of our discussion in breaking something, we're usually talking about some form of blackbox testing.

What should I do about it?
What I generally recommend to people learning about a new type of attack or vulnerability is to do 3 things (in a loop):

1. Build it
Let's say you were wanting to learn about SQL injection. In order to "break it", you have to have something to break. The first thing to do is go out and collect or build code samples that perform SQL queries. Some may be susceptible, and some may not, but you'll need code to get started. If you're talking about some other attack, or even an SQL injection attack through an application, you will need some type of running site. That means setting up local servers and applications, etc. There are lots of vulnerable applications available (ala webgoat) in order to practice your skills.

2. Break it
Once you have an environment, you can begin to try and break what you've created. Start simple and get a boost from exploiting something trivial. Work your way up to something more difficult. As you learn more techniques to exploit individual vulnerabilities (think of all the XSS options!), you'll want to try some of them out. A tool with progressive "lessons" like webgoat can be great for this reason. It can be enlightening (and frightening) to see your application fail in so many spectacular ways.

3. Secure it
Lastly, you'll want to work on building protections. It's great to show something can be broken, but that's not enough. Remediation can be very complex depending on the issue, but is always necessary. Also, you'll find that many of your attempts at remediation are either lacking in completeness or affect functionality in other portions of the system or both. Building appropriately scoped controls is a difficult but necessary task. The good news here is that there is a lot of good information available to help with this, but the bad news is that it's not in common developer guides. Most of the time, you have to go to a security-specific resource to get this information.

Here are a few extra thoughts around breaking software

- Keep your test cases around
It turns out to be really helpful to have a collection of small sample code snippets that express a single issue in a contained way. I end up using my rough collection of vulnerable code in lots of different unexpected ways. There's no reason to throw away work you've done anyhow. Also, it gives you that

warm feeling of fright when you see code you wrote from years ago.

- Sometimes breaking something is the only way to prove it can be broken.
This is true in a couple respects. First, sometimes the only way to prove to a developer that they're writing insecure code is to show it breaking (but make sure you're doing this legally :>). Second, while it is sad, this is often a helpful way to get non-technical folks on board with security. If they see something break, something just clicks for them. I've often seen this technique used to help justify security investment, though I'd argue that there are better alternatives for that.

- Try a bug bounty program and win!
Over the last few years, a number of organizations have come forward with bug bounty programs, essentially a financial incentive for finding bugs in their software and then being "nice" by reporting the bug to them and letting them fix it before exposing it to the world. This is an interesting concept for a lot of reasons, but for the person interested in learning to break something, it can be a great option since you can get paid if you find something.

In conclusion, breaking software can be a great way to learn how it works, and can be a crucial link in the effort to secure it. Go forth and break!

Week 43 - Build Something (And Give It Away)

What is it and why should I care?
This will admittedly be a short post because it's a pretty simple concept. Here's the simple idea in bullet form:

- Developers are builders of software (and security systems and even documentation sometimes)
- There is a need for software & docs
- Developers build software & docs and contribute it to the community
- Developers help others in the community, give back, and build credibility.

It's admittedly sometimes a hard sell to ask someone to come home after work and do more of the same. It's certainly not for everyone. However, we as developers certainly have a useful skill that can help others out if used properly. There are many drivers for why to do this, as well as benefits which I'll list below. A couple personal reasons for me are that a) I can, and others can't. It's a good feeling to help others in need, and b) I was helped when I couldn't. I received a lot from the community both in the way of software that I used as well as mentoring. It's nice to give back in some way.

What should I do about it?
So, if you buy into my notion above, here are a some ideas for a few concrete things you can do:

1. Build (better) documentation for an existing tool.
Lots of great projects exist that aren't being used or are chastised for being bad when they really just need documentation. Taking an existing tool and building some useful documentation (Spring framework and Twitter Bootstrap docs are great examples generally) can be a really helpful thing to the community. This is also a great way to get your feet wet with a project before jumping into coding.

2. Find some bugs in a project and patch them.
This is along the lines of my last post. Figure out how to break something, then do a responsible disclosure, and better yet – fix it! Developers will be grateful for the patch. If it's a project you're interested in working on, it's a good way to show the developers you write good code, and to get involved in a helpful way.

3. Write an open-source security (or other) tool
Write a tool that scratches an itch either for you or the community at large. It's a great use of your skill, you're likely to learn a lot and you benefit others as well as help secure software. It's tough to beat those benefits. You can use your own distribution channel or you can do something like make it an OWASP project if you want some good visibility and prefer to have it in a community ecosystem.

4. Do something with charity.
You can use your skills and donate them to charities. There are lots of things you can do in your community or around the world. Many find this to be a rewarding experience, and it can provide new opportunities for people and change communities for the better.

Projects like these also have a benefit that you build up a portfolio of work that you can share – a bit of a virtual resume. This is a concept that's become pretty popular in tech circles in the last few years with the so-called "social coding" movement. While there are lots of business and likely financial rewards for doing some of the things listed above, there are also a lot of intangibles, and those are what I see as the real benefit.

References

————–

http://www.hackersforcharity.org/

Week 44 - Follow A Secure SDLC

What is it and why should I care?

Software development has taken an interesting path over the short lifetime of the field. It began as a deeply technical field where only the best and brightest could participate, which is not unusual since it was born out of engineering, a very technical and structured field itself. However, as the field opened more widely to the general population due to the Internet as well as widespread access to computers and simpler programming paradigms, the barrier of entry was significantly lowered. There's recently been some educated guesswork that places the number of java developers in the world at 6-10 million. That's just one (albeit popular) language. In addition, the newer development platforms (web/mobile) and reduced time to market (days/weeks instead of months/years) have made the field even more popular and populous. Add to these points the fact that the web in particular was built to be open and that most developers haven't had significant security training, often including even those building platforms and languages.

From a software security perspective, that's a challenging environment in which to function: simple and accessible languages/frameworks that are fairly insecure being used by novices, or even professionals, with little to no security training. Historically in software, security has been an afterthought – we've done a poor job overall with basic concepts such as validating input, encoding output, authentication, access control, etc. As for more esoteric issues, we have typically engineered solutions well after the attacks are discovered and known (this is partially acceptable since "you don't know what you don't know"). However, we also seem to have very bad memories, and tend to re-introduce the same weaknesses of design repeatedly (see web->mobile).

But there is hope! I have no doubt that the software security field is going to be active for a long time (lots of problems), but I am a firm believer that we can make significant improvement, especially within our sphere of influence. What it requires though is baking security into the way we build and deliver software, our software development life cycle (SDLC). Many of the posts in this series (and on this blog generally) are point solutions to specific problems. However, securing the SDLC is far more broad and reaching. The SDLC is just the way you build software, the set of steps you follow to get from thought or need to working software to retirement of the product. Examples you've heard of are agile, waterfall, spiral, etc. The point is that whether yours is heavily structured or very ad-hoc, you follow some process.

The simple idea of securing the SDLC is that you modify your current process (or adopt a new one) in a way that accounts for security. The truth is that if you want specific attributes out of something you build, you must plan for those attributes before you build it, or the re-engineering effort is not acceptable. If I want a chair that rocks, but I just start cutting pieces of wood in the shape of a chair that I've seen before (copy-paste code from stackoverflow or previous projects), I'm going to end up with something like a chair … but not one that rocks, and certainly not one that rocks smoothly.

What should I do about it?

We want to build secure software, and we know we have to plan for it. Now, we need a process that allows that and helps us plan for that. Luckily there are several open source popular models that you can consider to help you get started. All of these should be customized and tailored for your environment, but they are very good at giving you ideas of areas of consideration. It's a good idea to get familiar with several, so even if you use one as your basic model, you can borrow from others to create a plan that works in your organization. While there are many good resources (the US CERT catalogs several), a few of noted interest are the Microsoft Security Development Lifecycle (SDL), Software Security Framework – Building Security In (SSF) and theSoftware Assurance Maturity Model

(OpenSAMM).

All of the models (including those I haven't mentioned) have strong and weak areas, but I personally like these three a lot, and for different reasons (though they're all reasonably similar, when push comes to shove). The Microsoft SDL has great documentation and lots of great tools and worksheets ready-built to help you get going. The SSF-BSI is grounded with real data (openly published, see BSIMM) and is extremely logical and simple and clean. The OpenSAMM is great because it shows very clearly how to extend and customize it for your environment. You get a great idea of the options available, and you can make an informed choice of what steps work for you.

It is actually pretty rare in our field that we have so many quality options that are open and available to solve a certain problem. In this case, we're fortunate that we have options. Make sure that whatever route you choose, that you follow it, and certainly improve over time. If you find a hole in your methodology, fill it in with a practice so it's no longer an issue.

In conclusion, software security is a long and arduous journey. Building secure software is no easy feat, but it helps tremendously to have a plan. Fortunately, there are several quality options for SDLC frameworks that account for security. Read about the different models to get ideas before you start – some cover certain areas better than others. Build a model that works in your environment, and by all means, use it!

References
———–

https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc/326-BSI.html
http://www.microsoft.com/security/sdl/
http://www.swsec.com/
http://www.opensamm.org/
http://bsimm.com/

Week 45 - Do Threat Modeling

What is it and why should I care?
After the last post covering secure the concept of a secure SDLC, this week we'll look at a specific activity recommended by the various secure SDLC models: threat modeling. From the view of the secure SDLC, this is an activity that takes place fairly early in the cycle.

Threat modeling is an exercise intended to improve the security of your application(s) by considering the attacks that a threat can perform against your specific application. That data is then used to inform the development of controls and tests to verify the prevention of those attack vectors.

There are various components in the above definition that will be discussed below. However, given the short post, I'd like to make a "further reading/watching" suggestion up front. If you're interested in the topic, please go look at the resources. I even included an excellent video of jOHN Steven talking about threat modeling. It's worth an hour as it's a good overview. jOHN is an expert on threat modeling and has a wealth of experience in making it useful and practical – his wisdom has certainly heavily influenced my own views on the topic.

What should I do about it?
A few initial caveats:
- This is fairly thin coverage of threat modeling as it is a complex process. Many complete books have been written on the topic.
- There are various models of threat modeling. I am attempting to portray the basic process only. You should use the model that most closely aligns with your organizational needs and then further tailor it to your specific environment.
- Along with various models, there is a wide array of terminology used to describe various components in threat modeling. A good example I like is the work Cigital has done on producing a useful vocabulary.

With those initial points out of the way, what do you actually do when threat modeling? Here are a few steps:

1. Identify what you want to protect
This should already be understood in your organization, but I mention it as a predecessor because writing down what is important can be a useful activity. One stakeholder won't often have all the information, and this process is really helpful to get everyone on the same page from the get-go. It also helps you determine what's worth protecting. This process should involve assigning a specific (or at least relative) value to your assets in order to determine priority.

2. Consider your application
Look at what you want to build, are building, or have built (depending on where you are in the life of the application) and make a picture of that (pictures are pretty and easier to understand). Do NOT use a network architecture diagram (too generic) or a set of UML class diagrams (too specific). My personal rule of thumb (informed by work of others) is that your picture should have 2 basic criteria. It should be specific to this application (the diagram of 2 similar apps in your portfolio should still have different components and therefore look somewhat different), and it should all fit onto 1 page in reasonably readable form (helps keep complexity at the right level). This diagram is not the only one that will exist. You can layer additional data on top of it with different views or you can expand certain subsections for more data, but there should be a single-page overall view. This view should include application-specific information, such as the software components along with design patterns used,

frameworks used, etc.

3. Add more views
In addition to the basic view of your system, you'll want to start to annotate your view with additional information. You can add data like:
- entry points: Think about areas of your application that can be invoked externally. These are important in that they provide attack surface so should be carefully considered.
- trust boundaries: These are areas where you make a decision about levels of trust granted to a given component. See the jOHN Steven video for an example of how trust boundaries aren't necessarily so cleancut.
- data flow: Consider how information flows through your system. Are there areas where data can flow in different paths depending on conditions such as role or other context? Think these through and map them out.
- critical functionality: The parts of the application that are most important deserve a look and some thinking about how they are different. Are there actually controls in place that make them different. Should there be?

4. Consider the attackers
We need to think about who is attempting to attack the application. Is a curious user the same as an insider in the types of attacks they can launch. What about an angry customer versus an angry employee or a customer that is not familiar with computers at all? They could all do things to attack the application, but they'll come in different forms and may require different controls. In this phase, you can build a simple spreadsheet to keep track of the different users, or try attack graphs or even attack trees. These vary in complexity and usefulness. The idea here is to give sufficient thought to the types of attackers you could encounter, determine which of those you care about, and then think about how those specific attackers may attempt to attack your application (specific attack vectors).

5. Rate the issues
Given the specific components of the application (what) and the attackers (who) and their attack vectors (how), we should have a decent picture of our threat environment, assuming we did a good job on the previous activities. At this point, we need to decide which threat scenarios (who attacking what how) we care about and how much. This prioritization process will rank issues for the next step.

6. Resolve the issues
Now we need to do something with this data that we've worked so hard to produce. Once we know the specific issues we care about preventing and which ones are most important, we go through the process of resolving them. I find I go through a few simple steps naturally.
- reduce attack surface: Are there things that can be resolved by removing unnecessary functionality or architecture from the application? There are tradeoffs here, but simplicity is your friend in security, and I try to look for this ways to apply this solution relentlessly.
- design around them: Can you build your system in a different way that doesn't require that specific technique. Are there alternative technologies or solutions that don't suffer that weakness? There is again a tradeoff here, as you'll need to reconsider your threat model in light of your given design alternative .
- find controls: If I can find quality reusable solutions, I use them. There are obvious benefits to reuse.
- build controls: If I can't find something that already does the job, I go the route of building something to handle the issue. When I do that, I also try to make it reusable.
- build tests: The threat modeling process should be a drive for test cases, just as requirements would. An additional benefit to doing this task as part of the process is that you can use a traceability matrix to tie the threat/attack vector to a test case, and show that it's covered.

7. Iterate
Threat modeling is an activity that, if done, is usually done once at the beginning of the project and not again. This should not be the case. Practically, I don't think you need to re-evaluate it for every development sprint, but you should re-evaluate it periodically. I generally recommend that you revisit the threat model in earnest when you hit a specific trigger (new attack, incident, operations data on attack scenarios, new assets, change in personnel) and on a periodic timed basis (3m, 6m, 1y).

The coverage I have given to threat modeling is admittedly basic, but should hit the high points well enough. This is an extremely useful security process that adds value, but is often not done because either 1) people don't know how to do it or 2) people think it's too heavyweight. Reason 1 is an education issue, and is easy enough to remedy for those who are interested. Reason 2 goes away in part with education, but also realize you should customize the process to your environment, which can save you time (certain threat actors may be of no concern to you, eg.) Also consider that we all generally build similar things for the most part (web, mobile, desktop) – our threat models often have similar or common components. We can build reuse across applications, internal groups, and even across organizations. There are publically available threat models (some better than others) for common application models that you can use as a starting point. From there, reuse what you can from app to app to save time.

In conclusion, threat modeling is a very useful exercise to evaluate and improve the security of your application. By considering your application and the threats against it, you can have a better understanding of how to effectively design the security of your system.

References
———–
jOHN Steven's threat modeling talk
http://software-security.sans.org/blog/2012/08/06/ask-the-expert-john-steven
https://www.owasp.org/index.php/Application_Threat_Modeling
http://www.cigital.com/justice-league-blog/2011/05/11/threat-modeling-vocabulary/
http://software-security.sans.org/blog/2012/08/06/ask-the-expert-john-steven
https://www.owasp.org/index.php/Category:Threat_Modeling
http://msdn.microsoft.com/en-us/library/ff648644.aspx
https://www.owasp.org/index.php/Threat_Risk_Modeling
http://www.schneier.com/paper-attacktrees-ddj-ft.html
http://www.cigital.com/justice-league-blog/2011/03/29/moving-to-mobile-new-threats/

Week 48 - You Will Get Hacked

What is it and why should I care?
You will get hacked. That is not meant to be a sensationalist line, but rather a functional reality in the environment we currently occupy. There are a few reasons I feel safe in stating that assumption:

- Many have already been openly hacked, including those that are well known for being "more secure" and even those that are security vendors. Also, I know of many companies that have been hacked and have not gone public, and I'm sure that's the case for many others.
- Information security is a relatively young field, with weak solutions compared to other "security" fields.
- The change of pace in the technologies we protect is significantly different than that of our solutions. New technologies are coming online faster than we can secure them, and the security of those technologies is no better (and often worse) than what we already have.

I should at least define what I mean by "hacked" in order to clarify my original point. I mean that there is some successful attack against your systems. That could be a data breach through a web service, a website defacement, an insider stealing data, a backup disk or laptop getting lost, hardcoded keys being found on your hardware, or any other of a number of possible successful attacks you've heard about recently. There are lots of ways you can lose, and as the old adage goes, it only takes an attacker one successful attempt, and you have to successfully defend against all of them.

