SEZG566/SSZG566

Secure Software Engineering

Security Requirements Engineering

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao

- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security Requirements Engineering

What is a requirement?

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is because requirements serve dual function

- May be the basis for a bid for a contract - therefore must be open to interpretation;
- May be the basis for the contract itself - therefore must be defined in detail;
- Both these statements may be called requirements.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

What is a "Requirement”

When the end result of the software development activity is a COTS (Commercial Off The Shelf) product, we term requirements as product specifications.

When the end result of the software development activity is to deliver the product to a single client in a project scenario, we use the term requirements

What is a "Requirement”

A *requirement is a need, expectation, constraint or interface of any stakeholders that must be fulfilled by the proposed software product during its development*

- **Need**—It is something basic without which the existence becomes untenable. It is the absolute minimum necessity if the system is to be useful
- **Expectation**—Expectation is an unstated need. It is expected that the development team brings expertise of software to bridge the gap in the needs stated by the user
- **Constraint**—It is a hurdle that the user has to live with
- **Interface**—It is the basis for interaction with the customers, suppliers, and peers of the user
- **Stakeholders**—A stakeholder is someone who is affected by the outcome of a endeavor, viz. end user, project team, marketing team to sell product, management

Importance of Requirements Engineering

Some studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once the system has become operational than if they were detected during requirements development.

According to Charette[2005], Requirements problems are among the top causes of the following undesirable phenomena

- Projects are significantly over budget, go past schedule, have significantly reduced scope, or are cancelled
- Development teams deliver poor-quality applications
- Products are not significantly used once delivered

Requirements Engineering Overview

Inception—ask a set of questions that establish …

- basic understanding of the problem
- the people who want a solution
- the nature of the solution that is desired, and
- the effectiveness of preliminary communication and collaboration between the customer and the developer

Elicitation—elicit requirements from all stakeholders

Elaboration—create an analysis model that identifies data, function and behavioral requirements

Negotiation—agree on a deliverable system that is realistic for developers and customers

Specification—can be any one (or more) of the following:

- A written document
- A set of graphical models
- A formal mathematical model
- A collection of user scenarios (use-cases)
- A prototype

Validation—a review mechanism that looks for

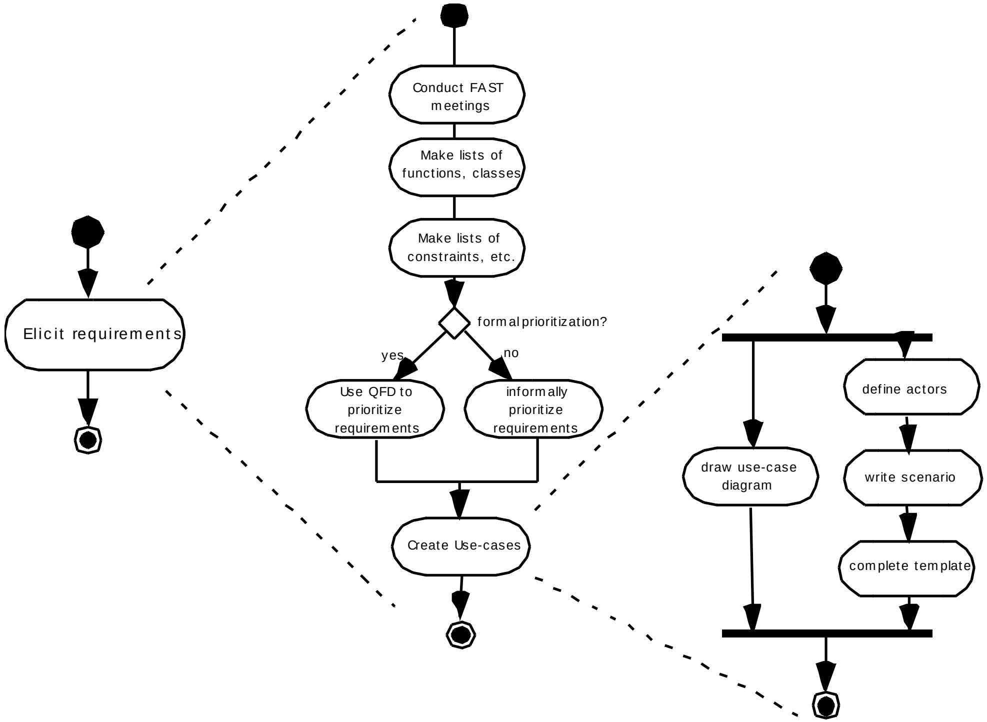
- errors in content or interpretation
- areas where clarification may be required
- missing information
- inconsistencies (a major problem when large products or systems are engineered)
- conflicting or unrealistic (unachievable) requirements.

