Some principles of secure design

Designing Secure Systems 2017/18

David Galindo

Based on slides by Mark Ryan

Caveat: No magic formulas…

We have no silver bullet

On the contrary:

Security is about trade-offs

Conflicting engineering criteria….

Conflicting requirements…

Overcoming human,
technology and market deficiencies

12 Principles

These principles draw on the ideas of **simplicity** and **restriction**:

- **Simplicity** makes designs and mechanisms **easy to understand**. Importantly, less can go wrong with simple designs.
- **Restriction minimizes the power of an entity**: it can access only information it needs; it can communicate with other entities only when necessary, and in as few (and narrow) ways as possible

Design principles for protection mechanisms

[Saltzer and Schroeder 1975]

Jerome H. Saltzer, Michael D. Schroeder.
The protection of information in computer systems.
Proceedings of the IEEE 63(9): 1278-1308 (1975)

12 Principles

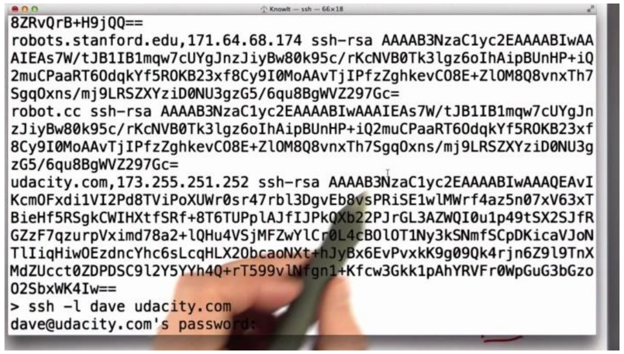
1. Secure the weakest link
2. Defence in depth
3. Fail secure
4. Grant least privilege
5. Economise mechanism
6. Authenticate requests
7. Control access
8. Assume secrets not safe
9. Make security usable
10. Promote privacy
11. Audit and monitor
12. Proportionality principle

1. Secure the weakest link

- Security practitioners often point out that **security is a chain**; and just as a chain is only as **strong as the weakest link**, a software security system is only as secure as its weakest component
- **Attackers go after the weakest point in a system**, and the weakest point is rarely a security feature or function. When it comes to secure design, make sure to consider the weakest link in your system and ensure that it is secure enough