If that sounds bleak, it should. However, I'm not suggesting we throw our hands up and quit, but rather we plan for the eventuality that an attack will succeed. Banks have been successfully robbed since they've been in existence. However, they don't just shut down and go out of business because they know they will get broken into, but rather they put security controls in place, they monitor, they work with law enforcement, as well as use lots of other techniques to prevent the attack in the first place, and when necessary, recover from a successful one.

What should I do about it?
If we work from the assumption that the successful attack will eventually occur, it changes our mindset a bit. We now don't just have to work towards preventative security, but we also have to think about recovery, or incident management. There are many components to a successful program to manage security when you plan ahead for successful attacks, and here are just a few for consideration:

Disaster Recovery Planning
While disaster recovery planning is traditionally associated with a natural disaster, there are significant benefits to performing this exercise for continuity (DR is also known as business continuity planning) in the wake of an attack. Tasks of interest here would include ensuring you have a solid backup plan for both your systems (servers, network, etc) and your data. For larger organizations that can afford it, this could mean having additional hardware at an off-site location generally a significant distance (far enough away that the same hurricane/tornado/flood, etc. won't affect it) that can be enabled if the primary site goes down. If you are "in the cloud", you should build your applications with the capability to roll over to different "regions", effectively accomplishing the same goal, but on rented hardware.

DEVOPS

Devops is all the rage these days, and has numerous benefits. In the context we're discussing today, one benefit worth noting is that updates and fixes can be pushed to production quickly and safely. You have processes in place that allow for code to be built, tested, reviewed, and pushed extremely quickly, and you also have the capability to roll back updates when needed. These are critical capabilities if you want to stay functional when under attack. If you found out you had an injection that could be (and was being) actively exploited, would you rather shut the site down until further notice, leave it up and vulnerable, or be able to fix it quickly and push out the update and keep the remainder of your site up?

Application Layer Intrusion Detection and Prevention
I've already written about this a couple of times before, but will provide a brief recap here for the relevant portions. This is an area where intrusion detection and prevention really shines. If we know we're being attacked, and we are keeping track of our users and the "bad" things they are doing, we can take some actions to protect ourselves. One of the side effects of knowing people are attacking you and tracking those users is that you start to think about how you'd like to be able to respond. For example, with AppSensor, you can respond by disabling the user account, notifying an administrator of nefarious activity, or you can block resources within the application. The capability to block resources is interesting for our context. Let's say you have a management console that allows you to do this, and you find out there's a vulnerability within your application, but it only resides on a certain page. With this capability, you can just block access to that page/function, and leave the rest of the site up. That will give you time to get the fix deployed, and then you can re-enable the feature.

Incident Management Team
This is more of a resource issue, but can greatly improve your ability to respond. Depending on the size of your organization, this could be part of a person's role, or the responsibility of a large group. By focusing effort here and building out a team to handle incidents, you can have focus and move towards better planning, root cause analysis, security controls, monitoring, etc.

Bug Bounty Program
Let me be clear that I would *not* suggest this for organizations that have either a)policy or regulatory concerns and consequences, or b)immature secure groups.
With those caveats, anecdotal evidence from organizations that have tried this suggest it can be very effective. You open up your applications to being attacked intentionally, and reward those who find issues and disclose them "responsibly", as defined by you. You essentially are getting a lot of cheap testing (though it's in prod). There's a lot of concerns to consider here, but it can certainly guarantee that you get a good testbed of attacking activity against your application.

Normal Security Program
The above are a few ideas for things that I haven't necessarily discussed in this series that come in handy when thinking about dealing with successful attacks. However, they all still fit into your normal security program and planning process. Also, they don't come in to the exclusion of anything in your existing program. DEVOPS does not win out over code review. They are complementary and both should be used. Actually, in most cases, you'll find that doing one is either a requirement for or benefits the other. Continue doing the things you're already doing, and just consider these additional program components.

In conclusion, at some point, your applications and systems will get successfully attacked – it is only a matter of time. It is wise to recognize that and deal with the situation in advance. With solid planning, you can put in place controls that include prevention, monitoring and detection, response, and recovery. These controls should be added to your overall security program in order to further strengthen your overall capabilities.

References

———–

https://www.owasp.org/index.php/OWASP_AppSensor_Project

Week 49 - Collect And Share Your Data

What is it and why should I care?
Today's topic is about two of the areas that are weakest in application security – data collection and sharing. We do a pretty terrible job as an industry in both areas, though there have been some marked improvements in the last couple of years that bring hope.

While there is no confusion around what data collection and sharing means in general, there is a lot of disagreement about both topics in specific areas. Let's briefly define both here for clarity:

Data Collection: The gathering and storage (collection) of specific points of interest (data).

Data Sharing: The distribution (sharing) of collected points of interest (data) with interested parties.

Both of these definitions are intentionally broad. IMHO, The issues brought up about what constitutes data vs. information (collection) and who gets the data (sharing) are fruitless when you consider how very little data we're talking about to start with. If we have broad data collection and sharing (different, better problem), we can then address the needs of the community as far as standardizing the what to share and with whom to share it.

The lack of data collection and sharing in any industry essentially means that you are unaware of what others are seeing and doing. That can be particularly challenging in security as we all share a common resource (the network) and we all are using a relatively small subset of tooling to perform very similar tasks. In many cases, data that comes from one organization could be helpfully utilized by another. This applies to our industry much more than in some other industries with wider variances on tools and processes. The reverse is also true: sharing positively benefits our industry more quickly and to a greater degree specifically because there is so much commonality.

What should I do about it?
My basic hope is that you look for ways to share your data. We all benefit from it. We are theoretically a science and engineering based field, but have a rough track record of sharing actual data to support our hypotheses. However, we do have some shining examples, and that should both give us hope and motivation to get better. On the collection side, we can look at folks like Etsy. They decided that data collection would be a central part of their DNA and invested engineering resources into building tools for data collection and monitoring – a great success story. For data sharing, there are a few popular ones, like the Veracode State of Software Security Report, the Verizon Data Breach Investigations Report (DBIR) and the WhiteHat Website Security Statistics Report. (Full Disclosure: I currently work for WhiteHat) All of these are great examples of organizations sharing the data they see for the benefit of the community. One other great example is that of Security Ninja sharing3 years worth of data that he's collected. He also makes a poignant quote in that article – "If you have the data don't hide it". *Note: He's speaking of sharing with internal teams – an extremely valuable (and nearly equally as uncommon) form of sharing in addition to sharing publicly.*

As I mentioned above we have some bright spots to give us hope that we can do better. Now let's touch on a few points you should consider before sharing to make sure you're doing your due diligence.

Stay Legal and Compliant
Certain organizations can't share certain data. That's a legal and regulatory reality. In general, security practitioners have been fairly conservative with what data is shared because it tends to be

sensitive, however sharing is becoming more commonplace, and that seems to be helping us all get a handle on the reality in practice. Just because you may have some restrictions on what or how you can share doesn't mean those restrictions are completely limiting. A good example is the FS-ISAC which shares data within the financial services industry, allowing similar organizations with similar concerns to share their data in a controlled environment. In short, make sure you are allowed to share something before you share it.

Always Anonymize
The data you share may or may not have privacy-related issues. If it does, you have to make sure you anonymize the data. No one's private information should ever be shared publicly, especially when the data that is desirable to share is not affected by the private information at all. Make sure you take care of your users and customers and don't share anything you shouldn't. There are lots of tools that can help with this process if you need them.

Try to Share Raw Data
As much as possible (making inherent inferences is difficult to avoid, even pre-collection), you should try to stick to sharing raw data (anonymized of course). That way, others can analyze your data and evaluate (support or contradict) your conclusions. For instance, my recent post about password storage referenced agreat spreadsheet considering the cost tradeoffs to attackers and defenders for password protection schemes. This raw (even generated) data gives others a way to make their own analysis and makes us all more aware of the actual data.

In conclusion, we discussed a weak spot for the application security industry: data collection and sharing. While we have historically been pretty bad at this, there are some bright spots and we're starting to see both collection and sharing happening on a broader scale, which is hopeful. Following a few simple due diligence tasks, we can make sure that we're sharing safely and can help the industry as a whole move forward.

References
———–

http://www.veracode.com/reports/index.html
http://www.verizonbusiness.com/about/events/2012dbir/
https://www.whitehatsec.com/resource/stats.html
http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/
http://www.securityninja.co.uk/application-security/application-security-data/
https://github.com/jsteven/psm/tree/master/Documents

Technology

Week 1 - Session Fixation Prevention

What is it and why do I care?
Session fixation, by most definitions, is a subclass of session hijacking. The most common basic flow is:

1. attacker gets a valid session ID from an application
2. attacker forces the victim to use that same session ID
3. attacker knows the session ID that the victim is using and can gain access to the victim's account.

Step 2 of forcing the session ID on the victim is the only real work involved in the attack. It's often performed by simply sending a victim a link to a website with the session ID attached to the URL.

Obviously, one user being able to take over another user's account is a *serious* issue, so …

What should I do about it?
Fortunately, resolving session fixation is usually fairly simple. The basic advice is:

Invalidate the user session once a successful login has occurred.

The usual basic flow to handle session fixation prevention looks like:

1. User enters correct credentials
2. System successfully authenticates user
3. Any existing session information that needs to be retained is moved to temporary location
4. Session is invalidated (HttpSession#invalidate())
5. New session is created (new session ID)
6. Any temporary data is restored to new session
7. User goes to successful login landing page using new session ID

A useful snippet of code is available from the ESAPI project that shows how to change the session identifier.
http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/reference/DefaultHTTPUtilities.java (look at the changeSessionIdentifier method)

There are other activities that you can perform to provide additional assurance against this issue. A few I thought of are listed below.

1. Check if a user tries to login using a session ID that has been specifically invalidated (requires maintaining this list in some type of LRU cache)
2. Check if a user tries to use an existing session ID already in use from another IP address (requires maintaining this data in some type of map)
3. If you see these types of obviously malicious behavior, consider using something like AppSensor (shameless plug) to protect your app, and to be aware of the attack.

As you can see, session fixation is a serious issue, but has a pretty simple solution. Your best bet if possible is to include an appropriate solution in some “enterprise” framework (like ESAPI) so this solution applies evenly to all your applications.

References

————–

https://www.owasp.org/index.php/Session_fixation
http://www.acros.si/papers/session_fixation.pdf
http://cwe.mitre.org/data/definitions/384.html
http://projects.webappsec.org/w/page/13246960/Session%20Fixation

Week 2 - Error Handling in web.xml

What is it and why do I care?
I've already discussed this particular entry in more detail as it's part of the OWASP Top 10, so you can find more detail here – http://www.jtmelton.com/2010/06/02/the-owasp-top-ten-and-esapi-part-7-information-leakage-and-improper-error-handling/. In this article, I'll just cover the important bits.

Error or exception handling is an important, often ignored, part of any application. There is a lot to be said about the topic in general, but for brevity's sake, I'm only going to cover a couple of the most critical cases in J2EE web applications.

Essentially, one of the biggest worries about exception handling is that you don't actually handle the exception. Instead your code or the code of some 3rd party library you're using allows an exception to bubble up. At that point, once it's at the boundary of your application and gets to the container, it depends on the specific container / application server you are using as to what semantics are applied in the handling of the exception. Often times, by default, there is a standard error page applied and the exception stack trace is simply printed on the screen in all it's glory. This is clearly a problem in that it gives attackers a lot of information about the system, and can certainly lead to further attacks.

What should I do about it?
Handling this issue is fairly straightforward. The basic advice is to provide error handlers for at least java.lang.Throwable (catches any Java exceptions or errors) and provide more specific handlers for specific exceptions as well as http error codes, the most common being 404 and 500. An example snippet that can be applied to the web.xml is below:

Note: error.jsp page should be generic and provide a canned response message of some type with no detail that could help an attacker fingerprint the app in any way.

```
<error-page>
  <error-code>404</error-code>
  <location>/error.jsp</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/error.jsp</location>
</error-page>
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/error.jsp</location>
</error-page>
```

There's plenty more to handling exceptions in your application, but from a security perspective, catching Throwable's and the specified error codes should do fairly well for you.

References
————
http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files
http://www.jtmelton.com/2010/06/02/the-owasp-top-ten-and-esapi-part-7-information-leakage-and-improper-error-handling/

Week 3 - Session Cookie Secure Flag

What is it and why do I care?

Session cookies (or the cookie containing the JSESSIONID to Java folks) are the cookies used to perform session management for web applications. These cookies hold the reference to the session identifier for a given user, and the same identifier is maintained server-side along with any session scoped data related to that session id. Since cookies are transmitted on every request, this is the most common mechanism used for session management in web applications.

The secure flag is an additional flag you can set on a cookie that instructs the browser to ONLY send this cookie on HTTPS (encrypted) transmissions, and _not_ on HTTP (unencrypted) transmissions. This ensures your session cookie is not visible to an attacker in, say, a man in the middle attack (MITM). This is not a complete solution to secure session management, but is an important step.

What should I do about it?

The resolution here is quite simple. You must add the secure flag to your session cookie (and preferably all cookies as any requests to your site should be HTTPS if possible).
Here's an example of how a session cookie might look without the secure flag:

```
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H;
```

And now, with the secure flag:

```
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H; secure;
```

Not much to it. You can obviously manually do this, but if you're working in a Servlet 3.0 or newer environment, there's a simple configuration setting in the web.xml that takes care of this for you. You should add this snippet to your web.xml.

```
<session-config>
  <cookie-config>
    <secure>true</secure>
  </cookie-config>
</session-config>
```

As you can see, resolving this issue is quite simple. It should be on everyone's //TODO list.

References

————
http://blog.mozilla.com/webappsec/2011/03/31/enabling-browser-security-in-web-applications/
http://michael-coates.blogspot.com/2011/03/enabling-browser-security-in-web.html
https://www.owasp.org/index.php/SecureFlag

Week 4 - Session Cookie HttpOnly Flag

What is it and why do I care?
Session cookies (or the cookie containing the JSESSIONID to Java folks) are the cookies used to perform session management for web applications. These cookies hold the reference to the session identifier for a given user, and the same identifier is maintained server-side along with any session scoped data related to that session id. Since cookies are transmitted on every request, this is the most common mechanism used for session management in web applications.

The HttpOnly flag is an additional flag that is used to prevent an XSS (Cross-Site Scripting) exploit from gaining access to the session cookie. Since gaining access to the session cookie, and subsequently hijacking the victim's session, is one of the most common results of an XSS attack, the HttpOnly flag is a useful prevention mechanism.

What should I do about it?
The resolution here is quite simple. You must add the HttpOnly flag to your session cookie (and preferably all cookies). Here's an example of how a session cookie might look without the HttpOnly flag:

```
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H;
```

And now, with the HttpOnly flag:

```
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H; HttpOnly;
```

And, if you were following along from last week, with both the secure and HttpOnly flags:

```
Cookie: jsessionid=AS348AF929FK219CKA9FK3B79870H; HttpOnly; secure;
```

Not much to it. You can obviously manually do this, but if you're working in a Servlet 3.0 or newer environment, there's a simple configuration setting in the web.xml that takes care of this for you. You should add this snippet to your web.xml.

```
<session-config>
  <cookie-config>
    <http-only>true</http-only>
  </cookie-config>
</session-config>
```

And, if you also use the secure flag, it looks like this:

```
<session-config>
  <cookie-config>
    <http-only>true</http-only>
    <secure>true</secure>
  </cookie-config>
</session-config>
```

As you can see, resolving this issue is quite simple. It should be on everyone's TODO list.

References
————-
http://blog.mozilla.com/webappsec/2011/03/31/enabling-browser-security-in-web-applications/
http://michael-coates.blogspot.com/2011/03/enabling-browser-security-in-web.html
https://www.owasp.org/index.php/HttpOnly

Week 5 - Clickjacking Prevention

What is it and why do I care?
Clickjacking is a type of "web framing" or "UI redressing" attack. What that simply means in practice is that:

1. A user (victim) is shown an innocuous, but enticing web page (think watch online video)
2. Another web page (that generally does something important – think add friends on social network) is layered on top of the first page and set to be transparent
3. When the user thinks they are clicking on the web page they see (video), they are actually clicking on the higher layered (framed) page that is transparent.

This attack is clever, and there are some interesting specifics in the actual execution of an attack (For more info, see the references), but here, I'm concerned with preventing the attack.

What should I do about it?
There is still no perfect answer on clickjacking, but things are getting better, especially as users upgrade to more modern browsers. The recommendation is two-fold:

1. Use the X-Frame-Options HTTP header
2. Include framebusting code

There is a future article in the series coming, I promise, on the X-Frame-Options HTTP header. All I'll say for now is that this header is the more robust solution, but that it requires a relatively modern browser. Fortunately, folks are slowly moving towards more modern browsers, so the situation is improving.