Requirements management

Eliciting Requirements

- Meetings are conducted and attended by both software engineers and customers
- Rules for preparation and participation are established
- An agenda is suggested
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- The goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

Eliciting Requirements



Quality Function Deployment

-Dr Yoji Akao et al (1966)

QFD is a focused methodology for carefully listening to the voice of the customer and then effectively responding to those needs and expectations – American Society for Quality

QFD translates needs of customer into technical requirements for software

Function deployment determines the "value" (as perceived by the customer) of each function required of the system

A requirement may be

- Normal - Stated
- Expected - Implicit
- Exciting - Beyond the above two

Information deployment identifies data objects and events

Task deployment examines the behavior of the system

Value analysis determines the relative priority of requirements

Requirements Engineering Challenges

Requirements Engineering on individual projects often suffers from the following problems:

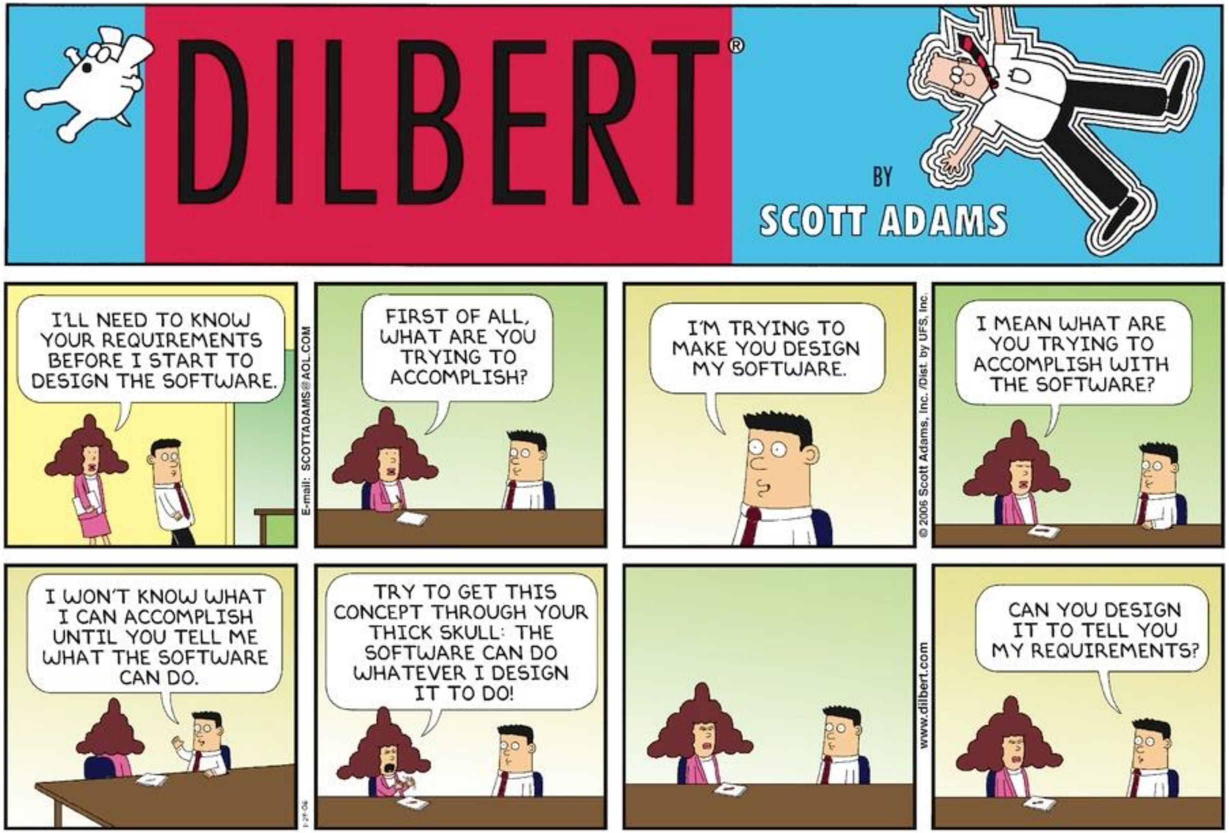
– Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.

– Requirements are often statements describing architectural constraints or implementation mechanisms rather than statements describing what the system must do.

– Requirements are often directly specified without any analysis or modeling. When analysis is done, it is usually restricted to functional end-user requirements, ignoring

1) quality requirements such as security,
2) other functional and nonfunctional requirements, and
3) architecture, design, implementation, and testing constraints.

Elicitation is not easy


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc. /Dist. by UFS, Inc.
www.dilbert.com

Requirements Engineering Challenges

Requirements specification is typically haphazard, with specified requirements being

- ambiguous,
- incomplete (e.g., nonfunctional requirements are often missing),
- inconsistent,
- not cohesive,
- infeasible,
- obsolete,
- neither testable nor capable of being validated, and
- not usable by all of their intended audiences.

Requirements management is typically weak, with ineffective forms of data capture

- e.g., in one or more documents (rather than in a database or tool) and missing attributes.
- often limited to tracing, scheduling, and prioritization, without change tracking or other configuration management.

Quality Requirements

Project teams often neglect *quality* requirements, such as performance, safety, security, reliability, and maintainability.

