Secure programming techniques and approaches I

Defence in depth

PA193 – Secure coding

You need this.

Petr Švenda

Zdeněk Říha

Faculty of Informatics, Masaryk University, Brno, CZ

CROCS

Centre for Research on Cryptography and Security

Defence in depth

- Code fails. We have to take it as a fact. All code has a nonzero likelihood of containing one or more vulnerabilities.
  - We have seen buffer overflow examples
- 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)

Defence in depth: Definition (Wikipedia)

- Non-IT: "**Defence in depth** (also known as deep or elastic defence) 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: "**Defence in depth** is an information assurance concept in which multiple layers of security controls (defence) 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."

Defence in depth

- It is an approach/concept/strategy
- You have to apply it in your concrete project
- This lecture will give you some hints
- You have to select appropriate measures
- You have to think as an attacker

Basic concepts

- Simplicity
  - Less things can go wrong
  - Fewer possible inconsistencies
  - Code is easier to understand
- Restriction
  - Minimize access (rights)
  - Inhibit communication

Basic concepts in more details

- Simplicity
  - keep it simple (stupid) - KISS
- Compartmentalization
  - Principle of least privilege
  - Minimize needed trust
- Defence in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Work in team
  - Do not reinvent wheel
  - Code review

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:
  1. Access to files
  2. Read 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.
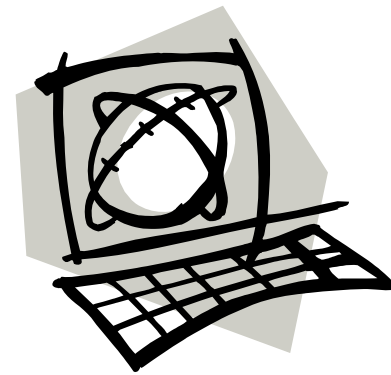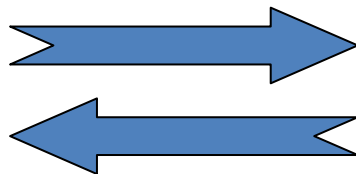
Least Privilege - example

- 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 priviledge.
- As soon as the port is bound the process should relinquish 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

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

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

Example: Web security (2)

Example: Web security (3)

- The server cannot trust that the input received from the web browser will be correct with respect to the limitations specified
  - E.g. MAXLENGTH attribute of the INPUT fields
  - E.g. values will be check by the Javascript functions
- It is easy to avoid these checks
  - Disable Javascript
  - Send the "filled" form directly
  - Tools (e.g. python request module)

Fail-Safe Defaults

- Default action is to deny access
- Blacklist & 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

Example - Blacklisting of HTML

- 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 attribues (like *formaction* of *<input>*,…)

- Syntax errors
  - How to recover from syntax errors

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, while fail-secure would lock doors to prevent unauthorized access to the building.

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.
}
```

Failing securely (2)

- This is a more secure alternative.
- Got it now?

```
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-01:56

II. Problem Description

The addition of a flawed check for a numeric result during reverse DNS lookup causes tcp_wrappers to skip some of its sanity checking of DNS results. These sanity checks are only enabled by the 'PARANOID' ACL option in the configuration file, and simply weaken the 'PARANOID' host checks to the level of assurance provided by the regular host ACLs.

III. Impact

An attacker that can influence the results of reverse DNS lookups can bypass certain tcp_wrappers PARANOID ACL restrictions by impersonating a trusted host. Such an attacker would need to be able to spoof reverse DNS lookups, or more simply the attacker may be the administrator of the DNS zone including the IP address of the remote host.

