

<https://github.com/christopherchristensen>

12. April 2019

Inhaltsverzeichnis

Security Requirements	3
Security	3
Safety	3
Important Notes about Security	3
Security vs. Safety	3
Perimeter Security	3
Why is Perimeter Security not sufficient anymore?	3
Why Perimeter Security often focused on before Insecurities in Software?	3
How to Intrusion happens	4
Malware	4
Infection	4
Exploit	4
Vulnerabilities	4
Example of an Indirect Error	4
Can Vulnerabilities and Exploits be prevented?	5
Ways to improve Security	5
Security Cost Considerations	5
Attributes of Software Security	5
What does Software Engineering deal with?	5
Scales of Software Project	5
Attributes of Software Quality	6
Software Quality	6
Development- vs. Maintenance-Costs	6
Where do bugs come from?	6
How much do Bugs cost in Relation to their Source?	6
Process Models	6
Different Process Models	7
Waterfall	7
Prototyping	7
Unified Process	7
Agile Development	8
SCRUM	8
DevOps	8
DREAD	8

DREAD: Damage (Potential)	8
DREAD: Reproducibility	9
DREAD: Exploitability	9
DREAD: Affected Users	9
DREAD: Discoverability	10
Why don't all organizations include Discoverability in DREAD?	10
DREAD Rating Calculation	10
Possible Exam Question to DREAD	10
OWASP Risk Rating	10
Attack Tree	11
Attack Tree - Node Values	11
Attack Tree - Node Value Calculation	11
Attack Tree - Countermeasures	11
UML	11
9 UML Diagrams	12
Use Case Diagram	12
4 Elements of UCD	12
Use Case	12
Actor	12
User Cases vs. User-Stories	12
Analysis with Use Cases	13
Context Model	13
Analysis with Use Cases including Security	13
Misuse Cases	13
Misuse Case Description Elements	13
Why are Systems Vulnerable?	13
Goals when securing Interfaces	14
Communication Model by Shannon	14
Problem with Communication Model by Shannon	14
How can Attacker exploit the above Problem?	14
Typical Exploit-Techniques used by Malware	14
Means to mitigate Exploits	15

Security Requirements

Security

- Defense of computers against
 - intrusion
 - unauthorized use of resources
 - humans with malicious or criminal intent

Safety

- Avoiding hazardous situations
- Alerting the correct systems if situation becomes unsafe
- Defending humans / environment against malfunctioning systems

Important Notes about Security

- Security
 - is not a programming problem only
 - must be considered from the beginning on
 - is relevant in all phases of software development
 - can't be fixed by using special methods / tools
 - can't be seen isolated from deployment / operations

Security vs. Safety

- No strict borderline between both concepts
- Security breaches may have serious safety consequences

Perimeter Security

- Measures to protect external perimeter of a system
 - Access Control
 - Firewalls
 - Anti Virus Software
- Not sufficient by today's standards (medieval concept of yesterday)

Why is Perimeter Security not sufficient anymore?

- More applications offer direct access to external users
- Modern applications often considered to be doors into OS (~ wall)

Why Perimeter Security often focused on before Insecurities in Software?

Bad Practice

- Secure coding often not considered to be important in the past
- Often, security professionals \neq software developers (full insight into software vulnerabilities missing)
- Functionality often focus of software developers
- Vendors want quick time to market, often no time for proper security architecture, design and testing

- Customers used to flaws and regular security updates (patches)
- Customers can't control software flaws, therefore depend upon perimeter protection

How to Intrusion happens

1. Infection
2. Exploit
3. Payload

Malware

Malicious Software

- Software doing something the user / owner of the computer doesn't like
- Special forms of malware
 - Virus
 - Worm
 - Trojan

Infection

- How malware gets into system
- When user trusts a piece of software (with malware)

Exploit

“to use something to one's own advantage”

- Taking advantage of weak points in systems / applications
- Piece of software, chunk of data, or a sequence of commands
 - that takes advantage of a bug / vulnerability
 - to cause unintended / unanticipated behavior on computer software, hardware, or something electronic

Vulnerabilities

- Programming errors (direct or indirect)
- Vulnerabilities lead to exploits
- Attacker must have applicable tool / technique that can connect to a system weakness (attack surface)

Example of an Indirect Error

- Flawed handling of memory management
 - may render a system vulnerable to denial-of-service attacks
 - doesn't effect system's functionality therefore may remain undetected during testing

Can Vulnerabilities and Exploits be prevented?

- Every system of minimum complexity contains errors
- No way of completely preventing it
- Relative security can be reached by
 - clean and intelligent software development
 - including security considerations in all phases of a system

Ways to improve Security

- Security and functionality need to be designed and integrated into the individual phases of a development life cycle
- Security considerations need to be interlaced into product core
- Understand security needs, implement the right controls and test thoroughly

Security Cost Considerations

- The higher the security efforts, the lower the costs for security breaches
- The higher the security efforts, the higher the costs for security measures
- With the right amount of security efforts the total costs can be kept at a minimum

Attributes of Software Security

- Quality
- Correctness
- Availability
- Robustness
- Verifiability
- Reliability
- Safety

What does Software Engineering deal with?

