Software Security

Security Principles

Erik Poll

Digital Security group

Radboud University Nijmegen

TRU/e Master in Cyber Security

Security principles

- Variations of lists of security principles appear in literature

- Security vulnerabilities often exploit violations of these principles
- Good security solutions & countermeasures often follow these principles

- NB there is some *overlap* & some *tension* between principles

Security principles

- secure the weakest link
- defence in depth
- least privilige
- 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
- 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

- A web application visible through firewall may be a soft target
  - *If so: improve web application security, not the firewall?*

    The website has to be visible through the firewall

- Or are attacks on the browser the highest risk?
  - *If so: improve browser security*

- Or are employee's smartphones a higher risk than company laptops?

  (aka the problem of BYOD – Bring Your Own Device)

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 laptop or own device 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

Defence in depth example

- prevention

and

- detection

Don’t assume security can be broken;
assume that it will be, and think about detection & reaction

- This typically requires logging
  - also: logging in a way that is well-suited to inspection & analysis
- thinking about how to recovering
  - thanks to back-ups

Defence in depth: counterexample

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

- Better
  - hash passwords

  and
  - have tight access control to the file

Principle of least privilige

- Be stingy with priviliges
  - 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

  An easy way to temporarily elevate higher privileges (with `sudo` on Linux, or ‘Run as administrator’ on Windows, helps users resist the temptation to not log in as root
- Device drivers running in kernel mode in some OSs

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 privilige

- In code
  - Minimise visibility

    eg use `private` or `protected` rather than `public`
  - Expose minimal functionality in interfaces of objects, classes, packages, applications

Principle of least privilige

- 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 privilige

Applying the principle of least privilige is hard in practice!
It requires more work & discipline!

Why?

- Running as root you won't get complaints about missing file permissions,
- Users will complain about missing rights, never about having too much rights
- Compiler will complain about breaking access restrictions in code, not about access restrictions being too liberal.
  – IDE and tools can help here
- …

Keep it simple (aka economy of mechanism)

Complexity important cause of security problems

- Complexity leads to mistakes eg incorrect use or insecure configuration by users and developers
- Complexity leads to unforeseen feature interaction

Aka KISS principle (Keep It Simple & Stupid)

Simplicity vs least privilige

NB there is a fundamental conflict between

- principle of least privilege

and

- kiss principle – keep it simple

Why?

- Principle of least privilege requires very fine-grained control with expressive policies (eg. complex access control matrix)
- ... which leads to 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 all or nothing for large chunks (*compartments)*

- 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, because compartementalisation provided by typical OSs is poor!
- Partition hard disk and install OS twice

Compartementalisation – at OS level

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

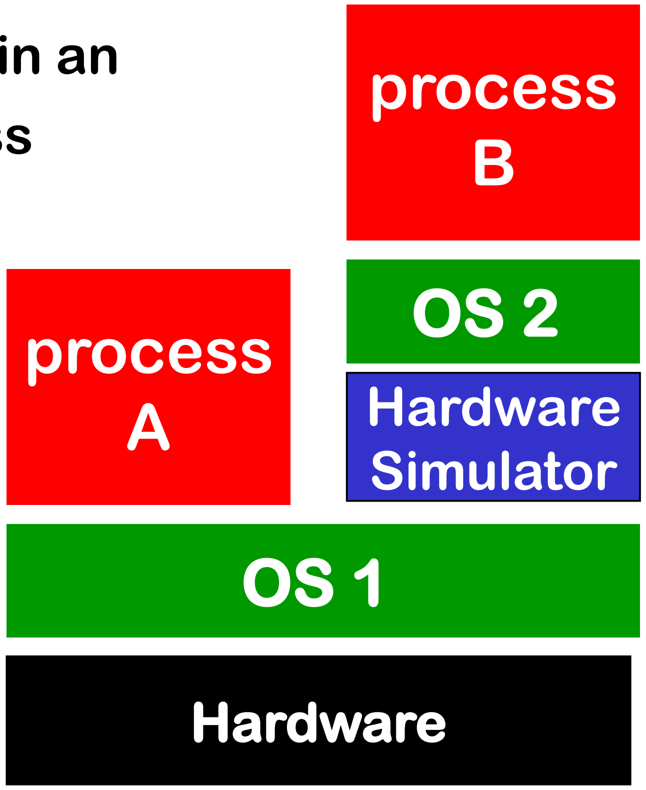
- operating system hypervisors (true microkernels)

  small, lightweight kernel, which partions hard disk & memory, to concurrently run several copies of the OS, in different compartments

