Software Security

Security Principles

Erik Poll
Digital Security (DS) group
Radboud University Nijmegen
Security principles

• Variations of lists of security principles appear in literature & online (see course website)
• Security vulnerabilities often exploit violations of these principles
• Good security solutions or countermeasures follow these principles
• Some overlap & some tension between principles
Security principles

• secure the weakest link
• defence in depth
• principle of least privilege
• minimise attack surface
• compartementalise
• secure defaults

• keep it simple
• fail securely
• promote privacy
• hiding secrets is hard
• use community resources
• be reluctant to trust
• ....
Security principles

These principles can be applied at *many levels*, eg.

- in *source code of a application*
- between applications on a machine
- at OS (operating system) level
- at network level
- within an organisation
- between organisations
- ...

Secure the weakest link

- Spend your efforts on improving the security the weakest part of a system, as this is where attackers will attack

- NB this requires a good risk analysis
  - what are the threats & attacker model?
  - which have the highest risk & impact?
Secure the weakest link

• Is the user maybe the weakest link?
  - eg think of phishing attacks, weak passwords
  *Maybe user education more important than technical measures?*

• Web application visible through firewall may be easier to break than the firewall?
  - *If so: improve web application security, not the firewall*
  Or are attacks on the browser the highest risk?
  - *If so: improve browser security*
Practise defence in depth

• have several layers of security
  - two controls are better than one.
• no single point of failure

• A typical violation:
  having a firewall, and only having firewall
  - a user bringing in a laptop circumvents firewall
    • this is an example of enviromental creep
Defence in depth example

• have a firewall

and

• secure web application software

and

• run web application with minimal priviliges
**Defence in depth example**

- use OS access control to restrict access to sensitive files 
  *and*
- encrypt them

  - Especially when files are written to removable media such as USB sticks (another example of environmental creep), to laptops, or PCs which might be disposed off
Defence in depth: counterexample

- Originally, on UNIX systems, the password file, /etc/passwd, which contains hashed passwords, was world readable

- better
  - hash passwords
  - have tight access control to the file
Principle of least privilege

• be stingy with privileges
  - only grant permissions that are really needed
  - resource permissions (eg memory limits, CPU priorities), network permissions, file permissions, ....

• typical violations
  - logging in as root/administrator
  - device drivers having to run in kernel mode
Principle of least privilege

• in organisation
  - don’t give everyone access to root passwords
  - don’t give everyone administrator rights

• on computer
  - run process with minimal set of privileges
    - Eg, don’t run web application as root or administrator
Principle of least privilege

- **in code**
  - not `public int x;`
    but `private int x;`
  
  - not `public void m();`
    but `package void m();`

- Expose minimal functionality in interfaces of objects, classes, packages, applications
Principle of least privilege

- for Java application
  not the default policy
  grant codeBase "file:${java.ext.dirs}/" {
    permission java.security.AllPermission;
  };
  but minimum required
  grant codeBase "file:./forum/*" {
    permission java.security.FilePermission;
    "/home/forumcontent/*","read/write";
  };
Principle of least privilege

- NB applying the principle of least privilege in code is **tricky & hard** and requires **lots of work and discipline**
- Why?
  - compiler complains about private field that should be public, not the other way around
  - compiler complains about missing import, not about superfluous import.
- Can this be improved?
  - tool support in compilers & IDEs, eg or separate source code analyzers, eg FindBugs and JAMIT tool for tightening visibility modifiers
    (http://grothoff.org/christian/xtc/jamit)
Keep it simple (aka economy of mechanism)

- **Complexity** important cause of security problems
  - complexity leads to unforeseen feature interaction
  - complexity leads to *incorrect use* and insecure configuration by users and developers
NB there is a fundamental conflict between
• principle of least privilege
and
• kiss principle – keep it simple

- principle of least privilege requires very fine-grained control with expressive policies
- ... which leads to more complexity
- ... which people then get wrong

• Compartementalisation can provide a solution using defence in depth
Compartementalise

• Access control is most comprehensible, and easiest to manage, if it is \textit{all or nothing} for large chunks – \textit{compartments} - of a system

• Motivations:
  - keeping it simple
  - containing attacker in case of failure

• Analogy: compartments on a ship
• Counterexample: OS that crashes if an application crashes
Compartementalise examples