As for the framebusting recommendation, it is breakable, but should still be done. It certainly raises the bar. There are many options for framebusting code, but the folks at Stanford put together a paper http://seclab.stanford.edu/websec/framebusting/ on framebusting where they evaluated the current code in the wild and showed ways to break it. They came up with their own proposed solution in the paper. I'll omit the code here, but it's at the top of page 11 of the pdf. The basic idea is to both:
a) use the stylesheet to disable display for the entire body of the page, and
b) use Javascript to either enable the display if not framed, or to bust out of the frame if framed.

This solution will probably (if it's not already) be broken, but it appears to be the best we have today. Clickjacking is unfortunately a less-than-clearcut issue to solve, but by combining a couple different approaches, you can resolve the issue with a fair amount of robustness.

Update 2/3/2012: The Stanford approach does not adequately support IE in all instances – here's a post from August Detlefsen explaining the solution. (h/t Chris Schmidt)

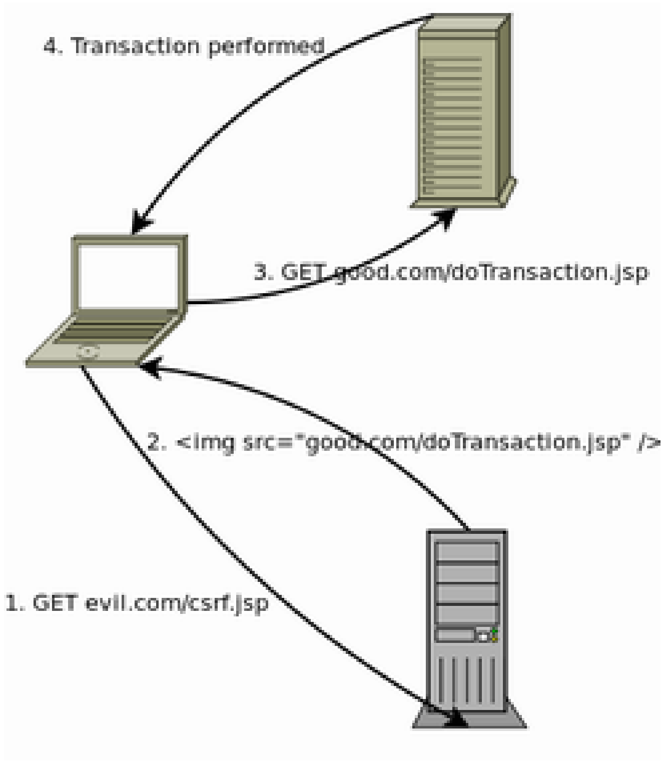
References
————
https://www.owasp.org/index.php/Clickjacking
http://seclab.stanford.edu/websec/framebusting/
http://michael-coates.blogspot.com/2010/08/x-frame-option-support-in-firefox.html
https://www.codemagi.com/blog/post/194

Week 6 - CSRF Prevention In Java

What is it and why should I care?
Cross Site Request Forgery (CSRF) is an attack wherein a victim is forced to execute unknown and/or undesired requests to a website at which he/she is currently authenticated. It exploits the fact that the “credentials” needed to perform a function on a website are generally loaded into a client-side cookie, which is transmitted automatically with every request. By exploiting this fact, an attacker can cause a victim to execute almost any request that isn’t protected against CSRF.

Here is a simple image that shows a basic CSRF attack. The steps are:

Step 1. The victim is lured to visit an evil site (evil.com)
Step 2. The victim’s browser loads the evil site, which contains a hidden image whose src points at an important transaction on good.com)
Step 3. The victim’s browser executes the transaction at good.com, passing along the session cookie (general authn/z mechanism).
Step 4. The transaction occurs, and the victim is not aware until later, if at all.


4. Transaction performed
3. GET good.com/doTransaction.jsp
2. <img src="good.com/doTransaction.jsp" />
1. GET evil.com/csrf.jsp

What should I do about it?
Now that we know what CSRF is and why it is dangerous, let’s consider some possible solutions.

Token

The classic solution to CSRF has been a per-session token (known as the synchronizer token design pattern). The basic flow is:

Step 1. User logs in and a randomized string (token) is placed in the user session
Step 2. On every form for a non-idempotent request (meaning essentially any request that changes server-side state – should be your HTTP POSTs), place the token in the form when it's submitted.
Step 3. On the request handler for the non-idempotent request, validate that the submitted token matches the token stored in the session. If either is missing or they don't match, do not process the transaction.

This solution has served pretty well for most situations over the years, but can be time-consuming to implement and often creates opportunities for forgetting validation on some requests.
The ESAPI project does have built in CSRF protection, but it is tied to the authentication scheme. Here is a writeup I previously did for ESAPI's CSRF protection.

CSRFGuard
A very good option with solid protection is the OWASP CSRFGuard project. This library makes it relatively simple to include CSRF protection into your application by simply mapping a filter and updating a configuration file. This is certainly a resource worth checking out.

Stateless CSRF Protection
One more thing I'd like to point out is some interesting work John Wilander (@johnwilander) is doing in the area of stateless CSRF protection. In the case where you can't or don't want to maintain the user token in server-side session state, John proposes a seemingly novel solution. The proposal is to allow the client side to create a cookie with the CSRF token (which gets submitted on every request) and to include the token as a request parameter. Since an attacker can't read both the cookie and request parameter, all the server side should have to do is validate that the token in the cookie and the request parameter match. This solution, to my knowledge, has not been widely reviewed or tested, but it is a great example of an elegant solution to a difficult problem. Only time will tell if this is the go-forward solution for stateless CSRF protection.

CSRF is a popular and dangerous attack, but with the protections described above, it is certainly manageable.

One last note here is that the token approach (classic or stateless) does break down when the site it is deployed on contains XSS (Cross Site Scripting) vulnerabilities. If an attacker can XSS your site, they can read the content and extract out the token you're using, effectively nullifying the protection. The lesson is simple: fix those XSS bugs too!

References
————
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet
http://www.cgisecurity.com/csrf-faq.html
http://www.jtmelton.com/2010/05/16/the-owasp-top-ten-and-esapi-part-6-cross-site-request-forgery-csrf/
https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project

Week 7 - Content Security Policy

What is it and why should I care?
Content Security Policy (CSP) is a new(ish) technology put together by Mozilla that web apps can use as an additional layer of protection against Cross Site Scripting (XSS), which is the primary goal of the technology. A secondary goal is to protect against clickjacking.

XSS is quite a complex issue, as is evidenced by the recommendations in the OWASP prevention cheatsheets for XSS in general and DOM based XSS. CSP does several things to help app developers deal with XSS.

Whitelist Content Locations
One reason XSS is quite harmful is that the browsers implicitly trust the content received from the server, even if that content has been manipulated and is loading from an unintended location. This is where CSP comes in: CSP allows app developers to declare a whitelist of trusted locations for content. Browsers that understand CSP will then respect that list and only load content from there, and will ignore content that references locations outside the list.

No Inline Scripts
Adding scripts inline to content is the way XSS is done, made possible because browsers have no way of telling whether the site actually sent that content, or if the attacker added the script to the site content. CSP prevents this entirely by forcing the separation of content and code (Great design!). This means that you have to move all your scripts to external files, which will require work for most apps, though it can be done (Twitter is one example of a high-profile site using CSP). The upside is in order for an attack to be successful with CSP, an attacker has to be able to:

Step 1) Inject a script tag at the head of your page.
Step 2) Make that script tag load from a trusted site within your whitelist.
Step 3) Control the referenced script at that trusted site.

This makes an XSS attack significantly more difficult.

Note: one common question here is what about event handling. CSP does still allow event handling through the onXXX handlers or the addEventListener mechanism.

No Code From Strings (eval dies)
Another welcome addition is the blacklisting of functions that create code from strings. This means that usage of the evil eval is out (along with a few others). Creating code from strings is a popular attack technique and is rather difficult to trace, so the removal of all these functions is actually quite helpful.

Another common question stemming from this is how to deal with JSON parsing. The right way to do this from a security perspective has always been to actually parse the JSON instead of eval it anyhow, and this functionality is still available, so nothing changes there.

Policy Violation Reporting
A rather cool feature of CSP is that you can configure your site to have a violation reporting handler, and then have that data available whether you run in report-only or in enforcing mode. In report only mode, you can get reports of locations in your site that will be prevented from execution when you enable CSP (a nice way to test). In enforcing mode, you can still get this data, but in production, you

can also use this as a simple XSS detection mechanism (bad guy tried XSS and it didn't run).

What should I do about it?
Well, you should use it! Actually, CSP doesn't really seem to have a downside. It's there to make your site safer, and even if a client's browser doesn't support it, it's entirely backwards compatible, so your site won't break for the client.

In general, I think the basic approach should be:
Step 1) Solve XSS through your standard security development practices (you should already be doing this)
Step 2) Learn about CSP – read the specs thoroughly
Step 3) Make the changes to your site and test and re-test (normal dev process)
Step 4) Run in report-only mode and monitor any violations to find areas you still need to fix. (could be avoided if you have a full functional test suite you can execute against your app in testing – good for you if you do!)
Step 5) After you're confident your site is working properly, turn it on in enforcing mode.

As for how you actually implement it, you have 2 basic options: an HTTP header and a META tag. The header option is preferred, and an example is listed below.

```
Content-Security-Policy: default-src 'self';
    img-src *;
    object-src media1.example.com media2.example.com *.cdn.example.com;
    script-src trustedscripts.example.com;
    report-uri http://example.com/post-csp-report
```

The example above says the following:
Line 1: By default, only allow content from 'self' or the site represented by the current url.
Line 2: Allow images from anywhere.
Line 3: Allow objects from only the listed urls.
Line 4: Allow scripts from only the listed url.
Line 5: Use the listed url to report any violations of the specified policy.

CSP is a very interesting and useful new technology to help battle XSS. It's definitely a useful tool to add to your arsenal.

Update (2/15/2012) Here's an interesting post on using ModSecurity to add CSP to your site – a nice look for the administrators. (h/t @ryancbarnett)

References
----
https://developer.mozilla.org/en/Introducing_Content_Security_Policy
https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html
https://wiki.mozilla.org/Security/CSP/Design_Considerations
https://wiki.mozilla.org/Security/CSP/Specification
https://dvcs.w3.org/hg/content-security-policy/raw-file/bcf1c45f312f/csp-unofficial-draft-20110303.html
http://www.w3.org/TR/CSP/
http://blog.mozilla.com/security/2009/06/19/shutting-down-xss-with-content-security-policy/
https://mikewest.org/2011/10/content-security-policy-a-primer
http://codesecure.blogspot.com/2012/01/content-security-policy-csp.html

https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
http://blog.spiderlabs.com/2011/04/modsecurity-advanced-topic-of-the-week-integrating-content-security-policy-csp.html

Week 8 - HTTP Strict Transport Security

What is it and why should I care?
HTTP Strict Transport Security (HSTS) is a new(ish) technology that allows an application to force browsers to only use SSL/TLS (HTTPS, not HTTP) when visiting their application. This occurs when the application sets an HSTS specific HTTP response header. Browsers that support HSTS recognize the response header and only communicate with that application over HTTPS for the specified time. Here are the basic steps of the flow:

Step 1: A user visits application, either over HTTP or HTTPS. For most users this will be HTTP as they often forget (or don't know) to type in https://
Step 2: The application renders the response, including the HSTS response header that specifies the site should be loaded over HTTPS only for the next 90 days.
Step 3: Supporting browser will not issue a non-SSL request to that application for the next 90 days.

What should I do about it?
Set the header! This is another no-brainer. There are no backwards compatibility issues and it only enhances your security posture. By adding the header, you give your users greater security with very little effort on your part. Here's a quick example of what setting the header might look like if you happen to code in Java.

```
HttpServletResponse ...;
response.setHeader("Strict-Transport-Security", "max-age=7776000;
includeSubdomains");
```

The snippet above sets the amount of time that the browser should only use HTTPS when accessing the application (or any subdomains) to the next 90 days.

max-age
The max-age attribute is required and sets the number of seconds that only HTTPS should be used. This can be set to whatever you want. Setting this to 0 tells the browser to delete the existing policy and not require HTTPS.

includeSubdomains
The includeSubdomains attribute is optional. It does not take a value; its' presence signifies that any sub-domains should abide by the same HSTS policy.

Expiration Reset
The time length (90 days in our example above) should be reset (overwritten) by the supporting browser every time the HSTS response header is received. What this means is that if you set the length to 90 days, and a person never goes more than 90 days without visiting your site, they'll always be visiting over HTTPS since the 90 days starts over every time they visit. Just keep in mind that if you change the setting, all browsers that access your site (post-change) will get the update. If you set it to 90 days today and 10 tomorrow, the browser throws away the 90 days and replaces it with 10.

Click-Through Insecurity
The HSTS spec also recommends (though the current spec doesn't seem to require) that the browser not send any data to the server if the channel is insecure for any reason (certificate issues, domain issues, etc.). This appears to be the route being taken by current browser implementers, which is the

most secure option. This means that those warning pop-ups users often see over HTTPS connections (mixed content, expired cert) don't appear any more and instead the user is simply shown an error.

Pre-loaded HSTS Lists
If you want to have even more assurance that your site is only ever visited over HTTPS, you can even request that your site be pre-loaded into a list of HSTS sites. This means the browser will require you to got that site over HTTPS on your very first visit! Paypal is one such well-known site that does this.

HSTS is a simple and effective control to let an application further protect its' users by forcing them over HTTPS. If your site is (or at least should be) HTTPS only, then implementing HSTS can provide you and your users with enhanced assurance.

References
————

http://hacks.mozilla.org/2010/08/firefox-4-http-strict-transport-security-force-https/
http://dev.chromium.org/sts
http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security
https://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec
http://michael-coates.blogspot.com/2011/07/enhancing-secure-communications-with.html
http://www.troyhunt.com/2011/11/owasp-top-10-for-net-developers-part-9.html

Week 9 - X-Frame-Options

What is it and why should I care?

X-Frame-Options (moving towards just Frame-Options in a draft spec – dropping the X-) is a new technology that allows an application to specify whether or not specific pages of the site can be framed.
This is meant to help deal with the clickjacking problem.

The technology is implemented as an HTTP response header specified per-page. Browsers supporting the (X-)Frame-Options header will respect the declaration of the page and either allow or disallow the page to be framed depending upon the specification.

What should I do about it?

Yet again, this is a very low-risk item that only adds additional assurance. There are some limitations that may prevent the header from offering protection in some instances, but it does NOT make you less safe. It is an additional layer of protection.

A page can specify 3 different options for how it wants to be framed.

Option 1: DENY
This option means this page can never be framed by any page, including a page with the same origin. A sample code snippet is below:

```
HttpServletResponse response ...;
response.addHeader("X-FRAME-OPTIONS", "DENY");
```

Option 2: SAMEORIGIN
This option means this page can be framed, but only by another page within the same origin. A sample code snippet is below:

```
HttpServletResponse response ...;
response.addHeader("X-FRAME-OPTIONS", "SAMEORIGIN");
```

Option 3: Allow-From
This option means the page can be framed, but only by the specified origin. A sample code snippet is below:

```
HttpServletResponse response ...;
response.addHeader("X-FRAME-OPTIONS", "Allow-From https://some.othersite.com");
```

As an additional help, the good folks at owasp have put together a simple example J2EE filter for X-Frame-Options.

(X-)Frame-Options is a good additional layer of protection to add to your site to prevent clickjacking. While it won't stop everything, it costs very little, and can help protect your users.

References

————
https://developer.mozilla.org/en/The_X-FRAME-OPTIONS_response_header
https://www.owasp.org/index.php/Clickjacking#Defending_with_response_headers

http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
http://blog.mozilla.com/security/2010/09/08/x-frame-options/
http://lcamtuf.blogspot.com/2011/12/x-frame-options-or-solving-wrong.html
http://www.jtmelton.com/2012/02/03/year-of-security-for-java-week-5-clickjacking-prevention/
http://tools.ietf.org/html/draft-gondrom-frame-options

Week 10 – X-CONTENT-TYPE-OPTIONS

What is it and why should I care?
X-Content-Type-Options is an HTTP header that can help prevent browser content-type sniffing problems.

The content-type for a given resource should match the "type" (too obvious?) of the resource. For example, an HTML page would use "text/html", a PNG image would use "image/png", and a CSS document would use "text/css". However, often times, the content-type is either not specified or is wrong. This has led to browsers having to implement "sniffing" algorithms to determine what the actual data is that is being served, and then apply the appropriate parsing and execution semantics for the sniffed type. This, however, has caused certain bugs. One well-known example allowed attackers to have files that were supposedly images be interpreted as javascript and executed.

What should I do about it?
There are actually 2 things to do here.

Step 1. When serving resources, make sure you send the content-type header to appropriately match the type of the resource being served. For example, if you're serving an HTML page, you should send the HTTP header:

```
Content-Type: text/html;
```

Step 2. Add the X-Content-Type-Options header with a value of "nosniff" to inform the browser to trust what the site has sent is the appropriate content-type, and to not attempt "sniffing" the real content-type. Adding this additional header would look like this:

```
X-Content-Type-Options: nosniff
```

These 2 simple steps will provide additional protection against content-type sniffing issues.

