Activity-Implementation View

At the core of CLASP are 24 security-related activities that can be integrated into a software development process. The activities phase translates into executable software the subset of the 24 security-related activities which were assessed and accepted in the implementation phase.

CLASP also has an impact on several key traditional software engineering activities, such as requirements specification. CLASP does not materially change the steps within such activities. Instead, it recommends extensions to common artifacts and provides implementation guidance for security-specific content.

Table: Roles and Related Activities

The following table relates the security-related project roles to the 24 CLASP activities to be implemented.

CLASP Activity	Related Project Role
Institute security awareness program	• Project Manager
Monitor security metrics	• Project Manager
Specify operational environment	• Owner: Requirements Specifier • Key Contributor: Architect
Identify global security policy	• Requirements Specifier
Identify resources and trust boundaries	• Owner: Architect • Key Contributor: Requirements Specifier
Identify user roles and resource capabilities	• Owner: Architect • Key Contributor: Requirements Specifier
Document security-relevant requirements	• Owner: Requirements Specifier • Key Contributor: Architect
Detail misuse cases	• Owner: Requirements Specifier • Key Contributor: Stakeholder
Identify attack surface	• Designer
Apply security principles to design	• Designer
Research and assess security posture of technology solutions	• Owner: Designer • Key Contributor: Component Vendor
Annotate class designs with security properties	• Designer
Specify database security configuration	• Database Designer
Perform security analysis of system requirements and design (threat modeling)	• Security Auditor
Integrate security analysis into source management process	• Integrator
Implement interface contracts	• Implementer
Implement and elaborate resource policies and security technologies	• Implementer
Address reported security issues	• Owner: Designer • Fault Reporter
Perform source-level security review	• Owner: Security Auditor • Key Contributor: Implementer;

CLASP Activity	Related Project Role
	Designer
Identify, implement and perform security tests	• Test Analyst
Verify security attributes of resources	• Tester
Perform code signing	• Integrator
Build operational security guide	• Owner: Integrator • Key Contributor: Designer; Architect; implementer
Manage security issue disclosure process	• Owner: Project Manager • Key Contributor: Designer

Project Manager: Responsibilities

The initial activities belong to the project manager. While his duties do not represent a significant time commitment, they do reflect the CLASP philosophy that effective security practices require organizational buy-in. For example, introducing a security awareness program should be about more than simply training developers that will be dealing with security functionality directly.

Everyone that has exposure into the development lifecycle should receive basic awareness training that will allow them to understand the macro-level issues that can impact a business. Particularly, people need to understand the immediate costs associated with security-related activities as well as the long-term benefits of an improved security posture. Otherwise, when a project begins to slip, security activities will risk being the first to be deferred if they do not have a concrete impact on the core feature set.

Requirements Specifier: Responsibilities

The primary security duty of a requirements specifier is to identify at a high level the core security model for the application. For example, the requirements specifier determines which resources might be at risk, the roles and responsibilities of users that may access those resources, and the potential consequences if these resources are compromised.

Not only do these activities provide a context for making choices about how to deal with particular security issues throughout the development lifecycle; these activities also define a framework for accountability that a project manager can apply if security problems are ultimately found in system design.

Implementer: Responsibilities

Most of the security activities traditionally assigned to implementers are actually best handled by the software architects and designers. Most software security issues can be addressed at architecture and design time, which is far more cost effective. This also allows an organization to concentrate security expertise among a very few of the most trusted members of the development organization.

Security Auditor: Responsibilities

Several key tasks are owned by a security auditor, which is a new role that CLASP introduces into the software development lifecycle. The invention of this role emphasizes the fact that development teams can easily get too close to its own systems to analyze them effectively.

Independent third-party security assessments are currently commonly accepted as a best practice. These assessments are also one of the simplest and most cost-effective measures that an organization can take to improve the security posture of its development efforts — whether the independent third party is a firm dedicated to security assessments or simply consists of members from another product team within the same organization.

Institute security awareness program

Purpose:	• Ensure project members consider security to be an important project goal through training and accountability. • Ensure project members have enough exposure to security to deal with it effectively.
Role:	Project Manager
Frequency:	Ongoing

Provide security training to all team members

Before team members can reasonably be held accountable for security issues, you must ensure they have had adequate exposure to those issues. Additionally, even those members of the team that do not directly deal with security issues should be aware of the project's security practices.

This is best done with a training program. Everyone on the team should receive training introducing them to basic security concepts and secure development process that is used within the organization.

Additionally, people within the organization should receive training targeted to their role. For example, Developers should receive detailed training on common basic causes and mitigation techniques, particularly as they relate to the development and deployment environment. Additionally, both developers and testers should receive training for automation tools that they should use in the course of doing their jobs.

Promote awareness of the local security setting

Everyone on a development project should be familiar with the security requirements of the system, including the basic threat model. When such documents are produced, they should be distributed and presented to team members, and you should solicit and encourage feedback from all parties on the team.

When other security-relevant documentation is produced — e.g., as code analysis results — that documentation should be made available to the team, even if not every member is required to review it.

Additionally, you should ensure that security implications are considered whenever a new requirement emerges. It is a best practice to explicitly address at the end of any technical meeting whether there are security ramifications.

Finally, we recommend promoting a culture where your team is externally security aware. Watch security news sources and/or article aggregators for security-relevant news that is related to your project at the end of any technical meeting — or appoint a designee to do this. Forward to your team anything that seems relevant to your project. This includes not only flaws in products you use on your project, but also interesting news, flaws, or other results that you feel will maintain awareness and/or further educate your team.

Institute accountability for security issues

Traditional accountability within development organizations is based primarily on schedule and quality. Security should be treated in much the same way as any other quality consideration.

First, the team should be given security goals. It is reasonable to expect that a team member will not be responsible for introducing "standard" risks into the system, without documenting and escalating those

risks before introducing them. This recognizes that security is not a “black-and-white” issue — i.e., there will always be some security risk in the system. It also helps ensure that development team members will consider and document any risks that are considered acceptable.

When the project manager becomes aware of a new security risk that was not caught before introducing it into the system, it is important that he not decide arbitrarily whether or not the risk should have been identified in advance. Instead, we recommend having in place a list of risks that can be used as a baseline. For example, developers should be given a list of coding security standards — such as the list in CLASP Resource E — that they are periodically assessed against. All members of the team should also be held accountable on the basis of a database of basic causes of vulnerabilities.

Note that sometimes security accountability may affect schedule accountability — i.e., finding a security issue that requires remediation can have a negative impact on schedule. We recommend that, whenever the decision is made to remediate a security risk in a way that will impact schedule, the accountability for the schedule slip should be tied to the accountability for the security problem.

Additionally, it is the responsibility of the project manager to ensure adoption of security activities into the development lifecycle and ensure that they are given the desired level of attention. Team members must, again, be accountable for performing these activities to a satisfactory level.

Appoint a project security officer

An excellent way to increase security awareness throughout the development lifecycle is to designate a team member as the project security officer, particularly someone who is enthusiastic about security.

The role of this person (or persons) can vary depending on the development organization but should encompass at least the first two of the following duties:

- Serve as a repository of security expertise for other project members.
- Take into account security concerns through the SDLC — such as during design meetings.
- Review work of other team members, as if an external security auditor, performing security assessments when appropriate.

Generally, independent auditors are far more effective than internal auditors, regardless of the level of security expertise, even if independent auditors are still inside the same company. Ultimately, more review is also preferable as a defense-in-depth measure.

Institute rewards for handling of security issues

Accountability is a necessity for raising security awareness, but another highly effective way is to institute reward programs for doing a job well done with regard to security. For example, it is recommended to reward someone for following security guidelines consistently over a period of time — particularly if the result is that no incidents are associated with that person.

Additionally, if team members identify important security risks that were not found in the course of standard auditing practices, these insights should be rewarded.

Monitor security metrics

Purpose:	• Gauge the likely security posture of the ongoing development effort. • Enforce accountability for inadequate security.
Role:	Project Manager
Frequency:	Ongoing

Identify metrics to collect

There is a wealth of metrics about a program that can offer insight into the likely security posture of an application. However, the goal of metrics collection goes beyond simply determining likely security posture; it also aims at identifying specific areas in a system that should be targets for improvement.

Metrics are also important for enforcing accountability — i.e., they should be used to measure the quality of work done by teams or individual project members. The information can be used to determine, for example, which projects need expert attention, which project members require additional training, or who deserves special recognition for a job well done.

One disadvantage of using metrics for accountability is that, when creating your own metric, it can take time to build confidence in a set of them. Generally, one proposes a metric and then examines its value over a number of projects over a period of time before building confidence that, for example, .4 instead of .5 is just as bad as .6 is just as good.

That does not make metrics useless. If the metric always satisfies the property that adding more risk to the program moves the metric in the proper direction, then it is useful, because a bar can be set for team members to cross, based on instinct, and refined over time, if necessary. One need not worry about the exact meaning of the number, just one's position relative to some baseline.

As a part of identifying metrics for monitoring teams and individuals, one must clearly define the range of artifacts across which the metrics will be collected. For example, if individual developers are responsible for individual modules, then it is suitable to collect metrics on a per-module level. However, if multiple developers can work on the same module, either they need to be accountable as a team, or metrics need to be collected — for example, based on check-ins into a version control system.

The range of metrics one can collect is vast and is easy to tailor to the special needs of your organization. Standard complexity metrics such as McCabe metrics are a useful foundation because security errors become more likely as a system or component gets more complex.

One of the key requirements for choosing a metric is that it be easy to collect. Generally, it is preferable if the metric is fully automatable; otherwise, the odds that your team will collect the metric on a regular basis will decrease dramatically.

