

# Trust, Assurance and Assumptions

What is Software Security?

Pascal Meunier, M.Sc., Ph.D., CISSP

# Costs of Insecurity Increase

*Axiom; proofs available elsewhere*

- Increased reliance on computers, increased risks, including catastrophic failures
- New and more severe laws (liability)
- Delayed security costs more and more
  - Early security (design) costs less than late security (patches, announcements, identity theft, blackmail and ransoms, consultants)



# Costs of Security Decrease

*Axiom; proofs available elsewhere*

- Availability of better development methods
- More experts, teaching materials, awareness
- Tools
  - Availability and power increases
  - Cost decreases



# Software Security

- Reliable software does what it is supposed to do.
- Secure software does that... and nothing else.
  - Ivan Arce
- How do we get there?

# Secure Programming: Art or Science?

- Artisanal work
  - Individual work
  - Completely dependent on an individual's unique skills and personal level of organization

# Engineering Sciences

- Objective
- Independent from one individual's perception
- Does not require truly unique skills
- Reproducible
- Predictable
- Systematic




# Software Engineering

- Aims to be a science, but is most often ad-hoc
- Uses reproducible processes to provide guarantees, controls, measurements
- Capability Maturity Models:
  - Characterize an organization's processes

## Levels of maturity (SEI)

- Level 1: Ad-hoc, individual effort and heroics (Art)
- Level 2: Repeatable
- Level 3: Defined
- Level 4: Managed
- Level 5: Optimizing (Science)

## Current Results

- >1000 new vulnerabilities reported every year
- About 50% of vulnerabilities are commonly repeated mistakes
- About 25% could be avoided by considering secure design principles
-  What's wrong?

# Current Folklore in Software Development Companies

- “What works in my day-to-day life should also work for the software I make”
- “Blocking known bad things from happening is good enough”
- “We’ll just keep that secret”
- “We can always patch it later”



## Contradictions from (different?) Security Experts

1. We say, "to improve security, watch out for this or that", and suggest processes that attempt to enumerate and consider known bad things
2. We also say "Don't use black lists, use white lists"
  - What are they?

## Black Lists

- Attempt to enumerate all possible bad inputs or events
  - They **assume** that this is possible
  - They **assume** that they have accounted for \*all\* possibilities
  - Widespread
    - e.g., Firewalls, web applications

# Assumptions and Vulnerabilities

- Most vulnerabilities can be expressed as an assumption that didn't hold in practice
  - Apart from silly mistakes such as off-by-one arithmetic errors
- An exploit is a way to prove an assumption wrong (in that context)

# Black Lists in Real Life

- Phone numbers
  - Block calls from known annoyances
  - Allows you to receive calls from new people
- Email
  - “real-time” black list block sources of spam
    - The list lags behind the sources so we get spam anyway (other disadvantages as well)

## Black List Failures

- Attacker figures out an input that was not blacklisted
  - Public phone booth
- Common failure example: directory traversal attacks (next slide)

# Example Directory Traversal Setup

- You have a networked application (e.g., web server using a database back-end)
- Server needs to store the password to the database somewhere, or there are other confidential documents on the server
  - Those are stored outside of a web-accessible directories ("document root")



# Black List Failure Enabling Directory Traversal Attacks

- “../” in a path to a file specifies going up a directory
  - An attacker could request a file that is not supposed to be public (e.g., passwords)
- Black list defense: block “..” in requested urls
- Attacker inputs an encoding for “..”:
  - “%2E%2E” is translated back to “..” later



## Why Black Lists Fail

- More often than not there is a way to circumvent or fool black list mechanisms
- Black lists are based on known bad things
  - Attacker only has to think of something new, or something you don't know about or forgot

# Teaching Security

- I teach secure programming by showing examples of bad things, and then showing the correct way to handle that situation
  - Train wreck stories keep up the interest of students and prove the relevance of the idea
- In effect, I am building a black list of often repeated mistakes in the minds of students!



