



Welcome to

# 1. Overview of Computer Security

Security in Web Development Elective, KEA

Henrik Kramselund he/him han/ham xhek@kea.dk @kramse  

Slides are available as PDF, kramse@Github

1-overview-computer-security-in-web.tex in the repo security-courses



2

## Plan for today

### Subjects

- Initial Overview of Software Security
- Confidentiality, Integrity and Availability
- Human Issues

### Exercises

- Linux introduction and familiarisation
- Run programs, servers and applications, how to
- Find the level of programming in the group
- Quick port scan intro with Nmap, more to come

Danish Criminal Code

Straffelovens paragraf 263 Stk. 2. Med bøde eller fængsel indtil 1 år og 6 måneder straffes den, der uberettiget skaffer sig adgang til en andens oplysninger eller programmer, der er bestemt til at bruges i et informationssystem.

Hacking can result in:

- Getting your devices confiscated by the police
- Paying damages to persons or businesses
- If older getting a fine and a record – even jail perhaps
- Getting a criminal record, making it hard to travel to some countries and working in security
- Fear of terror has increased the focus – so dont step over bounds!

Asking for permission and getting an OK before doing invasive tests, always!

### WAS preface and chapter 1: The History of Software Security

Quote, Summary page xxx,

This is a multifaceted book designed to be beneficial for those with both offensive and defensive security interests. It is also written to make it easily accessible for any type of developer or administrator with a sufficient web programming background (client + server) to understand and use.

Web Application Security walks you through a number of techniques used by talented hackers and bug bounty hunters to break into applications, then teaches you the techniques and processes you can implement in your own software to protect against such hackers.

This book is designed to be read from cover to cover, but can also be used as an on-demand reference for particular types of recon techniques, attacks, and defenses against attacks.

## par·a·noi·a

/ˌparəˈnoiə/ ⓘ

noun

noun: **paranoia**

1. a mental condition characterized by delusions of persecution, unwarranted jealousy, or exaggerated self-importance, typically elaborated into an organized system. It may be an aspect of chronic personality disorder, of drug abuse, or of a serious condition such as schizophrenia in which the person loses touch with reality.  
*synonyms:* [persecution complex](#), [delusions](#), [obsession](#), [psychosis](#) [More](#)
- suspicion and mistrust of people or their actions without evidence or justification.  
"the global paranoia about hackers and viruses"

### Origin

GREEK

para  
irregular

GREEK

MODERN LATIN

GREEK

noos  
mind

paranoos  
distracted

paranoia  
early 19th cent.

[More](#)

Source: google paranoia definition

From the definition:

suspicion and mistrust of people or their actions **without evidence or justification. the global paranoia about hackers and viruses**

It is not paranoia when:

- Criminals sell your credit card information and identity theft
- Trade infected computers like a commodity
- Governments write laws that allows them to introduce back-doors - and use these
- Governments do blanket surveillance of their population, implement censorship, threaten citizens and journalist

You are not paranoid when there are people actively attacking you!

I recommend we have appropriate paranoia (DK: passende paranoia)