Gene Spafford's story

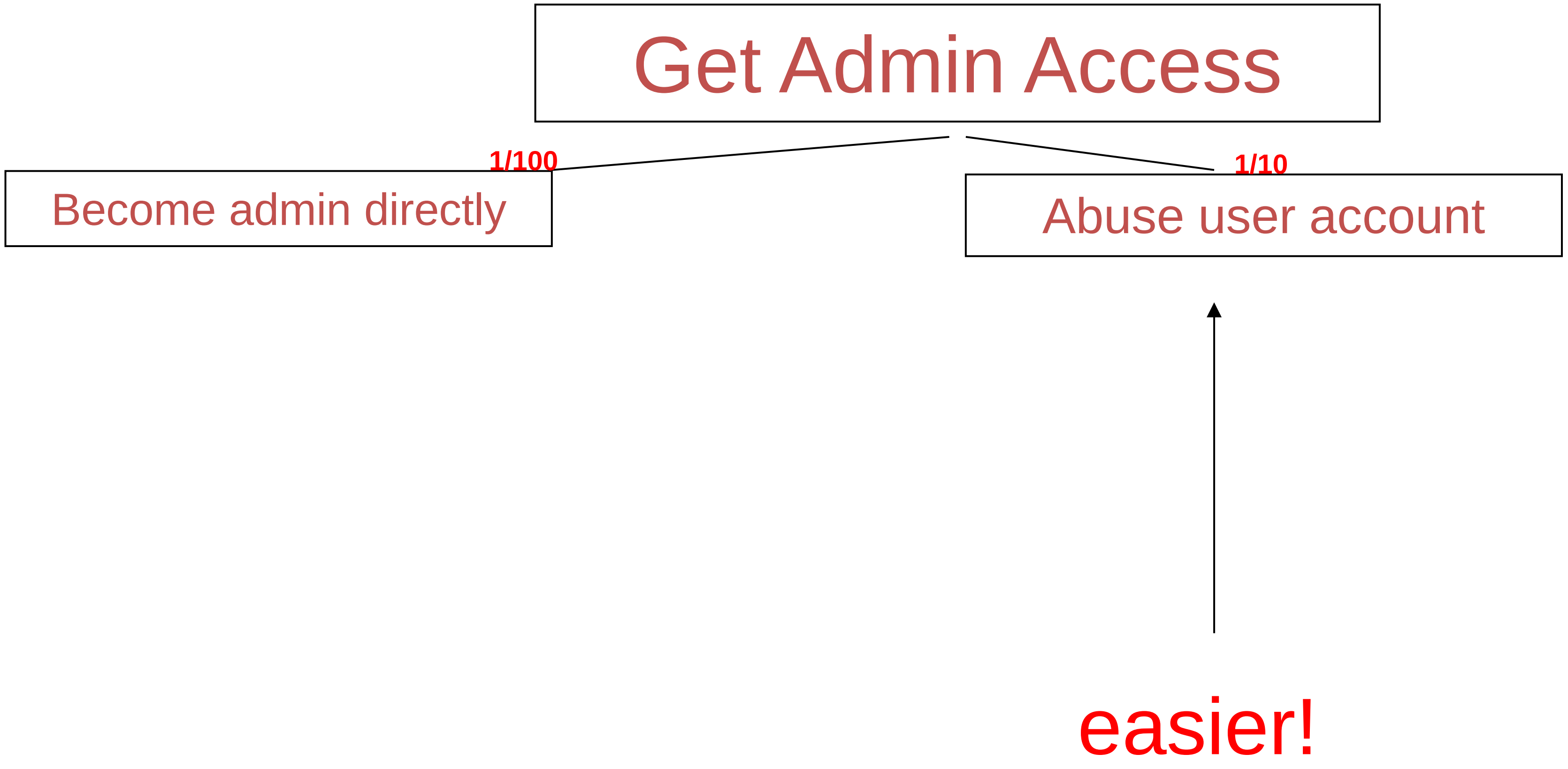
- Imagine you are charged with transporting some gold securely from one homeless guy who lives in a park bench (we’ll call him **Linux**) to another homeless woman who lives across town on a steam grate (we’ll call her **Android**). You hire an armoured truck to transport the gold. The name of the transport company is "**Applied Crypto, Inc**." Now imagine you’re an attacker who is supposed to steal the gold.
- Would you attack the Applied Crypto truck, Linux the homeless guy, or Android the homeless woman?



```
8ZRvQrB+H9jQQ==
robots.stanford.edu,171.64.68.174 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAA
AIEAs7W/tJB1IB1mqw7cUYgJnzJiyBw80k95c/rKcNVB0Tk3lgz6oIhAipBUnHP+iQ
2muCPaaRT6OdqkYf5ROKB23xf8Cy9I0MoAAvTjIPfzZghkevCO8E+ZlOM8Q8vnxTh7