# Other Software Security Black Lists

- Books (e.g., “19 Deadly Sins” etc...)
- SANS Top 10/ Top 20
- Software scanners
  - Check for buffer overflows, string format vulnerabilities, etc...
- Virus scanners
- Intrusion Detection Systems



# Black List Development Methods

- Threat modeling
  - Can you exhaustively list all the ways your software can be attacked?
- Risk Assessment
  - Enumerate and rank \*all\* the "significant" risks
- Penetrate and Patch



## Limited Success of Black List Development Methods

- Depend on the skills and experience of the actors (artisanal work, not science)
- Require expert knowledge
  - Expensive, irreproducible (unreliable)
- Better than nothing
  - Likely, common attacks can be parried or mitigated

# How Many Ways Are There to Make Mistakes?

- Axiom: **Infinite**
- Can you enumerate all of them?
  - Mathematically, **yes**
- How much time would that take?
  - **Infinity**

## Effort at MITRE: Common Weakness Enumeration

- List (or tree) of software weaknesses, idiosyncrasies, faults and flaws
- Black list of mistakes
- Job security!

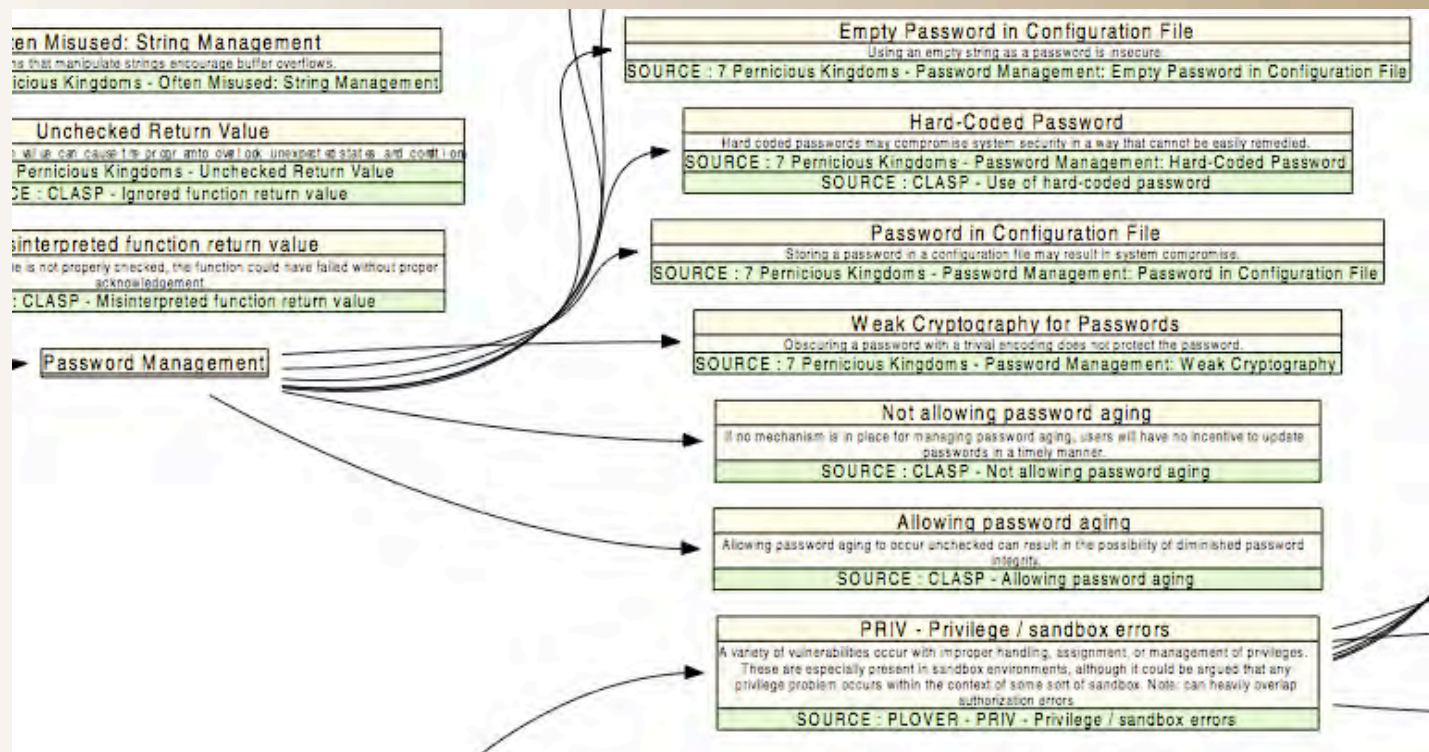
## CWEC (draft)

- Work in progress





# CWEC



## CWEC as a Black List?

- CWEC is work in progress and is already gigantic
- There will always be new additions
- Is it feasible, practical, or even effective to test software for the entire CWEC list of defects?
  - Better than nothing to look for some
    - Is that good enough?



