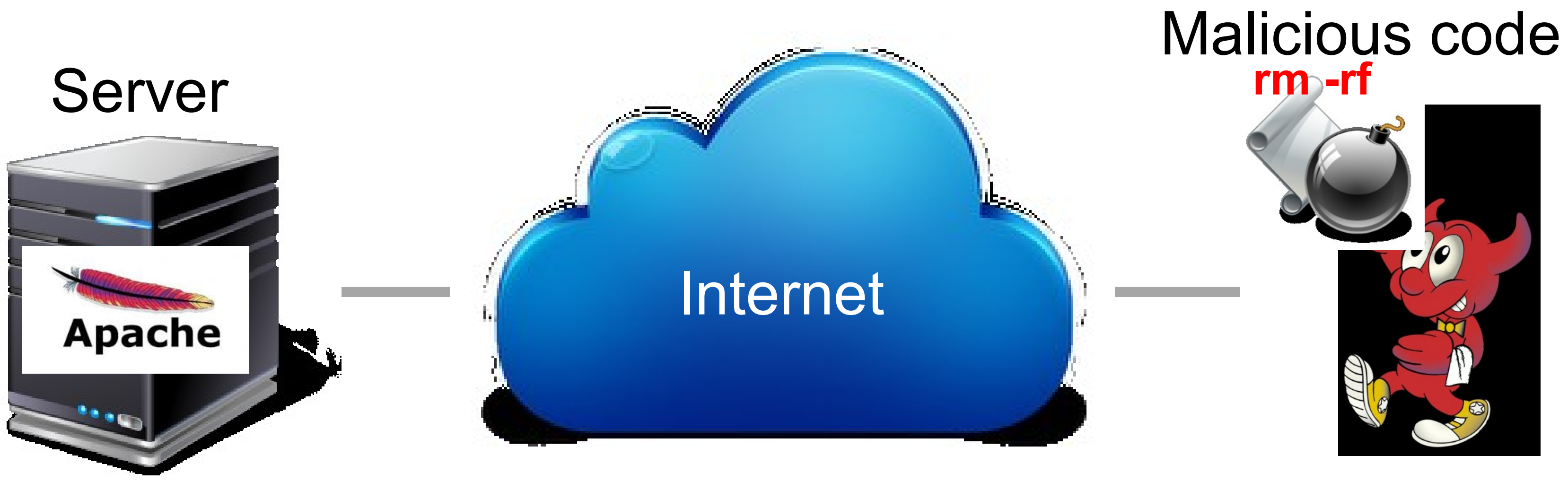
PA193 - Secure coding principles and practices

Defense in depth

Petr Švenda svenda@fi.muni.cz

Centre for Research on Cryptography and Security

What prevents malicious code on server?



Where would you attack?
Where is defense in depth applied?
How many layers are present?
Are these layers independent?

Defense in depth

- It is an general approach/concept/strategy
- You have to apply it in your concrete project
- You have to think as an attacker
  - Then select appropriate defenses/measures
- Your need to be able to find your weakest point
  - And make sure that the weakest point is strong enough

- This lecture will give you some hints

ACKNOWLEDGEMENT

Original slides made by Zdeněk Říha,
existing and new mistakes responsibility of me

Defense in depth: Definition (Wikipedia)

- Non-IT: “**defense in depth** (also known as deep or elastic defense) is a military strategy; it seeks to delay rather than prevent the advance of an attacker, buying time and causing additional casualties by yielding space.”
- IT: “**defense in depth** is an information assurance concept in which multiple layers of security controls (defense) are placed throughout an IT system. Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited that can cover aspects of personnel, procedural, technical and physical for the duration of the system's life cycle.”

Defense in depth – key features

- IT: “defense in depth is an information assurance concept in which multiple layers of security controls (defense) are placed throughout an IT system. Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited that can cover aspects of personnel, procedural, technical and physical for the duration of the system's life cycle.”