FreeBSD-SA-01:56

- The patch that is fixing the bug:
  http://ftp.sunet.se/pub/security/vendor/freebsd/patches/SA-01:56/tcp_wrappers.patch

```
--- contrib/tcp_wrappers/socket.c 2000/09/25 00:41:55  1.5
+++ contrib/tcp_wrappers/socket.c           2001/07/04 20:16:18  1.6
@@ -222,7 +222,7 @@
            hints.ai_family = sin->sa_family;
            hints.ai_socktype = SOCK_STREAM;
            hints.ai_flags = AI_PASSIVE | AI_CANONNAME | AI_NUMERICHOST;
-          if ((err = getaddrinfo(host->name, NULL, &hints, &res0) == 0)) {
+          if ((err = getaddrinfo(host->name, NULL, &hints, &res0)) == 0) {
                freeaddrinfo(res0);
                tcpd_warn("host name/name mismatch: "
                              "reverse lookup results in non-FQDN %s",
```

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.

Failing securely

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

```
                    at java.lang.Thread.run(Thread.java:619)
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

Do not expose system internals in case of errors

WordPress database error:
[Table './serkey/posts' is marked as crashed and should be repaired]
SELECT MAX(id) FROM posts;

WordPress database error:
[Table './serkey/posts' is marked as crashed and should be repaired]
SELECT * FROM posts WHERE id IN ('');

Failing securely

- – Many vulnerabilities are related to
  - • error handling,
  - • debugging,
  - • testing features,
  - • error messages.