# Overlapping Security Incidents

New data breaches nearly every week, these from danish news site version2.dk

Problem, we need to receive data from others

Data from others may contain malware

Have a job posting, yes

- then HR will be expecting CVs sent as .doc files

**Flere detaljer i gigantisk hotel-hack: 5,25 mio. ukrypterede pasnumre taget**

Jakob Møllerhøj | Sikkerhed | 07. jan 2019

3

**7,6 millioner spillerkonti lækket fra populært onlinespil**

Niels Møller Kjemtrup | Sikkerhed | 07. jan 2019

2

**Største læk i tysk historie: Politikeres og kunstneres data smidt på nettet**

Morten Egedal | Sikkerhed | 04. jan 2019

2

**Gentleman-aftale mellem politiske partier skal danne mur mod datalæk, hacking og fake news**

Louise Holst Andersen | Sikkerhed | 04. jan 2019

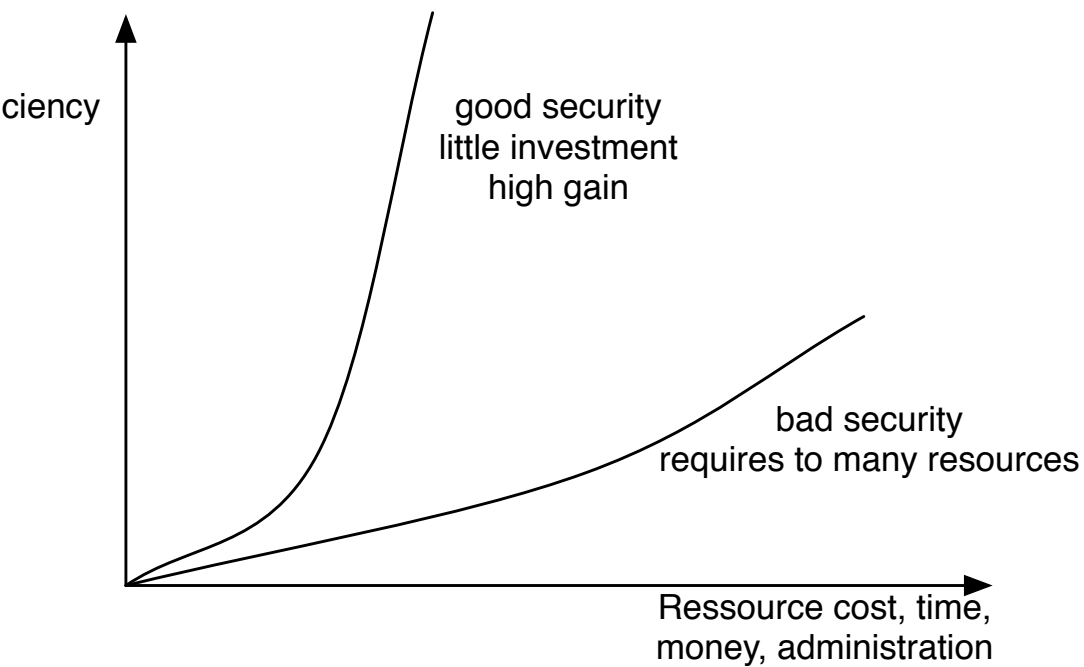
12

**Boligfond beklager læk af følsomme persondata: En menneskelig fejl**

Sikkerhed | 28. dec 2018

6





You always have limited resources for protection - use them as best as possible

### **Keep updated!**

- read web sites, books, articles, mailing lists, Twitter, ...

### **Always have a chapter on security evaluation**

- any process must have security, like RFC Request for Comments have

### **Incident Response**

- you WILL have security incidents, be prepared

### **Write down security policy**

- including software and e-mail policies

Use technology

Learn the technology - read the freaking manual

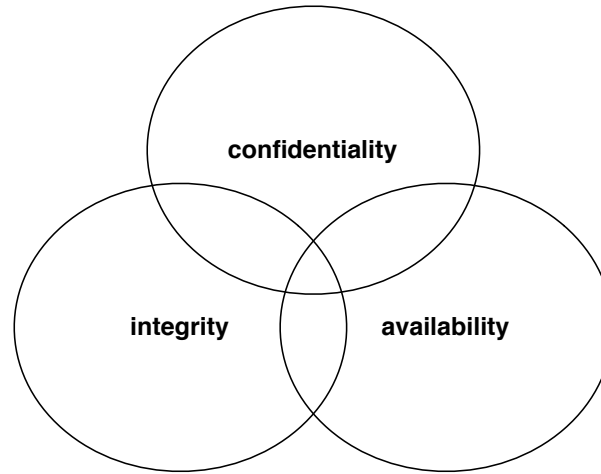
Think about the data you have, upload, facebook license?! WTF!

Think about the data you create - nude pictures taken, where will they show up?

- Turn off features you don't use
- Turn off network connections when not in use
- Update software and applications
- Turn on encryption: IMAPS, POP3S, HTTPS also for data at rest, full disk encryption, tablet encryption
- Lock devices automatically when not used for 10 minutes
- Dont trust fancy logins like fingerprint scanner or face recognition on cheap devices

But which features to disable? Let the security principles guide you

# Confidentiality, Integrity and Availability



We want to protect something

Confidentiality - data kept a secret

Integrity - data is not subjected to unauthorized changes

Availability - data and systems are available when needed

## Security is a process

Remember:

- what is information and security?
- Data kept electronically
- Data kept in physical form
- Dont forget the human element of security

Incident Response and Computer Forensics reaction to incidents

Good security is the result of planning and long-term work

Security is a process, not a product, Bruce Schneier

Source for quote: [https://www.schneier.com/essays/archives/2000/04/the\\_\\_process\\_of\\_secur.html](https://www.schneier.com/essays/archives/2000/04/the__process_of_secur.html)

## Work together



Team up!

We need to share security information freely

We often face the same threats, so we can work on solving these together

Prevention - means that an attack will fail

Detection - determine if attack is underway, or has occurred - report it

Recovery - stop attack, assess damage, repair damage

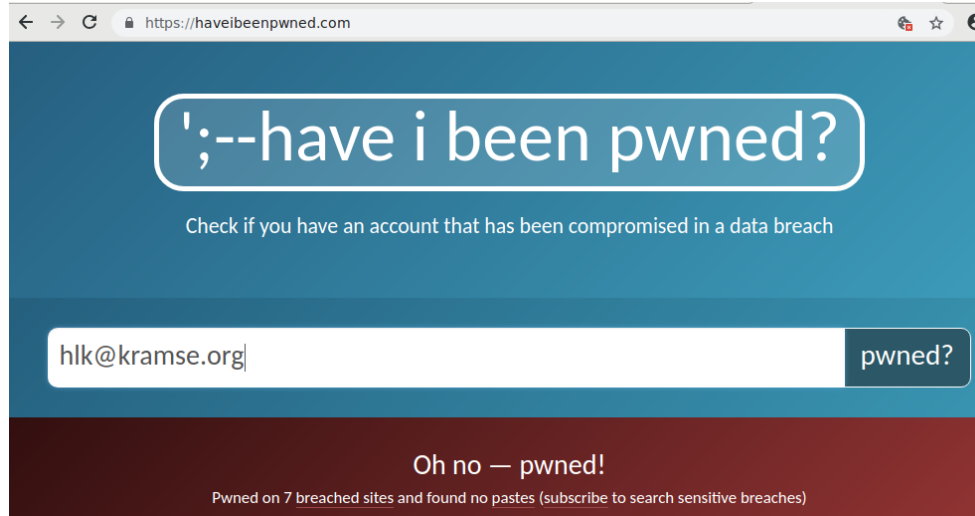
**Definition 1-1.** A *security policy* is a statement of what is, and what is not, allowed.