- Developers of certain kinds of mission-critical systems and systems in which human life is involved, such as the space shuttle, have long recognized the importance of quality requirements and have accounted for them in software development.
- In many other systems, however, quality requirements are treated in an inadequate way. Hence we see the failure of software associated with power systems, telephone systems, unmanned spacecraft, and so on.

This inattention to quality requirements is exacerbated by the desire to keep costs down and meet aggressive schedules.

Security Requirements Engineering

According to BSI[09], if security requirements are not effectively defined, the resulting system cannot be ***evaluated*** for success or failure prior to its implementation

Operational environments and business goals often change ***dynamically***, with the result that security requirements development is not a one-time activity.

Requirements engineering research and practice pay a lot of attention to the functionality of the system from the user's perspective, but little attention is devoted to what the system should ***not*** do [Bishop 2002]

Security Requirements Engineering

Users have implicit assumptions for the software applications and systems to be secure and are surprised when they are not. These user assumptions need to be translated into security requirements for the software systems when they are under development.

It is important for requirements engineers to think about the attacker's perspective and not just the functionality of the system from the end-user's perspective.

- An attacker is not particularly interested in functional features of the system, unless they provide an avenue for attack. Instead, the attacker typically looks for defects and other conditions outside the norm that will allow a successful intrusion to take place.

Security Is Not a Set of Features

Security features such as password protection, firewalls, virus detection tools etc. are, in fact, not security requirements.

They are rather implementation mechanisms that are intended to satisfy unstated requirements, such as authenticated access

A systematic approach to security requirements engineering will help avoid the problem of generic lists of features and take into account the attacker's perspective.

No convenient security pull-down menu that will let you select "security" and do the needful.

Security Is Not a Set of Features

Security is an ***emergent*** property of a system, not a feature

Because security is not a feature, it cannot be bolted on after other software features are codified, nor can it be ***patched*** in after attacks have occurred in the field. Instead, security must be built into the product from the ground up

Most cost-effective approach to software security incorporates thinking beyond normative features and maintains that thinking throughout the development process

Every time a new requirement, feature, or use case is created, the developer or security specialist should spend some time thinking about how that feature might be unintentionally misused or intentionally abused

Thinking Beyond Normal

When we design and analyze a system, we’re in a great position to know our systems better than potential attackers do.

We can leverage this knowledge to the benefit of security and reliability, by asking and answering the critical questions:

- Which assumptions are implicit in our system?
- Which kinds of things make our assumptions false?
- Which kinds of attack patterns will an attacker bring to bear?

Thinking like an attacker

System’s creators are not the best security analysts of that system.

- ‘Thinking like an attacker’ is extremely difficult for those who have built up a set of implicit assumptions

System Creators make excellent subject matter experts (SMEs).

- Together, SMEs and security analysts can ferret out base assumptions in a system under analysis and think through the ways an attacker will approach the software

Creating Useful Misuse Cases

Misuse cases is to decide and document *a priori* how software should react to illegitimate use

Unlike the functional requirements, designers/developers play the role of user and explain design and underlying assumptions to security expert documents

To guide brainstorming, software security experts ask many questions of a system's designers to help identify the places where the system is likely to have weaknesses. This activity mirrors the way attackers think.

The brainstorming covers user interfaces, environmental factors, and events that developers assume a person can't or won't do

- "Users can't enter more than 50 characters because the JavaScript code won't let them"
- "Users don't understand the format of the cached data, so they can't modify it."

Misuse vs. Abuse

According to Chun Wei,

Misuse cases are defined as "behavior that the system/entity owner does not want to occur"

- An interaction results in a session key being revealed to an actor who should not see the session key

Abuse cases are defined as "… where the results of the interaction are harmful to the system …"

- the actor posts the session key on a public website, then an abuse case takes place

But some authors do not distinguish

Chun Wei, Misuse Cases and Abuse Cases in Eliciting Security Requirements

Specifying Abuse Cases

The process of specifying abuse cases makes a designer very clearly differentiate appropriate use from inappropriate use.

The security expert/designer must ask the right questions:

- How can the system distinguish between good input and bad input?
- Can it tell whether a request is coming from a legitimate application or from a rogue application replaying traffic?
- where might a bad guy be positioned? On the wire? At a workstation? In the back office?
- Any communication line between two endpoints or two components is a place where an attacker might try to interpose himself or herself
- what can this attacker do in the system? Watch communications traffic? Modify and replay such traffic? Read files stored on the workstation? Change registry keys or configuration files? Be the DLL?

Trying to answer such questions helps software designers explicitly question design and architecture assumptions, and it puts the designer squarely ahead of the attacker by identifying and fixing a problem before it's ever created.

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

CMU SQUARE Process Model

SQUARE Process Model

Security Quality Requirements Engineering (SQUARE) is a process model that was developed at Carnegie Mellon University [Mead 2005].

SQUARE provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications.

The focus of the model is to build security concepts into the early stages of the SDLC. It can also be used for documenting and analyzing the security aspects of systems once they are implemented in the field and for steering future improvements and modifications to those systems.