SgqOxns/mj9LRSZXYziD0NU3gzG5/6qu8BgWVZ297Gc=
robot.cc ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAs7W/tJB1IB1mqw7cUYgJn
zJiyBw80k95c/rKcNVB0Tk3lgz6oIhAipBUnHP+iQ2muCPaaRT6OdqkYf5ROKB23xf
8Cy9I0MoAAvTjIPfzZghkevCO8E+ZlOM8Q8vnxTh7SgqOxns/mj9LRSZXYziD0NU3g
zG5/6qu8BgWVZ297Gc=
udacity.com,173.255.251.252 ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAvI
KcmOFxdi1VI2Pd8TViPoXUWr0sr47rbl3DgvEb8vsPRiSE1wlMWrf4az5n07xV63xT
BieHf5RSgkCWIHXtfSRf+8T6TUPplAJfIJPkQXb22PJrGL3AZWQI0u1p49tSX2SJfR
GZzF7qzurpVximd78a2+lQHu4VSjMFZwYlCr0L4cBOlOT1Ny3kSNmfSCpDKicaVJoN
TlIiqHiwOEzdncYhc6sLcqHLX2ObcaoNXt+hJyBx6EvPvxkK9g09Qk4rjn6Z9l9TnX
MdZUcct0ZDPDSC9l2Y5YYh4Q+rT599vlNfgn1+Kfcw3Gkk1pAhYRVFr0WpGuG3bGzo
O2SbxWK4Iw==
> ssh -l dave udacity.com
dave@udacity.com's password:
```