An important note to mention here is that while this is a useful protection, not all browsers have implemented it.
As of this writing (3/6/2012), only Chrome and IE support this protection (though NoScript does apparently add the protection to Firefox). Even though it won't save all your users, it's a useful mechanism to provide even more assurance for your users.

References
———–
http://www.browserscope.org/?category=security
http://blogs.msdn.com/b/ieinternals/archive/2011/02/11/ie9-release-candidate-minor-changes-list.aspx
http://blogs.msdn.com/b/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx
http://msdn.microsoft.com/en-us/library/gg622941(v=vs.85).aspx
http://htaccess.wordpress.com/2009/09/22/x-content-type-options-nosniff-header/
http://www.adambarth.com/papers/2009/barth-caballero-song.pdf
http://www.w3.org/TR/html4/types.html#h-6.7
http://hackademix.net/2010/10/30/x-content-type-options-noscript-and-browserscope/

Week 11 – X-XSS-PROTECTION

What is it and why should I care?
X-XSS-Protection is a Microsoft IE technology used to help prevent reflected XSS attacks in IE.

Note 1: This is not a "panacea" for XSS. There is no excuse for not developing your site in a secure manner to prevent XSS. This however is a protection offered by the browser itself (as opposed to an application), meant to protect the masses from the vast amount of XSS litter on the internet.

Note 2: Firefox (by way of NoScript), Chrome (by way of WebKit) and Safari(also WebKit) have similar protections, but apparently don't use the X-XSS-Protection header as a controlling mechanism.

The XSS protection provided essentially checks for request content that is matched in the response and would cause an XSS vulnerability to be exploited. The filter then performs some mangling of the content to prevent the attack from succeeding. According to the docs, IE has the protection turned on by default for most security zones, including the Internet zone, which is the primary concern for most users.

What should I do about it?
The first thing you should do is work towards resolving any and all XSS issues in your application. As a security minded developer, this is a must.

The recommendation for the use of this header is actually not so straightforward in my opinion. In general, the other HTTP headers I've described already in the series have had very little downside. However, the X-XSS-Protection header has had some problems in the past. As far as I'm aware, the IE folks have done a good job of dealing with the known vulns, but I still have concerns since some of the vulns have exposed security problems.

In general, I would recommend keeping the protection enabled, unless you are very sure you have XSS all cleaned up in your app. However, this comes with the caveat that you should at least put some thought into the use cases in your site first. Depending on your choice, here are the options you have available to use, and how you enable them in your application using the X-XSS-Protection HTTP header.

1. Enable the protection for all security zones in blocking mode (Blocking mode means the site won't display at all if an XSS attempt is found, but rather a simple warning to the user that the attack has been blocked):

```
X-XSS-Protection: 1; mode=block
```

2. Enable the protection for all security zones:

```
X-XSS-Protection: 1
```

3. Leave the protection enabled for the default zones:

Do nothing.

4. Disable the protection entirely (I only recommend this in 2 cases: either you're positive that you've completely resolved XSS in your app, or there's an issue in the XSS filter that you're aware of that causes an additional vulnerability) :

```
X-XSS-Protection: 0
```

The protection provided by the X-XSS-Protection header is not complete, but it does raise the bar against attackers and helps protect users. While there have certainly been some implementation issues, the fact that all the major browsers have some implementation of reflected XSS protection shows the importance of this issue. Be prudent in implementation, but certainly do everything you can to help your users be safe.

References

———–
http://blogs.msdn.com/b/ieinternals/archive/2011/01/31/controlling-the-internet-explorer-xss-filter-with-the-x-xss-protection-http-header.aspx
http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx
http://blogs.msdn.com/b/mikeormond/archive/2009/01/26/ie8-cross-site-scripting-xss-protection.aspx
http://msdn.microsoft.com/en-us/library/dd565647(v=vs.85).aspx
http://michael-coates.blogspot.com/2009/07/ie-8-anti-xss-bit-overblown.html
http://jeremiahgrossman.blogspot.com/2010/01/to-disable-ie8s-xss-filter-or-not.html
http://www.jtmelton.com/2009/01/12/the-owasp-top-ten-and-esapi-part-2-cross-site-scripting-xss/
http://michael-coates.blogspot.com/2009/11/ie8-xss-filter-bug.html
http://xforce.iss.net/xforce/xfdb/47442
http://hackademix.net/2009/11/21/ies-xss-filter-creates-xss-vulnerabilities/
http://www.theregister.co.uk/2009/11/20/internet_explorer_security_flaw/

Week 12 - Log Forging Prevention

What is it and why should I care?
Log forging is an issue that can occur if you allow un-trusted data to be written to a log storage mechanism. The intent of the attacker using log forging is to cover his tracks in the logs or at least make understanding what he was doing more difficult. Unfortunately, like most log-related issues, it's generally not a concern until something happens and you actually need the logs.

A simple example of log forging might look like this: (first the code)

```
String someVar = getRequestParameter("xyz");
log("Data is: " + someVar);
```

And now for what a normal request and the associated log entry might look like:

```
?xyz=my name is Bob

[2012-03-15 02:04:31] [bob] Data is: my name is Bob
```

And finally what a forged request and the associated log entry might look like:

```
?xyz=my name is Bob\r\n[2012-03-15 02:04:39] [mary] Mary created new
user\r\n[2012-03-15 02:04:46] [josh] Josh logged out\r\n[2012-03-15 02:04:55]
[susan] Susan performed an important transaction

[2012-03-15 02:04:31] [bob] Data is: my name is Bob
[2012-03-15 02:04:39] [mary] Mary created new user
[2012-03-15 02:04:46] [josh] Josh logged out
[2012-03-15 02:04:55] [susan] Susan performed an important transaction
```

The idea here is that the attacker has surmised what a standard log entry might look like and then using simple newline characters created what appear to be new legitimate log entries.

Note: If you are using a database for logging, you likely won't have as much of an issue since each entry is going to be in a single row. However, it could still affect you if your log viewer doesn't distinguish between rows. However, you still need to be aware of SQL injection here, which is actually a much more serious issue.

What should I do about it?
Fortunately, log forging has a relatively simple fix.

The general approach is to validate input (you should already be doing this) and encode output (you also should be doing this). Validating input alone is not generally going to stop this attack, since there are valid cases to allow input with newlines. Encoding output in addition to validating the input, however, should solve your problem. There are various options for encoding depending on your needs. A simple fix might be to strip out any user-supplied newlines or replace them with some benign character or character sequence. Another alternative might be to HTML encode the data before storing it. This allows you to decode the data later if you need to get back to the original data, as well as have it set up nicely for a web-based log viewing experience if that's desirable.

Log forging is a simple issue to understand and solve – it just takes some planning ahead to deal with properly. You’ll be glad you did though when you get that 3am call to look through the logs and figure out what’s happening!

I’ve actually already written a longer, more detailed article about log forging prevention here, but this shorter version was meant to show the essentials and fits in with the year of security for Java content.

References
———–

http://www.jtmelton.com/2010/09/21/preventing-log-forging-in-java/

Week 14 – Store JSPs In WEB-INF

What is it and why should I care?
Java Server Pages (JSPs) is an extremely common UI view technology used in J2EE development. JSPs represent the interface the end user interacts with while using an application. JSPs also usually include some business logic, and frequently there are portions of a page protected by some authorization constraints. Additionally, there are many times other protections in request handling (controller) code that perform some authorization before forwarding users to a given JSP. Bottom line, there is important logic (and at times data) stored in JSPs that shouldn't be generally accessible to end users without going through the appropriate controls to gain access to them.

Knowing that these resources are important to the application, we'd certainly like to protect them. In general, modern applications rely on frameworks that use an MVC model for accessing an application. What that practically means regarding JSPs is that there is no direct access to the JSP during normal use of the application. You make a request to some type of controller or handler that performs some business logic and then internally forwards to a JSP for rendering. This means JSPs are generally no longer directly accessed, at least on purpose. However, if you store your JSPs in an area accessible to the web without putting them in the WEB-INF (same place where your web.xml goes) directory, they can be directly browsed by users (unless you have other protections in place). This means your pages will most likely break functionally (since they won't have certain data they are expecting to come from the controller), and they will often times not have the appropriate authorizations performed since those are often expected to occur when a user follows the path through the controller.

Note: These same points could be made about certain types of configuration files, scripts and other types of data included in web applications, but JSPs are generally recognized as the most common resource type that is able to be force-browsed by the end user and contains important data. Also, configuration files for many frameworks work only when they are stored in WEB-INF, simplifying the issue there by forcing you to do it right.

What should I do about it?
The Servlet Specification defines that the WEB-INF directory of a deployed application is not to be directly accessible by external users. In other words, if a request is made for a resource in WEB-INF, it will be rejected. The only way this won't be the case is if there is a bug in the application server running your application.

For this reason, you should store your JSPs in WEB-INF. This way, your application will still be able to use them, since forwarding to a resource in WEB-INF happens server-side and doesn't present an access issue. Additionally, you prevent external users from being able to request them successfully by force browsing to them.

One thing to be aware of is that code that does file manipulation (read, write, move, etc.) on server-side files is still able to access the WEB-INF directory. This means there are still plenty of bugs out there where you can do directory traversal to read the web.xml file in the WEB-INF directory. The WEB-INF directory does prevent access to external direct requests, but not to code that runs server-side – definitely something to watch out for.

Storing your JSPs in the WEB-INF directory is a very simple and effective mechanism that offers protection against forced browsing attempts, and you can and should use it to provide an additional layer of protection to your applications.

Week 15 – Audit Security Related Events

What is it and why should I care?
Auditing security related events includes two basic concepts, so we'll begin by treating them individually.

Auditing
Auditing is a key part of any real software system. Many people treat logging and auditing as the same idea, though they're actually different. Definitions might vary, but mine boils down to the consumer of the output. In general, logging data is consumed by developers (most often for debugging problems), and possibly business owners to see basic trending information (likely through some basic log parsing for usage statistics, etc). Auditing, on the other hand, is meant to be used by auditors to reconstruct the events that occurred in the system. The view of these events is often constrained by a time period, a specific user or set of users, a specific function or set of functions, etc.

Usually, logged data is unstructured and can be or represent anything. Audit data, on the other hand, is generally structured, and can be thought of more like a database record where there are specific fields that are always filled in, and the only thing that changes is the data in the column, not the column itself (to use the DB analogy).

Security Related Events
Security related events are going to be determined by you as part of your development process, but there are several obvious candidates, such as login, logout, user management, credential management. etc. All of these are clearly security related and could be important to the security posture of your application either generally or specifically related to a single user or set of users.

Knowing that a security related event has occurred is important. Not knowing could lead to not only unauthorized access or usage of the system, but the inability to know that it even occurred.

What should I do about it?
Auditing is the option to choose when you're talking about security-related events. For any security-related event that occurs in the system, you should be auditing the activity. You should collect appropriate data on each event, such as event type (what happened), actor performing event (who did it), timestamp (when did they do it), etc. This type of data will allow you to filter the dataset by user, time, function, or any of the other data points when needed to determine specifically what occurred from an auditing perspective. Structured data in this form also lets you do helpful things like look at generic patterns and find that a specific user did a bunch of things outside work hours (unusual?) or everyone in the system all performed a single function within an hour of each other (maybe strange?). Some of these ideas are found in the concept of AppSensor, an OWASP project I work on.

I would also like to point out a great talk Gunnar Peterson gave at an OWASP chapter meeting called "Audit Logging Done Right". That video goes into detail about auditing and the power it has when used appropriately.

Auditing is not a new technology, and often is viewed as a boring "have to do", but it is actually a very powerful concept that lets us gain access and visibility into what the application is doing. It also gives us all of the nice capabilities of dealing with structured data . Once you recognize the utility, I hope you’ll start auditing a few more things out of the realization of it’s power, not just obligation!

References
————

http://vimeo.com/15423426 – Gunnar Peterson – Audit Logging Done Right
https://www.owasp.org/index.php/OWASP_AppSensor_Project

Week 16 - Set A Soft Session Timeout

What is it and why should I care?
A session timeout is an important security control for any application. It specifies the length of time that an application will allow a user to remain logged in before forcing the user to re-authenticate. There are 2 types: Soft Session Timeouts (today's topic) and Hard Session Timeouts (I'll cover this next week).

A soft session timeout is applied when the user does not interact with the system for a period of time.

As an example, lets say we have a system where:
1. Access to the application requires authentication
2. Attempting to access any portion of the application except login (and change/reset pw, etc.) redirects you to the login page.
3. A user logs into your system and walks away for 20 minutes and you have a 15 minute timeout
The net effect of this will be that the next interaction this user has with the system will then redirect them to the login page.

The section above shows what a soft session timeout is and does, but what is it protecting against? There are many related issues (authentication,authorization,auditing,session hijacking, etc.), but one of the primary issues is CSRF. By forcing a reasonably low session timeout, you add a security control that increases the difficulty of launching CSRF style attacks. Essentially, any attack that attempts to exploit the fact that the user is logged in is now either prevented or complicated by using this simple control.

What should I do about it?
Like many security controls, there is a functionality tradeoff with session timeouts. Many web apps we use have no soft session timeout configured, because they don't want to trouble a user with repeated logins. This is a risk decision - make things less secure for your users in order to make things simpler. If you have an application that protects sensitive data, or your users (or you) have a lower threshold of pain with risk decisions, you should opt for including a soft (and hard – see next week) session timeout.

In Java, there are several ways you can do this.

Option 1: Set the timeout in the web.xml

By far the most popular option, this is simple and allows you to configure this without having to set it in code. An example snippet showing a 15 minute timeout is below.

```
<session-config>
  <session-timeout>15</session-timeout> <!-- set in minutes -->
</session-config>
```

Option 2: Allow the app server to set the session timeout

This could mean that you allow the default (30 minutes for most app servers) or that you set the value specifically in your container. Either way, this is an option.

Option 3: Set timeout in code

This option allows you the ability to encode this setting in code. This does allow you the additional flexibility of setting differet timeouts for different users since it’s set on the session and not globally, but it’s far less common than it’s web.xml alternative.

```
httpSession.setMaxInactiveInterval(15*60); // set in seconds
```

Option 4: SSO sets the timeout

This is not a Java-only option, but still should be mentioned. Many enterprises use large single sign-on (SSO) identity systems to control access to applications. Many of these systems allow you to set the timeout for an application.

As you can see, the soft session timeout is a useful security control. It allows you to have another layer of protection for your application and your users.

References

————

http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files

Week 17 - Set A Hard Session Timeout

What is it and why should I care?
A session timeout is an important security control for any application. It specifies the length of time that an application will allow a user to remain logged in before forcing the user to re-authenticate. There are 2 types: Soft Session Timeouts (last week's topic) and Hard Session Timeouts (this week's topic).

A hard session timeout is applied when the user has been logged in for a specific period of time, no matter what.

As an example, lets say we have a system where:
1. Access to the application requires authentication
2. Attempting to access any portion of the application except login (and change/reset pw, etc.) redirects you to the login page.
3. A user logs into your system and uses the system, actively or inactively, for 9 hours and you have a hard session timeout that is set to 9 hours.

The net effect of this will be that the next interaction this user has with the system will then redirect them to the login page.

The section above shows what a hard session timeout is and does, but what is it protecting against? Whereas a soft session timeout is angled more towards preventing CSRF and similar attacks, a hard session timeout (while it does help protect against those as well) is helpful to prevent things like the permanent hijacking of an account. If an attacker does overtake an account, they can't use it forever without re-authentication. For this same reason, you should force authentication (validate old password) whenever a user attempts to change the password of the account.

What should I do about it?
Many applications, even those that avoid the soft session timeout, do include a hard session timeout. Unfortunately, it's not available simply to Java developers as an option for configuration. That means you have to either roll your own, or look for some existing software outside of the core Java/J2EE options.

In Java, there are a few ways you can enable a hard session timeout:

Option 1: Set timeout in code

There is no specific Java API call to do this. However, you could easily setup a filter (or your handler/ interceptor of choice) to perform this task. Essentially it would require you to store the last logged in time of every user and tie that to their authenticated session id. If a request is made using a session id tied to a user who has been logged in > X minutes, invalidate the session, and redirect the request to the login screen. Fairly simple idea.

Option 2: Use a third party library
Though I'm not aware of any libraries off the top of my head that do this, it wouldn't be hard to theoretically. (If one doesn't exist, you could always build it and donate it to the community!)

Option 3: SSO sets the timeout

This is not a Java-only option, but still should be mentioned. Many enterprises use large single sign-on (SSO) identity systems to control access to applications. Many of these systems allow you to set the timeout (both a soft timeout and a hard timeout) for an application.

As you can see, the hard session timeout is a useful security control. It allows you to have another layer of protection for your application and your users.