**Definition 1-2.** A *security mechanism* is a method, tool or procedure for enforcing a security policy.

Quote from Matt Bishop, Computer Security section 1.3



# Your data has already have been owned by criminals

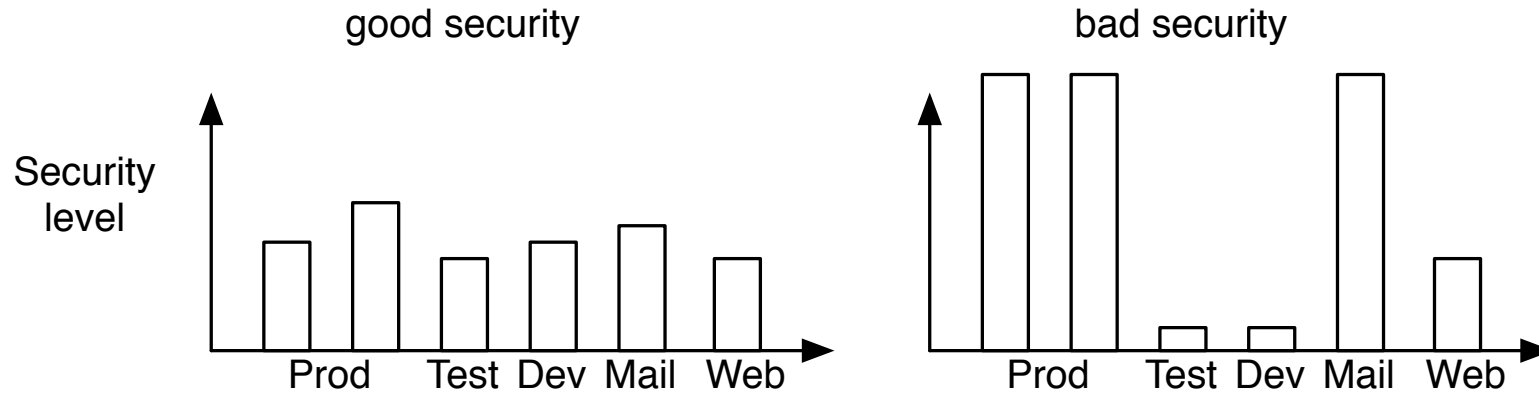


Your data is already being sold, and resold on the Internet

Stop reusing passwords, use a password safe to generate and remember

Check you own email addresses on <https://haveibeenpwned.com/>

Go ahead try the web site - hold up your hand if you are in those dumps



Better to have the same level of security

If you have bad security in some part - guess where attackers will end up

Hackers are not required to take the hardest path into the network

Realize there is no such thing as 100% security

## Human Issues and Organizational Problems

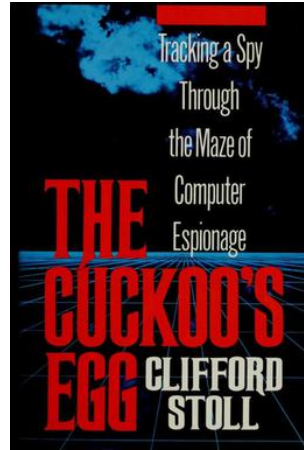
Returning to resources, it takes a lot of resources and people to secure systems:

- Time
- Money
- Skilled resources for designing, implementing, administer, monitor
- Computing resources

Often threats are focussed on outsiders, but insider threat can be common

Dont try to fix people problems with tech

## Cuckoo's Egg 1986 A real spy story



*Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage,*  
Clifford Stoll

*During his time at working for KGB, Hess is estimated to have broken into 400 U.S. military computers*

Source: [https://en.wikipedia.org/wiki/Markus\\_Hess](https://en.wikipedia.org/wiki/Markus_Hess)

## Morris Internet Worm - 30 years ago

Used multiple vulnerabilities:

- Sendmail Debug functionality, we have similar things and Google Hacking
- Buffer overflow in fingerd, we still have those
- Weak passwords/password cracking, list of 432 words and /usr/dict/words, same problem today
- Trust between systems rsh, rexec, think Domain Admin today
- Found new systems using /etc/hosts.equiv, .rhosts, .forward, netstat ...

Also known as the Morris Internet Worm

*The Internet Worm Program: An Analysis*

Purdue Technical Report CSD-TR-823, Eugene H. Spafford

Resulted in creation of the CERT, <http://www.cert.org>

## Internet Worms history repeats itself

Camouflage, tried to hide, malware today hides as well

- Program name set to 'sh', looks like a regular shell
- Used fork() to change process ID (PID)
- Worms in the 2000s spread quickly, like Code Red 2001 to approx 350.000 systems in a week
- SQL Slammer "It spread rapidly, infecting most of its 75,000 victims within ten minutes."

New malware today can use the same strategies

Except a lot of malware tries to stay hidden, less noisy

Using a small password list of 50 words it is possible to create your own botnet with 100.000s

- Security Testing Versus Traditional Software Testing
- Functional testing does not prevent security issues!
- SQL Injection example, injecting commands into database
- Attackers try to break the application, server, operating system, etc.
- Use methods like user input, memory corruption / buffer overflow, poor exception handling, broken authentication, ...

Where to start?



The OWASP Top Ten provides a minimum standard for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are.

The Open Web Application Security Project (OWASP)

OWASP produces lists of the most common types of errors in web applications

<http://www.owasp.org>

Create Secure Software Development Lifecycle



Common Vulnerabilities and Exposures (CVE):

- classification
- identification

When discovered each vuln gets a CVE ID

CVE maintained by MITRE - not-for-profit org for research and development in the USA.

National Vulnerability Database search for CVE.