•Weakest Link

Security like a chain:



The human (caveat and precious resource)

- Humans are often considered the “weakest link” in the security chain
- They ignore security policies and stick passwords to their computer monitors
- They can be coerced, blackmailed, “socially engineered” (=tricked)
- Even security experts are vulnerable to phishing
- But, “the user is not the enemy”...

Are people the weakest link in computer security?

“Cybersecurity professionals have spent the last 25 years saying **people are the weakest link**. That’s stupid! They cannot possibly be the weakest link – they are the people that create the value at these organisations”

“What that tells me is that the **technical systems** we’ve built **are not built for people**. Techies build systems for techies, they don’t build technical systems for normal people”

Ian Levy, NCSC Director

The Guardian, 22 Sept 2017

2. Defence in depth (aka “Castle Approach”)

- The idea behind the “defence in depth” approach is to defend a system against any particular attack using several independent methods
- It is a layering tactic, conceived by the US National Security Agency as a comprehensive approach to information and electronic security

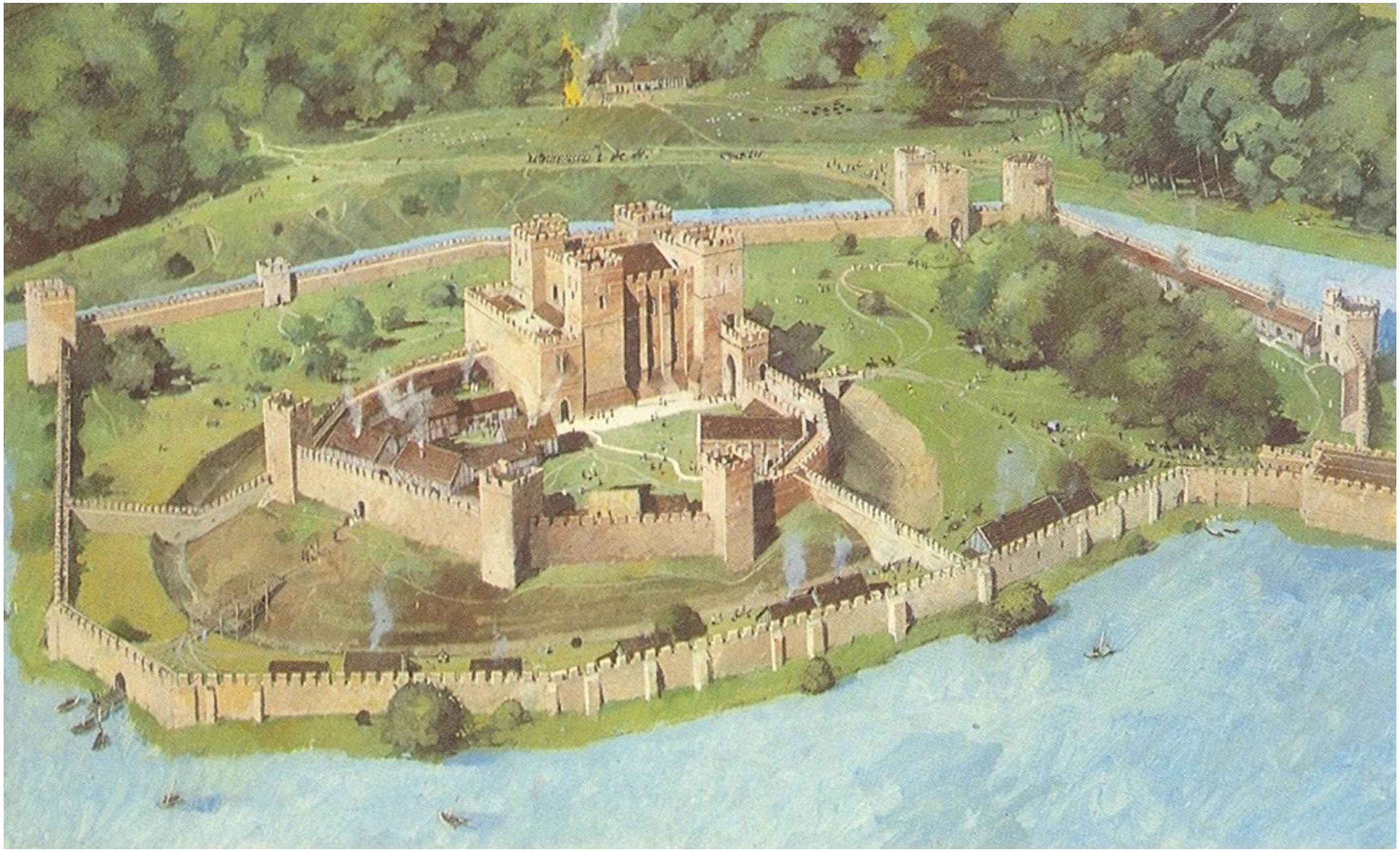
Redundancy and layering

- Examples: Don’t count on your firewall to block all malicious traffic; use an intrusion detection system as well. If you are designing an application, prevent single points of failure with security redundancies and layers of defence
- The idea behind defence in depth is to manage risk with diverse defensive strategies, so that if one layer of defence turns out to be inadequate, another layer of defence will hopefully prevent a full breach

Defence in depth: some mechanisms

- Anti virus software
- Authentication and password security
- Biometrics
- Demilitarized zones (DMZ)
- Encryption
- Firewalls (hardware or software)
- Hashing passwords (secure password management)
- Intrusion detection systems (IDS)
- Logging and auditing
- Multi-factor authentication
- Vulnerability scanners
- Physical security (e.g. deadbolt locks)
- Timed access control
- Internet Security Awareness Training
- Virtual private network (VPN)
- Sandboxing
- Intrusion Protection System (IPS)