References
———–
http://www.jtmelton.com/2012/04/17/year-of-security-for-java-week-16-set-a-soft-session-timeout/

Week 18 - Perform Application Layer Intrusion Detection

What is it and why should I care?
Application layer intrusion detection is a simple concept that I believe is very, very powerful when it comes to protecting applications. Most of the topics I've covered thus far have focused on the development portion of the software life-cycle, but this topic really covers the entire span of an application, from the requirements and planning to sun-setting.

The basic concept is that you plan for, implement and monitor "bad" things that occur in your application. With this type of system in place, you look for events that appear to be undesirable in some way and then keep track of them. Over time, you can make decisions about whether those individual events turn into an actual attack.

Many developers actually do most of the work of detection already. Consider the following pseudo-code:

```
if (user has access to record) {
    get data
    redirect to view/edit page
} else {
    log exception
    send user error message
}
```

I've seen code just like this lots of times. The problem here is the handling exceptional condition. In general, people don't review logs, so if there's an attacker trying to break your application, the only person seeing the error you've caught is the _*attacker*_. With one quick addition of sending a message to your intrusion detection engine, you can start tracking these events and actually gaining knowledge into the real-time (and historical if you choose to store it) usage of your application. After you've detected an actual intrusion, you also have the ability to respond to the activity in any [legal] way you see fit. Popular options include ideas like: increased logging, manipulating user account (logout, disable), or even blocking access to certain functionality.

What should I do about it?
Let's assume I've sold you on the idea of implementing something like this (hopefully I have). What now?
Well, you have a few options on how to proceed that I'm aware of: ESAPI, AppSensor or roll-your-own.
ESAPI does have an intrusion detection engine built-in that performs some of these ideas. It is admittedly not extensive, but the core is there and can certainly be extended.
AppSensor is one such extension of the ESAPI intrusion detection engine. The implementation is more extensive than what's available for ESAPI. Additionally, the project offers a book about the overall idea, as well as significant documentation in addition to the code. Lastly, there is actually a significant update being worked on currently on the project to update both the documentation and the code.
Rolling your own analysis engine can be a small or very large project depending on your needs. Nevertheless, you can certainly take the ideas and implement them in your applications and get significant benefit.
By just adding a little bit of effort, you can gain significant insight into the overall security health of your application(s). You can see who attacked/is attacking your application in real-time or the past, and you can actually respond to events as they occur. Who wouldn't like that?

Author note: I work on the AppSensor project, so this whole topic is near and dear to me. Please take advantage of the idea whether it's in our implementation or not!

References

————

https://www.owasp.org/index.php/ApplicationLayerIntrustionDetection
http://www.jtmelton.com/2010/11/10/application-intrusion-detection-with-owasp-appsensor/
http://www.owasp.org/index.php/OWASP_AppSensor_Project
http://www.owasp.org/
http://www.youtube.com/watch?v=6gxg_t2ybcE
http://www.clerkendweller.com/2010/11/12/Application-Intrusion-Detection-and-Response-Planning-Methodology
https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API

Week 21 - Anti-Caching Headers

What is it and why should I care?
Caching is a mechanism by which browsers and proxy servers store local copies of remote objects in order to improve performance of the system by not having to fetch these items repeatedly. (That's actually a decent description of caching in general.) Caching is wonderful for performance, assuming it's tuned properly and you know what you're doing. For security, however, it can be a very bad thing indeed.

Imagine you have your bank account information or maybe your medical records up on the screen. Later another person (or maybe a piece of malware) is on your machine and is able to access the data you were viewing on that screen without you even being logged into the site anymore. Not good!

Luckily, we can use the caching directives available to enable or prevent caching, or some of both, depending on what we want.

What should I do about it?
From a security perspective, you should disable caching altogether on sensitive resources. It's up to you to figure out what those are, but that's a simple enough problem to solve generally.

As for actually setting these directives, that's commonly done using HTTP headers, Cache-Control, Expires, and Pragma. The example below shows the setting of caching headers in Java to prevent caching altogether.

```
// for HTTP 1.1
response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
// for HTTP 1.0
response.setHeader("Pragma", "no-cache");
// setting to 0 means epoch + 0 seconds, or expired in 1970, thus invalid now.
// the setDateHeader method sets a date in the RFC-required date format
response.setDateHeader("Expires", 0);
```

Just a few LOC and you've prevented caching. Many folks I've worked with find it helpful to put this code into a J2EE filter, or at least a convenient utility method. However works best for you, cache prevention is a helpful security measure that helps protect users and the data in your application, so use it!

References
————
http://code.google.com/p/doctype-mirror/wiki/ArticleHttpCaching
http://stackoverflow.com/questions/49547/making-sure-a-web-page-is-not-cached-across-all-browsers
http://www.mnot.net/cache_docs/#CONTROL
http://securesoftware.blogspot.com/2008/02/web-caches-and-security-problems-in-web.html
https://www.golemtechnologies.com/articles/web-cache-security
http://palizine.plynt.com/issues/2008Jul/cache-control-attributes/
http://support.microsoft.com/kb/234067
http://owasp-esapi-java.googlecode.com/svn/trunk/src/main/java/org/owasp/esapi/reference/DefaultHTTPUtilities.java (setNoCacheHeaders method)

Week 22 - HTTP Parameter Pollution

What is it and why should I care?
HTTP Parameter Pollution (HPP) is a technique that allows you to "override or add HTTP GET/POST parameters by injecting query string delimiters". This term was created and popularized by a 2009 paper that showed you could tinker with request parameters, specifically by sending the same parameter multiple times, and cause some applications to behave unexpectedly. What happens if someone alters a request that looks like

```
/mypage.jsp?query=abc&acct=4
```

to look like

```
/mypage.jsp?query=abc&acct=5&acct=4
```

What value do you get when you call *request.getParameter("acct")*? What happens here depends on your server. Fortunately, there's a handy slide on page 9 of this deck that shows a number of web and application servers and how they deal with this HPP issue. In general for Java, you get the first instance of a parameter – meaning you would get the value "5".

What can go wrong here? Well, there are several ideas pointed out in the paper that are excellent, including overriding parameters, modifying behavior, accessing variables, and bypassing input validation.

One thought I had that's similar to the HTTP path parameter issue I wrote about previously is that you could be doing some type of custom url-parsing for authorization checks and then could be getting a different value in the actual request. In the example I gave above, that would mean you might get the last value for "acct" in your url parsing authorization code and get the value "4", determine the user is authorized to view that account, and then your actual database retrieval code uses *request.getParameter("acct")* to get the value of the account to retrieve from the DB, which is "5", and which the user does NOT have access to. Now you have an insufficient authorization problem.

What should I do about it?
There are a handful things you can do to address HPP.

1. Awareness
You must know your environment. Know the application server you are running on and how it handles HPP.

2. Consistency
Be consistent in the way you access and evaluate parameters. If you need to access a parameter multiple times,
make sure you access it in the same way both times. Don't parse the url manually in one case, and then use *request.getParameter()* the next time.

3. Validate Input

Various techniques are available for performing this attack, and encoding is very common in most of them. Validate all your input to ensure it's in the expected format. This requires canonicalization and validation. The ESAPI framework has a good validator that performs these tasks well.

4. Detect known attacks
You can use AppSensor or AppSensor-like detection to find out when an attacker is trying this type of an attack. You could do something simple like call *request.getParameterValues("acct")* and if you get a number of parameters that is not 1, then you know that either a) you have a bug, or b) someone's tinkering with the site. You can then keep track of these events and detect when an intrusion has occurred. This should help keep you in front of the attackers. (Note: this specific example won't work for some types of input like multi-select boxes, which you generally expect to return 0 -> many results.)

HPP is an interesting technique, and unfortunately not well known or understood. It's not difficult to prevent, but does take some forethought. Hopefully this starting list helps. Comment if you have other ideas.

References
————
http://blog.mindedsecurity.com/2009/05/http-parameter-pollution-new-web-attack.html
http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf
http://www.jtmelton.com/2011/02/02/beware-the-http-path-parameter/
https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
https://www.owasp.org/index.php/OWASP_AppSensor_Project

Week 23 - HTTP Header Injection

What is it and why should I care?
HTTP Header Injection is a specific injection attack that affects HTTP headers. It involves being able to manipulate the header data to cause various problems (response splitting, CRLF injection, cache poisoning, XSS, etc.). In general, it's a lesser known and understood attack, which is usually a recipe for minimal protection in applications to prevent it, and that is certainly the case with header injection.

Likely the most common would be CRLF injection (Carriage Return [%0d or \r] Line Feed [%0a or \n]), which involves adding the CRLF to the header, which results in proxy servers/caches/browsers mis-interpreting the response in an insecure manner, and gives the attacker control of part or all of the "split" response. This can lead to the other issues mentioned above.

What should I do about it?
The recommendation for header injection is the same as that for all injections – validate and encode. For header injection specifically, you should ensure that you:

1. Canonicalize – make sure the data is in its' simplest form before validation.

2. Validate – perform whitelist (only allow these few good characters) validation as opposed to blacklist (only reject these few bad characters) validation. Validation should ensure you don't allow CR/LF characters in any encoded form.

3. Encode – encode the resulting output if necessary. If your input validation is tight enough, this step is just a layered defense, but you should encode in case anything ever slips by validation for some reason. Again, take special care to encode CR/LF here.

References
————
http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf
http://www.securityfocus.com/archive/1/411585
http://packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf
http://lists.grok.org.uk/pipermail/full-disclosure/2006-February/042358.html
http://cwe.mitre.org/data/definitions/113.html
https://www.owasp.org/index.php/Interpreter_Injection#HTTP_Response_Splitting
https://www.owasp.org/index.php/HTTP_Response_Splitting
https://www.owasp.org/index.php/CRLF_Injection

Week 28 - Unit Test

What is it and why should I care?
Unit testing is the term generally associated with the process of writing code specifically purposed for testing your application functionality. You write test code to run your functional application code and verify the results.

Note: Unit testing is actually a specific subset of this idea focused on the individual unit (generally recognized as a single class and/or method). Other testing techniques (integration, end-to-end, functional) are also extremely valuable, but unit testing is the most well-known and most of the concepts are transferrable.

So, what does unit testing provide to me and why should I want to use it? Let me try to address that in two parts.

1. Development
Since unit testing is fundamentally an activity that goes on during the development of software, let me first consider what developers use it for. Considering the proper use of unit testing (I'll address improper use below), it provides you primarily with confidence and quality (which are arguably the same thing in my constrained definition here). I've had discussions with people who say it offers a lot more, and certainly you can argue that point, but for me all of the myriad reasons come back to confidence and quality. You've exercised the code enough (and you continually do so) that you feel confident that your code works the way you expected it to work. There's a lot that goes into this ideal world where you have good tests, but it can be practically done, and it makes an immense difference in the quality of code that you produce.

2. Security
So, unit testing inside the development process produces confidence and quality. What do we get from it when applied to security? … The same thing. Security is no different from other code from the perspective of correctness, though it is arguably less well understood in many organizations. At the end of the day, though, we should treat security requirements with the same rigor we would treat performance, functionality, or any other requirement. We should specify what we expect, build to that, test to that, and produce that. Unit testing is a fantastic tool that can be applied to produce confidence and quality in the implementation of security in an application.

What should I do about it?
"You should stop everything you're doing on the app you're currently working on and go back and make sure you have at least 80% code coverage with your tests." I put that in quotes because, sadly, I had a previous job (long ago) where I was informed that this was exactly what I needed to do. I thought they were joking at first, but I quickly found out that this was their real requirement. No matter that the number was seemingly arbitrary and that the tests they gave me as "good examples of what we're looking for" didn't actually examine the output of the test, but rather just ran the code (READ: not a real test). No, testing was all about hitting a number, and saying you did it.

If that's the way you look at testing, you get no value from it, and it actually costs you quite a bit in development, configuration and execution.
Ok, so if not that, then what?

Getting Started
Here are a couple ideas I'd recommend when first starting out on the path to good unit testing.

1. Read a good book or article on unit testing and the concepts and ideas. Kent Beck in particular has produced excellent resources on the topic.

2. Read somebody else's (good) test code. I always recommend this when folks are getting started. I usually recommend something like the Spring framework, which has some of the highest quality code around, to use as a starting point to get good ideas for how to test. Look at the class they are testing and the tests they wrote for it. This will get you a good idea of what's going on.

Once you've gotten through the initial learning phase regarding real unit testing, here are a few more ideas that I've seen work pretty well. Certainly add to or remove from this list to make it work in your environment, but at least consider the concepts. (Also, here is someone else's list if you'd like some additional ideas)

Good Ideas

1. Build a regression suite
A huge advantage of writing another full set of code to test your functional code is that the code stays around (also can have a bad result at times for maintenance, but let's focus on the positive here). That means you have an incredible regression test suite. This is one of the most powerful things about unit testing. You've written some function, and a few tests to cover it. You modify the function in order to add some new functionality, and now one of your tests breaks. This happens constantly, but is a strong safety net. This is one of the single best things I've ever had to learn the hard way that I needed.

2. Write good tests
Writing a test that exercises a method is trivial – call the method with necessary parameters … done. This is the cheap (and useless) way to do testing. What you should do (and you'll learn this in the book) is write positive and negative tests for the method. Think about nulls. Think about boundary conditions for your parameters. What happens if I send a negative number in or what if I send Integer.MAX_VALUE? Inspect the return values from functions and make sure they align with your expectations.

This type of "how can I break this" thinking usually comes natural to good security folks, so it's actually a good fit for them. However, good developers certainly get this mindset as they test more, and it's extremely beneficial for both functional and security tests. This is a cool and powerful way to get developers who aren't security minded going with security.

3. Start small
This piece of advice is given for almost everything, but that's because it works. Certainly begin practising on some insignificant code or even just make a sample app to get going. However, when getting good tests into your code-base, I generally recommend picking either/or a) your most important classes or b) your most buggy classes. You'll likely be surprised at the number of bugs you find when you start the process, but just remember those are bugs your customer won't find!

4. Capture metrics (and use them)
There are lots of tools that help you capture metrics over source code or bug repositories. It's really a fun exercise to see the number of reported bugs go down as your number of unit tests goes up. You can also use data like the number of bugs in code in a certain part of the application, or written by a certain developer to identify where to focus your testing efforts.

5. Use tools to help you out
You could certainly write your own unit testing framework, but why when others have already done it. In the Java world, the reigning king is JUnit. Martin Fowler is quoted as saying "never in the field of software development have so many owed so much to so few lines of code" when referencing JUnit. The idea behind JUnit is simple, but it's notable because the execution is great, and there are lots of additional tools, like IDE support and build tool support, that make using it so simple.

6. Do code coverage
A bit earlier, I knocked on code coverage because I feel it's a pretty weak metric in and of itself. However, it is a valuable tool in combination with a quality process. If you know your tests are good, and you know that you cover 85% of your significant code, then you're doing pretty well. Again assuming that you're tests are good, this can be another metric in the process that points to improvement.

As far as tools go, there are several, but I generally recommend EclEmma for this purpose.

7. Code review your tests with the code they're testing
Yes, you should code review your tests. This is part of the "writing good tests" process. You need to ensure that people are testing their code, and that they are doing it correctly. Just fold this into your code review process, and you're golden.

8. Test different requirements
Don't just test functionality. Test for performance. Test for security. Test for other types of requirements. All these things can be added, and they'll improve your confidence and quality.

9. Add a test when you get a bug
When you find a bug, do this:
- add a test that covers the bug
- make sure all the existing tests pass, but the new test fails
- fix the code
- make sure all tests pass

This process allows you to make sure you don't regress over time and that the same bug doesn't come back to haunt you.

10. Write your tests first … at least on paper
A lot of people swear by Test Driven Development, the idea that you write all your tests for a method/class _before_ you write the class. You ensure all of them fail. You then write the class, and by the time all the tests pass (assuming you've written tests to cover every scenario and that the tests are accurate and "good), then you're done with coding.

I'm personally not a stickler for this process, though there are advantages. If you're doing a brand-new project, this can be great, but a lot of the work developers do is on legacy code (even if only a few weeks/months old), and a lot of that doesn't have tests, so we have to have a process that allows for that.

What I will say is that you should write your tests separately from the code. All tests come from the requirements, not the code. This is an important point, and is where most new test writers fail – they write the tests to work with the code that's there, not to test the actual code. Of course if you're writing tests meant to work for the code you see, those tests should pass. It's divorcing yourself from the implementation when you're writing tests that's important. This is something you shouldn't sacrifice on.

11. Don’t let testing replace modelling altogether
In this superb talk, Glenn Vanderburg points out that software engineering relies quite heavily on modelling, when we can achieve many of the same desired results through testing. He talks about aerospace engineers using modelling, followed by a prototype, but says that of course they would use the prototype every time if it were as cheap as the modelling, since it’s actually testing the “real thing”. I think this point does ring true to an extent. I don’t personally like using modelling extensively because in practice it’s a) overkill and b) outdated as quickly as you can create it. However, I do think higher-level modelling adds significant value and that testing is never going to be able to effectively replace it because the value is in the mental exercise of considering the system at a higher level, which testing often doesn’t do properly, or at least effectively.