Sources: <http://cve.mitre.org/> og <http://nvd.nist.gov>

also checkout OWASP Top-10 <http://www.owasp.org/>

## Sample vulnerabilities

- CVE-2000-0884  
IIS 4.0 and 5.0 allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain UNICODE encoded characters, aka the "Web Server Folder Traversal" vulnerability.
- CVE-2002-1182  
IIS 5.0 and 5.1 allows remote attackers to cause a denial of service (crash) via malformed WebDAV requests that cause a large amount of memory to be assigned.
- Exim RCE CVE-2019-10149 June  
<https://www.qualys.com/2019/06/05/cve-2019-10149/return-wizard-rce-exim.txt>
- Exim RCE CVE-2019-15846 September  
<https://exim.org/static/doc/security/CVE-2019-15846.txt>
- CVE-2020 Netlogon Elevation of Privilege  
<https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-1472>
- Log4J RCE (CVE-2021-44228) - and follow up like CVE-2021-45046, also look at scanners like:  
<https://github.com/fullhunt/log4j-scan>

Source:

<http://cve.mitre.org/-CVE>



**CWE™** International in scope and free for public use, CWE provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

## CWE in the Enterprise

- ▲ [Software Assurance](#)
- ▲ [Application Security](#)
- ▲ [Supply Chain Risk Management](#)
- ▲ [System Assessment](#)
- ▲ [Training](#)
- ▲ [Code Analysis](#)
- ▲ [Remediation & Mitigation](#)
- ▲ [NVD \(National Vulnerability Database\)](#)
- ▲ [Recommendation ITU-T X.1524 CWE, ITU-T CYBEX Series](#)

<http://cwe.mitre.org/>

# CWE/SANS Monster mitigations

## Monster Mitigations

These mitigations will be effective in eliminating or reducing the severity of the Top 25. These mitigations will also address many weaknesses that are not even on the Top 25. If you adopt these mitigations, you are well on your way to making more secure software.

A [Monster Mitigation Matrix](#) is also available to show how these mitigations apply to weaknesses in the Top 25.

ID	Description
<a href="#">M1</a>	Establish and maintain control over all of your inputs.
<a href="#">M2</a>	Establish and maintain control over all of your outputs.
<a href="#">M3</a>	Lock down your environment.
<a href="#">M4</a>	Assume that external components can be subverted, and your code can be read by anyone.
<a href="#">M5</a>	Use industry-accepted security features instead of inventing your own.
<a href="#">GP1</a>	(general) Use libraries and frameworks that make it easier to avoid introducing weaknesses.
<a href="#">GP2</a>	(general) Integrate security into the entire software development lifecycle.
<a href="#">GP3</a>	(general) Use a broad mix of methods to comprehensively find and prevent weaknesses.
<a href="#">GP4</a>	(general) Allow locked-down clients to interact with your software.

See the [Monster Mitigation Matrix](#) that maps these mitigations to Top 25 weaknesses.

Source: <http://cwe.mitre.org/top25/index.html>

## Zero day 0-day vulnerabilities

Project Zero's team mission is to "make zero-day hard", i.e. to make it more costly to discover and exploit security vulnerabilities. We primarily achieve this by performing our own security research, but at times we also study external instances of zero-day exploits that were discovered "in the wild". These cases provide an interesting glimpse into real-world attacker behavior and capabilities, in a way that nicely augments the insights we gain from our own research.

Today, we're sharing our tracking spreadsheet for publicly known cases of detected zero-day exploits, in the hope that this can be a useful community resource:

Spreadsheet link: 0day "In the Wild"

<https://googleprojectzero.blogspot.com/p/0day.html>

- Not all vulnerabilities are found and reported to the vendors
- Some vulnerabilities are exploited *in the wild*

*Improving the Security of Your Site by Breaking Into it*

by Dan Farmer and Wietse Venema in 1993

Later in 1995 release the software SATAN

*Security Administrator Tool for Analyzing Networks*

Caused some commotion, panic and discussions, every script kiddie can hack, the internet will melt down!

We realize that SATAN is a two-edged sword – like many tools, it can be used for good and for evil purposes. We also realize that intruders (including wannabees) have much more capable (read intrusive) tools than offered with SATAN.

label Source: <http://www.fish2.com/security/admin-guide-to-cracking.html>

## Use hacker tools!

Port scan can reveal holes in your defense

Web testing tools can crawl through your site and find problems

Pentesting is a verification and proactively finding problems

Its not a silverbullet and mostly find known problems in existing systems

Combined with honeypots they may allow better security

## Local vs. remote exploits

**Local vs. remote** angiver om et exploit er rettet mod en sårbarhed lokalt på maskinen, pelvis opnå højere privilegier, eller beregnet til at udnytter sårbarheder over netværk

**Remote root exploit** - den type man frygter mest, idet det er et exploit program der når det afvikles giver angriberen fuld kontrol, root user er administrator på Unix, over netværket.

**Zero-day exploits** dem som ikke offentliggøres – dem som hackere holder for sig selv. Dag 0 henviser til at ingen kender til dem før de offentliggøres og ofte er der umiddelbart ingen rettelser til de sårbarheder



**Definition 23-14** A *computer worm* is a program that copies itself from one computer to another.

Definition from Computer Security: Art and Science, Matt Bishop ISBN: 9780321712332

Computer worms has existed since research began mid-1970s

Morris Worm from November 2, 1988 was a famous example

Virus, trojan or worm?

Unless you work specifically in the computer virus industry, call it all malware

## The Internet Worm 2. nov 1988

Exploited the following vulnerabilities

- buffer overflow in fingerd - VAX code
- Sendmail - DEBUG functionality
- Trust between systems: rsh, rexec, ...
- Bad passwords

Contained camouflage!

- Program name set to 'sh'
- Used fork() to switch PID regularly
- Password cracking using intern list of 432 words and /usr/dict/words
- Found systems to infect in /etc/hosts.equiv, .rhosts, .forward, netstat ...

