## broken approaches to integrating web app security

### Bolt-on approach

'make it work, then make it right' - security is left until the very end, which typically results in mistakes caused earlier in the development cycle, trying to be fixed in testing. The main problem is that a lot of important design decisions have cascading effects on the security, and vice versa.

### Do-it-all-up-front approach

The opposite of bolt-on. All potential security concerns are dealt with up front. This typically has two failure scenarios:

- frustration, and giving up
- it feels as though you've covered all bases, so you never touch security again until release
  It's wise to consider security up front, but not everything can be known up front and done at once.

### Big-bang approach

Similar to do-it-all-up-front, where everything is dealt with at once, but when this takes place can be chosen. The main issue once again is that some security benefits may be achieved, but a number of previously made design decisions may once again be found to not be suitable when it's too late.

### Buckshot approach

Try a bunch of security techniques on the app, hoping one does the job - the main idea is that you don't know what specifically requires security, thus there isn't a clear benefit. Random security additions may reduce performance, usability, and maintainability for no actual improved security.

### All-or-nothing approach

Generally the idea that some previous failure has resulted in a rapid change in approach, trying to implement all security on all fronts. By taking on everything at once, a lasting approach may not be made, and one may shortly return to doing nothing.

## successfully integrating web app security: life-cycle

Producing effective results in terms of web app security can be achieved through a life-cycle approach, otherwise known as 'baking' security into an app's life cycle. Security is addressed practically throughout development, rather than upfront or randomly.

**Table 1. Standard development activities and security add-ins at each phase.**

| MAIN ACTIVITIES | CORE SUBACTIVITIES | SECURITY ADD-INS |
|---|---|---|
| Planning | | |
| Requirement and analysis | Functional requirements<br>Nonfunctional requirements<br>Technology requirements | Security objectives |
| Architecture and design | Design guidelines<br>Architecture and design review | Security design guidelines<br>Threat modeling<br>Security architecture and design review |
| Development | Unit tests<br>Code review<br>Daily builds | Security code review |
| Testing | Integration testing<br>System testing | Security testing |
| Deployment | Deployment review | Security deployment review |
| Maintenance | | |

## high ROI activities

High return-on-investment (ROI) activities can be considered as a baseline set of techniques, independent of any security focus, to shape software throughout its life cycle. They're considered high ROI since if applied properly, they directly affect software's shape throughout the process, potentially transforming customer results for the better.
High ROI activities can include:

- Identifying design guidelines - putting together recommendations for development teams, including input from software vendor recommendations, corporate policies, industry practices
- Architecture/design review - evaluating the design against functional/non-functional requirements, as well as technical requirements and constraints. It is most valuable when performed early, to shape a design
- Code review - manual code review can be preferable to static analysis tools, since one can imagine the context of where code will be ran better, and what likely conditions may occur
- Testing - executable feedback for whether code works or not. Testing mainly focuses on functional requirements, but can be tuned for quality attributes if one can define what 'good' looks like
- Deployment review - evaluating an application once deployed to infrastructure. Preferably, early working builds of software can be deployed to provide as much preparation for production as possible

## security add-ins

Rather than necessarily including all security activities within the high ROI activities, they can exist on their own, overlaying the regular development life cycle activities.

### Objectives

Identify up front how important security is for the given project, potentially starting by stating what shouldn't happen with respect to security. Further security objectives can be determined with the following questions:

- what client data should be protected - does the app have user accounts/passwords, details, or financial history/transaction records?
- do you have any compliance requirements (e.g. privacy laws, regulations, standards)
- do you have specific quality of service requirements
- do you need to protect intangible assets, e.g. company rep, trade secrets, IP?
  Thinking in terms of some information's requirements in terms of confidentiality, integrity and availability (CIA), and how these link to functional requirements is one way to identify objectives. Creating a matrix of roles in the system, the objects they act on, and their permissions on those objects is another way.
  Don't try to figure out all security objectives up front. Identify ones which can have a cascading impact on design.