12. Continuous testing
Continuous integration platforms are common nowadays, but there are still lots of people not using them (but you SHOULD!). However, they really are extremely helpful for testing. They all include the idea that the unit tests should be run on every build. This is powerful because it forces you to find the broken tests quickly and fix them while what you changed is still fresh on your mind. If you don’t have a CI environment, your build system and configuration should at least allow you to do this every time you build, and then you should have your process include that step.

13. Get unit testing into your SDLC.
It sounds sad to say, but you have to force testing as a requirement or there will be those that won’t do it. Testing is a definite and important step in the SDLC and should be represented as such. I will note here that I fully believe your good developers will really enjoy the effect of the unit tests once they use them (even if they still don’t like writing them). Every good developer I’ve worked with has thought good tests were worth their weight in gold.

14. Add in tests for integration, functional, end-to-end, etc.
Unit testing is really a subset of the larger testing scope. There are tests at the component, business function, application, etc. levels and all of these should be tested. All of the same rules above apply to these as well. It’s really great to be able to hit a button and know that your application is being run through the gauntlet of unit, integration, functional, and end-to-end tests. It’s a pretty powerful concept.

One notable (and cool) example of this specifically as it relates to security is the use of security regression testing using the OWASP ZAP tool.

Conclusion
Unit testing is one of those ideas that’s not really specifically for security, but actually does quite a bit for security if applied correctly. It’s also a great way to get developers and security people working together for a common goal.

It’s also near and dear to my heart as I think it’s been the single most important idea that’s helped me improve as a developer over the years. I hope it’s as useful for you as it has been for me.

References
———–
http://www.confreaks.com/videos/282-lsrc2010-real-software-engineering
http://swreflections.blogspot.com/2012/01/static-analysis-isnt-development.html
http://www.readwriteweb.com/archives/12_unit_testing_tips_for_software_engineers.php

Week 29 - Manage Resources

What is it and why should I care?

Resource management has been an issue in programming for a very long time, and it's one of those issues that affects the A (Availability) of the classical C-I-A triad in information security. It's effectively where you gain access to some (generally expensive) resource (think database connection, file handle, etc.), and then don't properly return access, hence you don't release the associated resources (disk space, memory, file handles, network connectivity, etc.)

While not as high-profile as losing your user account database with all your passwords stored in roughly the equivalent as ROT-13 (the current flavour of the month in the security news), it can be pretty embarrassing to have your site fall over constantly due to resource leaks. Crashing, or at least instability, is the inevitable outcome of poor resource management. It's also not fun being the guy or gal that gets to restart the server every night at 2AM to make sure it will work without crashing the next day (yes, I have a fried who used to do this).

What should I do about it?

In order to prevent resource leaks, you need to properly release resources when they are no longer needed. In Java, that can look like a few different things.

1. finally block

The traditional way to close resources in Java is to use the finally block that's provided by the language. This is a fairly simple process once you know the idiom. It essentially looks like the following.

```
BufferedReader bufferedReader = null;
FileReader fileReader = null;

try {
    String line;
    fileReader = new FileReader("/tmp/myfile.txt");
    bufferedReader = new BufferedReader(fileReader);

    while((line = bufferedReader.readLine()) != null) {
    System.out.println(line);
    }

} catch(Exception e1) {
    e1.printStackTrace();
} finally {
    if(fileReader != null) {
        try {
        fileReader.close();
        } catch(Exception e2) {
        e2.printStackTrace();
        }
    }

    if(bufferedReader != null) {
        try {
        bufferedReader.close();
        } catch (Exception e2) {
```

```
        e2.printStackTrace();
        }
    }
}
```

Here we read a file, and print out each line. Notice that a good chunk of our code is devoted to resource management. However, if you generified this method by parameterizing the filename and returning a string representing the file contents, then you could make the method reusable. This is exactly what many file reader style utilities do.

2. try-with-resources
So why do people not close resources? Well, it's part ignorance of the issue, copying bad code snippets and laziness from what I've seen. What would be really nice is if the language itself gave you a way to get around this issue by making it automatic. Well in Java 7, after many, many requests for the functionality, it's finally been added as the try-with-resources statement.

Try-with-resources is only available in Java 7 and later, so you can't use it if you're on an older version of Java. However, if it is available to you, it's a nice little abstraction of syntactic sugar that allows you to push the resource management grunt work to the language itself. Below is a modified version of our initial code using try-with-resources.

```
try (FileReader fileReader = = new FileReader("/tmp/myfile.txt");
     BufferedReader bufferedReader = new BufferedReader(fileReader)
) {
    String line;

    while((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
    }

} catch (Exception e) {
    e.printStackTrace();
}
```

As you can see, the try-with-resources version is much shorter, cleaner and accomplishes the same thing. That's generally a good thing when it comes to code.

3. Use and create libraries with proper resource management
There are good examples of open source libraries that do this effectively. For instance, Hibernate and Spring both do proper resource management for database access. Just by using their libraries in the recommended configuration, you are accessing and releasing resources properly.

In addition to using open source libraries, you'll almost always find some utilities classes or libraries in applications related to resource management (think FileUtils, StreamUtils, DbUtils, etc.). If you build your resource utilization code into reusable libraries that perform proper management, then all your applications benefit.

In conclusion, resource management is one of those issues which doesn't get a lot of attention, but that can be critical to the stability of your application. Also, we showed that it's pretty easy to properly manage resources in Java if you know what to do, particularly when you move that functionality into reusable libraries.

References
————
http://www.mkyong.com/java/try-with-resources-example-in-jdk-7/
http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Week 30 - Authentication

What is it and why should I care?
Authentication is the process of verifying that someone is who they say they are. Essentially a user claims an identity and then must provide some form of proof of identity. In most systems, the identity is some form of username and the proof is a password. In the simplest case, if the username and password provided match a user record known to the application, then the user is authenticated and proven to hold the claimed identity.

Authentication is one of the most important aspects of security within an application because it's the starting point for most other conversations. Many security solutions start with the basic assumption of an authenticated user. That means we already got authentication right! Authentication is difficult to get right, but can be done.

What should I do about it?
I've already written down a few ideas about authentication in a previous series, but I'll try to add a few additional ideas here.

Let's start with a few easy ones that most people are very aware of.

Authenticate over a secure channel. For the web, this means SSL/TLS. Performing authentication over an insecure channel makes intercepting credentials trivial (unless other encryption mechanisms are used). This is an easy win.

Provide logout feature. Once you've logged in, you need to be able to logout. Putting a logout link/ button on every page in an easily visible place makes it simple for your users to terminate their authenticated session when they please.

Provide related functions. If you provide an authentication function within your application, that means you'll need to give your users a mechanism to update their credentials (passwords, tokens, etc.). For the common case of passwords, you'll need to provide additional functions such as password change and reset. In addition, you'll want to have an account lockout process to prevent brute-forcing of your authentication system. There are other functions you could add, but these are a good base set. These functions are also easy to get wrong, so pay attention to their implementation.

Use generic error messages. You'll need to ensure you don't leak information with error messages within the authentication subsystem. The classic example is someone logging in with a bad username and password and being told "user does not exist" versus logging in with a good username and bad password and being told "password does not match". Here, you've leaked that a user exists with that username, so you just have to guess their password and you should be in. What should be done is the user should be given a generic error message like "Authentication failure" and that way, you can't determine what the issue was.

Update (8/15/2012): There is far less benefit for generic error messages in systems where users can easily register for their own usernames, thereby circumventing the knowledge gap of whether or not an account actually exists.

Handle session management properly. Session management is a topic in and of itself. For J2EE folks, unless you have a good reason, you should be using the built-in session management capabilities of the platform (jsessionid). This is a solid implementation that you don't have to build yourself – use it.

One note here that is beyond this topic is stateless session management (think REST). Here, there are generally well-known practices (like http basic auth for REST) that are used instead of built-in session management. Just be aware of the systems you're using, and consider their security implications.

Consider single sign-on. Single sign-on can be a helpful option if you're in an organization that has a good implementation. Most of them work in a simple way. The user is authenticated by either being redirected to a common authentication function, or by each application accepting credentials and using these to authenticate against a single user repository. Either way, the actual user storage and authentication processes should only have to be built once, which is a good thing for reuse. This just makes it all the more important to get the process right.

Update (8/15/2012): While SSO does have it's uses, there are some concerns.

- You've now got a much larger repository of users all coming through a single point of failure, not to mention the increased risk of losing all of your accounts at once.
- You are also granting access across all applications (regardless of security requirements and risk tolerance) with a common time-out, as well as other services which are provided by the SSO implementation. There are situations where you certainly might want to choose the features and configuration of your authentication system depending on the risk profile.

All of these issues should be weighed, both good and bad, when considering an SSO implementation.

Now let's discuss a few of the less frequently implemented ideas.

Re-authenticate high value transactions. If you're application has an important transaction that needs to take place with an already authenticated user (e-commerce purchase, wire funds transfer, view medical records), you should re-authenticate the individual transaction. This ensures the user viewing your site at that moment has been properly authenticated. This can help prevent issues where users step away while still involved in an authenticated session and another user being able to use the system as the originally authenticated user.

Use multi-factor authentication. This is a costly measure, but can be a great way to increase security. Many online systems are starting to use a form of this by allowing you to have a code sent to your phone that you have to enter during the authentication process. That means an attacker would have to know (or guess) your password and have access to your phone in order to authenticate as you. This decreases the likelihood exponentially, and is an excellent option.

Outsource your authentication. In the last few years, it's become more common to let others do your authentication for you. Essentially, you can forego the authentication system altogether, and let your users authenticate to your site by sending you an authentication / access token from a well-known authentication provider (like google/twitter/facebook). All the authentication goes on at those other sites, and you are just told "OK, we know this is Bill and here's Bill's access token." As long as you check Bill's token and it's valid, then you're good to go. This is not an option for many organizations, but if you're building a site that could use one of these services, it can shrink the size of your application, and the general sense is that "those guys" do a better job at security than you would. In general, that's probably reasonably fair for the big providers in particular. Most of these providers are using the OAuth protocol right now.
In conclusion, authentication is a critical piece of most any application. It can be difficult to get right, but the steps listed above should put you on your way to getting it right.

Update (8/15/2012): Updates were made to the doc based on helpful review and feedback by Jim Manico.

References

————–

https://www.owasp.org/index.php/Authentication_Cheat_Sheet
https://www.owasp.org/index.php/Top_10_2010-A3-Broken_Authentication_and_Session_Management
https://www.owasp.org/index.php/Guide_to_Authentication
https://www.owasp.org/index.php/Testing_for_authentication
https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
http://www.jtmelton.com/2010/06/16/the-owasp-top-ten-and-esapi-part-8-broken-authentication-and-session-management/
http://oauth.net/

Week 31 - Access Control (1)

What is it and why should I care?
Access control, also known as authorization, is the step that comes after authentication. Access control is the process of "mediating access to resources on the basis of identity" [from here]. It assumes you have determined the identity of the user (whether known or anonymous) and are now making decisions about whether or not that user (or a group they represent) can access a given resource.

Access control is one of the most exploited security controls in many systems. Often times access control is not performed well or maybe not at all. When it exists, many times it is not applied consistently, and therefore leaves large uncovered gaps in the security of an application.

There's a good bit to unpack in the statements above, and there are different models for access control (MAC, DAC, RBAC, etc). However, there are some basic concepts for access control that generally need to be met in web applications, and those are the ideas I plan to lay out in the next few posts.
What should I do about it?
Note: This is part 1 of 3 in the sub-series related to access control. More is coming.

Note: While there are plenty of helpful best practices for implementation that I'm not going to touch on (like: do not depend on the client at all for access control decisions, apply principle of least privilege, centralize routines for access control, use a mechanism that allows simple policy changes, etc.), I've added some references at the bottom that include some of these recommendations. Please read if you're building access control systems or if you're just interested.

Given the caveats in the notes above, what I would like to talk about for this first post is one way that you should consider limiting your users' interactions with your application: by functionality. Limiting users by functionality simply means that certain users (or groups) can access certain functions on your site.

The first popular framework that made this really simple was Struts 1. In Struts 1, you could configure which roles had access to which actions in your application with a really simple mechanism that looked something like this:

```
<action name="..." path="..." roles="...">
    ...
</action>
```

J2EE in general has had a mechanism (roles and security constraints) that allows you to perform this function for a long time, but it can be a little clunky and requires a lot of configuration relative to the Struts 1 mechanism.

In more recent history, the Spring framework picked up the Acegi project and renamed it Spring Security (ok, not that recently). Spring security has a large focus on authorization since that's one if it's core functions.

Many frameworks provide a way to accomplish this in their own specific way, but you just have to be aware of the mechansims (or lack thereof) of any given framework you're using.

The key capabilities you will generally want at a minimum are feature and function management.

What I mean by feature management is a mechanism to control access to several functions at once grouped by general capability area. For instance, in SpringMVC, you might have a *TradeController* that controls access to all of your trading features within your application. Certain users/groups may need no access whatsoever to that feature, so having the capability to exclude components/features of the application to whole subsets of users is nice and simplifies management.

Function management means you can control access to the individual function that is being called for a specific request, ie. *tradeShares*(). Having the ability to specify which users/groups may access a specific function is critical.

A couple of finer points here:
1. If you only get one or the other (feature/function management), pick function management – you need the specificity.
2. If you get both, be careful regarding inheritance. The most flexible system I've seen thus far (that doesn't get you into a whole heap of trouble) is allowing inheritance from the feature to the function IFF the function does not specify any allowed users/groups of its' own. Otherwise, you get into merging allowed users/groups, and that can get messy, not to mention very confusing for those trying to do the right thing for security.

In conclusion, limiting users and groups by access to features and functions is a good first step to providing good access control. There are other ideas coming in the next couple weeks to watch out for and handle additionally, but this is a good start.

References

———–
https://www.owasp.org/index.php/Category:Access_Control
https://www.owasp.org/index.php/Guide_to_Authorization
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

Week 32 - Access Control (2)

What is it and why should I care?
We defined access control in part 1 of the access control sub-series, so let's move on to talk more about what we do about it.

What should I do about it?
In part 1 we discussed limiting your users' interactions with your application by functionality. This time I'd like to discuss an additional consideration for limiting interaction: by data. This simply means does your user have access to this specific data. A simple example should help clarify. Consider online banking, where millions of people might have access to the same function (view account), but only you (and bank employees) have access to view *your* specific account data.

One generally painful issue here is that there's no generic way for me to say how this specific recommendation will play out in your application. The "data" aspect of this issue means that it is usually coupled to your data model. If users are limited to their bank accounts, you need to consider the way you store data and see how you link users to bank accounts. You then need to limit your queries in the appropriate manner. There are more generically applicable techniques that generally involve data labeling or tagging, but they're often very involved and complex to get setup, and if you need a large solution like that, you probably are already aware of that fact.

A related issue is that most frameworks (save maybe ESAPI's interfaces) do not provide any API help with regards to access control related to data, or even a reminder that you should do it. For the most part, there are always custom APIs within an application to deal with this issue. That means that there's often spotty coverage. Some APIs are protected properly, but then others will be wide open.

This means that part of your development process will include identifying those pieces of data that should be restricted to only certain users or groups, and providing centralized access control APIs that are used to evaluate permissions to the given data. You might have a simple API that looks something like the following:

```
if(isAuthorized(Bill, VIEW, Account.class, 17398)) {
    //do real work
} else {
    //user is doing something bad, or there's a bug in the app, fire off event
to appsensor intrusion detector
}
```

The access control check above asks if "Bill" (user) has access to perform a "view account" (function) on the account (data) represented by id 17398. If you've done authentication correctly and performed a check like this, you're better off than many applications out there.

The next post will look at a few other data points to consider in your access control decision matrix. However, the basic function/data checks described in the first 2 posts are good enough for many uses. However, it's critical that both be applied correctly and consistently.

In conclusion, we saw that performing access control checks by considering access to data is an important part of any access control scheme. It can be more difficult to build than functionality checks since there's rarely help from common frameworks and it is dependent in large part on the data model of the individual application, but rigorous focus on building the appropriate common APIs can make it much easier to apply consistently.

References

————

https://www.owasp.org/index.php/Category:Access_Control
https://www.owasp.org/index.php/Guide_to_Authorization
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

Week 33 - Access Control (3)

What is it and why should I care?
We defined access control in part 1 of the access control sub-series, so let's move on to talk more about what we do about it.

What should I do about it?
In part 1 we discussed limiting your users' interactions with your application by functionality. In part 2 we discussed adding data access as a criteria. This time I'd like to discuss an additional consideration for limiting interaction: other. Yes, I mean to be generic. There are many other specific data points we can use to generate context in order to make decisions about access control. I'll outline a few below for your consideration, but consider this a starting list to get you going. You should definitely expand this list and make it specific to your environment.

The following are a few ideas to provide additional context to your access control decision matrix.