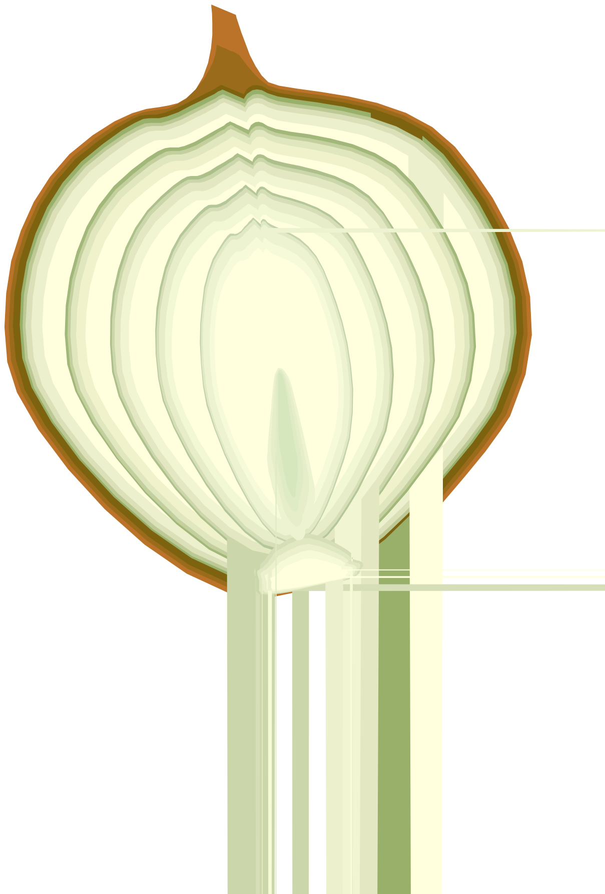
Military: Defence in Depth



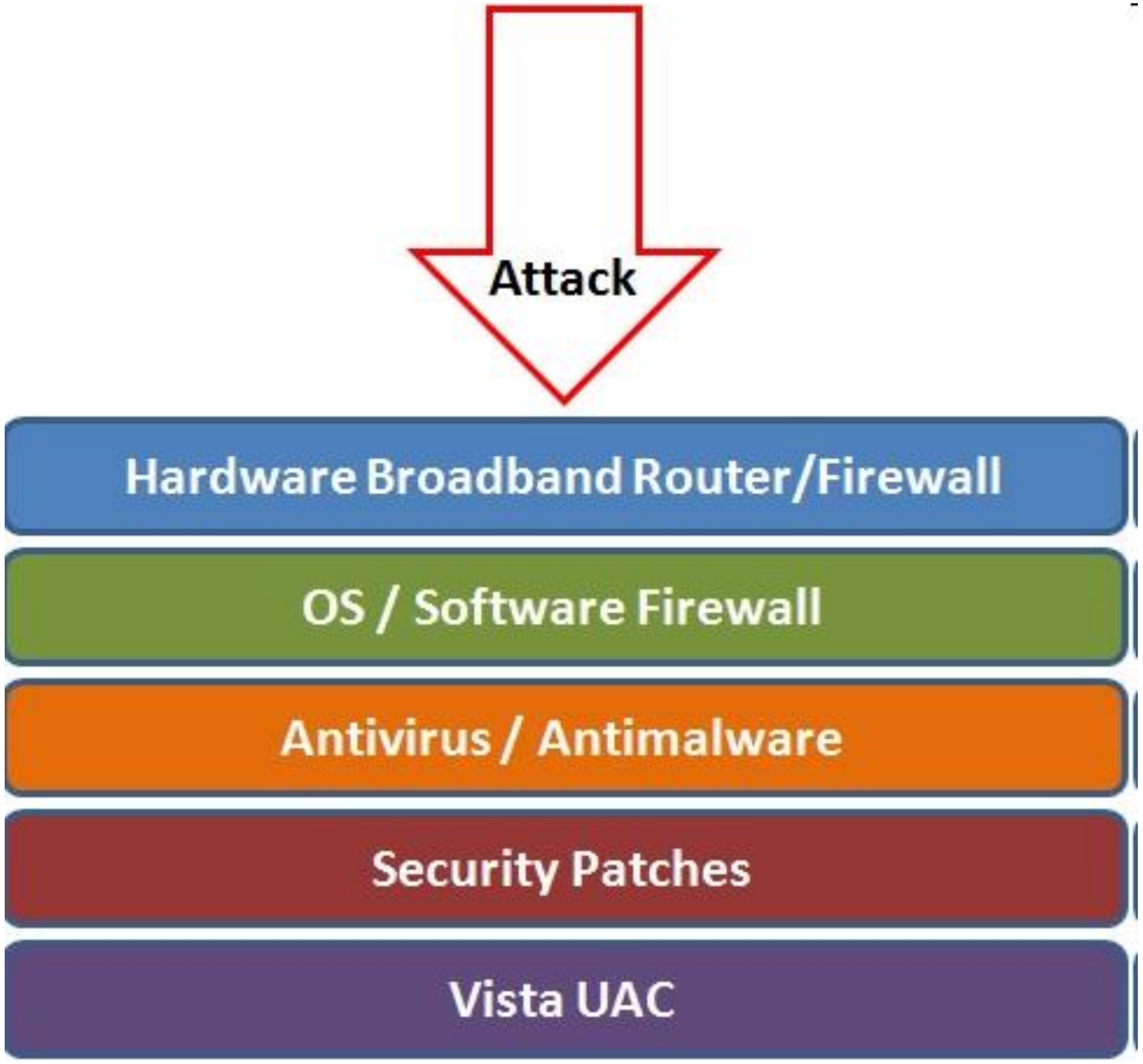
Layers

Computer systems have multiple layers, e.g.

- HW components
- Chipset/MB
- OS
- TCP/IP stack
- HTTP application
- Secure http layer
- Java script
- User/smart card interface



Example


Attack
Hardware Broadband Router/Firewall
OS / Software Firewall
Antivirus / Antimalware
Security Patches
Vista UAC

Brainstorming defence in depth

- Helps to consider these three aspects:
  - People
  - Technology
  - Operations

- Example questions:
  - Attacker may get my users' passwords
  - Attacker may cause his own code to run on my servers

Attacker may cause his own code to run on my servers

Defence in depth:

- P: Education against social engineering and phishing
- P: Education on granting less privilege principle
- T: Sandboxing
- T: Access control
- T: Put additional security if not from familiar IP addr
- T: Use an Intrusion Detection System
- O: Set up Penetration Testing team
- O: Set up Software Secure Development Lifecycle

Attacker may get my users' passwords

Defence in depth:

- P: Education against social engineering and phishing
- P: Education about choosing good passwords
- T: Defend pw file using firewall
- T: Hash/encrypt/... pw file
- T: Put additional security if not from familiar IP addr
- T: Use one-time passwords
- O: Log guessing attempts
- O: Rate-limit guesses

Chains vs. Layers

3. Fail secure

- A fail-secure system is one that, in **the event of** a specific type of **failure**, responds in a way such that **access or data are denied**
    - Related: a **fail-safe system**, in the event of failure, causes no harm, or at least a minimum of harm, to other systems or to personnel
- Fail-secure and fail-safe may suggest **different outcomes**