Made by Robert T. Morris, Jr.

Worm in 2010 intended to infect Iran nuclear program

Target was the uranium enrichment process

Infected other industrial sites

SCADA, and Industrial Control Systems (ICS) are becoming very important for whole countries

A small *community* of consultants work in these *isolated* networks, but can be used as infection vector - they visit multiple sites

More can be found in <https://en.wikipedia.org/wiki/Stuxnet>

**Definition 23-15** A *bot* is malware that carries out some action in coordination with other bots. The attacker, called a *botmaster*, controls the bots from one or more systems called *command and control (C&C) servers* or *motherships*. They communicate over paths called *C&C channels*. A collection of bots is a *botnet*.

Definition from Computer Security: Art and Science, Matt Bishop ISBN: 9780321712332

Internet Relay Chat has been popular for control channel to botnets

Botnets are popular for spamming campaigns or Distributed Denial of Service (DDoS) attacks

The site <https://malware.lu/> has interesting reads about botnets, and taking over the botnet infrastructures

- Following slides list some of the methods for eliminating security issues
- Sometimes when eliminating is not possible the runtime environment can be improved
- Other times the strategy involves the systems used for development - like alleviating the version control systems to enforce policies and restrict developers from doing bad things



Higher quality software is often more secure

This file specifies the preferred style for kernel source files in the OpenBSD source tree. It is also a guide for preferred user land code style. These guidelines should be followed for all new code. In general, code can be considered “new code” when it makes up about 50more of the file(s) involved. ...

Use queue(3) macros rather than rolling your own lists, whenever possible. Thus, the previous example would be better written:

```
#include <sys/queue.h>
struct foo {
    LIST_ENTRY(foo) link;  /* Queue macro glue for foo lists */
        struct mumble amumble; /* Comment for mumble */
        int bar;
};
LIST_HEAD(, foo) foohead;    /* Head of global foo list */
```

OpenBSD style(9)

## Coding standards functions

The following copies as many characters from input to buf as will fit and NUL terminates the result. Because strncpy() does not guarantee to NUL terminate the string itself, it must be done by hand.

```
char buf[BUFSIZ];  
  
(void)strncpy(buf, input, sizeof(buf) - 1);  
buf[sizeof(buf) - 1] = '\\0';
```

Note that strncpy(3) is a better choice for this kind of operation. The equivalent using strncpy(3) is simply:

```
(void)strncpy(buf, input, sizeof(buf));
```

OpenBSD strncpy(9)



## Compiler warnings - gcc -Wall

```
# gcc -o demo demo.c
```

```
demo.c: In function main:
```

```
demo.c:4: warning: incompatible implicit declaration of built-in  
function strcpy
```

```
# gcc -Wall -o demo demo.c
```

```
demo.c:2: warning: return type defaults to int
```

```
demo.c: In function main:
```

```
demo.c:4: warning: implicit declaration of function strcpy
```

```
demo.c:4: warning: incompatible implicit declaration of built-in  
function strcpy
```

```
demo.c:5: warning: control reaches end of non-void function
```

# Easy to do!

## No warnings = no errors?

```
# cat demo2.c
#include <strings.h>
int main(int argc, char **argv)
{
    char buf[200];
    strcpy(buf, argv[1]);
    return 0;
}
# gcc -Wall -o demo2 demo2.c
```

**This is an insecure program, but no warnings!**

(cheating, some compilers actually warn today)

## Version control sample hooks scripts

Before checking in code in version control, pre-commit - check

- case-insensitive.py
- check-mime-type.pl
- commit-access-control.pl
- commit-block-joke.py
- detect-merge-conflicts.sh
- enforcer
- log-police.py
- pre-commit-check.py
- verify-po.py

[http://subversion.tigris.org/tools\\_contrib.html](http://subversion.tigris.org/tools_contrib.html)

<http://svn.collab.net/repos/svn/trunk/contrib/hook-scripts/>

This references Subversion, which is not used much anymore. Just to show the concept is NOT new. Use hooks!

In a Java project I work on, we use log4j extensively. Use of `System.out.println()` bypasses the control that we get from log4j, so we would like to discourage the addition of `println` calls in our code.

We want to deny any commits that add a `println` into the code. The world being full of exceptions, we do need a way to allow some uses of `println`, so we will allow it if the line of code that calls `println` ends in a comment that says it is ok:

```
System.out.println("No log4j here"); // (authorized)
```