Date / Time
The time of day can be a critical factor for deciding whether or not to allow access in certain environments. A simple example might be that you only allow access to employees M-F 8-6. Outside of those hours, the employee may have little to no access within the system. Another example might be that you only perform certain tasks once a quarter at a certain time. You can set your policy to only allow a small window wherein those changes can be made.

Physical Location
If you expect a given user to only login from the UK and they start showing up logged in from Australia, you may not want to allow access. This geo-location capability becomes all the more important with mobile devices. You could theoretically limit a user to only logging in when they are at home or work, and nowhere else. The granularity of location differs depending on your geo-location provider, but many services are starting to get pretty accurate, particularly in the mobile device space.

Type of Device
You might want to limit certain applications to only be used from a mobile device or to never be used from a mobile device.

IP Address
You may want to limit access by specific IP addresses or ranges.

Browser Type / Version
People have been limiting applications by browser type/version for a long time. It's only been in the last few years, though, that I've heard of people doing it for security as opposed to functionality. I've seen apps now block older browser versions because they don't support the security capabilities required for a given application.

These are just a few ideas I've seen implemented to help make access control decisions. Some of these data points are logical and simple to get. Some are more difficult to find. Some have higher accuracy than others. The point of this article is to point out that there are additional data points that can be used to make access control decisions. Decide which ones make sense for your application and then use them.

In conclusion, we saw that we can use additional contextual data to make access control decisions that are more sophisticated than the norm. We can use data points that we already have access to in order to limit access to our systems in a more granular way. Done properly, this can greatly improve the security of our applications.

References

———–
https://www.owasp.org/index.php/Category:Access_Control
https://www.owasp.org/index.php/Guide_to_Authorization
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

Week 34 - Separate Admin Functionality

What is it and why should I care?
The idea of separating administrative functionality may strike some as odd. By administrative functionality, I’m just grouping those higher criticality functions (generally user/group/role management) that have the characteristic of affecting the application at large, generally through privilege escalation. The idea here is this:

- I have some critical functions that allow user management
- If these functions were exploited, privileges could be added to or removed from users
- I don’t trust that these functions are perfectly protected in all cases

You may disagree with that flow, particularly the last step, and you may feel you are perfectly secure. In my experience, most apps have holes, and exposing critical functionality *unnecessarily* can allow those latent vulnerabilities to have a much larger impact.

What should I do about it?
First, determine if this is even an issue in your application. If you don’t manage users in your app, you don’t need to think about this one. If your application is only internal-facing, maybe your risk profile means you don’t care about this issue in that circumstance.

Next, consider your options:

Status Quo
You could leave things as they are (assuming an existing app) and leave the admin functionality in place. Depending on the value of the protected resources, this may be a legitimate option, but for an important application, it likely isn’t.

Build another application
You could build a standalone app that is separate from the general application. This standalone app could have different authentication requirements (multi-factor), or it might only be deployed internally since all the admins are internal, or access to it might be restricted to certain IP addresses, etc. You have a lot of options in this scenario, but it does require a different codebase and/or deployment.

Subset the application
You could also build a specific subset of your application related to administrative tasks. This is probably the most common option across applications I’ve seen. The issue is that access to this functionality is treated the same way as the general application. You can improve this by having additional controls. Maybe you require client certificate authentication for this subsection, or maybe multi-factor authentication. You might also have additional source restrictions (IP addresses, browser, etc.) for this subsection of the application.

No matter which option you choose, make sure that you consider the issue as it applies to your environment, evaluate the alternatives, and make an informed decision based on the individual circumstances of the situation.

In conclusion, administrative functionality in an application is a high-priority target for exploitation by attackers given the valuable functionality it exposes. With some simple and straightforward changes to your application, you can greatly reduce the risk of this functionality being exploited.

Week 36 - Solve SQL Injection

What is it and why should I care?
SQL Injection (SQLi) is an issue that is caused because of poor code/data separation. The general issue is that a developer intends the user input to be interpreted as data, but an attacker can manipulate the input to cause the database to interpret the input as commands.

There have been a lot of devastating attacks recently using SQLi. You don't have to look very far to see the amount of damage it's caused, involving great financial, political and reputational impact. You probably won't have a tough time selling the C-Level exec on the idea that SQLi protection is important.

What should I do about it?
Note: One great resource which I drew from for both this post as well as the more in-depth post is the SQL Injection Prevention Cheat Sheet over at OWASP.

I've already written pretty extensively here on the basic problems with and solutions for SQLi. However, I'll cover the solutions briefly here for clarity's sake.

1. Use Parameterized Queries (Note that in Java, parameterized queries = prepared statements)
Parameterized queries are a great solution because they're fairly simple to learn,are ubiquitous and don't require your developers to learn another API – it's just SQL. (One nice additional benefit: it will make your security guy happy)

2. Stored Procedures
Stored procedures are a good option in some environments where you want to have the query management done separately from the code, and are often helpful for performance if properly tuned.

3. Output Encoding/Escaping
Output Escaping is more of a last resort. It can be properly done, but it's usually more work than the other 2 options, requires more customization, and is more error-prone for most developers. If you go this route, it's definitely advisable to use a common security framework such as ESAPI so that your controls are consistent across all your applications.

Now that we've covered the basic technology solution for SQLi in Java, let's consider the larger context of eradicating entire classes of vulnerabilities from our codebase as I discussed last week. Let's look at each specific sub-point I mentioned in that post and consider what you might do regarding SQLi.

1. Understand the problem.
I'd read available documentation on SQLi to make sure I understood the issue as well as I *think* I do. I'd also go read about SQLi for the database platform(s) that is used in my environment. I'd look for any specific weaknesses in implementation that I need to be aware of.

2. Consider all use cases where the issue can occur.
I would think about all the different types of database interactions that occur in my application portfolio. How many languages am I developing in or supporting (Java, C#, Cobol)? What platforms

am I developing for (web, services, desktop, mobile, mainframe, html5)? What interaction paradigms are they using (OLAP, OLTP, Warehousing, Batch)?

3. Evaluate solutions.
I would look at the SQLi cheat sheet from OWASP (and any associated weaknesses with the approaches that it espouses). Once I understood the recommended solution(s), I would consider whether said solution(s) would work in the environment in which I'm working. I would focus on using the best solution possible for everyone who can use it, then consider exceptions if it's not supported by a given interaction paradigm based on business need.

4. Institutionalize the chosen solution.
After deciding on a solution (say parameterized queries), you have to work with development teams to determine the current status of database interactions (are they using dynamic sql now or are they already on parameterized queries?). This step is a good place to (re)train developers on this specific topic from a security perspective, since they'll be implementing the solution. You then just have to do the work of fixing queries, and re-testing applications to make sure everything still works as expected. Note: In addition to solving SQLi from the "code" perspective, there are also additional steps you can implement here that increase assurance such as lowering the privileges/access of users so they can't cause certain types of harm even if they were to be malicious.

5. Add technology and processes for verification.
At this point, I would add a few tools in to make sure the technology is being used. I would make sure that there were static and dynamic analysis tools in place to check for SQLi. I would also make sure to code review closely any queries made by the application. I might go so far as to write custom rules for my static toolset that "trusts" my safe output functions to reduce any false positives. Lastly, I would monitor logs for suspicious activity. Using something like AppSensor is a way to automate some of that process.

Hopefully the approach I described makes sense to you. There are certainly additional steps you can take, but this should be a good start. The idea is that you don't just go and squash a few bugs. The hope is that you do solve the immediate problem, but make it impossible (or as close as you can get) for the problem to resurface.

References
———–
http://www.jtmelton.com/2009/12/01/the-owasp-top-ten-and-esapi-part-3-injection-flaws/
https://www.owasp.org/index.php/SQL_Injection
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Week 37 - Solve Cross-Site Scripting

What is it and why should I care?
Cross-Site Scripting (XSS) is another issue that is caused because of poor code/data separation. The general issue is that a developer intends the user input to be interpreted as data, but an attacker can manipulate the input to cause the browser to interpret the input as tags or commands.

XSS is exceedingly popular and well-known, and along with SQLi, is probably 1 of the 2 best-known vulnerabilities across web app security.

What should I do about it?
Note: One great resource which I drew from for both this post as well as the more in-depth post is the XSS Prevention Cheat Sheet over at OWASP.

I've already written here on the problems with and solutions for XSS. However, I'll cover the basic solutions briefly here for clarity's sake.

1. Canonicalize Input
Get your input data into it's simplest base form in preparation for validation so that you're more confident your validation routines aren't being circumvented.

2. Validate Input Using a Whitelist
Whitelist validation (accept only known good data) is a key tenet of good security. Check the content of the data, the length, data type, etc.

3. Contextual Output Encoding/Escaping
Output Escaping is the last step, and must be done for the appropriate context. You must understand where you're sticking data, and how that location is interpreted from the browser's view. This can be a sticky issue, so this one gets a bit problematic.

Now that we've covered the basic technology solution for XSS, let's consider the larger context of eradicating entire classes of vulnerabilities from our codebase as I discussed a couple weeks ago. Let's look at each specific subpoint I mentioned in that post and consider what you might do regarding XSS.

1. Understand the problem.
I'd read available documentation on XSS to make sure I understood the issue as well as I *think* I do. The cheat sheets at OWASP are good for basic theory, but there are books that talk about the issue in more detail, and there are additionally books that discuss how browsers work that can be helpful to understand. XSS is actually one of the more complex vulnerabilities around, and it actually is still changing because it's so closely tied to browser feature sets which are always expanding. In addition, javascript frameworks of significant capability have had a huge impact on web applications in the last 5-7 years, thereby compounding the problem.

2. Consider all use cases where the issue can occur.
I would think about all the different types of XSS interactions that occur in my application portfolio. How many languages am I developing in or supporting? What frameworks am I developing with? Do I need to allow users to enter HTML that is then displayed (JSoup / AntiSamy problem space)? Do I have a constrained browser environment (or do I still need to support IE 6)?

3. Evaluate solutions.

I would look at the XSS cheat sheet from OWASP (and any associated weaknesses with the approaches that it espouses). Once I understood the recommended solution(s), I would consider whether said solution(s) would work in the environment in which I'm working. For XSS, there's really not a lot of alternatives to how to solve the basic problem. However, there are differing implementations of solutions. For example, there are solutions where that require more effort to solve the issue, but do leave the maximum of flexibility (plain ol' JSP's), there are frameworks that default to solving the most common version of the problem (struts, springmvc, etc. – all default to basic html entity encoding), all the way up to templating solutions which constrain flexibility but offer a safer-by-default scenario (JXT). Consider also solutions for special use cases, such as the requirement to allow certain constrained amounts of HTML along with plain data, and how you might solve that class of issue (JSoup / AntiSamy).

4. Institutionalize the chosen solution.
After deciding on your collective solution, you have to work with development teams to determine the current status of vulnerability to XSS (beware: it's probably gonna be pretty bad). This step is a good place to (re)train developers on this specific topic from a security perspective, since they'll be implementing the solution. You then just have to do the work of implementing the chosen framework(s), making necessary modifications to the source, and re-testing applications to make sure everything still works as expected.

5. Add technology and processes for verification.
At this point, I would add a few tools in to make sure the technology is being used. I would make sure that there were static and dynamic analysis tools in place to check for XSS, though tools vary in usefulness here and coverage can sometimes be spotty. I would also make sure to code review UIs to see where dynamic data is emitted. I might go so far as to write custom rules for my static toolset that "trusts" my safe output functions to reduce any false positives. Lastly, I would specifically look for "known bad data". Using something like AppSensor or a WAF is a way to automate some of that process.

In conclusion, hopefully it's clear that solving XSS is a manageable task, but just takes some focused effort on research, training, implementation and verification. The pattern I've described for both SQLi and XSS are useful to follow for other vulnerability classes in order to really build a robust environment and suite of applications.

References
———–
https://www.owasp.org/index.php/XSS
https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
https://www.owasp.org/index.php/Abridged_XSS_Prevention_Cheat_Sheet

Week 38 - Create A Reusable Security Framework

What is it and why should I care?
Software reuse is a ubiquitous practice in software development. One study says that “80% of the code in today’s applications comes from libraries and frameworks”. That’s a lot. There is already a lot of research about software reuse and its benefits.

While the research exists, there’s no need to read it to get the basic answer, as any developer will tell you that it’s imperative to create and use reusable code in order to save time and increase quality. We don’t have time to re-write every basic function every time, and if you do that, you also won’t get the benefit of making a single function/feature more robust over time by sending it through all the rigorous quality measures you would with reusable code that’s been evaluated in lots of ways.

Given that reusable code is a must for software development in general, it follows that it would make sense to use it for security-sensitive code as well, right? While that logic seems sound, I’ve seen too much code to believe that it’s happening regularly in many organizations. Usually, at best, what you end up seeing is almost every project has a class named something like “SecurityUtils”, and it will have a couple “filterXYZ” or “escape/encodeXYZ” methods in it. That is the extent of software reuse for security for many, many applications.

Let’s be clear: Fixing XSS is hard. Performing proper access control across your site is hard. Performing appropriate input validation across your application is hard. These things require people, process and technology to be used to solve them properly, and even then, it’s still going to be a difficult process. Security is a difficult thing to get right (just like other aspects of software), so we may as well take any help we can get.

What should I do about it?
In order to help with some of these hard problems, we can leverage reusable frameworks for security. The idea is simple: those controls you put in place to protect your application, say from CSRF (like CSRFGuard), are likely useful for other applications in your organization. Of course, other teams might have a better solution for a different problem, such as a good context-aware encoding solution for XSS (like JXT). If you pool your resources across developers on your team, organization, or even the world (open source software), you get a lot of reuse of code. Additionally, the code is more likely to be good code necessarily since, with many people seeing it and using it, bad code should get weeded out more quickly. If you’re applying your other good processes, such as code review and testing, the quality continues to go up.

I do want to make a special note at this point. I’ve heard lots of folks (including my younger self) argue that we need more secure coding examples, ie. snippets. While I think there is certainly value in this approach (especially for those writing the reusable frameworks), I’ve come to believe that this in and of itself is not particularly useful. A developer will use your framework if it’s useful, provides value to him/her, and doesn’t get in the way (buggy or too difficult to use). A developer will, in general, NOT come and grab your snippet of code for doing XYZ. Snippets can work in very constrained environments and use cases, but I think the general approach should be towards the framework model.

At this point, I’d be remiss if I didn’t mention the excellent ESAPI library. It has about 10 useful controls that cover various parts of security across your applications. It’s in the process of being re-architected to make it more modern, but the idea that there is a security library that offers a holistic approach to security is awesome. If you’re not using ESAPI, give it a try or at least a look. If you’re responsible for building a security related framework, start with ESAPI or at least look at it for inspiration. It will certainly provide value.

In conclusion, software reuse is an old tried and true concept. There are lots of benefits to be realized from it. It’s common for standard application development frameworks, but there aren’t a lot of security libraries out there. Reuse applied to security makes sense just like it does for other features. Building a framework with a comprehensive approach to security is a big win for both development and security, so give it a try.

Week 46 - Store User Passwords Securely

What is it and why should I care?

Note 1: I've actually wanted to finish this post for quite a while, but every time I tried, I would do some more research and find more rabbit holes to enter. At this point, I'm going to cut my losses, and post what I have now. Unfortunately, any solution discussed on topic inherently has weaknesses – it's shades of gray even with good solutions.

Note 2: This post is about user passwords, not system account passwords. That is a different issue with different requirements, and therefore different solutions.

Password breaches have been unusually common over the last year or so, hitting many large and popular companies. What's been particularly disheartening is the weak protections applied to passwords in most of these cases. Several have been protected with a simple md5 or sha-1 hash. We've been able to hash our password locally and compare it to the publicly available list and see our hash right there in the list. That's not a pleasant feeling for anyone, much less those of us in security. But … what would we tell them to do better?

I've been looking at this issue for some time, and talking with various folks about what solutions they generally recommend. I've heard a handful of different ideas, some better than others, but all with weaknesses. At the end of the day, the best solutions recognize they are not perfect, and compensate for that. That is the course of action I would recommend.

What should I do about it?

There are a lot of different ideas about what to do in order to store passwords securely (just check the references :>). That being said, most fall under a basic collection of ideas(hash, salt + hash, salt + hash + key stretching, adaptive hash). While there are various opinions of what is best at this point, there is a reasonable amount of agreement on what's no longer acceptable. With that, let's look at a few bad ideas:

Plaintext Password Only (BAD)

This is a poor choice for obvious reasons. If your password data is breached, the attacker has no work to do in order to gain access to the records.

Encrypted Password Only (BAD)

This is a poor choice for a couple reasons. If you have a breach, an attacker has only to find the decryption key to gain access to the records. This may be a difficult task for external threat actors, but does nothing to mitigate internal attackers. Additionally, this is a poor privacy practice because the password is reversible, and the system can always see the password for the user, opening up another avenue for internal attackers to exploit.

Hashed Password Only (BAD)

This is a poor choice because it's not enough protection. Rainbow tables have made cracking hashed passwords trivial, and effectively equivalent to plaintext from the attacker's perspective.