    - For example, if a building catches fire, fail-safe systems would unlock doors to ensure quick escape and allow firefighters inside, while fail-secure would lock doors to prevent unauthorized access to the building

Fail secure

- Default is to deny access
- Programmers should **check** return values for **exceptions/failures**
- Base access decisions on **permission** rather than exclusion. This principle means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted
  - The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design

Secure-fail programming

- Look at the following (pseudo) code and see whether you can work out the security flaw:

-
```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == ERROR_ACCESS_DENIED) {
     // Security check failed
     // Inform user that access is denied
} else {
     // Security check OK
     // Perform task
}
```

- Secure-fail version:

-
```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
     // Security check OK
     // Perform task
} else {
     // Security check failed
     // Inform user that access is denied
}
```

Secure-fail programming

- Look at the following (pseudo) code and see whether you can work out the security flaw:

```
isAdmin = true;
try {
  codeWhichMayFail();
  isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex)
{
  log.write(ex.toString());
}
```

- Secure-fail version:

```
isAdmin = false;
try {
  codeWhichMayFail();
  isAdmin = isUserInrole( "Administrator" );
}
catch (Exception ex)
{
  log.write(ex.toString());
}
```

Fail-insecure designs

- Interrupted boot of OS drops user into root shell
- Browser unable to validate HTTPS certificate:
  - Tells user, allows user to click-through to proceed anyway
- Point-of-sale terminal unable to contact card-issuing bank:
  - Allows the transaction if less than a certain threshold
  - Risk management overrides security principle