The SQUARE work is supported by the Army Research Office through grant ("Perpetually Available and Secure Information Systems") to Carnegie Mellon University's CyLab.

SQUARE Process Steps

1. Agree on definitions
2. Identify security goals
3. Develop artifacts to support security requirements definition
4. Perform (security) risk assessment
5. Select elicitation techniques
6. Elicit security requirements
7. Categorize requirements as to level (e.g., system, software) and whether they are requirements or other kinds of constraints
8. Prioritize requirements
9. Inspect requirements

SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineers	Agreed-to definitions
2	Identify security goals	Definitions, candidate goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineers	Goals
3	Develop artifacts to support security requirements definition	Potential artifacts list (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineers	Needed artifacts: scenarios, misuse cases, models, templates, forms

SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
4	Perform (security) risk assessment	Misuse cases, scenarios, security goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including threat analysis	Requirements engineers, risk expert, stakeholders	Risk assessment results
5	Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost–benefit analysis	Work session	Requirements engineers	Selected elicitation techniques

SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
6	Elicit security requirements	Artifacts, risk assessment results, selected techniques	QFD, Joint Application Development, interviews, surveys, model based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirements engineers	Initial cut at security requirements
7	Categorize requirements as to level (e.g., system, s/w) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineers, other specialists as needed	Categorized requirements

SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
8	Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as Analytical Hierarchy Process (AHP - structured technique for organizing information) etc.	Stakeholders facilitated by requirements engineers	Prioritized requirements
9	Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan (formal approach with exit criteria) and peer reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

CMU SQUARE Work Products

Sample: Identify Security Goals

Work with the client to identify security goals that mapped to the company’s overall business goals.

Consider Asset Management System (AMS) of Acme Co.

Business goal of AMS: To provide an application that supports asset management and planning.

Security goals: Three high-level security goals were derived for the system:

- Management shall exercise effective control over the system’s configuration and use.
- The confidentiality, accuracy, and integrity of the AMS shall be maintained.
- The AMS shall be available for use when needed.

Attack Patterns

Attack patterns are descriptions of common methods for exploiting software. Act as a mechanism to capture and communicate the attacker’s perspective.

They derive from the concept of design patterns [Gamma 95] applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples

The following typical information is captured for each attack pattern:

- Pattern name and classification
- Attack prerequisites
- Description
- Targeted vulnerabilities or weaknesses
- Method of attack
- Attacker goal
- Attacker skill level required
- Resources required
- Blocking solutions
- Context description
- References

Attack Patterns

- **Pattern Name and Classification**: A unique, descriptive identifier for the pattern.
- **Attack Prerequisites**: What conditions must exist or what functionality and what characteristics must the target software have, or what behavior must it exhibit, for this attack to succeed?
- **Description**: A description of the attack including the chain of actions taken.
- **Related Vulnerabilities or Weaknesses**: What specific vulnerabilities or weaknesses does this attack leverage?
- **Method of Attack**: What is the vector of attack used (e.g., malicious data entry, maliciously crafted file, protocol corruption etc.)?

Attack Patterns

- **Attack Motivation-Consequences**: What is the attacker trying to achieve by using this attack?
- **Attacker Skill or Knowledge Required**: What level of skill or specific knowledge must the attacker have to execute such an attack?
- **Resources Required**: What resources (e.g., CPU cycles, IP addresses, tools, time) are required to execute the attack?
- **Solutions and Mitigations**: What actions or approaches are recommended to mitigate this attack, either through resistance or through resiliency?
- **Context Description**: In what technical contexts (e.g., platform, OS, language, architectural paradigm) is this pattern relevant? This information is useful for selecting a set of attack patterns that are appropriate for a given context.
- **References**: What further sources of information are available to describe this attack?