Defense in depth – application code

- Code fails. We have to take it as a fact. All code has a nonzero likelihood of containing one or more vulnerabilities
- You need to change your outlook from "my code is very good quality" to "though my code is the best it can be with today's knowledge, it likely still has security defects.“
  – Michael Howard, Attack Surface (MSDN)

Basic concepts

- Simplicity
  - Keep it simple (stupid) principle - KISS
- Compartmentalization
  - Principle of least privilege
  - Minimize needed trust
- Expect failures
  - Use more than one security mechanism (layered)
  - Secure the weakest link, Fail securely
- Work in team
  - Everyone can design defense he/she cannot breach
  - Do not reinvent wheel, Code review, tools

KISS principle

- Keep it as simple as possible
  - KISS – Keep it Simple Stupid
  - “Invented” in 1960s in aviation industry
- Simplicity
  - Less things can go wrong
  - Fewer possible inconsistencies
  - Code is easier to understand
  - When errors occur, they are easier to understand and fix
- Pay attention to interfaces and interactions

Keep it simple

Keep It Simple

- Don’t add unnecessary features
  - Additional functionality means more ways to attack
- Use simple algorithms that are easy to verify
  - Premature optimizations
  - ‘Hacks’ in code makes it
    - More difficult to understand
    - More difficult to maintain

FreeBSD-SA-11:08.telnetd

II. Problem Description

When an encryption key is supplied via the TELNET protocol, its length is not validated before the key is copied into a fixed-size buffer.

III. Impact

An attacker who can connect to the telnetd daemon can execute arbitrary code with the privileges of the daemon (which is usually the "root" superuser).

IV. Workaround

No workaround is available, but systems not running the telnet daemon are not vulnerable.

Compartmentalization

- Divide system into modules
  - Each module serves a specific purpose
  - Different modules will have different access rights
  - The access rights are related to activities
- Example application:
  1. Need access to files
  2. Reads user or network input
  3. Execute privileged instructions (under root UID)
- Real example:
  - Apache vs. suEXEC

suEXEC - example

- User “Alice” has a website including some CGI scripts in her own public_html folder, which can be accessed by http://server/~alice.
- Bob now views Alice's webpage, which requires Apache to run one of these CGI scripts.
- Instead of running all scripts as “wwwrun”, the scripts in /home/alice/public_html will be wrapped using suEXEC and run with Alice's user ID resulting in higher security and eliminating the need to make the scripts readable and executable for all users or everyone in the "wwwrun" group.

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
  - Rights added as needed and discarded after use (!)
- The original formulation from Jerome Saltzer
  - *"Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."*
- Dynamic assignments of privileges was later discussed by Roger Needham and others

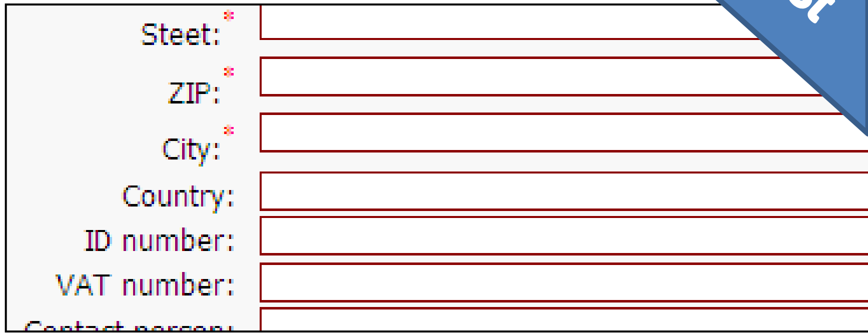
Least Privilege - example

- On UNIX-based systems binding a program to a port number <1024 requires root privilege.
  - (Let’s ignore modern ‘capabilities’ at this moment)
- Many internet servers listening on well known ports (like webserver on port 80, mailserver on port 25 etc.) need to be run with root privilege.
- As soon as the port is bound the process should drop the root privilege as it is typically not needed anymore.
- Many programs keep running with the root privileges.
  - After a successful attack against the process the attacker receives the power of root
  - “Sendmail” was well known for problems of this kind
- Visual Studio required Admin privileges for long time => developers were admins => programs were requiring admin privileges to execute