## What is Good Enough?

- Hackers are inventive and push back the limits of “good enough”
  - Dynamic, changing requirement
- Software is pushed towards higher and higher assurance requirements
- Black list development methods are doomed because they operate based on past knowledge

# Security Through Obscurity

- Leads to catastrophic failures
  - Secret phone numbers
    - Paris Hilton phone list made public
  - DVD encryption
    - Key found in software made public
- Don't assume that because it "almost" works in real life that it will work with software!



## Why Isn't Software Engineering More Like Civil Engineering?

- Bridges and buildings might collapse if they were ported from Earth 1.0 to Earth 2.0 with new Physics features (e.g., anti-gravity)
  - Roman arch wouldn't hold without gravity!
    - But it wasn't designed for that! Wait a minute...

# Civil Engineering

- Operates against natural threats
  - Risk analysis works in a stable environment
  - Past experience applies to the future
- Does not operate against
  - Intelligent, motivated attackers
  - Constantly changing environment

# Mathematical Tools and Models

- Civil Engineers have mathematical models that provide guarantees on the strengths of a structure
- Where are the equivalent for software engineering?
  - Formal methods
    - Historically been slow, extremely expensive, with results having very limited usefulness



## White Lists

- List known safe input and only accept that.
- Assumption that the inputs are safe can be validated and tested with defined criteria
- Not an assumption anymore
  - Software becomes more secure

## White Lists in Real Life

- “Priority Ringing”: assign a special ring to numbers you want to answer
- “Inbound Call Blocker” requires people to enter a secret code in order to ring your phone
  - The password is not a white list, but you use a white listing process when choosing who to give the password to

## What Looks Like White Lists?

- Best Practices
- List of things and people you trust
- Identification of assumptions
  - Changing assumptions into (justified) trust
- List of reasons for trusting (assurance)
- Better software development (formal) methods and tools

## Can I Trust This?

- The end goal or question for secure software development and deployment
- Measure and justification for trust:  
**Assurance**
- Recursive: what does the software need to trust? What are its assumptions?
  - What do those things need to trust?
    - What do those things need to trust?



## Basis for Trust

- Who and what do you trust?
- Does trust imply that you're taking a risk?
  - What is at stake?
- Are there assurances that something (or someone) can be trusted?
  - Why do you trust?



# Validating Trust

- What tools and techniques can be used to produce or evaluate those assurances?
  - Can you justify (verify/validate) trust?
- What is trusted computing?
  - Who doesn't trust who? (goals, perspectives)

# Why Do You Trust?

- Reduce costs
  - Economies of scale
  - Fewer tests
  - Simpler procedures
- Benefit from the skills of another
- Delegate and gain power and focus

## I Trust Because...

- Hopefully, not because you have no choice
  - that's not trust, it's being dependent and at the mercy of ...
- Need to balance risks and benefits
- What are the threats?
- What are the assurances?

## Example

- What would you think of a store where prices are indicated on a shelf, and the customer tells the cashier what the price was?
- Many web shopping carts had this defect: the server would tell the client browser the price of items, and the client would repeat the price back to the server when it was time to pay...

# Data Integrity and Trustworthiness

- Who has had control of the data?
- What was done to it?
- Who could affect operations done on the data?
- Is part of the data supplied by the user?
- Is the data used as part of commands?
- Many vulnerabilities result from not having a clear understanding of that



## How to justify (validate/ verify) trust

- Company security policies should describe standards, specifications and procedures for securing assets
- A formally trusted asset complies with all
- Asset is used with minimal checking
- Partial compliance means partial trust

# Partial Trust

- Requires additional safeguards in any application or system making use of the asset
- Trust is not binary (Yes/No)
- Calculated risk

## Hypothetical Question

- An external contractor lets your company access a database but will not let you audit their procedures (auditing wasn't part of the contract).
- How does that affect the trust put into that database for your applications that use it?