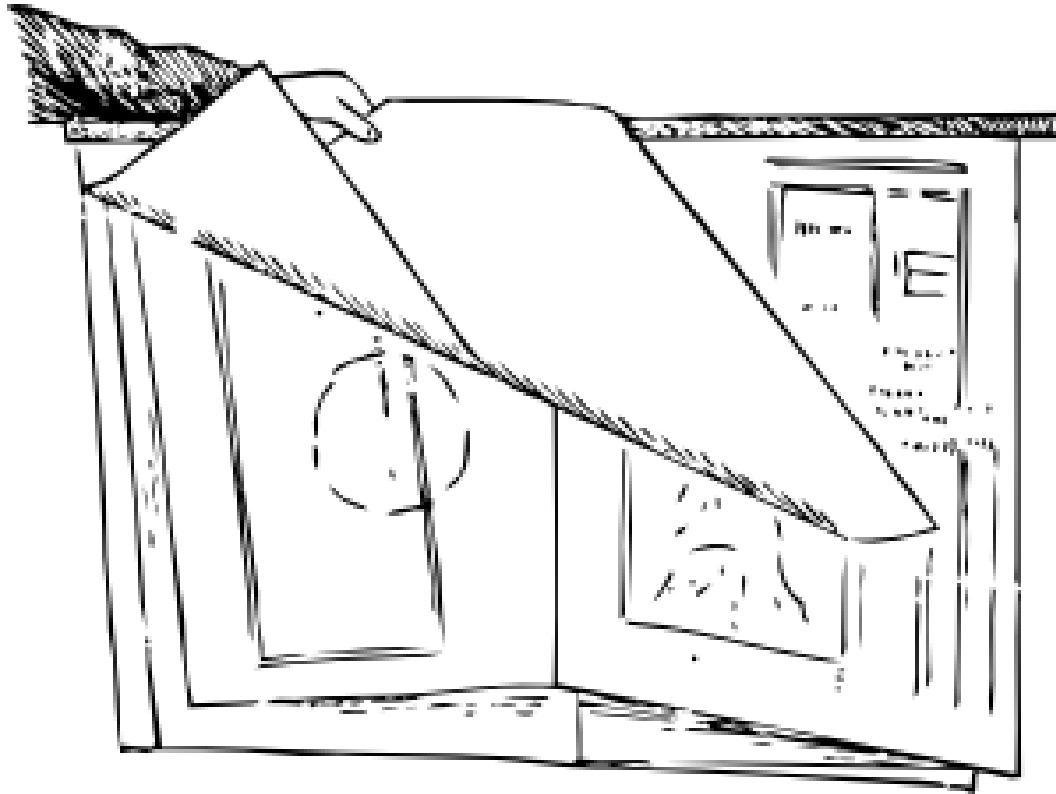
<http://svn.collab.net/repos/svn/trunk/contrib/hook-scripts/enforcer/enforcer>

## Example verify-po.py

```
#!/usr/bin/env python
"""This is a pre-commit hook that checks whether the contents
of PO files committed to the repository are encoded in UTF-8.
"""
```

<http://svn.collab.net/repos/svn/trunk/tools/hook-scripts/verify-po.py>

## Design for security - more work



Security is cheapest and most effective when done during design phase.

# Secure Coding starts with the design

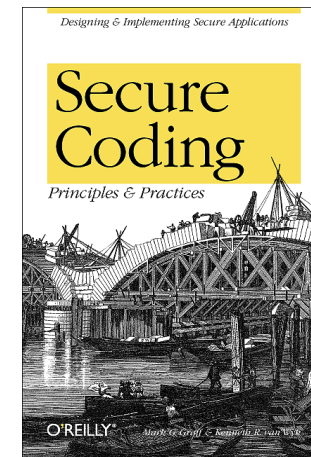
*Secure Coding: Principles and Practices* af Mark G. Graff, Kenneth R. Van Wyk 2003

Architecture/design – while you are thinking about the application

Implementation – while you are writing the application

Operations – After the application is in production

Ca. 200 pages, very dense.



## Design vs Implementation

Software vulnerabilities can be divided into two major categories:

- Design vulnerabilities
- Implementation vulnerabilities

Even with a well-thought-out security design a program can contain implementation flaws.



## Common Secure Design Issues

- Design must specify the security model's structure  
Not having this written down is a common problem
- Common problem AAA Authentication, Authorization, Accounting (book uses audited)
- Weak or Missing Session Management
- Weak or Missing Authentication
- Weak or Missing Authorization

Missing or flawed input validation is the number one cause of many of the most severe vulnerabilities:

- Buffer overflows - writing into control structures of programs, taking over instructions and program flow
- SQL injection - executing commands and queries in database systems
- Cross-site scripting - web based attack type
- Recommend centralizing validation routines
- Perform validation in secure context, controller on server
- Secure component boundaries

## Weak Structural Security

Our book describes more design flaws:

- Large Attack surface
- Running a Process at Too High a Privilege Level, dont run everything as root or administrator
- No Defense in Depth, use more controls, make a strong chain
- Not Failing Securely
- Mixing Code and Data
- Misplaced trust in External Systems
- Insecure Defaults
- Missing Audit Logs

More information about systems design and implementation can be found in the free resource:

Secure Programming for Linux and Unix HOWTO, David Wheeler

<https://dwheeler.com/secure-programs/Secure-Programs-HOWTO.pdf>

Chapter 5. Validate All Input details input validation in the context of Unix programs

Chapter 6. Restrict Operations to Buffer Bounds (Avoid Buffer Overflow)

Chapter 7. Design Your Program for Security

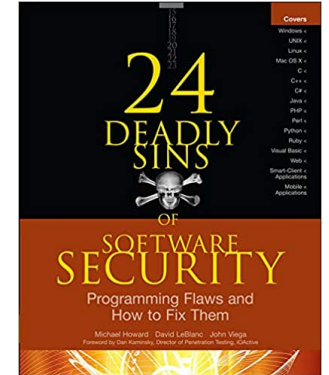
# Sins in Software Security

*24 Deadly Sins of Software Security* af Michael Howard, David Leblanc, John Viega 2010

Should be mandatory reading for all developers

Authors have written other great books

This book is very precise and gives a good overview



## Part I Web Application Sins 1-4

1) SQL Injection 2) Web Server-Related Vulnerabilities 3) Web Client-Related Vulnerabilities (XSS) 4) Use of Magic URLs, Predictable Cookies, and Hidden Form Fields

## Part II Implementation Sins 5-18

5) Buffer Overruns, 6) Format String, 7) Integer Overflows, 8) C++ Catastrophes, 9) Catching Exceptions, 10) Command Injection 11) Failure to Handle Errors Correctly 12) Information Leakage 13) Race Conditions 14) Poor Usability 15) Not Updating Easily 16) Executing Code with Too Much Privilege 17) Failure to Protect Stored Data 18) The Sins of Mobile Code