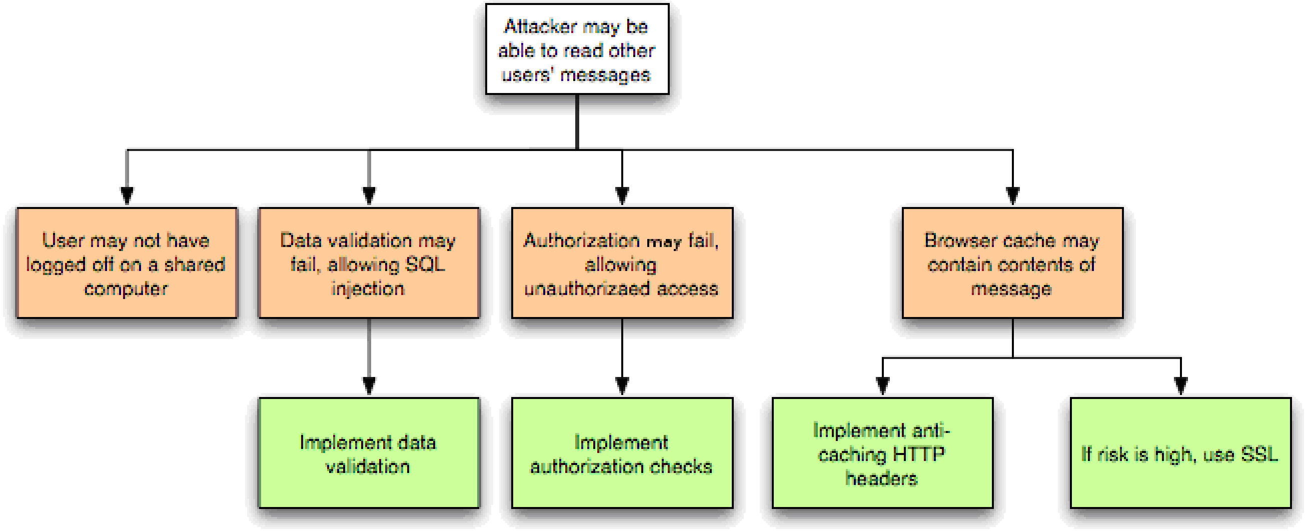
Attack Pattern Example

- **Pattern name and classification**: Shell Command Injection—Command Delimiters.
- **Attack Prerequisites**: The application must pass user input directly into a shell command.
- **Description**: Using the semicolon or other off-nominal characters, multiple commands can be strung together. Unsuspecting target programs will execute all the commands. An example may be when authenticating a user using a web form, where the username is passed directly to the shell as in: exec("cat data_log_" + userInput + ".dat").
  - The "+" sign denotes concatenation. The developer expects that the user will only provide a username. However, a malicious user could supply "username.dat; rm –rf / ;" as the input to execute the malicious commands on the machine running the target software. In the above case, the actual commands passed to the shell will be: cat data_log_username.dat; rm –rf /; .dat
  - The first command may or may not succeed; the second command will delete everything on the file system to which the application has access, and success/failure of the last command is irrelevant.

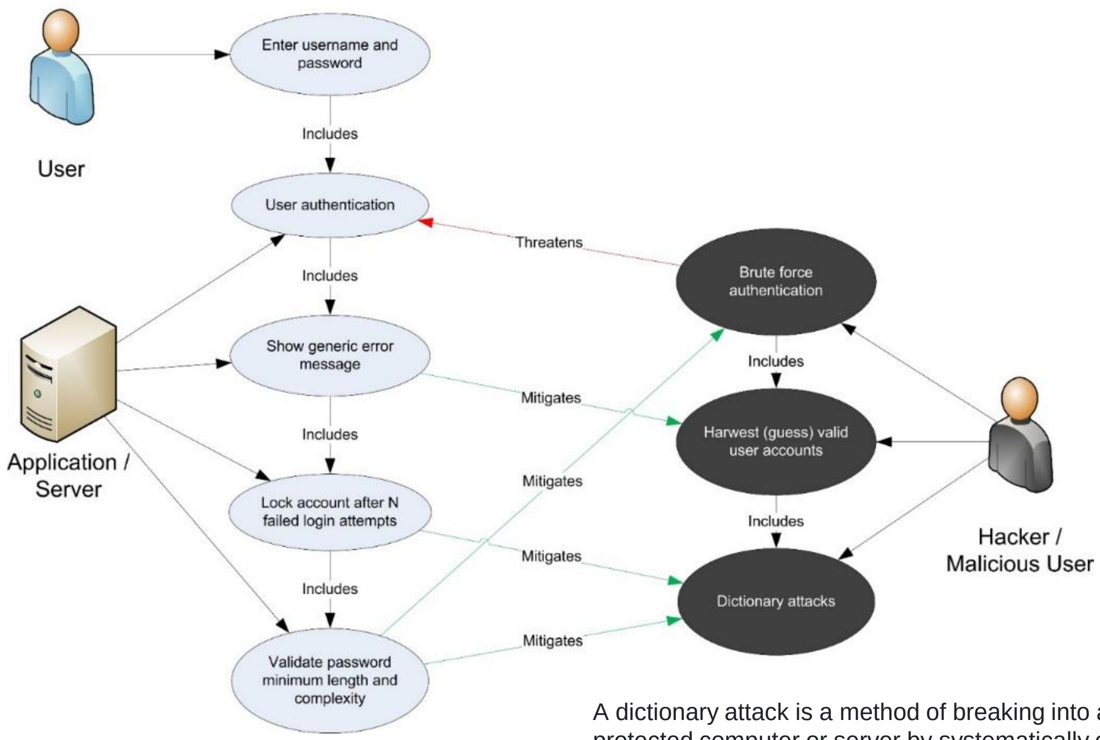
Attack Pattern Example

- **Related Vulnerabilities or Weaknesses**: : CWE-OS Command Injection, CVE-1999-0043, CVE-1999-0067, CVE-1999-0097, CVE-1999-0152, CVE-1999-0210, CVE-1999-0260, 1999-0262, CVE-1999-0279, CVE-1999-0365, etc.
- **Method of Attack**: By injecting other shell commands into other data that are passed directly into a shell command.
- **Attack Motivation-Consequences**: Execution of arbitrary code.
- **Attacker Skill or Knowledge Required**: Finding and exploiting this vulnerability does not require much skill.
- **Resources Required**: No special or extensive resources are required for this attack.
- **Solutions and Mitigations**: Define valid inputs to all fields and ensure that the user input is always valid. Also perform white-list and/or black-list filtering as a backup to filter out known command delimiters.
- **Context Description**: OS: UNIX.
- **References**: Exploiting Software [Hoglund 04].