There are metrics tailored specifically to security. For example, here are some basic metrics that can be used across a standard development organization:

- ***Worksheet-based metrics.*** Simple questionnaires — such as the system assessment worksheet in CLASP Resource F — can give you a good indication of your organizational health and can be a useful metric for evaluating third-party components that you want to integrate into your organization or product. Questions on that worksheet can be divided into three groups: "critical," "important," and "useful"; then a simple metric can be based on this grouping. For

example, it is useful enough to simply say that, if any critical questions are not answered to satisfaction, the result is a “0”.

The value of worksheet-based metrics depends on the worksheet and the ease of collecting the data on the worksheet. Generally, this approach works well for evaluating the overall security posture of a development effort but is too costly for measuring at any finer level of detail.

- ***Attack surface measurement.*** The attack surface of an application is the number of potential entry points for an attack. The simplest attack surface metric is to count the number of data inputs to the program or system — including sockets, registry lookups, ACLs, and so on. A more sophisticated metric would be to weight each of the entry points based on the level of risk associated with them. For example, one could assign a weight of 1.0 to an externally visible network port where the code supporting the port is written in C, 0.8 for any externally visible port in any other language, and then assign lesser ratings for ports visible inside a firewall, and small weightings for those things accessible only from the local machine. Choosing good weights requires sufficient data and a regression analysis, but it is reasonable to take a best guess.

  Attack surface is a complex topic, but a useful tool. See CLASP Resource A for a detailed discussion on the topic.

  Metrics based on attack surface can be applied to designs, individual executables, or whole systems. They are well suited for evaluating architects and designers (and possibly system integrators) and can be used to determine whether an implementation matches a design.

  Even with a weighted average, there is no threshold at which an attack surface should be considered unacceptable. In all cases, the attack surface should be kept down to the minimum feasible size, which will vary based on other requirements. Therefore, the weighted average may not be useful within all organizations.

- ***Coding guideline adherence measurement.*** Organizations should have secure programming guidelines that implementers are expected to follow. Often, they simply describe APIs to avoid. To turn this into a metric, one can weight guidelines based on the risk associated with it or organizational importance, and then count the occurrences of each call. If more detailed analysis tools are available, it is reasonable to lower the weighting of those constructs that are used in a safe manner — perhaps to 0.

  While high-quality static analysis tools are desirable here, simple lexical scanners such as RATS are more than acceptable and sometimes even preferable.

- ***Reported defect rates.*** If your testing organization incorporates security tests into its workflow, one can measure the number of defects that could potentially have a security impact on a per-developer basis. The defects can be weighted, based on their potential severity.

- ***Input validation thoroughness measurement.*** It is easy to build a metrics collection strategy based on program features to avoid. Yet there are many things that developers should be doing, and it is useful to measure those as well. One basic secure programming principle is that all data from untrusted sources should go through an input validation routine. A simple metric is to look at each entry point and determine whether input validation is always being performed for that input.

  If your team uses a set of abstractions for input validation, a high-level check is straightforward. More accurate checks would follow every data flow through the program.

  Another factor that can complicate collection is that there can be different input validation strategies — as discussed extensively in CLASP Resource B. Implementations can be judged for quality, based on the exact approach of your team.

- ***Security test coverage measurement.*** It can be difficult to evaluate the quality of testing organizations, particularly in matters of security. Specifically, does a lack of defects mean the testers are not doing their jobs, or does it mean that the rest of the team is doing theirs?

Testing organizations will sometimes use the concept of “coverage” as a foundation for metrics. For example, in the general testing world, one may strive to test every statement in the program (i.e., 100% statement coverage), but may settle for a bit less than that. To get more accurate, one may try to test each conditional in the program twice, once when the result is true and once when it is false; this is called branch coverage.

Directly moving traditional coverage metrics to the security realm is not optimal, because it is rarely appropriate to have directed security tests for every line of code. A more appropriate metric would be coverage of the set of resources the program has or accesses which need to be protected. Another reasonable metric is coverage of an entire attack surface. A more detailed metric would combine the two: For every entry point to the program, perform an attainability analysis for each resource and then take all remaining pairs of entry point and resource and check for coverage of those.

Identify how metrics will be used

This task often goes hand-in-hand with choosing metrics, since choice of metric will often be driven by the purpose. Generally, the goal will be to measure progress of either a project, a team working on the project, or a team member working on a team.

Besides simply identifying each metric and how one intends to apply it, one should consider how to use historical metrics data. For example, one can easily track security-related defects per developer over the lifetime of the project, but it is more useful to look at trends to track the progress of developers over time.

For each metric identified, it is recommended to ask: “What does this mean to my organization right now?” and “What are the long-term implications of this result?”. That is, it is recommended to draw two baselines around a metric: an absolute baseline that identifies whether the current result is acceptable or not, and a relative baseline that examines the metric relative to previous collections. Identified baselines should be specific enough that they can be used for accountability purposes.

Additionally, one should identify how often each metric will be collected and examined. One can then evaluate the effectiveness of the metrics collection process by monitoring how well the schedule is being maintained.

Institute data collection and reporting strategy

A data reporting strategy takes the output of data collection and then produces reports in an appropriate format for team consumption. This should be done when selecting metrics and should result in system test requirements that can be used by those people chosen to implement the strategy.

Implementing a data collection strategy generally involves: choosing tools to perform collection; identifying the team member best suited to automate the collection (to whatever degree possible); identifying the team member best suited to perform any collection actions that can not be automated; identifying the way data will be communicated with the manager (for example, through a third-party product, or simply through XML files); and then doing all the work to put the strategy in place.

Data collection strategies are often built around the available tools. The most coarse tools are simple pattern matchers — yet tools like this can still be remarkably effective. When using such tools, there are multiple levels at which one can collect data. For example, one can check individual changes by scanning the incremental change as stored in your code repository (i.e., scan the “diffs” for each check-in), or one can check an entire revision, comparing the results to the output from the last revision.

More sophisticated tools will generally impose requirements on how you collect data. For example, analysis tools that perform sophisticated control and data flow analysis will not be able to work on incremental program changes, instead requiring the entire program.

Where in the lifecycle you collect metrics is also tool-dependent. For example, many per-system metrics can be collected using dynamic testing tools — such as network scanners and application sandboxes, which are applied while the system is running. Code coverage tools also require running the program and therefore must be collected during testing (or, occasionally, deployment).

But static code scanners can produce metrics and can be run either at check-in time or during nightly builds. Tools like RATS that perform only the most lightweight of analysis may produce less accurate results than a true static analysis tool but have the advantage that they can operate over a patch or “diff” — as opposed to requiring the entire program. This makes assigning problems to team members much simpler.

Periodically collect and evaluate metrics

Periodically review the output of metrics collection processes (whether automated or manual). Act on the report, as appropriate to your organization. In order to maintain high security awareness, it can be useful to review metrics results in group meetings.

If it becomes clear — in the course of reviewing data produced by metrics — that those metrics do not adequately capture data needed to evaluate the project, teams or team members, use this information to iterate on the metrics collection process.

Specify operational environment

Purpose:	• Document assumptions and requirements about the operating environment, so that the impact on security can be assessed.
Role:	Requirements Specifier
Frequency:	As necessary; generally, once per iteration.

An operational environment specification allows team members to understand the operational context that they need to consider for designing protection mechanisms or building operational security guides. Much of the data required for an operational environment specification will already be produced in defining business requirements, and specifying the operational environment will often result in identifying new requirements.

Generally, this activity will result in changes to existing requirements and specifications, if necessary. However, it is also reasonable to produce stand-alone documentation. An operational environment worksheet is provided in CLASP Resource F.

Identify requirements and assumptions related to individual hosts

A host-level operational environment specification should identify anything that could potentially be security-relevant to other team members. In most circumstances, the large majority of considerations will be addressed by assuming nothing. For example, it is rare that, beyond the core OS, one will take actions to ensure that particular pieces of software will not be running on a machine, even if that software might pose a threat.

Still, there are properties that are worth specifying, even beyond hardware platforms and OS. For example, it is worth specifying which user the software is expected to run as, since this has security implications.

One can also enforce prerequisites, as long as they are necessary to product functionality. Any such prerequisites should be identified as early as possible. If the project is expected to interact with important system components or libraries that come bundled with the OS, it is recommended to note this as well, not only because those may be additional sources of risk to the resources the application exports, but also because the software should be concerned about the security of resources it is capable of using.

Additionally, one should consider what optional functionality might be in the environment that could have a security impact — positive or negative — that your project could explicitly leverage or protect, as necessary.

Example: Your customer base is government-focused and is likely to have a dynamic policy enforcement environment available. Note that — since providing policies for such an environment might be a way to remediate significant risks for those users — you can also serve other users by recommending a dynamic policy-enforcement environment to them. On the other hand, if your software is dependent on a component that is known to be risky, such as Microsoft's IIS server, it is good to know about the risk upfront.

Identify requirements and assumptions related to network architecture

In some environments, one can assume particular things about network topology, such as the existence and configuration details of a firewall or a single-sign-on mechanism. Often, however, assumptions cannot be made.

As with host-related concerns, it is recommended to define not only those things that will or will not be in the environment but also those things that may have an impact (either positive or negative) if present in the environment. For example, many applications assume implicitly that there is no network-attached storage, or if there is, it has its own security measures in place that make it as secure as the local disk. That is often not the case; and this is a concern that should ultimately be entered into an operational security guide if the risk is not addressed at the application level.

Additionally, focus on those network resources that must be present for the system to correctly function — such as a database, and possibly available bandwidth. Also, if your customers are expected to want integration with centralized authentication servers or other network resources, this should be noted as a requirement.

Identify global security policy

Purpose:	• Provide default baseline product security business requirements. • Provide a way to compare the security posture of different products across an organization.
Role:	Requirements Specifier
Frequency:	As necessary; generally, at least once per iteration.

Build a global project security policy, if necessary

If the organization is lacking a global project security policy, then the CISO, head of engineering and managers of significant projects (or the equivalents) should work together to determine whether a policy is valuable, and if so, produce the policy. It is generally a good idea to maintain this policy as a group, although it is particularly reasonable to entrust it to a single individual when the head of engineering has a strong security background.