  - SeL4, XEN, HyperV

Virtualisation by virtual machine

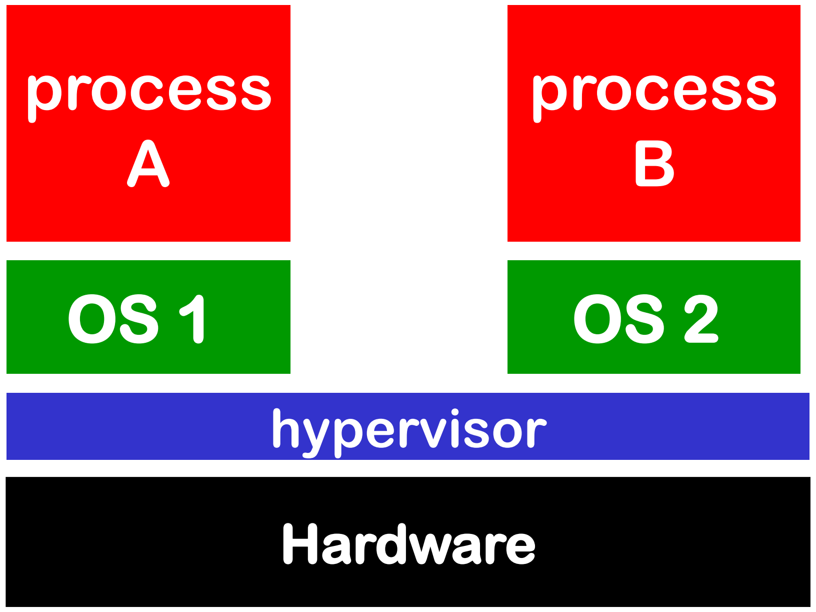
- We simulate the hardware in an OS process



Similar to Java VM, except that we simulate
the real hardware (which executes the normal CPU instructions) and not some abstract VM (which executes bytecode)

Virtualisation by hypervisor

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



See http://demo.tudos.org or http://www.osnews.com/story/15814 for a nice de

Compartementalisation in code

- ie. 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
  - So that only these have to be subjected to code reviews?

Compartmentalisation for web: CSP (Content Security Policy)

CSP is a form of sandboxing implemented in browser

- A webpage from bank.com could contain HTTP CSP header

```
 Content-Security-Policy:
default-src 'self';
img-src 'self' disney.com
child-src https://youtube.com
script-src self apis.google.com
```

to *only* allow

- images from bank.com itself or from disney.com
- embedded frames from youtube, included via https
- scripts from apis.google.com

To allow inline scripts, we'd have to add unsafe inline

CSP problems : COMPLEXITY ☹

CSP is very complex and therefore error-prone to use

- Typos in a CSP policy may mean that parts are silently ignored
- CSP distinguishes different types of content; if a policy only blocks one type but not the other, then it can be by-passed
- To help in configuring a policy, CSP can run in report-only mode. The browser than lets violations pas, but logs them, to report them to the server. Many sites run CSP in report-only mode without telling the browser to send the logs anywhere…
- If a CSP policy includes certain rich JavaScript libraries as trusted, it can be by-passed because the libraries can be abused to execute arbitrary code

  [Weichselbaum et al., CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy, CCS 2016]

  [Calzavara et al., Content Security Problems? Evaluating the Effectiveness of Content Security Policy in the Wild, CSS 2016]

Compartmentalisation for web: Sandboxing for iframes

- HTML5 introduced a sandbox option to restrict what an iframe can do
- Just turning on the sandbox with no further options

```
<iframe sandbox src="..."> </iframe>
```

imposes many restrictions, incl.

  - no JavaScript can be executed
  - pop-up windows are blocked
  - sending of forms is blocked
  - ...

- These restrictions can be lifted one-by-one, eg

```
<iframe sandbox allow-scripts allow-forms allow-pop-ups
                allow-same-origin src="..."> </ >
```

- For full list of options see
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe#attr-sandbox

Minimise attack surface

Mimimise

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

For the OS, this is also called OS hardening

This is just another instance of the principle of least privilege.