A Threat Tree Example

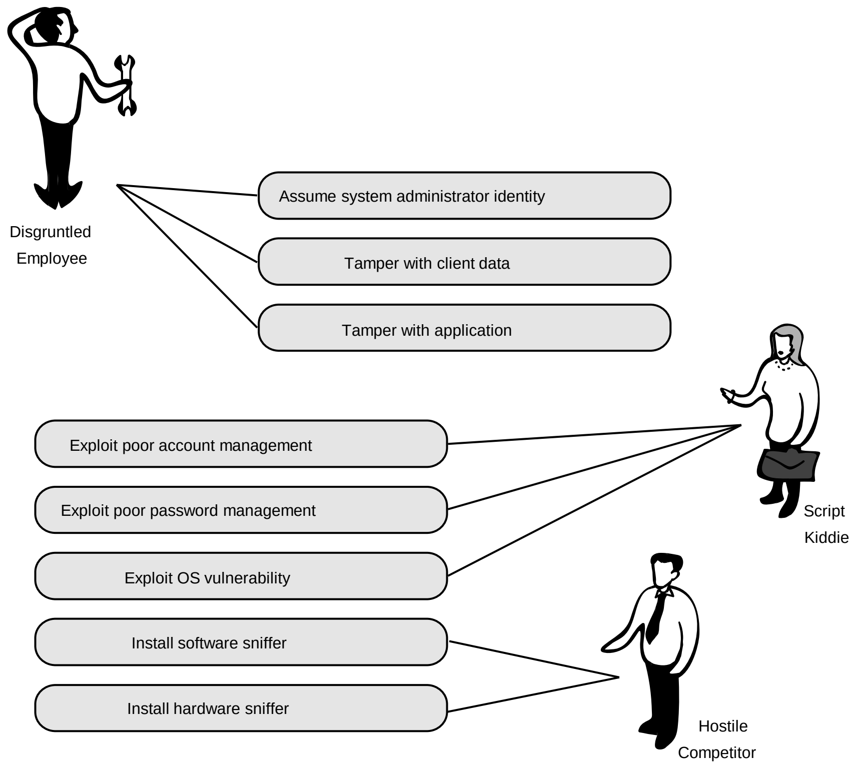

Attacker may be able to read other users' messages
User may not have logged off on a shared computer
Data validation may fail, allowing SQL injection
Authorization may fail, allowing unauthorizaed access
Browser cache may contain contents of message
Implement data validation
Implement authorization checks
Implement anti-caching HTTP headers
If risk is high, use SSL

Use Case with Abuse Case


User
Enter username and password
Includes
User authentication
Threatens
Includes
Show generic error message
Mitigates
Includes
Lock account after N failed login attempts
Mitigates
Includes
Validate password minimum length and complexity
Mitigates
Mitigates
Application / Server
Brute force authentication
Includes
Harwest (guess) valid user accounts
Includes
Dictionary attacks
Hacker / Malicious User

A dictionary attack is a method of breaking into a password-protected computer or server by systematically entering every word in a dictionary as a password

Abuse case example



Sample: Perform Risk Assessment

There are two essential assets in this system.

- The first is the Windows Server computer, which houses the majority of the production system’s intellectual assets (that is, the code that runs the system). This computer acts as a server that allows remote users to access the Asset Management System.

- The second essential asset is the information inside the Windows Server computer—specifically, the files stored in the Microsoft IIS server and the information stored in the Sybase database and MapGuide database are critical for making informed decisions. If this information is lost or compromised, the ability to make accurate decisions is lost.

Elicitation Methods

- Misuse/Abuse cases:
  - Misuse/abuse cases apply the concept of a negative scenario—that is, a situation that the system's owner does not want to occur—in a use-case context. Business leaders, military planners, and game players are familiar with the strategy of analyzing their opponents' best moves as identifiable threats
- QFD
  - As per Dr. Yoji Akao, who originally developed Quality Function Deployment (QFD) in Japan in 1966, it is a "method to transform qualitative user demands into quantitative parameters, to deploy the functions forming quality, and to deploy methods for achieving the design quality into subsystems and component parts, and ultimately to specific elements of the process"
- Joint Application Development (JAD)
  - The JAD methodology [Wood 1995] involves all stakeholders via highly structured and focused meetings. In the preliminary phases of JAD, the requirements engineering team is charged with fact-finding and information-gathering tasks. Typically, the outputs of this phase, as applied to security requirements elicitation, are security goals and artifacts. The actual JAD session is then used to validate this information by establishing an agreed-on set of security requirements for the product.