Particularly in large organizations with many separate projects, it is useful to have a set of baseline security requirements for software projects. Not only does this ease the burden of requirements specifiers in the long term, it also provides a way to compare the security posture of applications within the organization, and can be a framework for per-project accountability.

If some projects are deployed on the company's network, such requirements are even more valuable since they serve as a concrete documentation of internal procedures that documentation teams should be following. Some organizations even have separate policies for both internally deployed software and externally delivered software.

A global project security policy should detail a minimum baseline for protecting data resources, with respect to the basic security services. It can (and should) break resources up into categories (or specific technologies), providing different guidance for each, where appropriate. Such guidance should include when to apply technologies as well as how to apply technologies when they are used on a project.

When designing such requirements, one should avoid making choices that are arbitrary, and potentially limiting. For example, it is fine to specify a particular minimum key size for a cryptographic algorithm, but a policy shouldn't disallow a project from choosing larger keys, unless there is a strong reason for it.

We provide a sample list of global security requirements in CLASP Resource D and in Activity-Implementation View (activity: "Identify global security policy").

Determine suitability of global requirements to project

For each of the requirements in the global requirement list, one should determine whether it is appropriate to the project. If it is not appropriate to the project, that fact should be documented explicitly. Preferably, this would be done by maintaining an annotated copy of the global requirements document, so that one can easily demonstrate coverage of the global policy. However, it is also reasonable to incorporate irrelevant requirements directly into a requirements document, with an annotation indicating that it is believed to be irrelevant to the project, but must be followed per the global policy, if it becomes relevant.

If the global requirement is relevant to the project, determine how it is relevant:

- The global requirement is already addressed by one or more of the other system requirements. In this case, one should denote explicitly that the global requirement is addressed, and which project requirement(s) address it. This can be done either on a marked up version of the global policy, or in place in the system requirements document, depending on the organization's preferences.
- The global requirement contradicts the project requirements (implicit or explicit). Generally, this should result in a change of the project requirements. If not, it should be escalated beyond the project to the global policy maintainer(s), resulting either in a change of the global requirements or an exception that gets explicitly documented.
- The global requirement does not contradict existing requirements, but has not yet been addressed. The requirements specifier should determine how to incorporate the requirement. Sometimes the global requirement can be copied directly, and sometimes it will need to be elaborated. Often, however, global requirements will provide general, high-level guidance that an individual project may elaborate. For example, a global requirement may be to allow any cryptographic algorithm that was a finalist in the AES competition with 128-bit keys or larger for providing symmetric confidentiality, but a particular system may specify AES with 256 bit keys.

Identify resources and trust boundaries

Purpose:	• Provide a structured foundation for understanding the security requirements of a system.
Role:	Architect
Frequency:	As needed; at least once per iteration.

Identify network-level design

Describe the architecture of the system from the perspective of the network. Particularly, identify any components that could possibly be located on different logical platforms. For example, client software should be identified, as well as middleware and any database. If there is both middleware and a database, which might possibly live on a separate machine, they should be identified as logically separate.

As part of denoting components, denote trust boundaries. For example, the firewall is often a trust boundary — the client machines on the outside are less trustworthy. Individual hosts are often trust boundaries, and many multi-user systems can have multiple trust boundaries internally. Trust boundaries should be mapped to system roles that can be granted that level of trust.

A network-level design should be codified with a diagram in order to facilitate communication. This should be the same kind of diagram one would put on a whiteboard when asked about the architecture. The document should be kept up-to-date with changes and additions to the architecture. Particularly, as you identify protection mechanisms for resources and data links, you should annotate the diagram with these mechanisms.

Identify data resources

Identify data resources that may be used by a program. In conjunction with the next activity, this should ultimately be broken down into individual capabilities related to each resource. When the information is known, break down each resource as granularly as possible — e.g., by identifying individual database tables, instead of simply the database as a whole.

This information should be documented separately to facilitate analysis, but may be incorporated directly into business requirements.

Sample resources include:

- Databases and database tables
- Configuration files
- Cryptographic key stores
- ACLs
- Registry keys
- Web pages (static and dynamic)
- Audit logs
- Network sockets / network media

- IPC, Services, and RPC resources
- Any other files and directories
- Any other memory resource

Note: Network media is a resource of its own. Data resources will often be stored in memory, placed onto a wire, received in memory on the other end, and then stored on disk. In such a scenario, we often will not want to address the security of the data in a vacuum, but instead in the context of the resource the data is inhabiting. In the network media, we need to specify how to protect that data when it traverses the media, which may be done generically or specifically to the media.

Identify user roles and resource capabilities

Purpose:	• Define system roles and the capabilities/resources that the role can access.
Role:	Architect
Frequency:	As needed; at least once per iteration.

Identify distinct capabilities

Intelligent role division requires understanding the things in a system that users may be able to do (capabilities). Even if there is a heavy disposition to use a very limited number of roles, there is much value in identifying possible capabilities, then applying the principle of least privilege by binding capabilities to roles only when necessary. For example, even if the primary role abstraction is “user”, it is perfectly valid to restrict sensitive operations to a subset of those users.

Capabilities are interesting operations on resources that should be mediated via an authorization/access control mechanism. For example, the obvious capabilities for a file on a file system are: read, write, execute, create, and delete. However, there are other operations that could be considered “meta-operations” that are often overlooked, particularly: reading and writing file attributes, setting file ownership, and establishing access control policy to any of these operations.

Map system roles to capabilities

Roles are a way of mapping sets of capabilities to classes of users. Traditionally, people have thought of roles only at the highest level, breaking them down into administrator, users and guest, or whatever natural division suits the system. This is a reasonable high-level abstraction, but in many systems it does not serve the principle of least privilege, which states that one should have the minimal privileges necessary, and no more.

On the other end of the spectrum, one can define one role for every set of resource capabilities one might want to allow. But that can quickly get complex if users need to be able to assign capabilities to other users dynamically. As a result, it is usually best to map roles to static sets of capabilities. This should be done by specifying the default set of capabilities for the role as well as the maximum set of capabilities for the role.

In most situations, the system itself is an implicit role (or set of roles) that has all capabilities and mediates access to them — particularly in a client-server application.

Role to capability mappings can be expressed as requirements stating that the given role should have access to a particular set of capabilities. Optionally, role information can be captured in a separate artifact.

Identify the attacker profile (attacker roles and resources)

When defining system requirements, one must have a good model specifying where threats could originate. Particularly, one should attempt to identify potential groups that could be a threat as well as the gross resources one expects them to have.

For example, one should consider acknowledging the following attacker roles in an architecture:

- *Insiders* — particularly those who have physical access to the building where critical infrastructure is kept. Most crimes are caused by people with some sort of insider access, including friends, building workers etc. While many insider attacks are due to some form of disgruntlement, more often they are crimes of opportunity.
- *"Script Kiddies"* — are those people who leverage exploits that are easy to find in the underground community. This group generally targets widely deployed software systems, due to the ready availability of exploits and targets. Such systems are often present as components in more complex systems.
- *Competitors* — who may have a reasonable budget and may be willing to fund illegal or borderline activity that is unlikely to be traced back to them (e.g., due to outsourcing to Russia).
- *Governments* — who are generally extraordinarily well funded.
- *Organized crime* — who choose few targets based on financial gain but are well funded.
- *Activists* — who will target organizations that are particularly unliked. This threat vector is easy to ignore, but could be a source of risk. For example, there are non-traditional activists, such as those that target security companies perceived to be untalented.

An attacker profile should be documented independently but could be incorporated into business requirements.

Document security-relevant requirements

Purpose:	• Document business-level and functional requirements for security.
Role:	Requirements specifier
Frequency:	As needed; at least once per iteration.

In this activity, we describe how to take a resource-centric approach to deriving requirements. This approach results in much better coverage of security requirements than do ad-hoc or technology-driven methods. For example, many businesses will quickly derive the business requirement “Use SSL for security,” without truly understanding what requirements they are addressing. For example, is SSL providing entity authentication, and if so, what is getting authenticated, and with what level of confidence? Many organizations overlook this, and use SSL in a default mode that provides no concrete authentication.

All requirements (not simply security requirements) should be SMART+ requirements — i.e., they should follow a few basic properties:

- *Specific.* There should be as detailed as necessary so that there are no ambiguities in the requirement. This requires consistent terminology between requirements.
- *Measurable.* It should be possible to determine whether the requirement has been met, through analysis, testing, or both.
- *Appropriate*. Requirements should be validated, thereby ensuring that they not only derive from a real need or demand but also that different requirements would not be more appropriate.
- *Reasonable.* While the mechanism or mechanisms for implementing a requirement need not be solidified, one should conduct some validation to determine whether meeting the requirement is physically possible, and possible given other likely project constraints.
- *Traceable.* Requirements should also be isolated to make them easy to track/validate throughout the development lifecycle.

SMART requirements were originally defined by Mannion and Keepence. We have modified the acronym. The original “A” was “Attainable”, meaning physically possible, whereas “Reasonable” was specific to project constraints. We have combined these two requirements since their separation is somewhat arbitrary and since we believe there should be a focus on appropriateness. Due to this change, we distinguish our refinement as SMART+ requirements.

The original paper on SMART requirements is good elaboration on these principles. See http://www.win.tue.nl/~wstomv/edu/2ip30/references/smart-requirements.pdf.

Document explicit business requirements

Security requirements should be reflected in both business and functional requirements. Generally, business requirements will focus on demands from the customer and demands that are internal to the organization. As a result, business requirements may be somewhat unstructured.

A starting point for internally driven requirements can be taken from a global security policy, if present. Be aware that individual projects may have specific requirements that are not covered by the global policy or are in conflict with it.

Since customers often are not adequately security-aware, one should not expect to derive an exemplary set of security requirements through customer interaction. It is recommended to explicitly bring up issues that may become important with system users after deployment, particularly:

- Preferred authentication solutions;
- Preferred confidentiality solutions for network traffic;
- Preferred confidentiality solutions for long-term storage of key data; and
- Privacy concerns (particularly for personal data).