Minimize needed trust

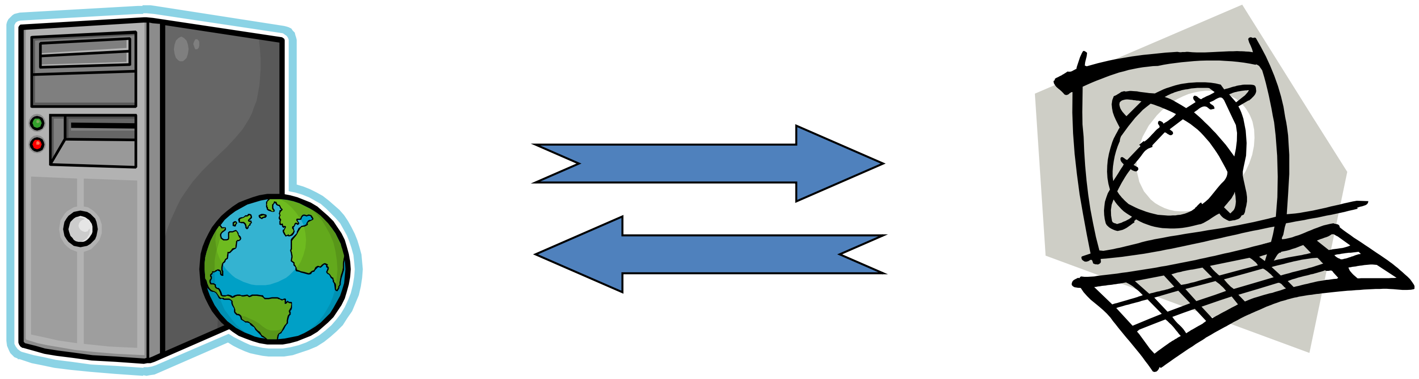
- Minimize trust relationships
- Clients, servers should not trust each other
  - all can get hacked
  - can be manipulated by users
- Trusted code should not call untrusted code
- Do not trust the input (!)
  - Separate lecture on input validation will follow
- Do not trust the communication channel
  - Use encryption, data authentication etc.
  - Separate lecture on secure channel will follow

Example: Web security

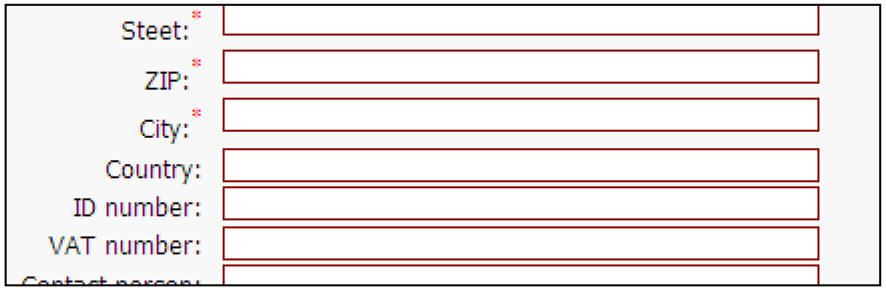


Steet: *	
ZIP: *	
City: *	
Country:	
ID number:	
VAT number:	
Contact person:	

- Web server + web client
- Simple HTML form (FORM,INPUT,TEXT,MAXLENGTH, …)
- Validity of fields checked by Javascript on clientside



Example: Web security (2)