## Possible Answers

- Although we "trust" them to do a best effort at security, we design applications by assuming that their database could have been compromised (proactive stance)
- We trust their data, and if their database should get compromised we will sue them for damages (reactive stance)

## Another Answer

- Since the contract specifies which procedures they were supposed to follow, we will mostly trust the database but include several basic "sanity checks"
  - Perhaps we trust their reputation
- It depends on your company's security policies, security stance, and perhaps just the policies for that project (and its criticality).



# Trust Types

- Direct
- Transitive
- Assumptive

## Direct Trust

- You perform the validation, proofs, and assurance instruments yourself
- No delegation

## Transitive Trust

- Entities with the same standards and criteria trust the evaluation results of the others
- A trusts B
- B trusts C
- A trusts C transitively through B

# Assumptive

- A.k.a. "spontaneous"
- Trusting the results of an evaluation conducted with different standards, criteria or procedures
- Most dangerous trust type
- As its name suggests, it often involves an assumption, probably not clearly identified

## Example Assumptive Trust

- A friend tells you that Brand Z's mint ice cream is great, so you buy some.
- Your tastes are somewhat different but you trust the evaluation
- Can you provide another example?

# Foundations of Trust

- Identification
- Authentication
- Accountability
- Authorization
- Availability
- Assurance

# Who and What Do You Trust, Why and How

- Trust binds a property or attribute (expected behavior) to an identifiable asset
  - What are the assurances of the expected behavior? Why is it trusted?
- Need authentication to assert identity
- Need access control to provide just enough authority for desired capabilities

# AAA

- Accountability
  - Proofs of misconduct; systematic logs
- Authorization
  - Provide guarantees about the transfer of capabilities and the properties of assets
- Availability
  - Is the trust data sufficiently available to make timely and informed decisions?



# Trusting White Lists

- Definition: white list software development methods attempt to bind properties and expected behaviors to planned and systematic processes and activities or even programs themselves
  - Mathematical, formal proofs provide assurance that the software does what it is supposed to and nothing else
  - Closer to repeatable, engineering science



# White List Development Methods

- Design by contract
- Correctness by construction
- Model checkers
- Automated analysis tools

# Design By Contract

- Specify relationship between a software object class and its clients as a contract
  - Pre-conditions for execution
  - Post-conditions after execution
  - Invariants
  - Bertrand Meyer, "Object Oriented Software Construction"

## Pre-Conditions

- Things that must be true for the code to perform correctly
- Example: In order to calculate the square root, the function must be given a positive number
  - Function is unable to handle imaginary numbers

# Post-Conditions

- What the method changed
  - Side-effects
  - Results
- Example: A function that fetches a record advances a cursor
- Analogy: You leave the shopping cart next to your car after doing groceries

# Invariants

- What holds true throughout a section of code
  - Perhaps what **MUST** hold true for the program to be correct
- Examples:
  - Some data is read-only
  - Program can write only inside this space
  - Gravity on earth =  $9.80665 \text{ m} / \text{s}^2$



# Identification of Invariants

- Automatic software analysis methods attempt to identify program invariants
- Can flag updates and changes to code made by new programmers, that violate the invariants
  - Mistakes or incomplete training of new coder
  - Loss of expertise through personnel turnover
- Ernst et al. 2001



# Correctness by Design

- High integrity software
- Recognized as one of the best ways to create high assurance software
- Uses SPARK, a derivative of the Ada language
- Formal description of the code is checked against what the code actually does



# Model Checkers

- Check invariants throughout code execution
  - Limitation: does it exercise all code paths?
    - No execution == no check
    - State “explosion”: the combination of all possible values for variables results in an exponential increase in the number of things to keep track of
      - Some simplifications are made