Develop functional security requirements

Functional security requirements should show how the basic security services are addressed for each resource in the system, and preferably on each capability on each resource. This generally calls for abstraction to make the process manageable. Security requirements should be, wherever possible, abstracted into broad classes, and then those classes can be applied to all appropriate resources/capabilities. If there are still resources or capabilities that do not map to the abstractions, they can be handled individually.

For example, end-user data that is generally considered highly sensitive can often be placed into a “User-Confidential” class, whereas public data could be placed into a “User-Public” class. Requirements in the first class would tend to focus on circumstances in which access to that data can be granted to other entities.

Classes can be applied either to data resources or to individual capabilities by specifying a requirement that the specific resource or capability should be handled in accordance with the security policy of the particular protection class. When applied to data resources, requirements should be specified in the abstracted class for any possible capability, even if some data elements will not have the capability.

Whereas most data resources will lump into a few reasonable abstractions, it is often the case that other system resources such as the network, local memories, and processors do not conform to user data requirements.

For each identified category, specify protection requirements on any resource in that category, relative to the basic security services:

- *Authorization (*access control*)*: What privileges on data should be granted to the various roles at various times in the life of the resource, and what mechanisms should be in place to enforce the policy. This is also known as access control and is the most fundamental security service. Many other traditional security services (authentication, integrity, and confidentiality) support authorization in some way.
- Consider here resources outside the scope of your system that are in the operating environment which need to be protected — such as administrative privileges on a host machine.
- *Authentication and integrity*: How is identity determined for the sake of access to the resource, and must the resource be strongly bound to an identity? For example, on communication channels, do individual messages need to have their origin identified, or can data be anonymous?

  Generally, requirements should specify necessary authentication factors and methods for each endpoint on a communication channel and should denote any dependencies, such as out-of-band authentication channels — which should be treated as a separate system resource.

  Integrity is usually handled as a subset of data origin authentication. For example, when new data arrives over a communication channel, one wants to ensure that the data arrived unaltered (whether accidentally or maliciously). If the data changes on the wire (whether by accident or malice), then

the data origin has changed. Therefore, if we validate the origin of the data, we will determine the integrity of the data as well.

This illustrates that authentication — if it is necessary in a system — must be an ongoing service. An initial authentication is used to establish identity, but that identity needs to be reaffirmed with each message.

Identity is the basis for access control decisions. A failure in authentication can lead to establishing an improper identity, which can lead to a violation of access control policy.

- *Confidentiality* (including privacy): Confidentiality mechanisms such as encryption are generally used to enforce authorization. When a resource is exposed to a user, what exactly is exposed: the actual resource or some transformation? Requirements should address what confidentiality mechanism is required and should identify how to establish confidentiality — usually requiring identity establishment.
- *Availability*: Requirements should focus on how available a resource should be for authorized users.
- *Accountability* (including non-repudiation): What kind of audit records need to be kept to support independent review of access to resources/uses of capabilities — i.e., what logging is necessary? Remember that log files are also a data resource that need to be specified and protected.

After building a set of abstractions and mapping it to resources, one needs to ensure that all resources (and preferably capabilities) have adequate coverage for security requirements. This generally entails walking through each resource identified in the system and attempting to determine whether there are special requirements relative to each of the core security services.

The output should not only consist of security requirements, but also documentation of what threats were considered. Considered threats should be documented on a per-resource — or per-capability — basis and should address each security service. These should be cataloged in the threat model.

Explicitly label requirements that denote dependencies

All external dependencies should be captured in requirements to whatever degree reasonable. All third-party components used should be specified. Any required functionality in the operational environment specification should be specified.

Any requirements denoting external dependencies should be explicitly labeled as such in order to facilitate subsequent analysis.

Determine risk mitigations (compensating controls) for each resource

At the business requirement level, one generally identifies what resources need to be protected — i.e., what risks on individual resources need to be addressed — and may document customer-driven technology decisions for ways to mitigate risks on those resources.

Functional requirements should specify what mechanisms should be put in place to provide security services on resources. Such mechanisms address particular risks. A requirements specifier should not worry about determining specific risks. This means that the requirements specifier should not spend too much time identifying how particular services might be compromised. Instead, he should prefer specifying general mechanisms that assume any method of compromise.

While this may not address all risks, it shifts the need for security expertise into the analysis process (usually, architectural analysis). Of course, as risks that are more granular are identified, requirements and mitigations should be updated.

Functional security requirements should focus on how potential security risks are to be addressed in a system. As with business requirements, functional security requirements can be derived in a structured way from either a set of resources (including those that are not explicitly data resources, such as the CPU) or, preferably, a set of capabilities defined over a set of resources.

Risks on capabilities differ throughout the lifetime of a system, and when specifying functional requirements for protecting data, one should explicitly consider this. If and when data-flow diagrams are available for the system, one should trace each resource through the diagram, assessing risk relative to each core security service at each step, particularly assessing whether currently identified controls are valid at each trust level.

It can be useful to carefully consider data flow through the system as opposed to just data considered statically. Realistically, requirements on that data can change, depending on the subsystem in which the data is passing — particularly as the data passes through system trust boundaries.

Particularly, one should realize that data belonging to one user could often have accidental (unauthorized) flows to other users in the system and to people with insider access to the system. Seek to protect data as soon as feasible and for as long as possible — particularly, while data is in storage.

For each resource capability tracked through the system, identify on trust boundaries what risks could be considered (iterating through the basic security services), then identify solutions for addressing those risks. If an action is to be taken as part of the system being built, document it as a functional requirement, mapping it explicitly to the capability, resource, and any relevant business requirements.

If no protection is to be implemented in the context of the system, the risk should be documented for the benefit of the end user. Additionally, when feasible, one should recommend compensating controls — mitigation techniques that can be implemented by the customer. Similarly, even when risks are addressed internal to the system, there will generally be lesser lingering risks, and these too should be documented in an operational security guide. See the activity on *Building operational security guide* for more detail.

One should iterate on security requirements as new risks are presented — such as through risk analysis.

Resolve deficiencies and conflicts between requirement sets

Many systems will have multiple levels of requirements, all of which will address security. For example, a project may have a set of business requirements, a set of functional requirements, and a set of global requirements that are effectively requirements for the project — particularly if they are not directly incorporated into either of the other artifacts.

One should map each set of requirements to the others in order to determine omissions and conflicts. For example, one can annotate a copy of global requirements, specifying which business or functional requirements map to each global requirement by iterating through the business or functional requirements that are security-relevant.

Conflicts, when noticed, should be resolved as appropriate. If a global requirement is to be exempted, an organization should have an approval process involving the owner of the global requirements and resulting in explicit sign-off. Otherwise, conflicts should be resolved by mutual agreement of appropriate contributors.

When business requirements fail to address a global requirement, or functional requirements fail to elaborate on business requirements adequately, create a new requirement as appropriate.

Detail misuse cases

Purpose:	• Communicate potential risks to stakeholder. • Communicate rationale for security-relevant decisions to stakeholder.
Role:	Requirements Specifier
Frequency:	As required; typically occurring multiple times per iteration, and most frequently in Inception and Elaboration iterations.

Identify misuse cases

Misuse cases are identical to use cases, except that they are meant to detail common attempted abuses of the system. Like use cases, misuse cases require understanding the actors that are present in the system. Those actors should be mapped to capabilities, if possible. Misuse cases should be designed for each actor, and one should also consider uses cases for nefarious collaborating actors.

As with normal use cases, one should expect misuse cases to require adjustment over time. Particularly, it is common to start with high-level misuse cases, and refine them as the details of the system are better understood.

Determining misuse cases generally constitutes a brainstorming activity. There are three good starting points for structured brainstorming:

- First, one can start with a pre-existing knowledge base of common security problems and determine whether an attacker may have cause to think such a vulnerability is possible in the system. Then, one should attempt to describe how the attacker will leverage the problem if it exists.
- Second, one can brainstorm on the basis of a list of system resources. For each resource, attempt to construct misuse cases in connection with each of the basic security services: authentication, confidentiality, access control, integrity, and availability.
- Third, one can brainstorm on the basis of a set of existing use cases. This is a far less structured way to identify risks in a system, yet is good for identifying representative risks and for ensuring the first two approaches did not overlook any obvious threats. Misuse cases derived in this fashion are often written in terms of a valid use and then annotated to have malicious steps.

Describe misuse cases

A system will have a number of predefined roles, and a set of attackers that might reasonably target instances of the system under development. These together should constitute the set of actors that should be considered in misuse cases.

As with traditional use cases, you should establish which actors interact with a use case — and how they do so — by showing a communicates-association. Also as traditionally done, one can divide use cases or actors into packages if they become too unwieldy.

Important misuse cases should be represented visually, in typical use case format, with steps in a misuse set off (e.g., a shaded background), particularly when the misuse is effectively an annotation of a legitimate use case.

Those misuse cases that are not depicted visually but are still important to communicate to the user should be documented, as should any issues not handled by the use case model.

Identify defense mechanisms for misuse cases

As one identifies defense mechanisms for various threats specified in a use case model, one should update the use case model to illustrate the defense mechanism. If there is no identified mechanism at a particular point in time, the use case should be annotated to say so.

Defense mechanisms either should map directly to a functional requirement, or, if the defense mechanism is user-dependent, to an item in an operational security guide.

Evaluate results with stakeholders

Review and discuss the misuse case with stakeholders, so that they have a clear understanding of the misuse case and agree that it is an adequate reflection of their requirements.

Identify attack surface

Purpose:	• Specify all entry points to a program in a structured way to facilitate analysis.
Role:	Designer
Frequency:	As needed; usually once after design, and ongoing during elaboration.

The attack surface can be defined explicitly in requirements, but is generally defined in the threat model document.