Steet:*

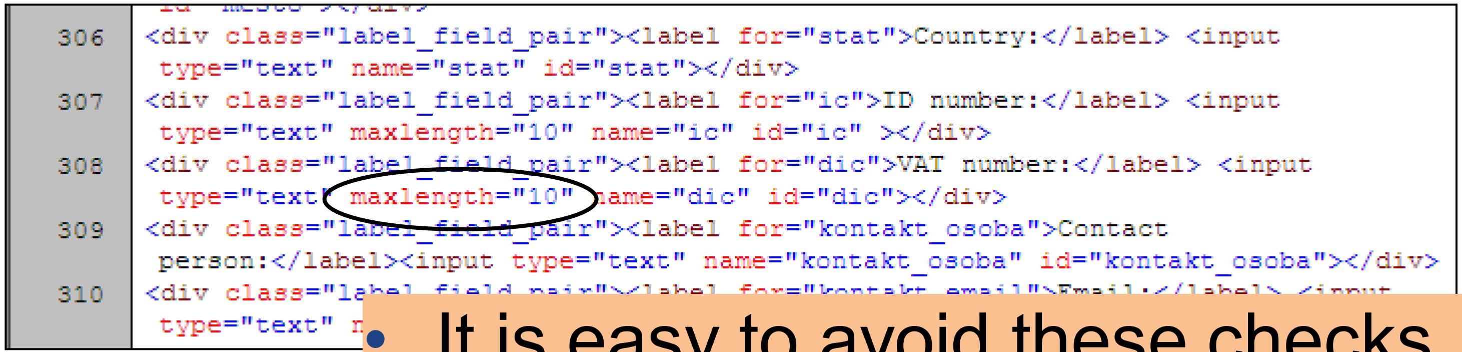
ZIP:*

City:*

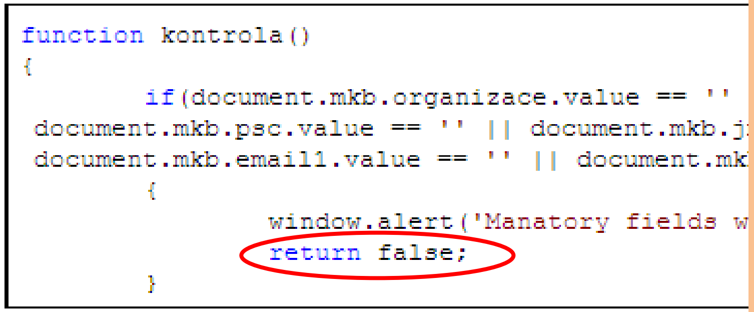
Country:

ID number:

VAT number:



```
<div class="label_field_pair"><label for="stat">Country:</label> <input
 type="text" name="stat" id="stat"></div>
<div class="label_field_pair"><label for="ic">ID number:</label> <input
 type="text" maxlength="10" name="ic" id="ic" ></div>
<div class="label_field_pair"><label for="dic">VAT number:</label> <input
 type="text" maxlength="10" name="dic" id="dic"></div>
<div class="label_field_pair"><label for="kontakt_osoba">Contact
 person:</label><input type="text" name="kontakt_osoba" id="kontakt_osoba"></div>
<div class="label_field_pair"><label for="kontakt_email">Email:</label> <input
 type="text" n
```



```
function kontrola()
{
        if(document.mkb.organizace.value == ''
 document.mkb.psc.value == '' || document.mkb.j
 document.mkb.email1.value == '' || document.mk
        {
                window.alert('Manatory fields w
                return false;
        }
```

- It is easy to avoid these checks
  - Disable Javascript
  - Send the “filled” form directly
  - Tools (e.g. python request module)

Fail defaults

- Blacklist vs. Whitelist
- Example: firewall
  - Default action is to drop packets
  - The administrator configures the firewall to allow only the packet types deemed acceptable though
- Example: input filtering
  - E.g. HTML tags in blog posts

Fail-Secure

Example - Blacklisting of HTML tags

- E.g. blocking the tags
  - *'applet', 'body', 'bgsound', 'base', 'basefont', 'embed', 'frame', 'frameset', 'head', 'html', 'id', 'iframe', 'ilayer', 'layer', 'link', 'meta', 'name', 'object', 'script', 'style', 'title', 'xml'*
- A new version of HTML arrives (e.g. HTML5)
  - New tags (like <*audio*>, <*video*>, …)
  - New attributes (like *formaction* of <*input*>,…)
- Syntax errors
  - How to recover from syntax errors

Fail-Secure

Fail-safe vs. Fail-secure

- **Fail-safe** means that a device will not endanger lives or properties when it fails
- **Fail-secure** means that access or data will not fall into the wrong hands in a failure
- Example - if a building catches fire:
  - fail-safe systems would unlock doors to ensure quick escape and allow firefighters inside
  - fail-secure would lock doors to prevent unauthorized access to the building

Fail-Secure

Failing securely (1)

- What’s wrong with the following code?

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

Fail-Secure

Failing securely (2)

- This is a more secure alternative

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

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

FreeBSD-SA-11:09.pam_ssh

I. Background

The PAM (Pluggable Authentication Modules) library provides a flexible framework for user authentication and session setup / teardown. It is used not only in the base system, but also by a large number of third-party applications.