Making `public` fields `private` also minimises the attack surface (from malicious or buggy code components)

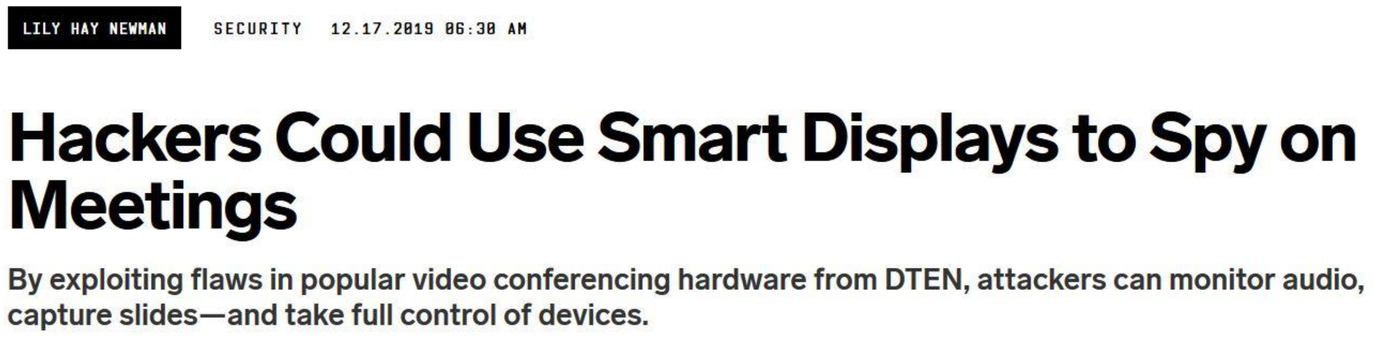
Minimise attack surface in *time*

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

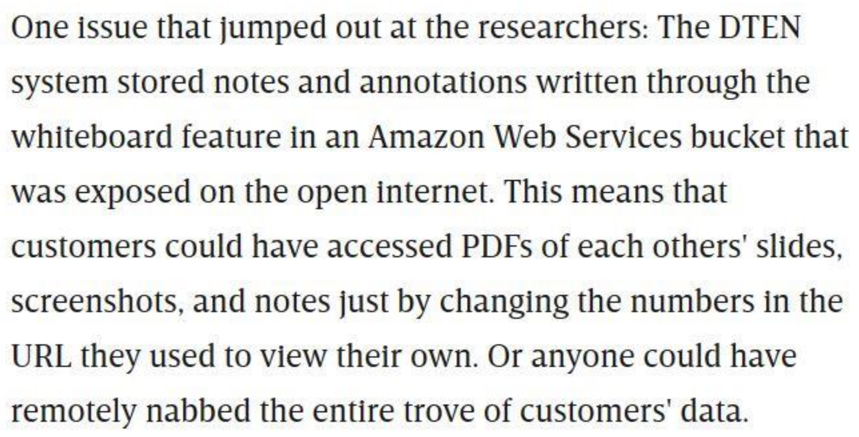
- On smartcards, it’s good practice to zero-out arrays that contains sensitive information (usually, decrypted information) as soon as it’s no longer needed