- – Make sure you handle errors
- – **Test**
  - • **Test if your system fails securely as you expect**
  - • There may be nontrivial consequences, relationships, …

KISS principle

- Keep it as simple as possible
  - KISS – Keep is simple stupid
  - "Invented" in 1960s in aviation industry
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Pay attention to interfaces and interactions

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 make 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.

CR⚙CS

KISS principle – Best practices

- Break down your tasks into sub tasks that you think should take no longer than 4-12 hours to code.
- Break down your problems into many small problems. Each problem should be able to be solved within one or a very few classes.
- Keep your methods small, each method should never be more than 30-40 lines. Each method should only solve one little problem.
- Solve the problem, then code it. Not the other way around.
- Test driven development
  – Prepare tests first

Mediation is difficult

- Check permissions at every access
- Quite often done only with the first action
  - File open
- If permissions change after the access could be unauthorized
- Mediator with higher privileges
  - Kernel
  - Services (daemons running as root)
  - …

FreeBSD-SA-08:03.sendfile

II. Problem Description

When a process opens a file (and other file system objects, such as directories), it specifies access flags indicating its intent to read, write, or perform other operations. These flags are checked against file system permissions, and then stored in the resulting file descriptor to validate future operations against.

The sendfile(2) system call does not check the file descriptor access flags before sending data from a file.

III. Impact

If a file is write-only, a user process can open the file and use sendfile to send the content of the file over a socket, even though the user does not have read access to the file, resulting in possible disclosure of sensitive information.

"Security by Obscurity" is NOT secure

- "Security by Obscurity" vs. "Open design"
- Security should not depend on secrecy of design or implementation
- "Security by Obscurity" does not work
  - Reverse engineering
  - Disassembler: machine code to assembly language
  - Discomplier: 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

Separation of Privilege

- Require multiple conditions to grant privilege
  - Separation of duty
- Failures are seen frequently
  - Edward Snowden (2013)
    - US lost classified information
    - Now asylum seeker in Russia
  - 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

- Share the minimal number of mechanisms
  - Information can flow along shared channels
  - Covert channels
- Use isolation
  - Virtual machines
  - Sandboxes

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).

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

CR⊙CS