- Use different machines for different tasks
  - eg run web application on a different machine from employee salary database
- Use different user accounts on one machine for different tasks
  - unfortunately, security breach under one account may compromise both...
  - compartementalisation provided by typical OSs is poor!
- Partition hard disk and install OS twice
Improved compartmentalisation – at OS level

- **chroot jail**
  restricts access of a process to subset of file system, ie. changes the root of file system for that process

  Eg run an application you just downloaded with
  ```
  chroot /home/sos/erikpoll/trial;/tmp
  ```

  Nice idea, but chroot is hard to get working, and hard to get working correctly.
Compartementalisation – at OS level

- virtual machines
  - VMWare
  - very popular these days, but mainly for reasons of convenience & costs, not security

- operating system hypervisors (true microkernels)
  small, lightweight kernel, which partions hard disk & memory, to concurrently run several copies of the OS, in different compartments
  - XEN, L4Linux, Fiasco, Nova hypervisors from TU Dresden
Virtualisation by virtual machine

- We simulate the hardware in an OS process

Similar to Java VM, except that we simulate the real hardware, and don't provide some abstract VM

This is solution proposed by VMware.com
Virtualisation by hypervisor

- We simulate the hardware below the operation system, in a so-called hypervisor aka micro-kernel
Virtualisation by hypervisor

• We simulate the hardware below the operation system, in a so-called hypervisor aka micro-kernel
Chrome software architecture uses Windows-specific features for compartmentalisation.

- (complex!) rendering engine is black box for browser kernel
- Plugins run as different processes

Internet Explorer 8 runs tabs in different processes.

Compartementalisation in Chrome

- One rendering engine per tab, plus one for trusted content (e.g., HTTPS certificate warnings)
- No access to local file system and to each other
- One browser kernel with full user privileges

- Rendering engine: Handling HTML, CSS, Javascript, XML, DOM, rendering
- Browser kernel: Cookie & passwd database, network stack, SSL/TLS, window management

- Chrome software architecture uses Windows-specific features for compartmentalisation
Compartementalise

• in code, aka modularisation,
  - using objects, classes, packages, etc.

Restrict sensitive operations to small modules, with small interfaces

• so you can concentrate efforts on quality of these modules
• eg only these have to be subjected to code reviews?
Compartementalisation – when surfing

• Use different web browsers for different types of activities, eg
  • one for e-banking & e-shopping at “trusted” websites, another for downloading movies
  • one for Google+ and another for gmail

• Better still (*defence in depth!*):
  • run these browsers under different user accounts
  • run these user accounts under different OS
Minimise attack surface

Minimise
• number of open sockets
• number of services
• number of services running by default
• number of services running with high privileges
• number of dynamic content webpages
• number of accounts with administrator rights
• number of files & directories with weak access control
• ...

OS hardening also aims to reduce the attack surface

Effectively another form of the principle of least privilege
Minimise the attack surface

- in code
  - not `public int x;`
    but `private int x;`
  - not `public void m()`
    but `package void m()`

This is applying principle of least privilege, and also reduces attack surface, from buggy or hostile code
Minimise attack surface in *time*

Examples

- Automatically log off users after n minutes
- Automatically lock screen after n minutes
- Unplug network connection if you don’t use it
- Switch off computer if you don’t use it

- On smartcards, it’s good practice to zero-out arrays that contains sensitive information (usually, decrypted information) as soon as it’s no longer needed
Fail securely

• Incorrect handling of unexpected errors is a major cause of security breaches

• Counterexamples:
  - fallback to unsafe(r) modes on failure
    • sometimes for backward compatibility
    • asking user if security settings can be lowered
  - crashing on failure, leading to DoS attack
  - leaking interesting information for an attacker

• Of course, having exceptions in a programming language has a big impact
Fail securely example

isAdmin = true; // enter Admin mode
try {
    something that may throw SomeException
}
catch (SomeException ex) {
    // should we log?
    log.write(ex.toString());
    // how should we proceed?
    isAdmin = false;
    // or should we exit?
}
Variants of failing insecurely

• information leakage
  - information revealed by error message can be useful for attacker

• ignoring errors
  - Easier in a programming language without exceptions!
    • eg forgetting to check for -1 return value in C

• misinterpreting errors

• useless errors
  - why does strncpy return an error value at all?