- Cost-effective development of high-quality software
- ... of software systems
 - Construction...
 - Control...
 - Rollout...
 - Operation and maintenance...

Scales of Software Project

- Trivial
 - Staff: 1, 4-6 Weeks, Lines of code < 500
 - e.g. Simple administrative programs
- Small
 - Staff: 1, 1-6 Months, Lines of code: 1'000-2'000
 - e.g. Small commercial applications
- Medium
 - Staff: 2-5, 1-2 Years, Lines of code: 10'000-50'000

- e.g. Warehouse management
- Big
 - Staff: 5-20, 2-3 Years, Lines of code: 50'000-100'000
 - e.g. Small operating system
- Very Big
 - Staff: 100-1'000, 4-5 Years, Lines of code: ca. 1'000'000
 - e.g. Database system
- Extremely Big
 - Staff: 2'000-5'000, 5-10 years, Lines of code > 10'000'000
 - e.g. Air traffic control

Attributes of Software Quality

- Correctness / Verifiability
- Reliability
- Robustness
- User-Friendliness
- Maintainability
- High Performant
- Portability / Compatability
- Reusability

Software Quality

- Features of software necessary for meeting requirements
- Features of security and safety (partly competing)
- Importance of features dependant on project

Development- vs. Maintenance-Costs

- Cost-effective = ca. 20% Dev / 80% Maintenance

Where do bugs come from?

- 56% Analysis and Specification
- 27% Design
- 7% Coding
- 10% Other

How much do Bugs cost in Relation to their Source?

- 82% Analysis and Specification
- 13% Design
- 1% Coding
- 4% Other

Process Models

Description of a process

- Activities must be
 - defined exactly
 - arranged along a time scale
- Process model is the starting point for
 - project planning
 - project management
- Process model must be adapted to
 - the project
 - the development environment (personnel / tools)

Different Process Models

- Waterfall
- Prototyping
- Unified Process
- Agile Development
- SCRUM
- → Devops

Waterfall

- Strict linear sequence of activities
- Simple definition of milestones
- Relatively easy project management
- Little freedom for developers
- Problems
 - Not flexible

Prototyping

- Iterative
- Flexible reactions to user requests possible
- More freedom for developers
- Problems
 - Hard to manage (hard to define milestones)
 - “quick-and-dirty” solutions stay in product forever

Unified Process

- Compromise between Waterfall and Prototyping
- Iterative
- Flexible reaction to user requests possible
- Little freedom for developers
- Function-Oriented
- Problems
 - Often too slow, complicated, inefficient (cumbersome)

Agile Development

- Agile Values
 - Individuals / Interactions > Processes / Tools
 - Working Programs > Elaborate Documentation
 - Constant Cooperations with Customer > Contracts
 - Courage / Openness for Change > Obeying fixed plans
- Agile Principles
 - Reuse existing resources
 - KISS
 - Functional / Customer-Oriented
 - Collective Code Ownership
- Agile Methods
 - Pair Programming
 - Testdriven Programming
 - Refactoring
 - ...

SCRUM

- Process Model for Agile Methods
- Time-Oriented

DevOps

- Build
- Deploy
- Test
- Release
- Build

[Content missing here]...

DREAD

Methodology for risk rating

- Damage
- Reproducibility
- Exploitability
- Affected Users
- Discoverability

DREAD: Damage (Potential)

- How bad would an attack be?
- Rating High (3):
 - Attacker can undermine security system

- Get full trust authorization
- Run as admin
- Upload content
- Rating Medium (2):
 - Lead sensitive information
- Rating Low (1):
 - Lead trivial information

DREAD: Reproducibility

- How easy to reproduce attack?
- Rating High (3):
 - Attack reproducible every time
 - Attack doesn't require timing window
- Rating Medium (2):
 - Attack reproducible
 - Only within a timing window and partial race situation
- Rating Low (1):
 - Attack difficult to reproduce
 - Even with knowledge of security hole

DREAD: Exploitability

- How much work to launch attack?
- Rating High (3):
 - Novice programmer could make attack in short time
- Rating Medium (2):
 - Skilled programmer could make attack, then repeat steps
- Rating Low (1):
 - Attack requires extremely skilled person
 - Requires in-depth knowledge every time

DREAD: Affected Users

- How many people will be impacted?
- Rating High (3):
 - All users
 - Default configuration
 - Key customers
- Rating Medium (2):
 - Some users
 - Non-default configuration
- Rating Low (1):

- Very small percentage of users
- Obscure feature
- Affects anonymous users

DREAD: Discoverability

- How easy is it to discover threat?
- Rating High (3):
 - Published info explains attack
 - Vulnerability found in most commonly used feature
 - Very noticeable
- Rating Medium (2):
 - Vulnerability in seldom-used part of product
 - Only some user should come across it
 - Requires some thinking to see malicious use
- Rating Low (1):
 - Bug is obscure
 - Unlikely that users will find damage potential

Why don't all organizations include Discoverability in DREAD?