Don't reinvent the wheel

- Use standard, tested components
- Use SW, libraries, designs, protocols that other 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.
- JVM is a hot target these days
  - Java as a language has always been considered a bit more secure language than C/C++
- Early versions of PHP, Flash, Silverlight, ….

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" (!)

Secure the weakest link

- Think about possible attacks
  - What want attackers achieve?
  - How can they attack your system?
  - What do they need to succeed?
- Find weakest link
  - Analyze the ways to attack the system
    - The security analysis
  - Improve the security of the weakest link

The weakest link

- Encryption example
  - The system encrypts data
  - Encryption is done in a standard crypto library
    - The library will typically not be the weakest link
  - Data is stored in encrypted form
  - The weakest link will typically be centered around the cryptographic key / password
    - How and where is the passport stored?
    - How is it processed during the data encryption/decryption?

Software design pattern

- SW design pattern
  - General solution to a standard problem
  - The problem is common and being solved frequently
  - The solution is general and can be reused
- The aim is to speed up the development process and to avoid issues that can be recognized later (or too late).
- Design pattern is not code
  - Design pattern ≠ code reuse

Software design pattern

- Examples
  - Computational design patterns
    - Identify key computations
  - Execution patterns
    - Execution of stream of tasks & synchronization
  - Implementation strategy patterns
   - Program organization and data structures

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

Examples of security patterns

- Security patterns for highly available systems
  - Check pointed system
    - Replication and recovery from component failure
  - Standby pattern
    - Resuming the service of a failing component
  - Comparator-checked fault tolerant system
    - monitoring the failure free behavior of a component
  - Replicated system
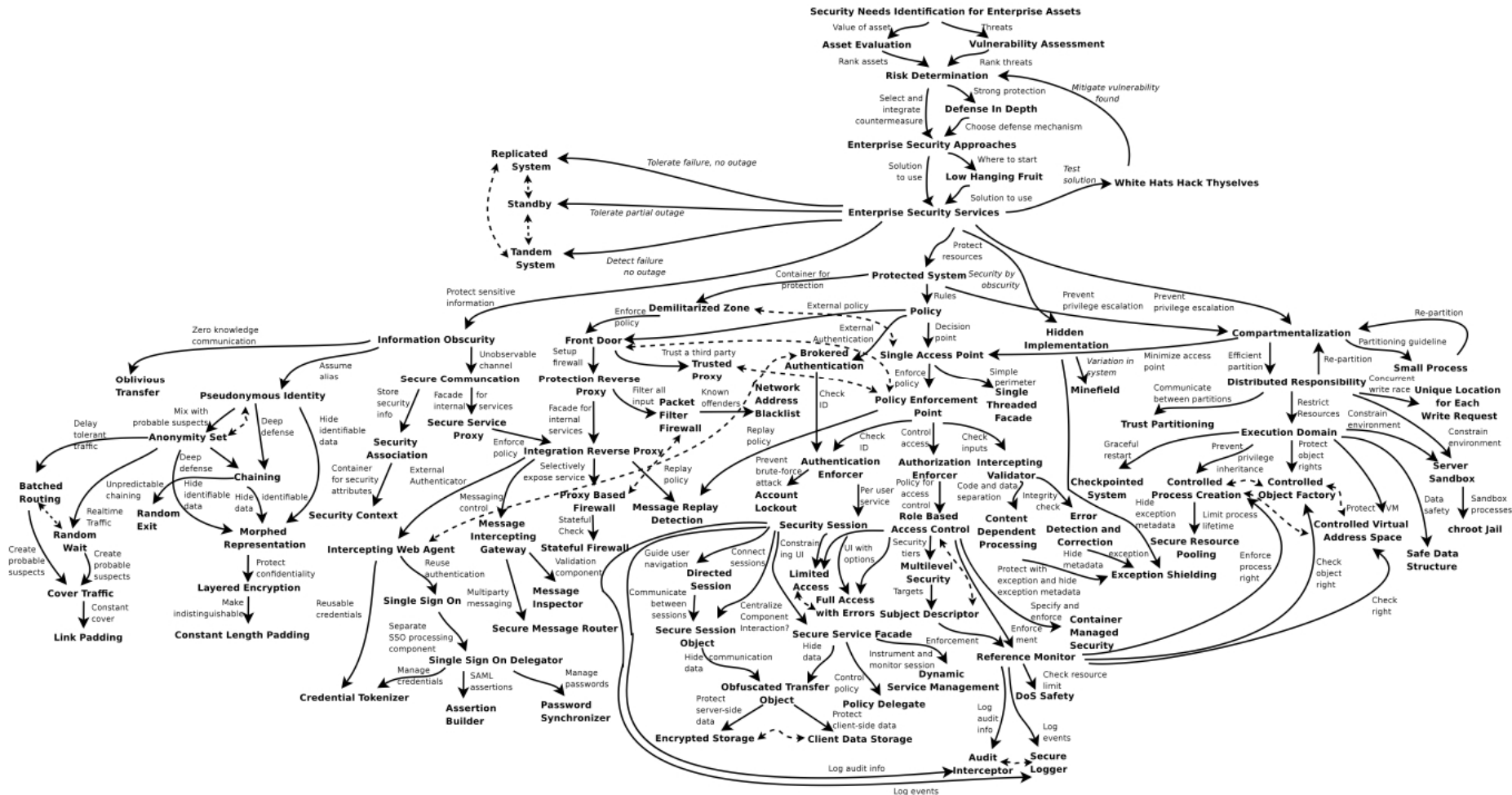   - The use of redundant components, load balancing, …

Security Pattern Catalog

Source: http://www.munawarhafiz.com/securitypatterncatalog/index.php
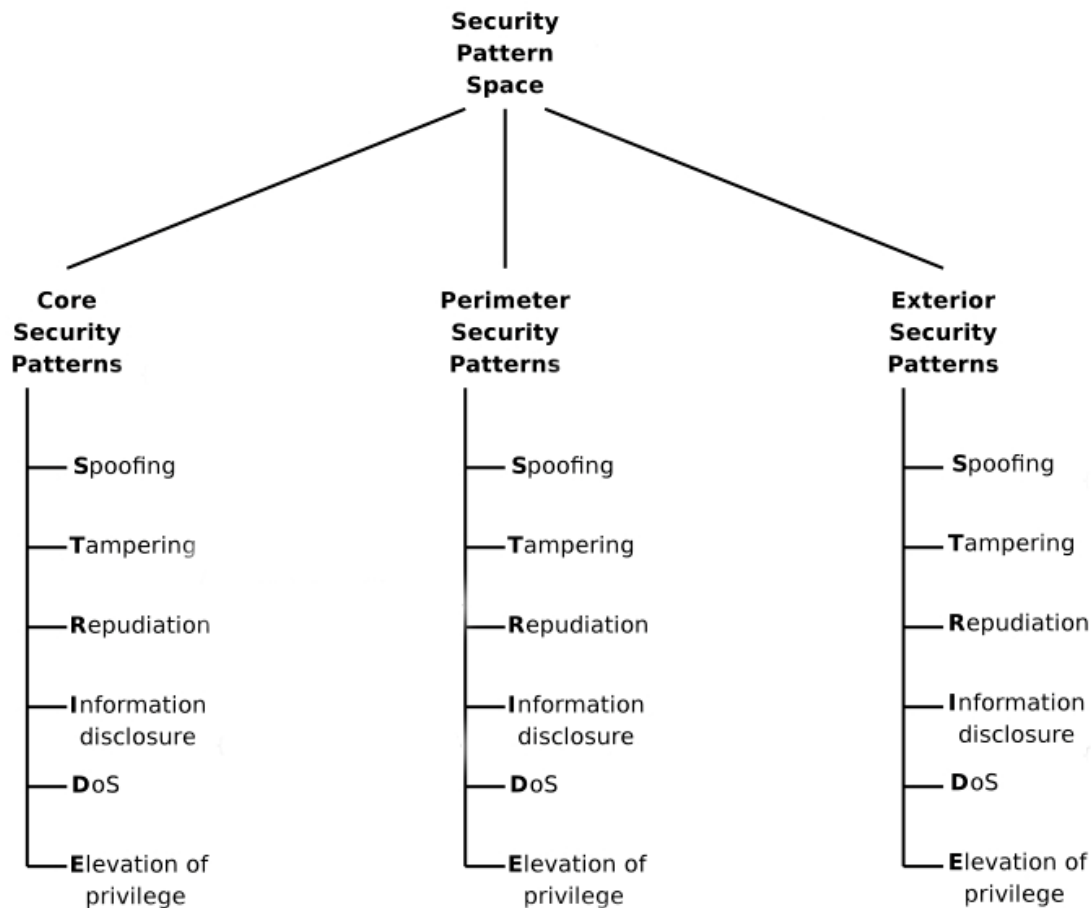
Security Pattern Catalog

Security Pattern Catalog - Example

- 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

OWASP Top 10 of WEB application vulnerabilities

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Session Management
4. Cross Site Scripting (XSS) Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management
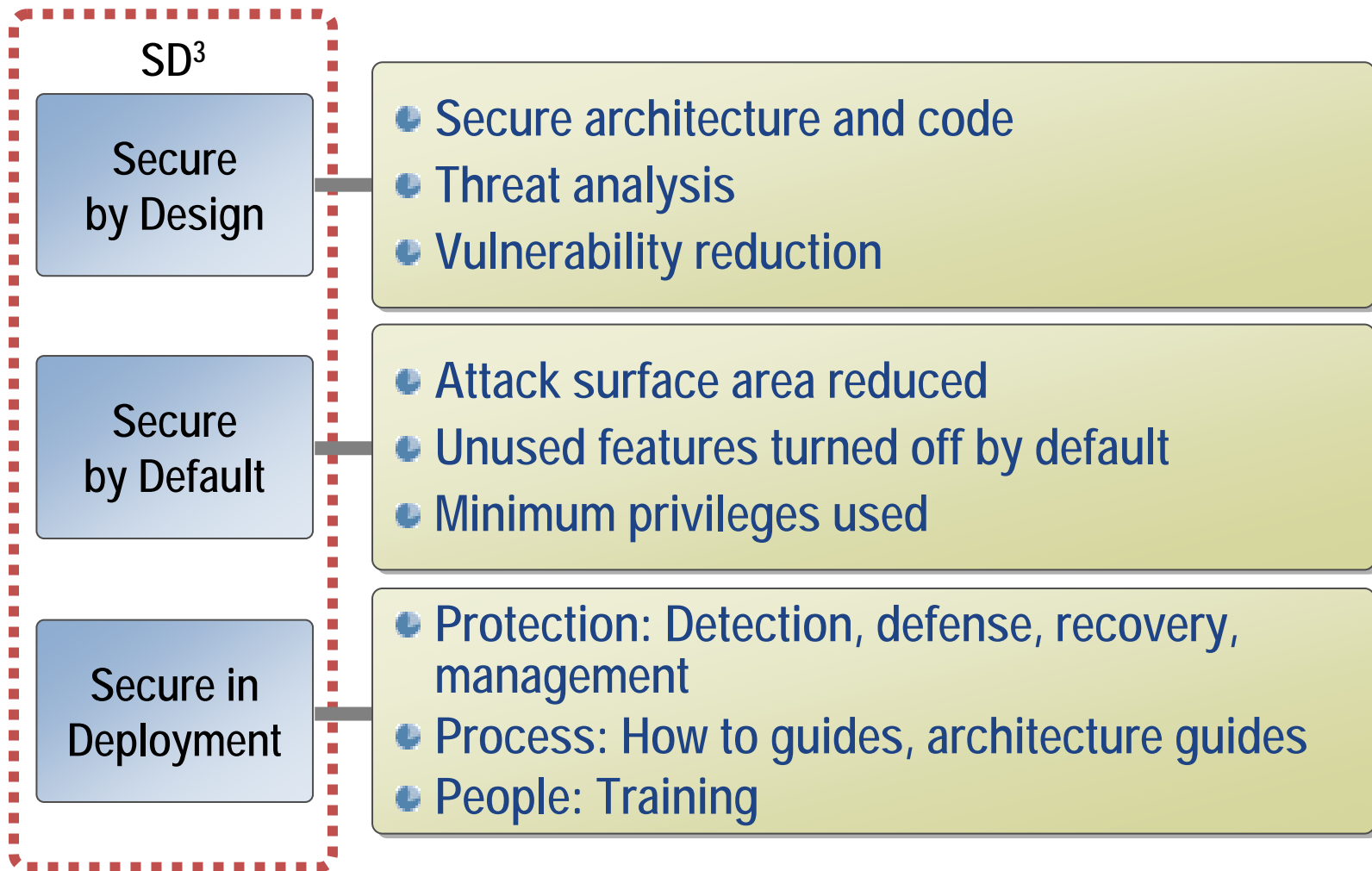
Security in Software Development Life Cycle
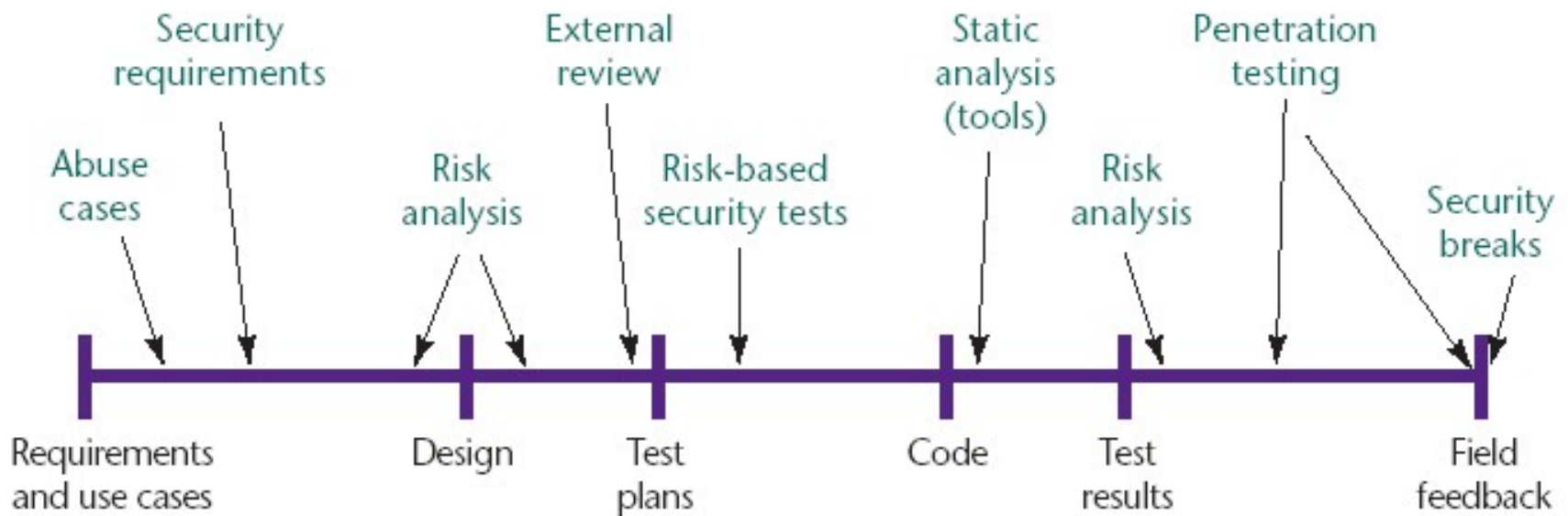
[Source: Gary McGraw, *Software security*, Security & Privacy Magazine, IEEE, Vol 2, No. 2, pp. 80-83, 2004.]