Sample: Elicit and Categorize Security Requirements

Security requirements are identified and then organized to map to the high-level security goals (from Step 2).

Examples include :

- Requirement 1: The system is required to have strong authentication measures in place at all system gateways and entrance points (maps to Goals 1 and 2).
- Requirement 2: The system is required to have sufficient means to govern which system elements (e.g., data, functionality) users can view, modify, and/or interact with (maps to Goals 1 and 2).
- Requirement 3: A continuity of operations plan (COOP) is required to assure system availability (maps to Goal 3).

Incorporating SQUARE in SDLC

- All nine steps of SQUARE fall under the requirements analysis and specification phase.
- The software requirements specification (SRS) should accommodate the outcome of the first eight SQUARE steps
  - The SRS must clearly specify the security definitions agreed on (Step 1). It is necessary to document security goals (Step 2) along with the project goals and constraints. Develop artifacts (Step 3) such as misuse cases and scenarios to support security requirements definition
  - Categorize the security requirements (Step 7) and prioritize them (Step 8) along with documented functional requirements. (For clarity, it is preferable to separate security requirements from functional requirements.)

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

OWASP Recommendations

OWASP SAMM

SAMM (Security Assessment Maturity Model) divides activities associated with software development into 5 business functions for incorporating security

- Governance
  - Includes strategy, metrics, policy, compliance, education and guidance
- Design
  - Includes threat assessment, security requirements, and secure architecture
- Implementation
  - Includes secure build, secure deployment, and defect management
- Verification
  - Includes design review, implementation review, and security testing
- Operations
  - Includes issue management, environment hardening, operational enablement

SAMM Security Requirements

SAMM expects the following as part of security requirements:

–Consider security explicitly during the software requirements process

–Increase granularity of security requirements derived from business logic and known risks.

–Mandate security requirements process for all software projects and third-party dependencies.

Consider security explicitly – SR1
(during the software requirements process)

Goal	Activities	Quality Criteria
Consider security explicitly during the software requirements process.	Identify security requirements	• Teams derive security requirements from functional requirements and customer or organization concerns • Security requirements are specific, measurable, and reasonable • Security requirements are in line with the organizational baseline
	Perform vendor assessments	• Consider including specific security requirements, activities, and processes when creating third-party agreements • A vendor questionnaire is available and used to assess the strengths and weaknesses of your suppliers

Increase granularity – SR2

(of security requirements derived from business logic and known risks)

Goal	Activities	Quality Criteria
Increase granularity of security requirements derived from business logic and known risks.	Standardize and integrate security requirements	• Security requirements take into consideration domain specific knowledge when applying policies and guidance to product development • Domain experts are involved in the requirements definition process • You have an agreed upon structured notation for security requirements • Development teams have a security champion dedicated to reviewing security requirements and outcomes
	Discuss security responsibilities with suppliers	• You discuss security requirements with the vendor when creating vendor agreements • Vendor agreements provide specific guidance on security defect remediation within an agreed upon timeframe • The organization has a templated agreement of responsibilities and service levels for key vendor security processes • You measure key performance indicators

Mandate security requirements process – SR3
(for all software projects and third-party dependencies)

Goal	Activities	Quality Criteria
Mandate security requirements process for all software projects and third-party dependencies.	Develop a security requirements framework	• A security requirements framework is available for project teams • The framework is categorized by common requirements and standards-based requirements • The framework gives clear guidance on the quality of requirements and how to describe them • The framework is adaptable to specific business requirements
	Align security methodology with suppliers	• The vendor has a secure SDLC that includes secure build, secure deployment, defect management, and incident management that align with those used in your organization • You verify the solution meets quality and security objectives before every major release • When standard verification processes are not available, you use compensating controls such as software composition analysis and independent penetration testing

Assessment Matrix for Security Requirements

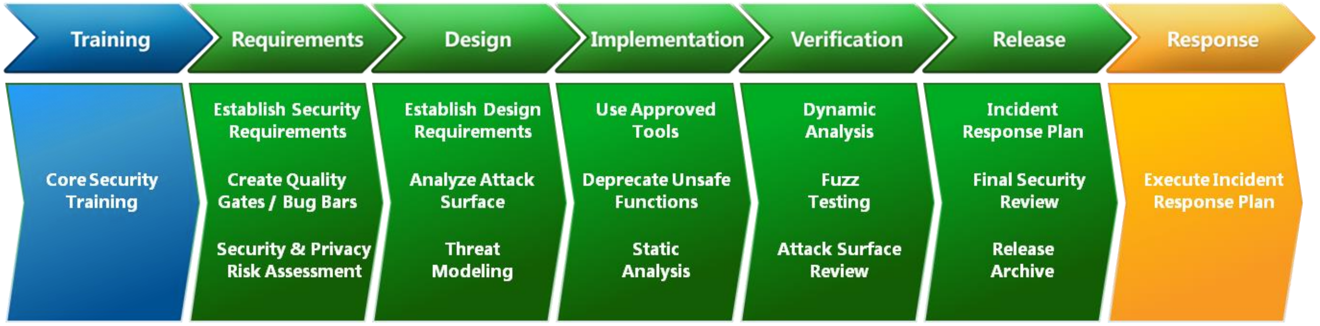
Score ->	0.0	0.2	0.5	1.0
Do project teams specify security requirements during development?	No	Yes, for some applications	Yes, for at least half of the applications	Yes, for most or all of the applications
Do stakeholders review vendor collaborations for security requirements and methodology?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do you define, structure, and include prioritization in the artifacts of the security requirements gathering process?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do vendors meet the security responsibilities and quality measures of service level agreements defined by the organization?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do you use a standard requirements framework to streamline the elicitation of security requirements?	No	Yes, for some applications	Yes, for at least half of the applications	Yes, for most or all of the applications
Are vendors aligned with standard security controls and software development tools and processes that the organization utilizes?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