Identify system entry points

The system attack surface is the collection of possible entry points for an attacker. Generally, when performing a network-level design, one will already have defined the components with which an attacker can interact, giving the highest-level notion of entry points.

In this task, define the specific mechanisms through which anyone could interact with the application regardless of their role in the system. For example, document all network ports opened, all places where the file system is touched, any local UI elements, any inter-procedural communication points, and any public methods that can be called externally while the program is running.

For each entry point, provide an unambiguous description and a unique identifier. Generally, this information — as well as the supporting information collected below — can be stored in a table-based format much like a requirements matrix.

Program entry points should be documented as they are identified. Often, as a project transitions from specification to elaboration, entry points become more granular. This increased granularity should be handled by defining attack surfaces hierarchically. For example, data communication over a network port will have a corresponding handler in the code where input from the network socket is read and will sometimes have multiple handlers. Such handlers should be identified as input points that are parented under the specific network socket.

Another example is a web application. There may be one or more ports that are entry points, and there may be multiple web pages on the port that are entry points. Also, each web page may have one or more forms that are entry points.

Map roles to entry points

For each point in the attack surface, identify all roles that could possibly access the entry point. This should map to trust boundaries previously defined — i.e., all entry points in the same trust boundary should have the same set of roles attached. Otherwise, ensure that there really is a control enforcing access control to the resource and update trust boundaries appropriately.

Map resources to entry points

For each entry point, document the resources that should be accessible from that entry point — and capabilities that should be accessible if the system is specified to this level. This will facilitate building data flow diagrams, if part of your process. It will also facilitate security analysis — as will data flow diagrams, if available.

Apply security principles to design

Purpose:	• Harden application design by applying security design principles. • Identify security risks in third-party components.
Role:	Designer
Frequency:	As necessary; at least once per iteration

Refine existing application security profile

This activity is performed on an existing design. If it follows other CLASP activities, the team will have done the following before this point:

- Identified resources in the system and capabilities on those resources;
- Identified roles in the system;
- Identified trust boundaries; and
- Identified requirements for providing security services on a resource-by-resource basis, throughout the lifetime of the resource.

Often, all of this information will be identified in the requirements. If any of the information is not present, it should be produced at this time.

If the information does exist, it should be updated to account for additional detail and refinements that have since been added to the architecture.

At the end of this subtask, one should understand the security needs for each role resource in the system, throughout the complete lifetime of the application, including security requirements for data links and long-term data storage.

Determine implementation strategy for security services

Security requirements should specify what needs to be done in relation to core security services. The purpose of design is to elaborate on how those requirements will be met.

Identify solutions for meeting security requirements at each identified point in the design by adhering to the following principles:

- Look for third-party solutions, starting the search with a preference for well-vetted off-the-shelf solutions to untrusted solutions or in-house solutions. For example, when cryptography is viewed as a solution to a problem, look first to see if there are recent standards from well-regarded standards bodies that address the problem.

  For example, the recent trend for standards by organizations such as the IETF, IEEE, ITU, and NIST is to adopt well-vetted research ideas into standards, then bring in external security review. Do enough diligence to build confidence that the research community is not worried about the standard. If no good standard exists, try to leverage software that has a clear lineage from peer-reviewed academic research and avoid designing your own solutions without the guidance of a well-respected cryptographer.

- When considering off-the-shelf technologies, perform a risk assessment of the technology before designing it into the system, as discussed in the next activity. When choosing to integrate the technology, go back and integrate additional security requirements into the product requirements as appropriate.
- Design appropriate validation mechanisms — input validation, authentication, and authorization — wherever data can enter a system or cross a trust boundary. For example, in a multi-tier system with a firewall, it is insufficient to perform either input validation or authentication on data of external origin, because insiders behind the firewall would be able to inject data without being validated.

  A more reasonable solution is to validate on every architectural tier and to pass credentials securely between architectural components.
- Ensure that identified solutions address risks to the desired degree. For example, input validation routines that only perform basic pattern matching and do not perform syntactic validation can often be circumvented. See the discussion in CLASP Resource B on input validation.
- Prefer the simplest solution that meets requirements. Complex solutions tend to both have additional inherent risks and be harder to analyze.
- When multiple security solutions are necessary to better alleviate risks — i.e., a single solution is left with risk that still needs to be mitigated using another solution — be sure that, if there is an instance of a risk that one of the solutions can address, the risk does get addressed. For example, if using multiple authentication factors such as passwords and smart cards, a user should need to validate using both technologies, instead of simply one.

  If this “defense-in-depth” strategy is taken, the attacker has to thwart both authentication mechanisms. Otherwise, the system is only as good as the weaker of the two mechanisms — the “weakest-link” principle.
- Look for ways to minimize exposure if defenses are somehow compromised: e.g., fine-grained access control, trusted systems, or operational controls such as backups, firewalls, and the like.

Build hardened protocol specifications

While it is desirable to use high-level protocols for security such as SSL/TLS, most applications will ultimately define their own semantics and thus their own protocols when communicating.

No matter how simple, protocols that are developed in-house should be well-specified so that they can be analyzed. They should always be rigid in what they accept. This means that the method for performing input validation should be apparent in the protocol specification.

A cryptographer should analyze any system containing new protocols for secure communication or identity establishment authored by the development organization. Protocols should also be as simple as feasible so as to be as easy to analyze as is feasible.

One should also specify what happens on error conditions. Generally, when errors are not related to well-known classes of accidental user error, it is best to fail safely and reset, even if there is minimal lack of availability created, because secure recovery from unexpected and infrequent classes of errors is generally quite difficult to perform.

Design hardened interfaces

API interfaces themselves define protocols, and should be treated in the same way, with well-defined specifications, including specifications defining valid input. Note that — as discussed in the Input Validation concept — checking the range of each parameter in isolation is not always a sufficient specification. Be thorough in defining under which circumstances data is semantically valid. For example, if the first parameter affects what values are valid for the second parameter, this should be noted in a specification.

APIs should also come with well-specified error handling mechanisms. Callers should be forced to deal with unusual conditions when they occur. Particularly, do not specify use of error codes that a developer will often ignore. Instead, specify use of an exception that — if all else fails — will be caught at the top level; in this case, the program should fail securely and reset.

Additionally, one should focus on exporting a few simple APIs, which will minimize the attack surface.

Research and assess security posture of technology solutions

Purpose:	• Assess security risks in third-party components. • Determine how effective a technology is likely to be at alleviating risks.
Role:	Designer
Frequency:	As necessary.

Get structured technology assessment from vendor

If a technology is to be integrated into your system — even if it is for the purposes of mitigating risk in your own system — you will generally assume the risks associated with that technology.

For this reason (among others), it is most desirable to assess the security risks of such components in the same way as your own software. Vendors are rarely cooperative in giving the access required for this; and in cases where they are (e.g., open source software), the effort involved in a full assessment is rarely cost-effective.

Instead, one will generally want to collect relevant data that will provide insight into the likely security posture of software through interaction with the vendor. See CLASP Resource F for a sample “self-assessment worksheet” that either the vendor can fill out, or (more often) you can fill out, based on interaction with the vendor.

A good product assessment worksheet should give insight into the following:

- At a high level, what are the trust boundaries, resources, and roles in the system?
- Has an independent assessment been performed by a respected third-party? And if so, what business risks did it identify, and what has changed since the assessment?
- What are the security qualifications of the development team?
- What are the major areas of risk in the product?
- What were the security requirements that were used for development (implicit and explicit)?

This assessment should essentially be a structured interview with the purpose of collecting as much documentation as possible about the product and the process used to develop that product.

Perform security risk assessment

Perform due diligence on the vendor-reported assessment information to the degree possible. For example, validate data with other customers and/or through information available on the Internet.

Perform a requirements analysis from the material collected to assess resource risks that may be present but that are not addressed by the product. For any risk that would not be acceptable if incorporated into your effort, identify possible mitigating controls, the likely cost to implement, and who would need to implement the control — particularly if it is the vendor.

If desirable, attempt to resolve risks with the vendor. Based on the assessment, make a determination on whether to proceed with the technology.

Receive permission to perform security testing of software

A way to gain additional confidence in software is to test it. However, testing software for security vulnerabilities may be in violation of a software licensing agreement. To avoid any potential issues, vendor acknowledgement should be sought.

Perform security testing

Perform security testing as described in the CLASP activity *Identify, implement and perform security tests*.

Annotate class designs with security properties

Purpose:	• Elaborate security policies for individual data fields.
Role:	Designer
Frequency:	Once per iteration

Map data elements to resources and capabilities

Each data element in the system should have a security policy for it that is defined by the system requirements and design, either explicitly or explicitly. While security requirements should be defined on a per-resource or a per-capability basis, data elements will often not be a resource on their own, but will be a component of a more abstractly defined resource.

Each data element should be mapped back to the requirements to determine the requirements on that data in relation to the basic security services. Often, this task will lead to a refinement of requirements.

For example, consider a system that defines user data as a resource. There may be an access control requirement stating the data should be available only to the individual user and the administrator — except as allowed by the user. In such an example, it may be that not all data should have this flexibility. Maybe the user could choose to export his name and address to others but not his social security number.

Realistically, such refinement of requirements happens frequently, and in an agile environment, these changes may not be incorporated directly into requirements; in this case, documenting information either in a class diagram or as a structured annotation to the code helps ensure correct implementation and facilitates review.

Annotate fields with policy information

Note that access control policy on a resource depends on the operation on that resource (i.e., the capability). In a class diagram, capabilities are generally identified by methods operating on that data.

Data fields should define the owning role or roles and should also define generically which role or roles have access to which basic capabilities throughout the lifetime of the data — e.g., read, write, modify, execute, assign permissions to a capability, and add or transfer ownership.

An important goal of such a specification is to allow an auditor to determine whether data could ever flow in a way that violates the access control policy. The policy should be as coarse as possible to make it easy to specify and check.