Counter examples to many of these principles


LILY HAY NEWMAN SECURITY 12.17.2019 06:30 AM

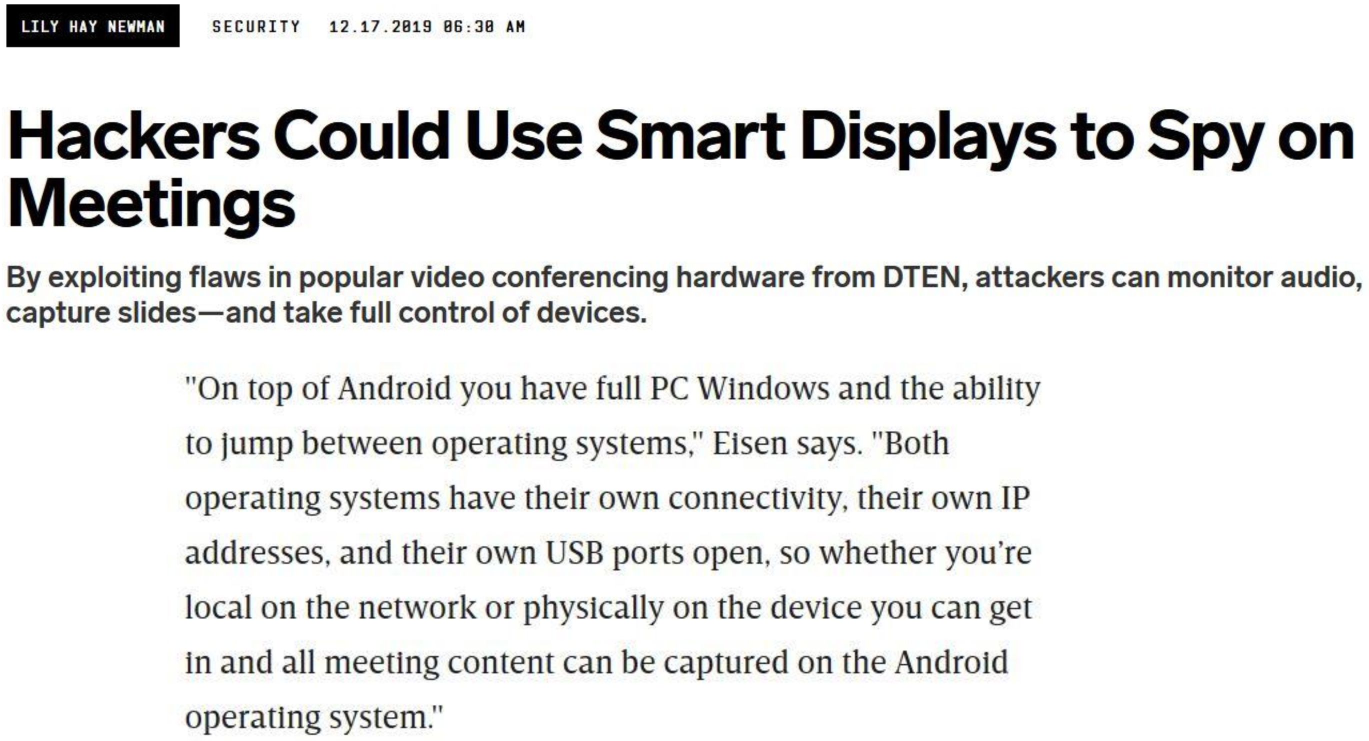
Hackers Could Use Smart Displays to Spy on Meetings

By exploiting flaws in popular video conferencing hardware from DTEN, attackers can monitor audio, capture slides—and take full control of devices.


One issue that jumped out at the researchers: The DTEN system stored notes and annotations written through the whiteboard feature in an Amazon Web Services bucket that was exposed on the open internet. This means that customers could have accessed PDFs of each others' slides, screenshots, and notes just by changing the numbers in the URL they used to view their own. Or anyone could have remotely nabbed the entire trove of customers' data.

https://www.wired.com/story/dten-video-conferencing-vulnerabilities/

Counter examples to many of these principles



LILY HAY NEWMAN SECURITY 12.17.2019 06:30 AM

Hackers Could Use Smart Displays to Spy on Meetings

By exploiting flaws in popular video conferencing hardware from DTEN, attackers can monitor audio, capture slides—and take full control of devices.

"On top of Android you have full PC Windows and the ability to jump between operating systems," Eisen says. "Both operating systems have their own connectivity, their own IP addresses, and their own USB ports open, so whether you're local on the network or physically on the device you can get in and all meeting content can be captured on the Android operating system."

https://www.wired.com/story/dten-video-conferencing-vulnerabilities/

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

- nformation 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 strncopy return an error value at all?
- handling wrong exceptions
- handling all exceptions

Failing *in*securely example

Example code in `Local System` service in Windows, which temporarily reduces its access rights

```
// running with Local System permissions
ImpersonateNamedClient(someUser);
// running with lower access rights of someUser
DeleteFile(fileName);
RevertToSelf();
// become Local System again
```

What's wrong here ?

- What happens if ImpersonateNamedClient fails?

Failing *in*securely example

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

What's probably/possibly wrong here ?

One catch block to handle SecurityException, XMLException, IOException, FileNotFoundException, SocketException, ...