The base system includes a module named pam_ssh which, if enabled, allows users to authenticate themselves by typing in the passphrase of one of the SSH private keys which are stored in encrypted form in the their .ssh directory. Authentication is considered successful if at least one of these keys could be decrypted using the provided passphrase.

By default, the pam_ssh module rejects SSH private keys with no passphrase. A "nullok" option exists to allow these keys.

II. Problem Description

The OpenSSL library call used to decrypt private keys ignores the passphrase argument if the key is not encrypted. Because the pam_ssh module only checks whether the passphrase provided by the user is null, users with unencrypted SSH private keys may successfully authenticate themselves by providing a dummy passphrase.

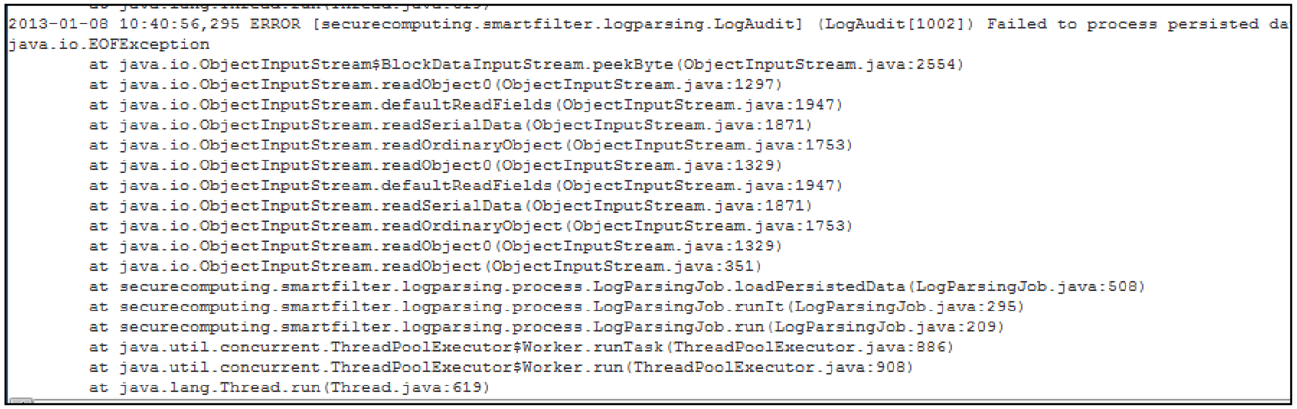
III. Impact

If the pam_ssh module is enabled, attackers may be able to gain access to user accounts which have unencrypted SSH private keys.