DREAD-D, “DREAD minus D”

- They feel Discoverability rewards “Security through Obscurity”
- They then ignore it or always assume Discoverability at maximum rating (3)

DREAD Rating Calculation

- After rating each Bug or Threat
- Put all into rows of table and sum up DREAD ratings

Possible Exam Question to DREAD

- Threat given
- Attacker Profile given (Skill, Motivation, ...)
- Rate the Threat

OWASP Risk Rating

Part of Threat Modelling

- Threat Agent
- Vulnerability
- Technical Impact
- Business Impact

Attack Tree

Part of Threat Modelling

- Conceptual diagrams showing how an asset, or target, might be attacked
- Nodes combined by AND / OR
 - OR: Represents different ways to achieve same goal
 - AND: Represents different steps in achieving goal
- Costs assigned to activities

Attack Tree - Node Values

- Values can be assigned to leaf nodes
 - Boolean
 - Cost to attack
 - Cost to defend
 - Time to achieve
 - Time to repulse
 - Probability of success of attack
 - Likelihood that attacker will try attack

Attack Tree - Node Value Calculation

- AND-Nodes: Values of subnodes must be added
- OR-Nodes: Take one (the minimum) value of subnodes

Attack Tree - Countermeasures

- Add further nodes with countermeasures

UML

Unified Modeling Language

- Modeling requirements during system analysis
- Standard language for...
 - specifying,
 - visualising,
 - constructing,
 - documenting
- ...software systems

9 UML Diagrams

- (Use case diagram)
- Class diagram
- Object diagram
- State diagram
- Activity diagram
- Sequence diagram
- Collaboration diagram
- Component diagram
- Deployment diagram
- Timing diagram

Use Case Diagram

UCD

- To identify primary elements (actors) and processes (use cases) of system
- Shows which actors interact with each use case
- Actions described from actor's view

4 Elements of UCD

- Actors (elements that interact with system)
- System
- Use Cases (services that system performs)
- Lines (relationships)

Use Case

- Distinct business functionality or behaviour of system
- Sequence of related activities done by actor together with system
- Represents some action the user might perform to complete a task

Actor

- External interface of system
- Any entity that performs certain roles in system
- Activates flow of actions in system and influences them

User Cases vs. User-Stories

- User-Stories
 - common in Agile Dev. / Scrum
 - used for planning Use Cases
 - cut into pieces until they are implemented in one iteration
- Use Cases
 - bigger, more context information
 - define and document functionality

Analysis with Use Cases

- Draw a context model
- Write a glossary
- Detail the context model
- Define scenarios with sequence diagrams
- Use activity diagrams to generalize

Context Model

- Draw one Use Case for whole system
- Includes all actors that interact with system
- For each actor all functions shown

Analysis with Use Cases including Security

- (Steps above)
- Add Misuse Cases

Misuse Cases

- Based on Use-Case terminology
- Defined and completed by Mitigation Cases
- Process
 - Insert Misuse Cases
 - Describe Misuse Cases
 - Find Mitigation Use Case

Misuse Case Description Elements

- Misuse Case Name
- Misuser Profile
- Description
- Basic Path
- Alternative Paths
- Triggers
- Assumptions
- Mitigation

Why are Systems Vulnerable?

Interfaces as Special Problems

- Systems consist of components
 - developed independently
 - have input data
 - exchange data
- Components need interfaces
 - rules for input data define a language

- language must be defined exactly
- language must be interpreted exactly in same way on both sides of interface

[missing content here ... (interface types)]

Goals when securing Interfaces

- Secure Input Validation
 - Accept only valid input
 - Always interpret input the same way
 - No unforeseen reactions on any input
- Identical interpretation on both sides of interface
 - Language of sender must equal language of receiver
 - Design contracts must be defined explicitly
 - Design contracts must be followed in provable way

Communication Model by Shannon

1948

- Sender: sends Message M that is $\text{Encoded}(M)$ and send over a Channel
- Receiver: receives $\text{Encoded}(M)$, then message is $\text{Decoded}(\text{Encoded}(M))$

Problem with Communication Model by Shannon

- The decoded message can be wrong
- Receiver can react differently as expected by sender,
- Because implementation of $\text{Decoded}(\dots)$ may be wrong

How can Attacker exploit the above Problem?

- Find a combination of M , $\text{Encoded}(\dots)$, $\text{Decoded}(\dots)$ with these properties:
 - $M \neq \text{Decoded}(\text{Encoded}(M))$
 - M is semantically valid for sender
 - $\text{Decoded}(\text{Encoded}(M))$ is semantically valid for receiver
 - Processing of M at receiver is not equal to processing of $\text{Decoded}(\text{Encoded}(M))$ at receiver

Typical Exploit-Techniques used by Malware

- SQL Injection
- Command Injection
- XSS
- Buffer Overflows
- String Overflows

Means to mitigate Exploits

- Check all Input Data
 - Which data, where, when
- Problems
 - Amount
 - Efficiency
 - Bypassing
 - Streaming