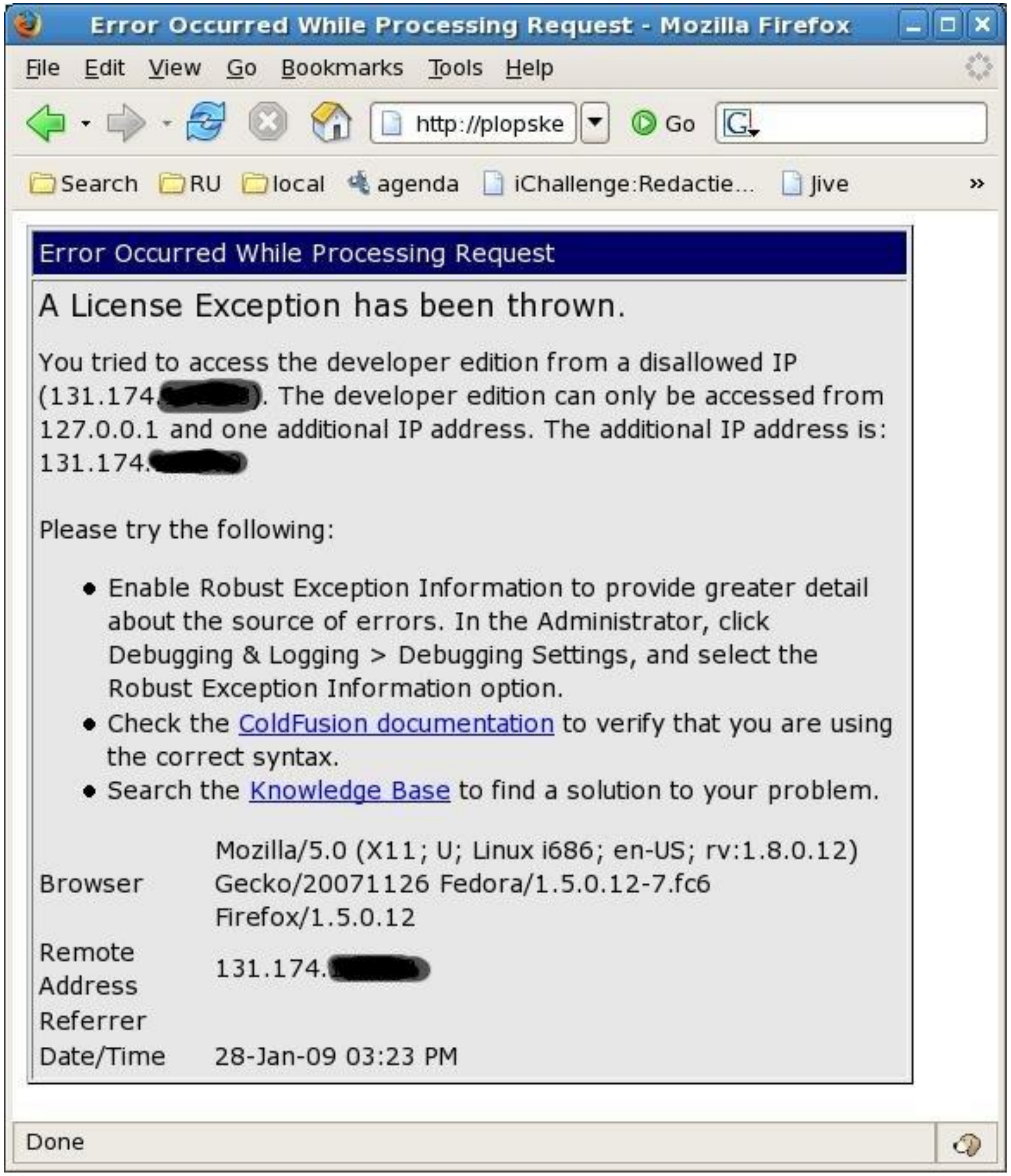
Failing *in*securely example

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

What's possibly/probably wrong here ?

- empty catch block is suspicious...
- overly broad catch, for all Exceptions, is suspicious

Error report of department online roostergenerator


Error Occurred While Processing Request - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://plopske
Go
Search RU local agenda iChallenge:Redactie... Jive
Done

Error Occurred While Processing Request

A License Exception has been thrown.

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

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.

Browser	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.12) Gecko/20071126 Fedora/1.5.0.12-7.fc6 Firefox/1.5.0.12
Remote Address	131.174.[redacted]
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.
```

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!
- But obscurity can help: it may require extra work that puts of attackers.

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)`

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 7.0S
      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

```
public void process(String str)
```

is the caller or callee responsible for checking if for instance

```
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

- assuming that `malloc` will succeed and won't return NULL
- assuming that `new SomeObject()` won't throw an `OutOfMemoryException`
- assuming that Javascript code won't try to 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)
- …

Recurring tragedy: data formats tend to grow in expressivity (and complexity) until it because code.

Of course, in the end it is hard to make any fundamental distinction about which data can be regarded as “code”

Be reluctant to trust

- Understand what you are trusting
- Minimise what you are trusting

  ie keep Trusted Computing Base (TCB) as small as possible.

TCB = part of the system that *has* to be trusted

NB

- Trust is *not* a good thing
- 'Trusted' is not the same as 'trustworthy'
- Trust is transitive – hence there often is a *trust chain*

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**
   ```
   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**

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

for the specific languages and systems that you use

Eg read the IEEE Top 10 Security Design Flaws