Part III Cryptographic Sins 19-21

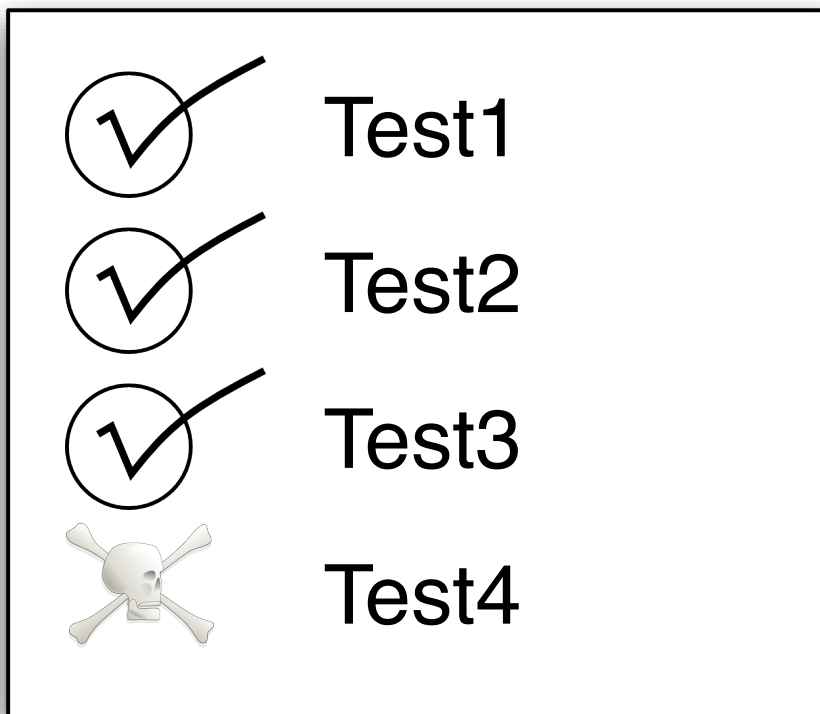
19) Use of Weak Password-Based System 20) Weak Random Numbers 21) Using Cryptography Incorrectly

Part IV Networking Sins 22-24

22) Failing to Protect Network Traffic, 23) Improper use of PKI, Especially SSL, 24) Trusting Network Name Resolution

Still want to program in C?

## Testing - more work now, less work in the long run





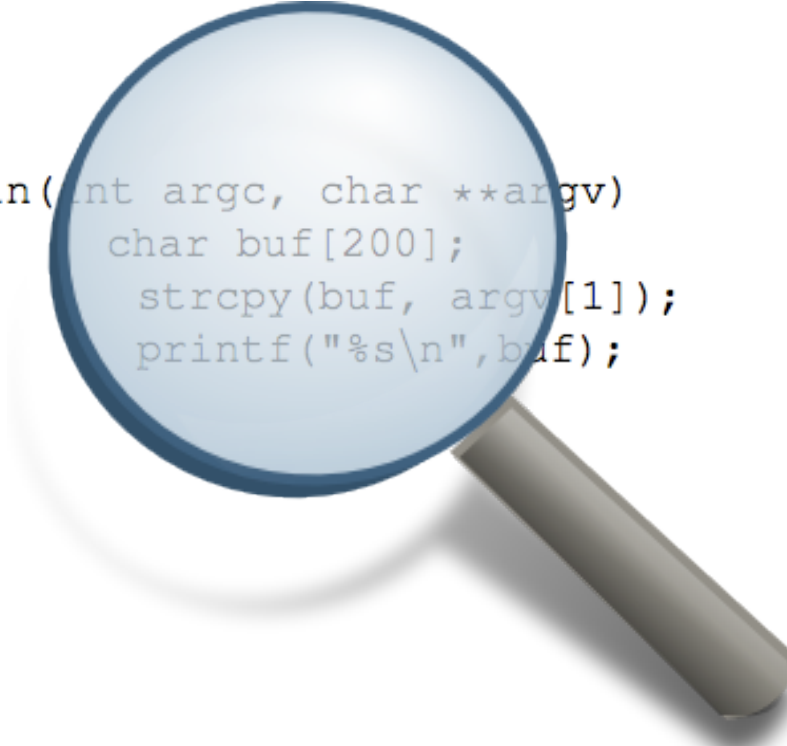
## Unit testing - low level / functions

```
public class TestAdder {  
    public void testSum() {  
        Adder adder = new AdderImpl();  
        assert(adder.add(1, 1) == 2);  
        assert(adder.add(1, 2) == 3);  
        assert(adder.add(2, 2) == 4);  
        assert(adder.add(0, 0) == 0);  
        assert(adder.add(-1, -2) == -3);  
        assert(adder.add(-1, 1) == 0);  
        assert(adder.add(1234, 988) == 2222);  
    }  
}
```

Test individual functions

Example from [http://en.wikipedia.org/wiki/Unit\\_testing](http://en.wikipedia.org/wiki/Unit_testing)

Avoid regressions, old errors reappearing



```
main(int argc, char **argv)
{
    char buf[200];
    strcpy(buf, argv[1]);
    printf("%s\n", buf);
}
```

Use tools for analysing code and applications

## Types of analysis

### static analysis

Run through the program source code or binary, without running it Can find bad functions like strcpy

### dynamic analysis

Running the program with a test-harness, monitoring system calls, memory operations etc.