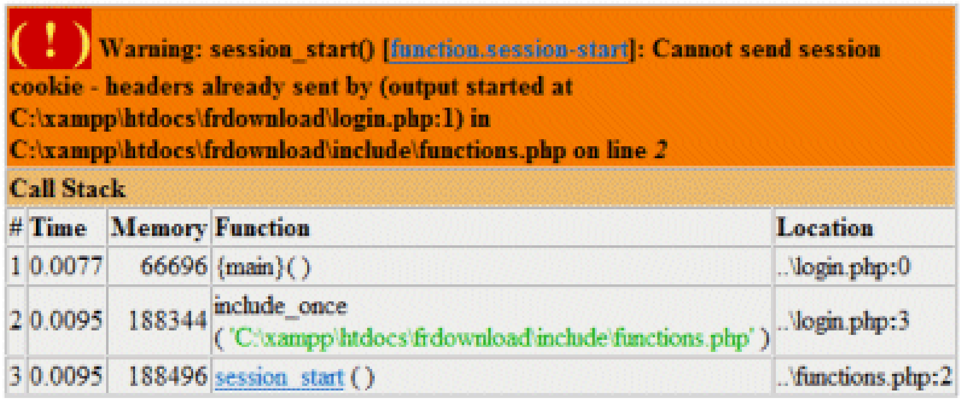
Fail-Secure

Failing securely

- Do not expose system internals even in case of errors
  - Stack traces
  - Internal errors
  - Paths



```
2013-01-08 10:40:56,295 ERROR [securecomputing.smartfilter.logparsing.LogAudit] (LogAudit[1002]) Failed to process persisted da
java.io.EOFException
        at java.io.ObjectInputStream$BlockDataInputStream.peekByte(ObjectInputStream.java:2554)
        at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1297)
        at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:1947)
        at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1871)
        at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1753)
        at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1329)
        at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:1947)
        at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1871)
        at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1753)
        at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1329)
        at java.io.ObjectInputStream.readObject(ObjectInputStream.java:351)
        at securecomputing.smartfilter.logparsing.process.LogParsingJob.loadPersistedData(LogParsingJob.java:508)
        at securecomputing.smartfilter.logparsing.process.LogParsingJob.runIt(LogParsingJob.java:295)
        at securecomputing.smartfilter.logparsing.process.LogParsingJob.run(LogParsingJob.java:209)
        at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
        at java.lang.Thread.run(Thread.java:619)
```



(!) Warning: session_start() [function.session-start]: Cannot send session cookie - headers already sent by (output started at C:\xampp\htdocs\frdownload\login.php:1) in C:\xampp\htdocs\frdownload\include\functions.php on line 2

Call Stack

#	Time	Memory	Function	Location
1	0.0077	66696	{main}( )	..\login.php:0
2	0.0095	188344	include_once ('C:\xampp\htdocs\frdownload\include\functions.php')	..\login.php:3
3	0.0095	188496	session_start ( )	..\functions.php:2

Fail-Secure

Failing securely

- Many vulnerabilities are related to
  - error handling,
  - debugging,
  - testing features,
  - error messages.
- Make sure you handle errors
  - Manual testing and review
  - Static analysis for missing return values checks
  - Fuzzing

Fail-Secure

Failing securely

- Errors as side-channel for an attacker
  - Analysis of system behavior (probing)
  - Padding oracle attack (RSA PKCS1, CBC padding)
- Test
  - Test if your system fails securely as you expect
  - There may be nontrivial consequences, relationships, …
  - Negative unit/integration tests

“Security by Obscurity” is NOT secure

- “Security by Obscurity” vs. “Open design”
- Security should not depend on secrecy of design or implementation (Kerckhoff)
- “Security by Obscurity” does not work (in long time)
  - Reverse engineering
  - Disassembler: machine code to assembly language
  - Decompiler: machine code to higher-level language
- Assume an attacker knows everything you know
  - Insider attacks are common
  - If attacker has 1-in-a-million chance, and there are a million attackers, you are out of luck

Security by Obscurity vs. Open Design

- Open design does not mean that the full source code must be available to everyone
- Logically crypto keys, passwords, … must remain secret ☺

Security by obscurity

- Examples where security by obscurity did not work
  - GSM encryption algorithms: A5/1, A5/2, …
  - WEP encryption
  - CSS encryption on DVDs
  - Mifare classic smartcards
  - Car remotes (Keeloq, VW Group immobilizer…)
    - Weak proprietary cipher, few global master keys…
- Obscurity adds additional burden to analysis
  - Good because attacker needs to overcome (short term)
  - Bad because less analysis is performed and system is almost always vulnerable after first release (long term)

Separation of Privilege

- Require multiple conditions to grant privilege
  - Separation of duty
- Failures are seen frequently
  - Edward Snowden (2013)
    - System admin, US lost classified information
  - Unauthorized trading in UBS (Kweku Adoboli, 2010)
    - Loss of 2 billion USD
  - Fraudulent trades Societe Generale (Jerome Kerviel, 2008)
    - Loss of 7.2 billion USD

Do not share (runtime resources)

- Sharing often introduced to increase performance
  - But often decrease original security
  - (virtual perimeter)
- Share the minimal number of mechanisms
  - Information can flow along shared channels
  - Covert channels
- Use isolation
  - Sandboxes
  - Virtual machines
  - Physical separation

Vulnerability Note VU#911878 (CVE-2005-0109)

Description

Hyper-Threading (HT) Technology allows two series of instructions to run simultaneously and independently on a single processor. With Hyper-Threading Technology enabled, the system treats a physical processor as two "logical" processors. Each logical processor is allocated a thread on which to work, as well as a share of execution resources such as cache memories, execution units, and buses.