A coarse policy will often require exceptions to implement a policy that is more complex. That is, there may be conditions where it may be valid to pass data in a way that would not be allowed by a high-level policy. For example, consider a simple policy that user data should not go to other users. Instead of specifying fine-grained capabilities around granting read and write access, one can mark the data as relaxable.

Points where such decisions are made are called relaxation points. How relaxation can occur should be well specified in the requirements, and the number of points in the program should be minimized to lessen the chance of error and facilitate analysis.

If policy relaxation should never be necessary for a data element, it should be annotated as non-relaxable. Otherwise, it should be annotated as relaxable, along with a description under the conditions where relaxation can occur; this may be done by identifying a requirement by reference.

Annotate methods with policy data

Methods operate on data, and may use one or more capabilities on that data. Methods should be annotated to identify which operations they perform on data, and whether they are relaxation points for any data element.

Specify database security configuration

Purpose:	• Define a secure default configuration for database resources that are deployed as part of an implementation. • Identify a recommended configuration for database resources for databases that are deployed by a third party.
Role:	Database Designer
Frequency:	As necessary; generally once per iteration.

Identify candidate configuration

Choose a candidate database configuration for the database.

While an out-of-the-box configuration is an acceptable starting point, it is usually more efficient to start with a third-party baseline or to go through a process to identify a candidate baseline. For example, see the NIST database security checklist: http://csrc.nist.gov/pcig/cig.html.

In the case of third-party deployments, the configuration will generally be defined relative to the default configuration.

Validate configuration

For the resources specified that interact with the database, validate that the baseline configuration properly addresses the security requirements for that data.

Also, unnecessary functionality (e.g., stored procedures) can introduce unanticipated risk vectors. Practice the principle of least privilege by removing unnecessary functionality from the configuration.

In the case of third-party deployments, it is sufficient to specify which functionality is absolutely necessary in the operational security guide, then to recommend that all other features be disabled.

If appropriate, perform testing with a database configuration tool for any candidate platforms to determine non-obvious security risks to resources. Again, make changes to the configuration if necessary, and documenting them in the operational security guide, if appropriate.

Perform security analysis of system requirements and design (threat modeling)

Purpose:	• Assess likely system risks in a timely and cost-effective manner by analyzing the requirements and design. • Identify high-level system threats that are documented neither in requirements nor in supplemental documentation. • Identify inadequate or improper security requirements. • Assess the security impact of non-security requirements.
Role:	Security Auditor
Frequency:	As needed; generally, once initial requirements are identified; once when nearing feature complete.

Develop an understanding of the system

Before performing a security analysis, one must understand what is to be built. This task should involve reviewing all existing high-level system documentation. If other documentation such as user manuals and architectural documentation exists, it is recommended to review that material as well.

To facilitate understanding when the auditor is not part of the project team, it is generally best to have a project overview from a person with a good customer-centric perspective on the project — whom we assume is the requirements specifier.

If feasible, documentation should be reviewed both before and after such a review so that the auditor has as many opportunities to identify apparent constancies as possible. If documentation is only to be read once, it is generally more effective to do so after a personal introduction.

Anything that is unclear or inconsistent should be presented to the requirements specifier and resolved before beginning analysis.

Determine and validate security-relevant assumptions

Systems will be built with assumptions about the attacker and the environment in which the software will be deployed. If the proper CLASP activities have been incorporated into the development process, the following key information should be documented before starting a requirements assessment:

- A specification of the operational environment;
- A high-level architectural diagram indicating trust boundaries;
- A specification of resources and capabilities on those resources; this may be incorporated into the requirements;
- A specification of system users and a mapping of users to resource capabilities; this also may be incorporated into the requirements;
- An attack surface specification, to whatever degree elaborated;
- Data flow diagrams, if available;

- An attacker profile (again, this may be part of the requirements); and
- Misuse cases, if any.

With the exception of misuse cases, if the development process does not produce all of these artifacts, the security auditor should do so. Sometimes reviewers will forego data-flow diagrams, because the flow of data is well understood on the basis of the architectural diagram.

If the artifacts have been produced previously, the auditor should validate the security content of these documents, particularly focusing on inconsistencies, technical inaccuracies, and invalid assumptions. Particularly, review should address the question of whether the attacker profile is accurate since many organizations are not attentive enough to insider risks.

Any assumptions that are implicit should be validated and then incorporated into project documentation.

Review non-security requirements

For requirements that are not explicitly aimed at security, determine whether there are any security implications that are not properly addressed in the security requirements. This is best done by tracing resources that are relevant to a requirement through a data-flow diagram of the system and assessing the impact on each security service.

When there are security implications, identify the affected resource(s) and security service(s), and look to see if there is a requirement explicitly addressing the issue.

If you are using a correlation matrix or some similar tool, update it as appropriate after tracing each requirement through the system.

Also, correlate system resources with external dependencies, ensuring that all dependencies are properly listed as a resource. Similarly, perform a correlation analysis with the attack surface, making sure that any system entry points in third-party software are reflected.

Assess completeness of security requirements

Ensure that each resource (or, preferably, capability) has adequate requirements addressing each security service. A best practice here is to create a correlation matrix, where requirements are on one axis and security services on capabilities (or resources) are on another axis. For each security requirement, one notes in the appropriate boxes in the matrix which requirements have an impact.

The matrix should also denote completeness of requirements, particularly whether the security service is adequately addressed. As threats are identified in the system that are not addressed in the requirements by compensating controls, this documents what gaps there are in the requirements.

Identify threats on assets/capabilities

Iterate through the assets and/or capabilities. For each security service on each capability, identify all potential security threats on the capability, documenting each threat uniquely in the threat model.

In an ideal world, one would identify all possible security threats under the assumption of no compensating controls. The purpose is to demonstrate which threats were considered, and which controls mitigate those threats. However, one should not get too specific about threats that are mitigated adequately by compensating controls.

To achieve this balance, one identifies a threat and works to determine whether the threat can be applied to the system (see next subtask). If the auditor determines that the threat cannot be turned into a vulnerability based on controls, avoid going into further detail.

For example, a system may use a provably secure authenticated encryption system in conjunction with AES (e.g., GCM-AES) with packet counters to protect against replay attacks. There are many ways that the confidentiality of this link might be thwartable if this system were not in place. But since the tools are used properly, the only possible threat to confidentiality is breaking AES itself, which is a result of the GCM security proof. Since — assuming that the tools are used correctly — all possible on-the-wire threats are mitigated except for this one, threat analysis should focus on determining whether the tool was used correctly and not on determining what threats might exist if the tool is used incorrectly (or if a different tool is used).

Identifying security threats is a structured activity that requires some creativity since many systems have unique requirements that introduce unique threats. One looks at each security service and ask: “If I were an attacker, how could I possibly try to exploit this security service?”. Any answer constitutes a threat.

Many threats are obvious from the security service. For example, confidentiality implemented using encryption has several well-known threats — e.g., breaking the cipher, stealing keying material, or leveraging a protocol fault. However, as a baseline, use a list of well-known basic causes of vulnerabilities as a bare minimum set of threats to address — such as the set provided with CLASP. Be sure to supplement this with your own knowledge of the system.

This question of how to subvert security services on a resource needs to be addressed through the lifetime of the resource, from data creation to long-term storage. Assess the question at each trust boundary, at the input points to the program, and at data storage points.

Determine level of risk

Use threat trees to model the decision-making process of an attacker. Look particularly for ways that multiple conditions can be used together to create additional threats.

This is best done by using attack trees (CLASP Resource A). Attack trees should represent all known risks against a resource (which is the root of the tree), the relationships between multiple risks (particularly, can risks be combined to result in a bigger risk), and then should characterize the likelihood of risk and the impact of risk on the business to make decisions possible.

Risk assessment can be done using a standard risk formula for expected cost analysis, but the data is too complex to gather for most organizations. Most organizations will want to assign relative values to important concerns and use a weighted average to determine a risk level.

Most of the important concerns going into such an average can be identified using Microsoft’s DREAD acronym:

- *Damage potential.* If the problem is exploited, what are the consequences?
- *Reproducibility*. How often does an attempt to exploit a vulnerability work, if repeated attempts have an associated cost. This is asking: What is the cost to the attacker once he has a working exploit for the problem? In some cases, a vulnerability may only work one time in 10,000, but the attacker can easily automate attempts at a fixed additional cost.
- *Exploitability.* What is the cost to develop an exploit for the problem? Usually this should be considered incredibly low, unless there are mitigating circumstances.
- *Affected users.* What users are actually affected if an exploit were to be widely available?

- *Discoverability.* If unpatched, what is the worst-case and expected time frame for an attacker to identify the problem and begin exploiting it (generally assume a well-informed insider risk with access to your internal process in the first case, and a persistent, targeted reverse engineer in the second).

Additionally, proper risk assessment requires an estimation of the following factors:

- The effectiveness of current compensating controls. If the control is always effective, there is little point in drilling down farther after that fact is well documented.
- The cost associated with implementing compensating controls — as the cost of remediation — must be balanced against the expected loss.

For existing compensating controls, map them to the specific threat you have identified that they addressed, denoting any shortcomings in the control.

If it is unclear, use data flow diagrams and available resources to determine where the threat is or is not adequately addressed, focused particularly on storage, input points (the attack surface), and trust boundaries (generally, network connections).

Unfortunately, detailed values for each of these concerns are difficult to attain. Best practice is to assign relative values on a tight scale (for example: 0-10), and assign weights to each of the categories. Particularly, damage potential and affected users should generally be weighted most highly.

For each risk identified in the system, use the present information to make a determination on remediation strategy, based on business risk. At a bare minimum, make a determination such as: “Must fix before deployment”; “Must identify and recommend a compensating control”; “Must document the problem”; or “No action necessary”.

Identify compensating controls

For each identified risk with inadequate compensating controls, identify any feasible approaches for mitigating the risk and evaluate their cost and effectiveness.

Evaluate findings

The auditor should detail methodology and scope, and report on any identified security risks that may not have been adequately addressed in the requirements.