• handling wrong exceptions

• handling all exceptions
Failing insecurely example

Example code in Local System service in Windows

```c
ImpersonateNamedClient(someUser);
    // lower access rights to become someUser
DeleteFile(fileName);
RevertToSelf();
    // become Local System
```

What's wrong here?
• What happens if ImpersonateNamedClient fails?
Failing *insecurely* example

```java
try { ... // (1) Load XML file f from disk
    ... // (2) Use some data from f to get URI
    ... // (3) get X509 certificate
    ... // (4) access URI with certificate
} catch (Exception ex) {
    ....
}
```

What's probably/possibly wrong here?

- one catch block to handle `SecurityException`, `XMLException`, `IOException`, `FileNotFoundException`, `SocketException`, ......
Failing *insecurely* example

```java
try {
    ...
    ...
    } catch (Exception ex) {
        // do nothing
    }
```

What's possibly/probably wrong here?

- empty catch block is suspicious...
- overly broad catches is suspicious
A License Exception has been thrown.

You tried to access the developer edition from a disallowed IP (131.174). The developer edition can only be accessed from 127.0.0.1 and one additional IP address. The additional IP address is: 131.174.

Please try the following:

- Enable Robust Exception Information to provide greater detail about the source of errors. In the Administrator, click Debugging & Logging > Debugging Settings, and select the Robust Exception Information option.
- Check the ColdFusion documentation to verify that you are using the correct syntax.
- Search the Knowledge Base to find a solution to your problem.

Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.12)
Browser Gecko/20071126 Fedora/1.5.0.12-7.fc6
Firefox/1.5.0.12
Remote Address 131.174.
Referrer
Date/Time 28-Jan-09 03:23 PM
error trace of our department’s online diary

Database error: Invalid SQL: (SELECT egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM egw_cal JOIN egw_cal_dates ON egw_cal.cal_id=egw_cal_dates.cal_id JOIN egw_cal_user ON egw_cal.cal_id=egw_cal_user.cal_id LEFT JOIN egw_cal_repeats ON egw_cal.cal_id=egw_cal_repeats.cal_id WHERE (cal_user_type='u' AND cal_user_id IN (56,-135,-2,-40,-160)) AND cal_status != 'R' AND 1225062000 < cal_end AND cal_start < 1228082400 AND recur_type IS NULL AND cal_recur_date=0) UNION (SELECT egw_cal_repeats.*,egw_cal.*,cal_start,cal_end,cal_recur_date FROM egw_cal JOIN egw_cal_dates ON egw_cal.cal_id=egw_cal_dates.cal_id JOIN egw_cal_user ON egw_cal.cal_id=egw_cal_user.cal_id LEFT JOIN egw_cal_repeats ON egw_cal.cal_id=egw_cal_repeats.cal_id WHERE (cal_user_type='u' AND cal_user_id IN (56,-135,-2,-40,-160)) AND cal_status != 'R' AND 1225062000 < cal_end AND cal_start < 1228082400 AND cal_recur_date=cal_start) ORDER BY cal_start mysql

Error: 1 (Can't create/write to file '/var/tmp/#sql_322_0.MYI' ....

File: /vol/www/egw/web-docs/egroupware/calendar/inc/class.socal.inc.php

... Session halted.
### Blind SQL injection

Suppose `http://newspaper.com/items.php?id=2` results in SQL injection-prone query

\[
\text{SELECT title, body FROM items WHERE ID=2}
\]

Will we see difference response to URLs below?

1. `http://newspaper.com/items.php?id=2 AND 1=1`

What will be the result of

`../items.php?id=2 AND SUBSTRING(user,1,1) = 'a'`

The same as 1 iff `user` starts with a; otherwise the same as 2!