Information could potentially be deduced by local users using programs capable of shared memory cache eviction analysis. Proof of concept code using timing and cache eviction analysis techniques have demonstrated that cryptographic keys can be deduced on Intel processors with Hyper-Threading technology (HTT) . It is likely that similar techniques could be employed on other processor architectures that support simultaneous multithreading.

This vulnerability is applicable to many operating system platforms running on a hardware platform that supports simultaneous multithreading (Intel HTT in particular).

Vulnerability Note (CVE-2015-0565)

- DRAM: privilege escalation via Row Hammer

Description

The DRAM memory is used by most recent computers.

Researchers found that reading at a memory address can trigger a bit flip in a page located near. To ease this attack, DRAM without ECC (Error Correcting Code) was used, and the processor cache was flushed with the CLFLUSH assemble instruction.

A local attacker can therefore alter the content of DRAM memory, in order to corrupt data. If these data are located in a page used by a privileged process, this attack can lead to a privilege escalation.

Human Acceptability

- Security mechanisms complicate accessing resources and performing duties
  - Hide complexity introduced by security mechanisms to users
- Chernobyl nuclear power plant
  - Some safety mechanisms disabled/bypassed
- Unpopularity of User Account Control (UAC) in Microsoft Vista
  - Number of alerts reduced in subsequent Windows versions
- Certificate validation errors in Web browsers

Human nk

KeePass+Dropbox
LastPass
1Password
MozillaSync
…

Google:
Sfdlk2c&432mo%
Skype:
*(&21mefd872!&

More than 60% of users have weak passwords

Use proven

Don’t reinvent the wheel

- Use standard, tested components
- Use SW, libraries, designs, protocols that others are successfully using
- In particular use standard crypto and crypto libraries
  - Use standard good random number generators
  - Use standards parsers etc.
  - Don’t implement your own cryptography
- Bad examples
  - Bad use of crypto: 802.11b
  - Protocols without expert review: early 802.11i
  - Ad-hoc changes to OpenSSL key generation: Debian (2008)

Avoid High-Risk Technologies

- Some technologies are considered more insecure than others
  - This includes programming languages, services and protocols
- Statistics of published vulnerabilities
  - E.g. comparison of web browsers
- If the technology must be used, integrate security wrappers, application firewalls etc.
- Java VM is a hot target these days
  - Java as a language has always been considered a bit more secure language than C/C++