Additionally, for any risks, the auditor should recommend a preferred strategy for implementing compensating controls and should discuss any alternatives that could be considered.

If any conflicts or omissions are brought to light during requirement review, the security auditor should make recommendations that are consistent with project requirements.

The security auditor should be available to support the project by addressing problems adequately.

The project manager is responsible for reviewing the findings, determining whether the assessments are actually correct to the business, and making risk-based decisions based on this information. Generally, for those problems that the project manager chooses to address, the risk should be integrated into the project requirements and tasking.

Integrate security analysis into source management process

Purpose:	• Automate implementation-level security analysis and metrics collection.
Role:	Integrator
Frequency:	As required

Select analysis technology or technologies

There are a number of analysis technologies that could be integrated into the development process. One broad way to categorize them is dividing them into two classes:

- *Dynamic analysis tools*, which require running the program in order to perform an analysis, often in its full operational context for maximum effectiveness; and
- *Static analysis tools*, which analyze the program entirely without running the program.

Generally, dynamic analysis tools are better suited to be run manually as part of the quality assurance process, as they require running many tests to exercise the code thoroughly, and often those tests must be driven by a human.

There are several available static analysis tools.

Determine analysis integration point

Source code analysis can be integrated into source management as part of the check-in process, as part of the build process, or independently. CLASP recommends integrating it into check-in and into build, using efficient but less accurate technology to avoid most problems early, and deeper analysis on occasional builds to identify more complex problems.

Integration at check-in can be used to prevent check-in of code into a primary branch that does not meet coding standards or to assign potential new security defects to committers. The first goal is not well suited to legacy software applications, unless a baseline of tool output is used for comparison. The second goal also requires baseline output used for comparison that is updated incrementally.

Deep analysis can be done as a result of check-in, but frequent deep analysis is not necessary. Developers should get more immediate feedback; security auditors should get more detailed feedback, but not as frequently as with every check-in.

Integrate analysis technology

Analysis technology should be integrated into the source management process in an automated way if possible. If the technology does not support such integration out-of-the-box, one could consider building integration. Otherwise, it must be performed manually, which will generally rule out per-check-in analysis.

Integrating analysis technology should involve the following:

- Producing a version of the source to be tested which is suitable for input into the analysis tool. Most analysis tools will require the code to compile as well as instructions for turning the code into an actual executable, even though the executable is not run.
- Performing the analysis.
- Eliminating results that have been previously reported by the tool and have yet to be resolved.
- Presenting any new results or the lack of results to the appropriate parties — usually the developer or the security auditor. This may occur through a common interface, such as a bug tracking system. Potential problems should go through a single process for reported security problems.
- Storing information about the analysis for use in future analyses, and also store any metrics collection.

Implement interface contracts

Purpose:	• Provide unit-level semantic input validation. • Identify reliability errors in a structured way at the earliest point in time.
Role:	Implementer
Frequency:	As needed; generally as functions or methods are modified.

Interface contracts are also commonly known as assertions. They can be a formidable tool for preventing security problems — particularly if applied consistently, and rigorously.

In many application development processes, interface contracts are not enabled in production software. They are removed by habit in order to improve efficiency. If the efficiency impact is nominal for the project, CLASP strongly recommends leaving such checks in the code for the sake of security.

Otherwise, checks of security critical parameters should be implemented using a permanent mechanism, such as code directly at the top of the function, as discussed in activities below.

Implement validation and error handling on function or method inputs

For each method or function visible outside its compilation unit, specify in code what the expectations are for valid input values. One should validate that each input variable has a valid value in and of itself, and should determine validity in relation to other inputs. Validation checks should contain no side effects. Failures should be handled as specified in design. See CLASP Resource B for the concept on input validation.

Input variables should not be constrained to parameters. Any variable read by the function or method should be considered an input variable — including global variables, and class and method variables. Note that some interface contract facilities will allow specifying invariants for an entire class — i.e., things that must always be true about class data before and after each method invocation — once.

Implement validation on function or method outputs

Perform the same validation between relationships before exiting a function or method. Output specifications are meant to provide a clear behavioral specification to calling code to prevent accidental misuse.

Generally, output validation code is most useful in implementation. It is reasonable to disable such code for deployment or even use pseudo-code if absolutely necessary.

Implement and elaborate resource policies and security technologies

Purpose:	• Implement security functionality to specification.
Role:	Implementer
Frequency:	As necessary.

Review specified behavior

The developer should identify any remaining ambiguities in the specification of security properties or technologies, including any further information necessary to build a concrete implementation.

Perceived ambiguities should be addressed with the designer.

Implement specification

As with most development, implementers should build software to specification. Even when security is a concern, this is not different. As is the case when implementing traditional features, the implementer should ensure that all coding guidelines are met — especially security guidelines.

Address reported security issues

Purpose:	• Ensure that identified security risks in an implementation are properly considered.
Role:	Designer
Frequency:	As required.

Assign issue to investigator

When a security issue is identified in a system, further investigation should be assigned to the appropriate designer if it can be determined from known information about the problem. Otherwise, it should be assigned to the chief architect until the determination of the most appropriate designer can be made.

Assess likely exposure and impact

If the problem exists in released software and was reported by a security researcher, attempt to reproduce the exploit in order to determine whether the vulnerability actually exists. If it cannot be reproduced, work with the researcher to determine whether the problem does not actually exist or whether it could have been a side effect of something in the researcher's test environment.

When reproducing the exploit is too difficult or when there is no risk of disclosure, at least determine whether there is enough evidence to demonstrate that the vulnerability is likely to exist.

Determine the circumstances when the vulnerability could potentially be exploited in order to get a sense of the overall risk level, focusing on the following:

- Which builds of the product contain the risk, if any?
- Which configuration options are required in order for the risk to exist?
- What must the operational environment look like for the risk to be relevant?

This information will allow you to determine how many customers will — or would be — at risk.

Determine what the worst case and likely consequences are for the risk. From this information, determine how responding to this risk will be handled from a resourcing perspective. That is, will it be handled at all, immediately, or at a particular point in time? Further: Will there be an effort to provide more immediate remediation guidelines to customers while a permanent patch is being devised?

If the risk involves software that may be in use by other vendors in their products, contact either the vendors directly or a coordinating body — such as the CERT (Computer Emergency Response Team) coordination center.

Determine and execute remediation strategies

Identify how the problem is to be addressed, in the short term and in the long term, if the short-term solution is not a permanent fix. Incorporate the task of addressing the problem into the development lifecycle if appropriate.

If part or all of the remediation strategy involves implementing external controls, task an appropriate party to document the implementation of those controls in the operational security guide.

The architect should review all remediation strategies that impact the code base before they are implemented in order to ensure that they are valid in the context of the entire system.

Validation of remediation

Perform testing to ensure that the risk was properly addressed. This should include production of regression tests meant to detect the vulnerability if accidentally introduced. See the CLASP activity on testing for more information.

Perform source-level security review

Purpose:	• Find security vulnerabilities introduced into implementation.
Role:	Security Auditor
Frequency:	Incrementally, at the end of each implementation iteration.

Scope the engagement

It is rarely possible to look at each line of code in a system, particularly if someone needs to understand its relationship with every other line. Therefore, it is important to collect as much information as feasible about the system architecture and overall development process in order to help scope out the areas that merit the most attention.

The auditor should always start by collecting the most recent documentation for the system — including requirements, architecture, API docs, and user manuals. If previous steps in the process were followed, the material needed to scope a source-level security review should have already been produced and would be included in this material. The auditor should ensure that all documentation seems to be present and should work to collect anything that is not. While the auditor can perform an initial sanity check of the material collected, this check should not be the initial focus since much of the auditing work will involve performing such validation.

The auditor should be collecting the following material (and generally producing it if it does not exist):

- *System requirements and specification.* An auditor is expected to identify places where security requirements are violated and to make recommendations for remediating risks.
- *A threat profile for the system*. Possible threats: governments, employees, etc., and the associated capabilities they are assumed to have.
- *Any previous assessments*, including architectural assessments.

The data one should be capturing in the scoping of the engagement is collected in the assessment worksheet in CLASP Resource F.

If the auditor did not produce the threat profile — or if the threat profile is not current —, one should perform an incremental assessment, focusing on changes and shortcomings in the original.

Run automated analysis tools

Automated analysis may be incorporated into the build process, in which case the auditor can use results from a current analysis, instead of running an additional analysis.

Evaluate tool results

For each potential risk identified by the tool, assess whether the risk is relevant to the development effort. Risks that are not relevant should be marked as not relevant for one of the following reasons:

- The risk is mitigated by an existing or recommended compensating control that is not within the scope of analysis for the tool.

- The risk is not in the threat profile for the program. For example, attacks that require local user access to the same machine running the software may have already been deemed outside the scope of consideration.
- The risk is a false positive in the analysis itself.

Evaluating the results requires tool-dependent processes. Determining absolutely whether a tool result is a real vulnerability or a false positive is often not necessary, as it often involves attempting to craft an exploit. Instead, the investigator should deem it a likely risk in the case of those risks that the investigator cannot rule out as a risk based on examining the tool output and the code.

For those risks that are relevant, determine impact and recommend remediation strategies in the same manner as performing an architectural analysis, documenting results in an implementation security review report.

Identify additional risks

Analysis tools are not capable of finding all security risks in software. Many classes of risk can be identified in an architectural analysis that is not conclusively controlled. Additionally, some classes of risk may not be considered in an architectural analysis because they are artifacts of implementation error.

Compose a list of possible risks by reviewing both those risks identified in the architectural analysis and a database of common risks. See the CLASP Vulnerability database in the section CLASP Vulnerability View.

For each potential risk, identify system resources that might be susceptible to the risk. Follow execution through the code from any relevant input points to the data resource, looking at each appropriate point whether there is a likely instantiation of the risk.

As with examining tool output, the investigator should not look to prove risk beyond a doubt. Identifying likely risks is sufficient, where a likely risk is one that the auditor cannot rule out on the basis of a detailed manual analysis of the code.