Why we often have fail-insecure

- Fail-insecure often introduced by desire to support legacy (insecure) versions.
  - E.g., TLS allows old clients to use weak crypto keys

- Sometimes introduced because it is in general easier to work with short “blacklist” than with long “whitelist”
  - List of known-phishing websites
  - List of known-malicious users
  - List of known…

4. Grant least privilege

The principle of least privilege

- also known as the principle of minimal privilege or the principle of least authority

requires that in a particular abstraction layer of a computing environment, **every module** (such as a process, a user, or a program, depending on the subject) **must be able to access only the information and resources that are necessary for its legitimate purpose**

Least privilege: rationale

- When code is limited in the system-wide actions it may perform, vulnerabilities in one application cannot be used to exploit the rest of the system.
  - For example, Microsoft states “Running in standard user mode gives customers increased protection against inadvertent system-level damage caused by malware, such as root kits, spyware, and undetected viruses”

Least privilege: example

- A user account gets only those privileges which are essential to that user's work
  - A backup user does not need to install software: hence, the backup user has rights only to run backup and backup-related applications. Any other privileges, such as installing new software, are blocked.

Least privilege: example

- In UNIX systems, root privileges are necessary to bind a program to a port number less than 1024.
  - For example, to run a mail server on port 25, the traditional SMTP port, a program needs the privileges of the root user
- Once set up, the program should relinquish its root privileges, not continue running as root
- A large problem with many e-mail and other servers is that they don't give up their root permissions once they grab the mail port (Sendmail is a classic example)

Why we often fail to assign least privilege

- It's an effort to figure out what the least privilege needed actually is
- The unaware (or lazy or overworked) designer or programmer will often assign the max privilege, because that's easy
- "If we don't run as admin, stuff breaks"
- It's very hard to design systems that don't have some root/sysadmin that has all the privileges
  - E.g., Google employees getting fired for reading users gmail

Related: separate privileges

- Design your system with many different privileges
  - “file access” is not one privilege, but many
  - “network access” translates to many privileges
  - reading vs writing

Separation of Privileges

Split system into pieces, each with limited privileges

<u>Implementation in software engineering:</u>

Have computer program fork into two processes:

- –The main program <u>drops</u> privileges (e.g. dropping root under Unix)
- –The smaller program keeps privileges in order to perform a certain task
- –The two halves then communicate via a socket pair
- –

Benefits:

• A successful attack against the larger program will gain minimal access.

–even though the pair of programs will perform privileged operations

Related: Segregation of Duties

Achieved by the **Principle of Functional Separation**

- Several people should cooperate
- Examples:
    - one developer should not work alone on a critical application
    - the tester should not be the same person as the developer
    - If two or more steps are required to perform a critical function, at least two different people should perform them, etc

This principle makes it very hard for one person to compromise the security, on purpose of inadvertently.

A particular case is the **Principle of Dual Control**

- Example 1: in the SWIFT banking data management system there are two security officers: left security officer and right security officer. Both must cooperate to allow certain operations
- Example 2: nuclear devices command
- Example 3: cryptographic secret sharing

5. Economise mechanism

- Complexity is the enemy of security
- It’s just too easy to screw things up in a complicated system, both from a design perspective and from an implementation perspective

Economise mechanism: related

- Keep it simple, don’t look silly!
  - A design principle noted by the U.S. Navy in 1960
- Minimalistic design
  - Developers may create user interfaces made to be as simple as possible by eliminating buttons and dialog boxes that may potentially confuse the user
- *Worse* is better
  - Quality does not necessarily increase with functionality. Preference given to software that is limited, but simple to use
- "You ain't gonna need it" (YAGNI)
  - A principle of *extreme programming*, says that a programmer should add functionality only when actually needed, rather than when you just foresee that you need them