SDL Recommendations

Security Development Lifecycle (SDL)



SDL Requirements Phase

The Requirements phase of the SDL includes

- Project Inception—when you consider security and privacy at a foundational level
  - Development teams identify key objectives and integrate security and privacy to minimizes disruption to plans and schedules
- Cost analysis—when you determine if development and support costs for improving security and privacy are consistent with business needs.

SDL Requirements Practices

The major SDL practices during requirements analysis are

– Establish Security & Privacy Requirements

– Create Quality Gates/Bug Bars

– Create Security/Privacy Risk Assessments

Establish Security & Privacy Requirements

Define and integrate security and privacy requirements early

- identify key milestones and deliverables and minimize disruptions to plans and schedules.

Security and privacy analysis including

- assigning security experts,
- defining minimum security and privacy criteria for an application, and
- deploying a security vulnerability/work item tracking system.

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

Create Quality Gates/Bug Bars

Defining minimum acceptable levels of security and privacy quality

- the team understand risks associated with security issues,
- Team identifies and fixes security bugs during development, and
- Team apply the standards throughout the entire project.

Set a bug bar to clearly define the severity thresholds of security vulnerabilities

- (for example, no known vulnerabilities in the application with a “critical” or “important” rating at time of release)

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

Security/Privacy Risk Assessments

Identify portions of a project requiring threat modeling and security design reviews before release

Determine the Privacy Impact Rating of a feature, product, or service

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

SDL - Agile

- SDL was developed by Microsoft for securing very large products, such as Windows and Microsoft Office
- The SDL team at Microsoft developed an approach that melds agile methods and security—the Security Development Lifecycle for Agile Development (SDL-Agile)
- There is a need to reconcile
  - Agile release cycles are short, there isn't enough time for teams to complete all of the SDL requirements for every release
  - There are serious security issues that the SDL is designed to address, and these issues simply can't be ignored for any release

SDL-Agile Requirements

- SDL requirements are represented as tasks and added to the product and sprint backlogs
- SDL tasks are added to the backlog as non-functional stories
- SDL-Agile places each SDL requirement into one of three categories defined by frequency of completion
  - Every-sprint requirements - that are so essential to security that no software should ever be released without these requirements being met
  - Bucket Requirements - tasks that must be performed on a regular basis over the lifetime of the project but that are not so critical as to be mandated for each sprint
  - One-Time Requirements - that need to be met when starting a new project with SDL-Agile

Every-Sprint Requirements

- SDL requirements that are so essential to security that no software should ever be released without these requirements being met
- Examples are:
  - Run analysis tools daily or per build
  - Threat model all new features
  - Ensure that each project member has completed at least one security training course

Appendix P of Microsoft-SDL V5.2

Bucket Requirements

- Consists of tasks that must be performed on a regular basis over the lifetime of the project but that are not so critical as to be mandated for each sprint
- The table below contains a sampling of the tasks

Verification Tasks	Design Review	Planning
ActiveX fuzzing	Conduct a privacy review	Create privacy support documents
Attack surface analysis	Review crypto design	Update security response contacts
Binary analysis (BinScope)	Assembly naming and APTCA	Update network down plan
File fuzz testing	User Account Control	Define/update security bug bar

- Appendix Q of Microsoft-SDL V5.2

One-Time Requirements

- SDL requirements that need to be met when you start a new project with SDL-Agile
- SDL-Agile allows a grace period to complete each one-time requirement. The period generally ranges from one month to one year after the start of the project
- For example,
  - Choosing a security advisor has a one-month completion deadline,
  - Updating your project to use the latest version of the compiler is considered a potentially long, difficult task and has a one-year completion deadline

- Appendix R of Microsoft-SDL V5.2

Software Security Engineering, Julia H. Allen, et al, Pearson, 2008.
Security in Computing by Charles P. Pfleeger, Shari L. Pfleeger, and Deven Shah Pearson Education 2009
Computer Security: Principles and Practice by William Stallings, and Lawrie Brown Pearson, 2008.
www.owasp.com
www.microsoft.com

Thank You!