Salted and Hashed Password Only (BAD)

This is a poor choice also because it's not enough protection. Though salting is a good practice, today's (late 2012) hardware makes simple hash salting a sufficiently weak protection that it's no longer considered viable. Brute forcing a salted hashed password is a practical option for even a poorly funded attacker today.

So, we've seen some bad options … what are the possible solutions? Here are the available proposals that I'm currently aware of that _may_ be considered reasonable depending upon your environment.

Salted and Hashed and Iterated Password
In this option, you essentially perform a salted hash, then hash that, then hash that, then hash that … repeatedly for some large number of iterations (50,000X, 100,000X, 500,000X). The goal here is to slow down your password verification process. By doing this, you slow down your code when the user is logging in, but the theory is that login is a fairly rare request for applications, so you'll only run it, say, once per user per day, but the attacker has to run it for every try. You make brute force ineffective again. While this theory is nice, verify whether the assumption is accurate for your environment. In this helpful spreadsheet, jOHN Steven points out that the cost of performing login on a sufficiently large site can certainly cost money in additional hardware, while it may not slow down an attacker as much as you think. This may or may not be a concern for your environment, but it's certainly a consideration.

Adaptive Hash Functions
This option is very similar to the salted/hashed/iterated password option externally, but with varied internal operations, most calling the "iteration" bit a work factor. The currently popular implementations of these concepts are PBKDF2, bcrypt and scrypt. These options will have the same consideration as above with regards to hardware cost and attacker prevention.

Encrypted Adaptive Hash Functions
This is one of the options proposed in the threat model for secure password storage that jOHN put together (a fantastic piece of work). It is an idea which has some interesting strengths, but has not been heavily discussed yet in the industry (at least not openly). It does solve some of the problems that other solutions have, but not all. Additionally, there is a measure of extra effort and complexity added as part of the solution.

So, which solution should you choose? Obviously pick a reasonable one, but the answer is – it depends. In my mind, however, as much or more important than the specific encryption/hash solution you choose (assuming you choose a reasonable one) are the additional related tasks you should undertake.

Threat Model
You honestly need to consider your threat model and who you are protecting against and let the results of that inform the steps you take. If you're protecting against a low-grade attacker, you have different concerns than if you are looking at a "hacking" collective, a disgruntled internal employee, or a nation-state. Each of these scenarios will have different requirements and will inform your protection scheme.

Know That You Will Lose
This is critical. Go into your planning with the assumed certainty of a breach (though I hope that never happens to you, I promise). What will you do when you see your name on CNN under the heading "120M User Accounts Stolen"? What will your story be? What will your process be?

You need to have a plan to deal with a compromise of your data. You should have a way to 1)protect your users whose accounts have been breached and 2)roll out the necessary updates to your system. All of this is a lot simpler if you've planned (and practiced) it beforehand. See the "workflow under attack" section of the threat model for more info.

Consider Multi-Factor Authentication
While multi-factor authentication doesn't preclude you from appropriately handling all of the steps discussed above, it can significantly lower the likelihood of compromise of your customers. If it is possible in your situation, it's an option to heavily consider.

In conclusion, while password storage is clearly no easy feat, and no solution is perfect, hopefully you see there are much better ways to handle it than the current common scenario. There are technical solutions for the specific storage mechanism, but that is just one part of the puzzle. You also need to take into account how you will deal with issues like your specific threat environment, planning for what to do when you do lose your password data, and additional protections like multi-factor authentication. There's a lot to account for, but if you take each step into account, it's possible to significantly improve the password storage in your applications.

References
————

Secure Password Storage Threat Model
jOHN Steven PSM code / docs
http://www.cigital.com/justice-league-blog/2012/06/11/securing-password-digests-or-how-to-protect-lonely-unemployed-radio-listeners/
https://blog.whitehatsec.com/hash-length-extension-attacks/
http://www.cigital.com/justice-league-blog/2009/08/14/proper-use-of-javas-securerandom/
http://throwingfire.com/storing-passwords-securely/
http://www.analyticalengine.net/2012/06/should-we-really-use-bcryptscrypt/
http://www.analyticalengine.net/2012/06/one-of-many-problems-with-appsec/
http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html
http://blog.mozilla.org/webdev/2012/06/08/lets-talk-about-password-storage/
https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines#Password_Storage
https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
https://www.owasp.org/index.php/Hashing_Java
http://en.wikipedia.org/wiki/Key_derivation_function
http://en.wikipedia.org/wiki/Key_stretching
http://krebsonsecurity.com/2012/06/how-companies-can-beef-up-password-security/

Week 47 - Store Encryption Keys Securely

What is it and why should I care?
Encryption (specifically talking symmetric encryption here) is a critical component of many applications, and the storage of the encryption key can be tricky to get right. Encryption falls under that area of secure programming that you don't come into contact with casually, hence you might not be practiced in secure key storage. You might run across SQLi since it's common to use a database, or XSS because you're programming for the web, but you usually run into encryption explicitly. There are a couple of extremely common usages of encryption in normal programming: encrypting application data (credit card, SSN, etc.) or encrypting system account credentials (user/pw pair for connecting to a DB or web service call). Both of these cases require the keys to be accessible to the application at runtime in order to function properly, so we have to figure out how to solve that problem correctly.

In addition to being less common or prolific than other security issues (SQLi, XSS), there is generally less information available about how to accomplish the task properly. To be fair, that's not necessarily because of its less common nature, but more because there is no one right answer for how you should protect your encryption key – there are various options and your specific circumstances will at least partially dictate the solution. It's more nuanced than "use prepared statements and no dynamic sql!". While there's no single right answer, there are some common good and bad approaches which I'll attempt to address.

What should I do about it?
Before looking at solutions, I want to briefly address 2 base assumptions I've made which are worth considering before digging in deeper:

1. *Store as little sensitive data as needed*
You should only store the sensitive data that you must. If you can throw it away, then by all means throw it away. There are lots of factors that go into this decision, but from a security perspective, you can't lose data you don't have.

2. *Use well-vetted implementations of strong encryption algorithms in the proper configuration*
There are 3 subpoints here:
- Strong encryption algorithms: Check NIST for the most up-to-date list of acceptable encryption algorithms. A simple example is that you should no longer be using DES for encryption, but AES is currently ok (as of late 2012)
- Proper configuration: Again, check NIST. You should be using the appropriate mode and padding for your encryption algorithm in order for it to be more resistant to cryptanalysis. A simple example here is that you should not be using ECB as a mode – it is not safe.
- Well-vetted implementations: You should be using algorithms and configurations that are approved, but the implementation should be well-known and reasonably vetted. The level of rigor for the vetting will differ depending upon your environment, but in general, java programmers are usually safe with either the built-in implementations from Oracle, or using BouncyCastle, a popular crypto library.

Now that we've considered our assumptions, let's look at a few things you should not do when considering how to store your encryption key:

- *Don't store your key with the data it is protecting.*
You shouldn't store your key together with your data, because if the attacker gets one, they likely have the other. That means the attacker has nothing but time to get your data. Two common ways this shows itself is either with the key stored in the database with the data, or the key stored in a flat

file with the username and password it's protecting. This is a bad practice.

- *Don't store your key on the same host as the data it is protecting.*
This is very similar to the previous point, but also takes into account the key in one file and the username/password in another file. If they're both on the same server, it's more likely the attacker will be able to get to both. Also, if the DB is local to the appserver as is the key file, then the likelihood increases that both could be grabbed at the sametime for an offline attack.

- *Don't rely on security by obscurity.*
Encryption key storage is an area where I see obscurity used more often than in other places in applications. People try to do clever things like break the key up across several files or obfuscate the key in some way, and then reverse it out in the application. I'm not saying these tactics couldn't increase the security marginally – they might depending on the attack vector. However, they also usually complicate processes like key rotation and data re-keying. My vote is to stick to simpler solid practices. If you do go this route, make sure you have a complete solution that takes into account your full key lifecycle.

We've looked at a few bad techniques, now let's consider some good practices.

- *Separate key and protected data*
I've already covered this above with the bad practices, but making sure the encryption key and the data it is protecting are separate is certainly a good practice. Make sure it's separate across tiers as well.

- *Lock down your permissions*
Make sure that your account credentials are locked down appropriately, including DB, filesystem, web service calls, etc. Ensure that you are using strong authentication and access control as well as least privilege. For instance, there's usually no reason for the encryption key to be writable by an application – read-only makes sense in this case.

- *Consider various storage options.*
There are a multitude of ways you could store your encryption key depending on your needs. A few might be a keystore (java keystore), an HSM (hardware security module), flat file, DB, etc. You may also allow for the key to be manually entered by operations personnel at application startup time. This puts key storage out of band generally, but does mean that there are additional runtime requirements you must live with, but that may be the right option for you. Again, I want to point out that not all of these will work for everyone, but it's good to know they are available so you can see which solution will work in your environment.

- *Plan for key rotation and re-keying data*
Whether it's driven by internal or external requirements, a data breach, or just good practice, you should be doing key rotation and data re-keying. A key should have a general lifecycle that takes it from generation to destruction. By following this process, you shorten the window for which the key is useful, and you have a recourse if you are breached. The hope during a breach is that either the data or the key is captured but not both. When planning for this step, be sure to consider issues like disaster recovery and backed up data that may need to be retrieved. Those processes should play into the key lifecycle as well.

- *Know your policies, standards and regulations*
Depending on your industry or even just your organization, you will have both internal and external requirements that you must account for. For instance, if you process and store credit cards, you have

to live with PCI/DSS. There are certain requirements that these standards levy upon you, and those may impact your key management process. Be aware of what they are and make sure you are abiding by those when you build out your plan.

In conclusion, encryption key storage is not a simple, clear-cut security issue. It is nuanced, and the appropriate solution depends on your policy and regulatory environment. However, there are some well known good and bad practices you should consider before coming up with a solution. With attention to these details, you can come up with a solution that fits your needs and provides solid security as well.

References

———–

https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet

Conclusion and Links

This will serve as the conclusion to a year-long series on security topics for Java. Let's first look at the original motivations from the series introduction.

There are several motivations for this series:
1. Get some old topics written down
2. Research some new technologies
3. Write
4. Learn
5. Answer questions from Java friends

I can safely say that I've achieved all of these. I covered a pretty wide variety of topics along the way and noticed a few interesting trends:

- The posts that got the most reads were about technical controls, specifically those that were related to configuration settings or response headers. I'm not sure what this means, but it's possible these were read more because they don't specifically apply to Java and can be handled at the web server level or with a WAF, etc.

- The posts that got the most interaction (comments/emails/etc) were process topics that are repeated constantly in the security echo chamber (audit, access control, thread modeling, security training). The interesting thing to me was that these topics were poorly understood by many commenters, and those "experts" that do understand them often have no hard data backing up their assumptions about the topics.

- As I worked through the year, the vast majority of topics I wrote about had less to do with Java specifically – they were security topics that applied across languages – this was actually to be expected. The unexpected thing to me was that many of the topics were referenced in the context of security, but are really just a specific use case of basic software engineering best practice. I knew good development usually spurs much better security, but writing on this many topics really drove the point home for me.

It's my sincere hope that some, if not all, of these topics are helpful to you. I (mostly) enjoyed writing them and had some great discussions with folks along the way.

Below I've added the full list of links for all the posts from the series. Hope you enjoyed it!
----
http://www.jtmelton.com/2012/01/02/year-of-security-for-java-introduction/
http://www.jtmelton.com/2012/01/02/year-of-security-for-java-week-1-session-fixation-prevention/
http://www.jtmelton.com/2012/01/10/year-of-security-for-java-week-2-error-handling-in-web-xml/
http://www.jtmelton.com/2012/01/17/year-of-security-for-java-week-3-session-cookie-secure-flag/
http://www.jtmelton.com/2012/01/25/year-of-security-for-java-week-4-session-cookie-httponly-flag/
http://www.jtmelton.com/2012/02/03/year-of-security-for-java-week-5-clickjacking-prevention/
http://www.jtmelton.com/2012/02/07/year-of-security-for-java-week-6-csrf-prevention-in-java/
http://www.jtmelton.com/2012/02/14/year-of-security-for-java-week-7-content-security-policy/
http://www.jtmelton.com/2012/02/21/year-of-security-for-java-week-8-http-strict-transport-security/
http://www.jtmelton.com/2012/02/28/year-of-security-for-java-week-9-x-frame-options/
http://www.jtmelton.com/2012/03/06/year-of-security-for-java-week-10-x-content-type-options/
http://www.jtmelton.com/2012/03/13/year-of-security-for-java-week-11-x-xss-protection/

http://www.jtmelton.com/2012/03/20/year-of-security-for-java-week-12-log-forging-prevention/
http://www.jtmelton.com/2012/03/29/year-of-security-for-java-week-13-know-your-frameworks/
http://www.jtmelton.com/2012/04/03/year-of-security-for-java-week-14-store-jsps-in-web-inf/
http://www.jtmelton.com/2012/04/10/year-of-security-for-java-week-15-audit-security-related-events/
http://www.jtmelton.com/2012/04/17/year-of-security-for-java-week-16-set-a-soft-session-timeout/
http://www.jtmelton.com/2012/04/27/year-of-security-for-java-week-17-set-a-hard-session-timeout/
http://www.jtmelton.com/2012/05/01/year-of-security-for-java-week-18-perform-application-layer-intrusion-detection/
http://www.jtmelton.com/2012/05/09/year-of-security-for-java-week-19-reduce-the-attack-surface/
http://www.jtmelton.com/2012/05/16/year-of-security-for-java-week-20-trust-nothing/
http://www.jtmelton.com/2012/05/23/year-of-security-for-java-week-21-anti-caching-headers/
http://www.jtmelton.com/2012/05/31/year-of-security-for-java-week-22-http-parameter-pollution/
http://www.jtmelton.com/2012/06/06/year-of-security-for-java-week-23-http-header-injection/
http://www.jtmelton.com/2012/06/13/year-of-security-for-java-week-24-use-static-analysis/
http://www.jtmelton.com/2012/06/20/year-of-security-for-java-week-25-use-dynamic-analysis/
http://www.jtmelton.com/2012/06/27/year-of-security-for-java-week-26-do-code-reviews/
http://www.jtmelton.com/2012/07/05/year-of-security-for-java-week-27-penetration-testing/
http://www.jtmelton.com/2012/07/11/year-of-security-for-java-week-28-unit-test/
http://www.jtmelton.com/2012/07/17/year-of-security-for-java-week-29-manage-resources/
http://www.jtmelton.com/2012/07/25/year-of-security-for-java-week-30-authentication/
http://www.jtmelton.com/2012/08/02/year-of-security-for-java-week-31-access-control-1/
http://www.jtmelton.com/2012/08/07/year-of-security-for-java-week-32-access-control-2/
http://www.jtmelton.com/2012/08/14/year-of-security-for-java-week-33-access-control-3/
http://www.jtmelton.com/2012/08/21/year-of-security-for-java-week-34-separate-admin-functionality/
http://www.jtmelton.com/2012/08/30/year-of-security-for-java-week-35-solve-security-problems-one-at-a-time/
http://www.jtmelton.com/2012/09/07/year-of-security-for-java-week-36-solve-sql-injection/
http://www.jtmelton.com/2012/09/12/year-of-security-for-java-week-37-solve-cross-site-scripting/
http://www.jtmelton.com/2012/09/21/year-of-security-for-java-week-38-create-a-reusable-security-framework/
http://www.jtmelton.com/2012/09/27/year-of-security-for-java-week-39-dont-reinvent-the-wheel-unless-its-square/
http://www.jtmelton.com/2012/10/05/year-of-security-for-java-week-40-get-a-security-person-or-some-people-if-you-can/
http://www.jtmelton.com/2012/10/12/year-of-security-for-java-week-41-spend-wisely-on-developer-security-training/
http://www.jtmelton.com/2012/10/19/year-of-security-for-java-week-42-break-something/
http://www.jtmelton.com/2012/10/27/year-of-security-for-java-week-43-build-something-and-give-it-away/
http://www.jtmelton.com/2012/11/03/year-of-security-for-java-week-44-follow-a-secure-sdlc/
http://www.jtmelton.com/2012/11/09/year-of-security-for-java-week-45-do-threat-modeling/
http://www.jtmelton.com/2012/11/17/year-of-security-for-java-week-46-store-user-passwords-securely/
http://www.jtmelton.com/2012/11/24/year-of-security-for-java-week-47-store-encryption-keys-securely/
http://www.jtmelton.com/2012/12/01/year-of-security-for-java-week-48-you-will-get-hacked/
http://www.jtmelton.com/2012/12/08/year-of-security-for-java-week-49-collect-and-share-your-data/
http://www.jtmelton.com/2012/12/13/year-of-security-for-java-week-50-think/

http://www.jtmelton.com/2012/12/22/year-of-security-for-java-week-51-document-everything/
http://www.jtmelton.com/2012/12/29/year-of-security-for-java-week-52-never-stop-improving/