Use proven

Learn from Mistakes

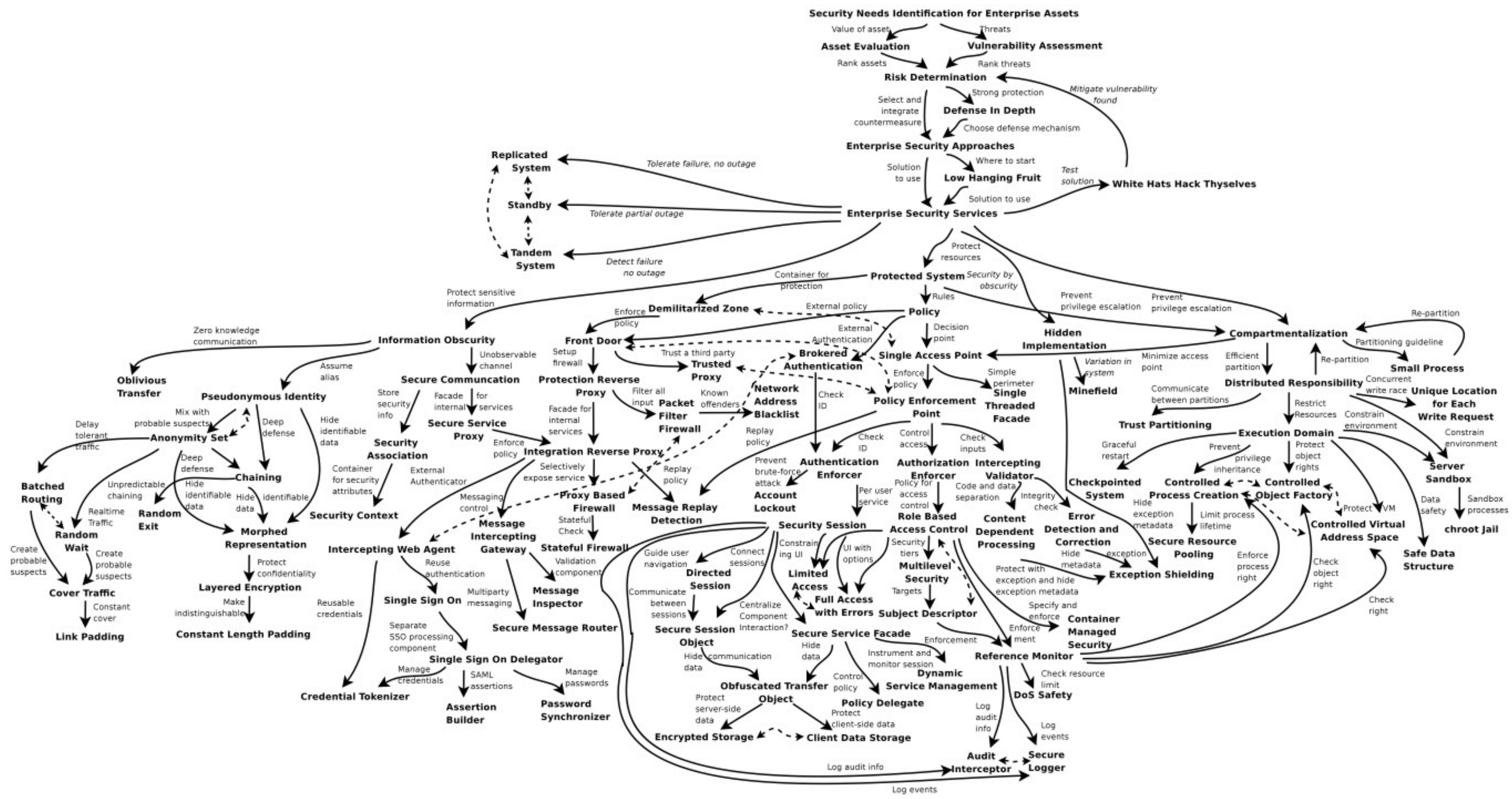
- Learn from your mistakes and mistakes of others
  - How did the security error occur?
  - Is the same bug repeated in the code?
  - How could it have been prevented?
    - Change your education/practices to avoid repeating the same errors
- Examine mistakes/bugs of your "competitors" (!)

SOFTWARE DESIGN PATTERNS

Security patterns

- Applying the idea of Software design pattern to the area of computer security
- Aim is to achieve some IT security goals
  - Like confidentiality, integrity, … or some specific goal
- Comprehensive catalogs of security patterns exist
  - E.g. Munawar Hafiz. Security Pattern Catalog
  - http://www.munawarhafiz.com/securitypatterncatalog/index.php

Security Pattern Catalog



Source: http://www.munawarhafiz.com/securitypatterncatalog/index.php

Example: defense in depth

- Problem
  - A security failure in a compartment can cause the whole system to crash. How can we make the system robust against security failures?
- Solution
  - Employ security measures at multiple layers of an application and throughout its operating environment. defense In Depth is more a security principle. In fact this is considered to be the core security principles for system architecture.
- Known Uses
  - qmail does not employ only one security mechanism, rather it has security solutions built in different levels of architecture.
- Source
  - Hafiz et. al.
- Tags
  - Deep defense

http://www.munawarhafiz.com/securitypatterncatalog/patterns.php?name=defense%20in%20Depth

The SD^3 Security Framework (Microsoft)

SD^3

Secure by Design
- Secure architecture and code
- Threat analysis
- Vulnerability reduction

Secure by Default
- Attack surface area reduced
- Unused features turned off by default
- Minimum privileges used

Secure in Deployment
- Protection: Detection, defense, recovery, management
- Process: How to guides, architecture guides
- People: Training

Summary

- Never assume impenetrable defense, completely secure code, will-never-happen situations…
  – Ask yourself: What will happen if defense fails?
- Defense in depth is general technique
  – Tries to remove single point of failure
- Proper secure coding is one layer of defense
  – Or better make it multiple layers
- Good design supports defense in depth!

Questions