# White List Configuration Method

- Suse Linux “AppArmor”
  - Uses “capabilities” defined by the operating system
  - In learning mode, registers all the needed capabilities for normal operation
  - Afterwards, it denies operations that need a new capability (it could be the result of an exploit)

## Limitations of AppArmor

- Instead of being a list of known safe capabilities, AppArmor is a list of needed capabilities
  - Not a “pure” white list
- Capabilities are granular
- Capabilities can be composed together in complex ways (possibly unexpected)
- Allows more than is known safe



# Types of White Lists

- Permissive white list
  - Allows more than necessary (e.g., due to granularity)
- Restrictive white list
  - Denies more than necessary
- Exact white list
  - Matches exactly what is safe to do

## Which Are Most Alike?

1. Assumptive Trust

A. White List methods

2. Transitive Trust

B. Black List methods

i. Engineering Science

ii. Artisanal Work

# How Do You Know It Worked?

- Can never be 100% certain that software is perfectly secure
- How can you have transitive (justified) trust in software?
  - Assurance is the best we can do
    - Assurance provided by the development methods
    - Assurance that the development methods were followed



## What Provides Assurance?

- “Software Quality Engineering”
- “Software Quality Assurance”
- “Software Assurance”
- Verification and Validation
  - For our purposes (purists will object)
- Software security assurance requirements
- Software security functional requirements

## Software Assurance is not...

- A software maintenance plan with benefits, with photos of smiling people
  - That's a service plan, insurance, not assurance
- Marketing ploy
  - People vaguely know that assurance is good
    - So let's market our insurance as "assurance"
  - Brought to you by a company with **enormous software assurance problems** (go figure)



# Assurance

- Planned processes and systematic activities that improve correctness
- How do you measure assurance?
- International Standard: Common Criteria
- Defines Evaluation Assurance Levels (EALs) 1-7 and more

# What role does assurance play in computer security?

- a. It protects vendors that get sued
- b. It's how you deal with customers afraid of the latest vulnerability
- c. It describes how the application of specific software engineering practices lowers risks

# Common Criteria

- <http://niap.nist.gov/cc-scheme/index.html>
- Vocabulary
  - PPs: Protection profiles
  - STs: Security Targets
  - TOE: Target of Evaluation (i.e., product)
  - CCTL: Common Criteria Testing Laboratory

## CC Main Contents

- Security functional requirements
  - e.g., logging, encryption
- Security assurance requirements
  - e.g., code review and audit for vulnerabilities, secure storage of code during development, training and education of programmers in secure coding
- Common Evaluation Methodology



# Protection Profiles

- Specify customized security requirements (by the customer)
- Moving from assumptive trust to transitive trust
  - PPs define \*your\* criteria and requirements
- Example PPs: Anti-Virus, Biometrics, Operating Systems



# Security Targets

- Specific to a product and customer
  - What does it need to do, security-wise, for a particular kind of customer? (by vendor)
- ST Evaluation:
  - Check if ToE really meets the ST security functional requirements
    - To a degree specified by security assurance requirements

# CCTL

- Trusted to perform evaluations with defined criteria
- Better than assumptive trust because PPs are matched against certifications
  - Quite flexible
  - Strictly speaking, not full transitive trust because the available criteria are pre-determined, but it's very close

## Validated Products

- Evaluate security objectives, requirements, development methods to some EAL
- Example:
  - Microsoft Windows Server 2003 Certificate Server, EAL 4 Augmented ALC\_FLR.3
    - ALC\_FLR.3 defines additional requirements

# Evaluation Assurance Levels (EALs)

- EALs 3-4 commonly requested by governments and security-demanding organizations
- EAL 4 evaluation typically costs \$1 million
- High assurance (EALs 5-7) requires formal methods (~white lists!)
  - Becoming more and more feasible now



## **Name an International Assurance Standard?**

- a. Common Criteria
- b. Frequent Criteria
- c. The EAL

## Conclusion

- Thesis: Software engineering, to be a science, needs more and better white list methods supported by formal methods
  - This will lower costs (less reliance on expensive artisan-experts), increase security and make higher assurance software practical
- Assurance gives you confidence that processes were executed properly, and evaluations give you a justification for transitive trust