Determine the impact of likely risks that are identified and recommend remediation strategies in the same manner as if performing an architectural analysis, documenting results in an implementation security review report.

Identify, implement, and perform security tests

Purpose:	• Find security problems not found by implementation review. • Find security risks introduced by the operational environment. • Act as a defense-in-depth mechanism, catching failures in design, specification, or implementation.
Role:	Test Analyst
Frequency:	As necessary; generally multiple times per iteration.

Identify security tests for individual requirements

For any requirement previously identified to have security relevance, identify an implementable testing strategy, looking to provide as complete assurance as possible and noting that some testing may be best performed statically — which is therefore potentially outside the scope of the actual QA organization. However, it is a good idea to dynamically test even those things that are assured statically, particularly if something in the operational environment could adversely affect the original test result.

Build these security tests into your test plan as with any other test. For example, specify the frequency at which the test should be run.

See the security testing techniques in CLASP Resources A, B and C.

Identify resource-driven security tests

Usually, a system will not have resource-driven security requirements, or those requirements will somehow be inadequate if only in minor ways.

If necessary, identify the resources available to the system on the basis of the architectural documentation and use of the software.

For each resource, identify whether that resource was addressed adequately by testable security requirements — i.e., that it had testable protection mechanisms in place for the core security services.

Note that in many cases security requirements will be left implicit, leaving the tester or analyst to guess what a violation of security policy entails. In such cases, the analyst should particularly focus on identifying tests that can ferret out non-obvious users of resources. That is, identify tests that will determine which system roles can gain access to each resource, paying attention to the case of unauthorized parties, as well as valid users attempting to access the resources that should only be accessible to the owning user.

Again, integrate any identified tests into the existing test plan.

Identify other relevant security tests

Using a common testing checklist, determine what other security tests are appropriate to the system. For an example, see the checklists in the book *How to Break Software Security* by Whittaker and Thompson.

Missing tests will point out a weakness in the resource-driven security requirements, and the gap should be communicated to the requirement specifier. Often, these gaps will be a failure in specifying the operational security requirements. If security testing determines that the security depends on the operational

environment, or if it is obvious that security depends on the operational environment, then the test analyst should inform the owner of the operational security guide, who should document the issue appropriately.

Implement test plan

Implement the test plan as normal. For example, the test plan may indicate acquiring tools, writing test scripts, or other similar activity.

Execute security tests

Perform the identified security tests as specified in the test plan.

Verify security attributes of resources

Purpose:	• Confirm that software abides by previously defined security policies.
Role:	Tester
Frequency:	Once per iteration.

Check permissions on all static resources

Using a standard install on a clean system, inspect the permissions and access controls placed on all resources owned by the system, including files and registry keys. The permissions granted by the system's default install should exactly match those put forth by the resource specifier in the security requirements, or from the global security policy.

If no specific permissions are identified by resources, determine whether roles other than the owning role can access the resource, based on its permissions.

Any deviation from specified or expected behavior should be treated as a defect.

Profile resource usage in the operational context

The requirements, a security profile the or operational security guide should specify what resources the system should be able to access. When performing functional and non-functional testing, use profiling tools to determine whether the software abides by the policy. In particular, look for the following:

- Access to network resources (local ports and remote addresses) that are — or appear to be — invalid.
- Access to areas of the local file system outside the specification.
- Access to other system data resources, including registry keys and inter-process communications.
- Use of system privileges in situations that are not specified.

Again, any deviation from specified or expected behavior should be treated as a defect.

Perform code signing

Purpose:	• Provide the stakeholder with a way to validate the origin and integrity of the software.
Role:	Integrator
Frequency:	Once per release build.

Obtain code signing credentials

A prerequisite for code signing are credentials that establish your identity to a trusted third party. Most PKI (public key infrastructure) vendors (also known as certification authorities, or CAs), offer Software publishing Certificates (i.e., code signing credentials), including Verisign. Process for obtaining credentials differs, depending on the CA.

Identify signing targets

Signatures are generally performed on a unit that contains all parts of an application, such as a single archive file (JAR, WAR, or CAB). Generally, the unit is an installable package. Any other granularity requires multiple signature checks per application install, which is inconvenient for the end user.

Sign identified targets

Running the code signing tools usually will automatically add a signature to the packaging unit, which can then be distributed directly.

Build operational security guide

Purpose:	• Provide stakeholder with documentation on operational security measures that can better secure the product. • Provide documentation for the use of security functionality within the product.
Role:	Implementer
Frequency:	Once per iteration.

In the course of conception, elaboration, and evaluation, there will generally be many items identified that should be communicated to one or more roles at deployment. This information should all be collected in a role-driven implementation guide that addresses security concerns.

Document pre-install configuration requirements

Begin by documenting the environmental requirements that must be satisfied before the system is installed. See the task on operational environment assumptions for more detail.

Document application activity

Document any security-relevant use of resources, including network ports, files on the file system, registry resources, database resources etc. See the activity on *Resource identification* for more detail.

Document the security architecture

Document the threat profile assumed in design and the high-level security functionality of the system as relevant to the user — including authentication mechanisms, default policies for authentication and other functions, and any security protocols that are mandatory or optional. For protocols used, document the scope of their protection.

Document security configuration mechanisms

List, and explain all security configuration options present in the system, and make note of their default and recommended settings. Be explicit about how they work, referencing any technologies utilized.

Document significant risks and known compensating controls

Any known security risks that the customer may find reasonable should be documented, along with recommended compensating controls, such as recommended third party software that can mitigate the issue, firewall configurations, or intrusion detection signatures.

Manage security issue disclosure process

Purpose:	• Communicate effectively with outside security researchers when security issues are identified in released software, facilitating more effective prevention technologies. • Communicate effectively with customers when security issues are identified in released software.
Role:	Project Manager
Frequency:	As needed.

Many security researchers find security problems in software products, often by intentional investigation. Except in a very few cases, researchers will release information about the security vulnerability publicly, usually to either the BUGTRAQ mailing lists or the Full Disclosure mailing list.

Most security researchers act responsibly in that they attempt to give the software vendor adequate time to address the issue before publicly disclosing information. This activity addresses how to interface with responsible security researchers.

Industry best-practice guidance for responsible security vulnerability research can be found at: http://www.whitehats.ca/main/about_us/policies/draft-christey-wysopal-vuln-disclosure-00.txt

Provide means of communication for security issues

If reasonable, the communication mechanism should be published on the vendor web site in a security area devoted to the product since this is where researchers will first look.

Otherwise, vendors should be prepared to handle security alerts at the following standard addresses:

- security@
- secalert@
- contact@
- support@
- sales@
- info@
- The listed domain contact information.

A researcher attempting to be responsible may still not be well informed, and so may only try one of these addresses. Some researchers will only attempt communication until they successfully send the vendor an E-mail that does not bounce. Sometimes that E-mail will be sent to a high-volume alias or to an individual who receives a high volume of E-mail, such as the CEO or CTO.

A central security response alias should be established, such as security@ or secalert@ and published on the web site if possible. Additionally, owners of various E-mail addresses that might receive security alerts should be notified of the central alias and be asked to forward any relevant communication.

Acknowledge receipt of vulnerability disclosures

On receipt of the vulnerability disclosure, respond with acknowledgement of receipt, as well as a reasonable timetable for addressing the vulnerability. This should never take more than a calendar week from receipt and should generally be handled as quickly as possible.

The time line should indicate at a bare minimum when the vendor expects to be able to provide remediation for the problem, if validated. Responsible security researchers often will inform the vendor that they will go public if the time frame given is seen as an attempt to keep the information from the public. Generally, target 30 days, but let the researcher know that you may require 30 to 60 days more if circumstances warrant. Also, inform him that you expect the researcher to act responsibly by not disclosing before you can ready a remediation strategy for customers (as long as you act in a reasonable time frame), and show that you are doing so in such a way that the researcher can determine good faith.

Good faith is best shown by providing weekly status updates, which should be offered in the acknowledgement E-mail.

If the vulnerability is found in a version of the software that is no longer supported, this should be communicated. However, you should attempt to ascertain whether the vulnerability affects supported versions of the software, and this fact should also be communicated to the researcher.

The process and policies for security disclosure should be communicated clearly to the researcher, either by E-mail or by publishing it on the web, in which case the web page should be referenced in the E-mail.

Address the issue internally

The reported vulnerability should be entered into the process for dealing with reported security issues. Communication information for the researcher should be passed along, in case further contact is necessary to better understand the report.

The researcher should be given the opportunity to test any remediation strategies implemented before they are distributed publicly. The researcher will generally make an effort to determine whether the vulnerability has been addressed adequately. In cases where it is not addressed adequately, the researcher should give the vendor additional time to address the problem, if required.

Communicate relevant information to the researcher

As the issue is internally addressed, the vendor should provide the researcher with the following information on update, as the information becomes available:

- Whether the vulnerability has been reproduced.
- Timing and distribution mechanism for any patches or fixed releases.
- Work-arounds to the problem for those that will be unwilling or unable to patch in a timely fashion.

Additionally, if a longer resolution period is necessary, then this should be communicated to the researcher. If the time frame is already 45 days from report, the researcher will be unlikely to grant an extension unless the vendor can clearly demonstrate to the researcher that the problem requires extensive changes, usually as the result of a fundamental design change. The vendor will also likely need to show that there are no adequate mitigating controls, which will generally require demonstrating why the researcher’s proposed work-arounds are inadequate.

Provide a security advisory and customer access to remediation

The vendor should provide its own security advisory of the issue, but may also choose only to endorse the researcher's advisory, after assuring that it contains adequate information for customers to protect themselves.

If the advisory only points to compensating controls, not an actual fix, it should provide a time line and distribution information for a permanent fix.

The advisory should also present an overview of the problem, denoting what resources are at risk, as well as information on how to assess whether an installation is at risk.