So we can use this to find out things about the database structure & content!
Blind SQL injection

Blind SQL injection: a SQL injection where the response itself is not interesting, but the type of the response, or lack of response, leaks information to an attacker

- **Errors** can also leak interesting information: eg for
  
  ```sql
  IF <some condition> SELECT 1 ELSE 1/0
  ```
  
  error may reveal if `<some condition>` is true

- More subtle than this, **response time** may still leak information
  
  ```sql
  .. IF(SUBSTRING(user,1,1) = 'a',
       BENCHMARK(50000, ...), null)..
  ```

  time-consuming **BENCHMARK** statement only executed if `user` starts with ‘a’
It’s hard to keep secrets

- Don’t rely on security by obscurity
  [Kerckhoffs principle]
- Don’t assume attackers don’t know the application source code, and can’t reverse-engineer binaries
  - Don’t hardcode secrets in code.
  - Don’t rely on code obfuscation

- Example
  - DVD encryption
  - webpages with hidden URLs
  - passwords in javascript code – this happens!
Promote privacy

• Privacy of users, but also of systems

• Counterexamples
  – > telnet somemachine
     Trying 123.1.2.3
     Connected to somemachine (123.1.2.3)
     Red Hat Linux release 7.0 (Hedwig)
     Kernel 2.2.16 on an i686
     login:

  - Smartcard chips still do this
Clearly assign responsibilities

At organisational level:
• eg. make one person responsible for something rather than two persons – or a whole group.

At coding level:
• make one module/class responsible for input validation, access control, ...
• for a method
  
  ```java
  public void process(String str)
  ```
  is the caller or callee responsible for checking if for instance
  ```java
  str!=null & !(str.equals(""))
  ```
But still practice defence in depth...
Identify your assumptions

- including obvious, implicit assumptions
  - these may be sources of vulnerability, and may change in long run, due to *function creep*

Examples

- laptops invalidate implicit assumption that computers don’t move past the company firewall
- assumption that user is a human may cause you to ignore possibility of brute-force password guessing
- assumption that `new LoginContext()` won’t throw an `OutOfMemoryException`
- assumption that a Java applet won’t use all CPU time
Principle of psychological acceptance

• If security mechanism is too cumbersome, users will switch it off, or find clever ways around it

• User education may improve the situation, but only up to a point

• How many security pop-ups can we expect the user to cope with, if any?
Don’t mix data & code

This is the cause of *many* problems, eg

- **traditional buffer overflow attacks**, which rely on mixing data and code on the stack
- **VB scripts in Office documents**
  - leads to attacks by hostile .doc or .xls
- **javascript in webpages**
  - leads to XSS (cross site scripting attacks)
- **SQL injection** relies on use data (user input!) as part of SQL query
Use community resources

Use google, books, webforums, etc. to learn & reuse

• learn about vulnerabilities
  - and avoid making the same mistakes
• learn about solutions and countermeasures
  - and reuse them
for the specific system, language, etc you use

Eg read information about security principles at
https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/358-BSI.html
Be reluctant to trust

- Understand and respect the chain of trust
- NB trust is transitive
- NB trust is *not* a good thing
- NB “trusted” is not the same as “trustworthy”

- Minimize Trusted Computing Base (TCB), ie that part of the system (software and hardware) that *has* to be trusted

*Exercise: define the TCB for the VM & hypervisor scenario’s on slides 24 & 25*
Ken Thompson (Reflections on trusting trust)

Backdoor in UNIX and Trojan in C-compiler revealed during Turing award lecture

1. backdoor in `login.c` of UNIX
   
   ```c
   if (name == "ken") {
       don't check password;
       log in as root;
   }
   ```

2. code in C compiler to add backdoor when recompiling `login.c`

3. code in C compiler to add code (2 & 3!) when (re)compiling a compiler
Security principles

- secure the weakest link
- defence in depth
- principle of least privilege
- minimise attack surface
- compartementalise
- secure defaults
- keep it simple
- fail securely
- promote privacy
- hiding secrets is hard
- use community resources
- be reluctant to trust
- ...

Read information about security principles at
https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/358-BSI.html
Sun tarball problem (1993)

- Every tarball (zip-file) produced on Solaris 2.0 contained fragments of the password file /etc/password

- How did this happen?
  - tar looked up some user info directly prior to producing tarball:
    - password file was loaded in heap memory for this
    - this heap memory was then released
  - then tar allocated memory for constructing the tarball
    - allocated memory was always the memory just released
    - memory not zeroed out on allocation by program or OS...

- Solution: replacing `char *buf (char*)malloc(BUFSIZE)` by `char *buf (char*)calloc(BUFSIZE)`