Why systems get complex

- Desire for features; mission-creep
- New technologies, new possibilities
  - Stretches the designs we already have
  - Causes interfaces to get used in ways not intended
  - Causes new software to be layered on top of old software
- Desire for legacy compatibility
  - Desire to keep things interoperable

Why complexity leads to insecurity

- Cognitive overload
  - Designer can't keep all the possibilities in her head at once
  - Security analyst can't reason about all the access possibilities

6. Authenticate requests

Be reluctant to trust

- Assume that the **environment** where your system operates **is hostile**. Don’t let just anyone call your API, don’t let just anyone gain access to your secrets!
- When using a cloud component, put in some checks to make sure that it has not been spoofed or otherwise compromised
- **Anticipate attacks** such as: command-injection, cross-site scripting, and so on

A Few Words on Trust

Definitions:

- A trusted system is one that can break the security policy

(paradoxical definition)

- Trustworthy system is one that won't fail us

An employee who is selling secrets is trusted **and** not trustworthy

Suppose Dropbox is trustworthy, and yet Alice encrypts her files before putting them there. Thus for Alice, Dropbox is trustworthy **but** not trusted

Trust vs Trustworthiness

Questions:

What is trusted **and** trustworthy?

What is trusted **and** not trustworthy?

What is not trusted **but is** trustworthy?

Be Trustworthy

Public scrutiny promotes trust

Commitment to security **is visible** in the long run

7. Control access

- Every access and every object should be checked, every time
- Make sure that if permissions change on the fly in your system, that access is systematically rechecked
- Don’t cache permission results that grant authority or wield authority
- In a world where massively distributed systems are pervasive and machines with multiple processors are the norm, these principles can be tricky to implement

8. Assume secrets not safe

- Security is not obscurity, especially when it comes to secrets stored in your code. Assume that an attacker will find out about as much about your system as a power user, and more
- The attacker’s toolkit includes decompilers, disassemblers, and any number of analysis tools. Expect them to be aimed at your system
- Finding a crypto key in binary code is easy. An entropy sweep can make it stick out

9. Make security usable

- If your security mechanisms are too annoying and painful, your users will go to great length to circumvent or avoid them
- Make sure that your security system is as secure as it needs to be, but no more
- If you affect usability too deeply, nobody will use your product, no matter how secure it is. Then it will be very secure, and very near useless

You can Even Be More Friendly

- Don’t discourage users
  - –Minimize the number of clicks
  - –Minimize the number of things to remember
  - –Make security easy to understand and self-explanatory
  - –Security should NOT impact users that obey the rules
- Established defaults should be reasonable
  - –People should not feel trapped
- It should be easy to
  - –Restrict access
  - –Give access
  - –Personalize settings

10. Promote privacy

- Collect only the user *personally identifiable information* (PII) you need for your specific purpose
  - EU General Data Protection Regulation (GDPR) applies in UK from May 2018
    - data breaches can cost up to 4% of a company’s yearly revenue
- Store it securely, limit access
- Delete it once your purpose is done
- Let the users own their data (only store encrypted data)

11. Audit and monitor

- Record what actions took place and who performed them
  - This contributes to both disaster recovery (business continuity) and accountability

12. Proportionality Principle

Reminder: security isn't the only thing we want.

Maximize security???

-vs-

Maximize “utility” (the benefits)
while limiting risk

to an acceptable level
within reasonable cost
commensurate with attacker's resources

Twelve principles

1. Secure the weakest link
2. Defend in depth
3. Fail secure
4. Grant least privilege
5. Economise mechanism
6. Authenticate requests
7. Control access
8. Assume secrets not safe
9. Make security usable
10. Promote privacy
11. Audit and monitor
12. Proportionality principle

Remember the caveat:
There are no magic formulas…

We have no silver bullet

Security is about trade-offs

Conflicting engineering criteria….

Conflicting requirements…

The goal is to have enough security to thwart the attackers
we care about