### Threat modelling

A threat model is an organised representation of relevant threats/attacks/vulnerabilities a system may face. In design, this allows one to play out various 'what if' scenarios, and evaluate what countermeasures are necessary/relevant. These threat models may be more successful when partitioned between threats to infrastructure, and threats to the application. The following guidelines can be useful for building threat models:

- incrementally render the threat model, and use a question-driven approach to reveal high-risk engineering decisions
- use complimentary techniques, for instance use-case analysis, dataflow analysis, question driven approaches, role matrices and brainstorming
- don't do everything up front, and consider threat modelling as a key activity, rather than a replacement for all other stages of security activities
- the focus lies more on the issues being faced with security, rather than minute specifics and

countermeasures which may be suitable

### Design guidelines

Gathering relevant security design advice from vendors, industry and other individuals then organising it such that it's useful for a development team, will give better chances of success.
A good guideline will aim to consist of three main points:

1. What to do
2. Why
3. How
   Each of these points should use simple wording schemes, and organise information to be insightful - focus on guidelines with immediate impact on requirements, which can be integrated into activities/the general workflow.

### Architecture and design review

Architecture/design reviews focused on security should be just that - by having a separate security focused design review, it avoids overcomplicating a potentially already existing design review, and doesn't limit the effectiveness of the security design review itself.
The best way to focus for maximum security impact is by asking questions which pull out high-risk decisions - therefore, continue to use question based approaches, while partitioning security design review into smaller chunks where appropriate. The overall idea is that security focused design reviews should be a separate, yet continual occurrence while focusing on making the most impact.

### Code review

A security code review focuses on whether code meets security objectives, rather than if it just works. There are 4 common steps to performing security code reviews:

1. Identify security code review objectives
2. Perform a preliminary scan - e.g. static analysis to find an initial set of issues, and figure out where to look further
3. Review code for security issues - thorough review of code to find vulnerabilities
4. Review security issues unique to your architecture - focusing on any features designed specifically to handle known security issues
   Contextual analysis (i.e. having security objectives in mind while reading code) becomes much more key than static analysis - similar to the security design review, these should ideally be separate to a general code review.

### Testing

When talking about security, this typically focuses on penetration testing. White-box testing and black-box testing are both appropriate, but evidence suggests white-box testing (where you know the internal workings and try to test those specifically) may be far more effective for finding security issues. Testing should revolve around previously agreed on security objectives and threat models - additionally, third-party security assessment is still industry practice.

### Deployment review

An app can behave differently during development and production for various reasons. Examples include:

- infrastructure constraints
- corporate policy developer was oblivious to is violated (?)
- different privilege-level account is used in production
  The goal of testing deployment is to eliminate as many of these potential surprises as possible, ideally through build cycles which evaluate an app's runtime behaviour. Considering as many different

configurations for an app (potentially categorising them) with deployment reviews can help to identify what may behave differently in production.

## Categories + Context precision

The primary advantage of security specific activities outlined above over general development activities, is the increased focus on security. Categories and context precision help to improve focus and produce more effective results

### Web Application Security Frame

The Web Application Security Frame is a set of categories used to organise recurring security issues. It contains the following categories:

- input/data validation
- authentication
- authorization
- configuration management
- cryptography
- sensitive data
- exception management
- session management
- auditing/logging
  They all generally address root-cause issues rather than one off issues

### Context precision

Context precision is the idea of evaluating a given problem's context based on the following:

- application type (web app, web service, component/library, desktop app, mobile app)
- deployment scenario (intranet, extranet, the Internet)
- project type (in-house IT, third party)
- life cycle (Waterfall, Agile, Spiral etc)
  Keeping context in mind allows for more specific design choices and specifics involving issues which may fit into categories stated by the Web Application Security Frame.
  The aim isn't to make a new technique for every context, but to ensure what was designed fits the context at hand, and make any optimisations if necessary.

## Overall

The main takeaway is that to integrate security throughout a development life cycle, security focused activities which follow similar styles to main development activities is effective. These activities let you look through a security lens, to make calls on how important security is and what should be prioritised. All security should never be implemented at a single point, since important design choices throughout will have cascading effects on security.