Flawfinder <http://www.dwheeler.com/flawfinder/>

RATS Rough Auditing Tool for Security, C, C++, Perl, PHP and Python

PMD static ruleset based Java

[http://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)

## A Fool with a Tool is still a Fool

1. Run tool
2. Fix problems
3. Rinse repeat

Fixing problems?

```
char tmp[256]; /* Flawfinder: ignore */  
strcpy(tmp, pScreenSize); /* Flawfinder: ignore */
```

Example from <http://www.dwheeler.com/flawfinder/>

# Pseudo Random Number Generator

## Debian OpenSSL [\[edit\]](#)

In May 2008, security researcher **Luciano Bello** revealed his discovery that changes made in 2006 to the random number generator in the version of the **OpenSSL** package distributed with [Debian GNU/Linux](#) and other Debian-based distributions, such as **Ubuntu**, dramatically reduced the entropy of generated values and made a variety of security keys vulnerable to attack.<sup>[10][11]</sup> The security weakness was caused by changes made to the openssl code by a Debian developer in response to compiler warnings of apparently redundant code.<sup>[12]</sup> This caused a massive worldwide regeneration of keys, and despite all attention the issue got, it could be assumed many of these old keys are still in use. Key types affected include SSH keys, OpenVPN keys, DNSSEC keys, key material for use in X.509 certificates and session keys used in SSL/TLS connections. Keys generated with GnuPG or GNUTLS are not affected as these programs used different methods to generate random numbers. Non-Debian-based Linux distributions are also unaffected. This security vulnerability was promptly patched after it was reported.

[https://en.wikipedia.org/wiki/Random\\_number\\_generator\\_attack#Debian\\_OpenSSL](https://en.wikipedia.org/wiki/Random_number_generator_attack#Debian_OpenSSL)

The random number generator is VITAL for crypto security

Break it

kea



Use fuzzers, hackertools, improve security by breaking it

## Simple fuzzer

```
$ for i in 10 20 30 40 50
>> do
>> ./demo `perl -e "print 'A'x$i"`
>> done
```

AAAAAAAAAA

AAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAA

Memory fault

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

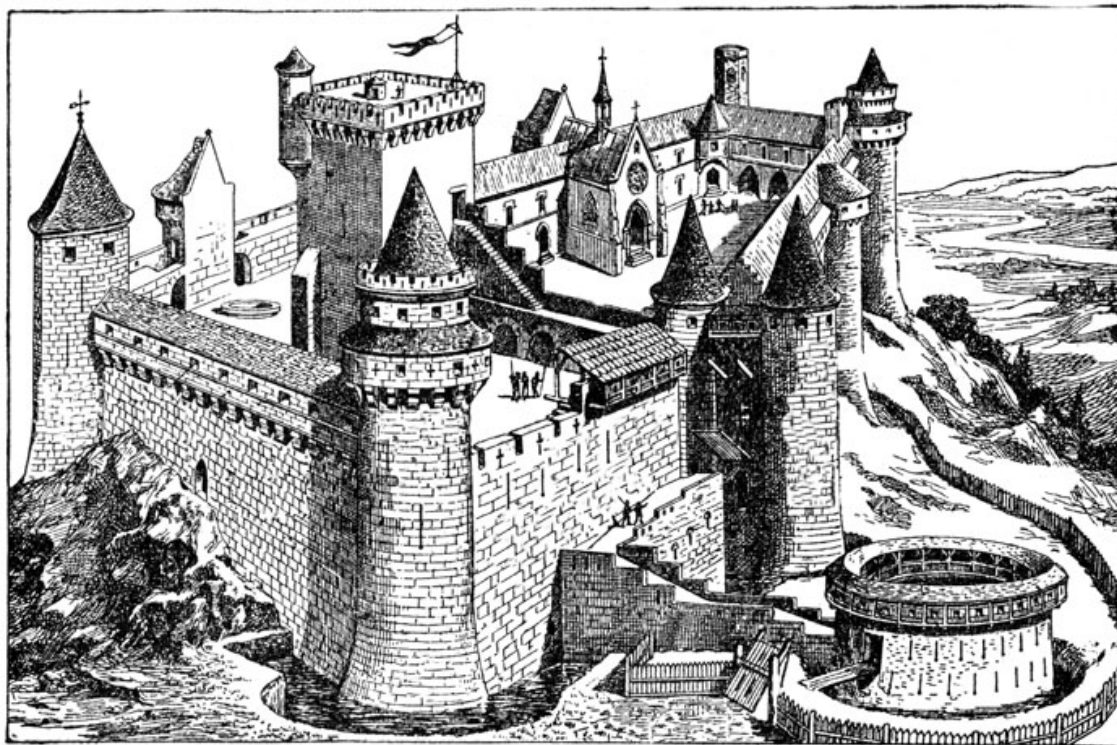
Memory fault

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Memory fault

Memory fault/segmentation fault - juicy!





Sidste chance er på afviklingstidspunktet

Der findes mange typer *jails* på Unix

Ideer fra Unix chroot som ikke er en egentlig sikkerhedsfeature

- Unix chroot - bruges stadig, ofte i daemoner som OpenSSH
- FreeBSD Jails
- SELinux
- Solaris Containers og Zones - *jails på steroider*
- VMware virtuelle maskiner, er det et jail?

Hertil kommer et antal andre måder at adskille processer - sandkasser

Husk også de simple, database som `_postgresql`, Tomcat som `tomcat`, Postfix postsystem som `_postfix`, SSHD som `sshd` osv. - simple brugere, få rettigheder

## JVM security policies

```
// ===== WEB APPLICATION PERMISSIONS =====  
// These permissions are granted by default to all web applications  
// In addition, a web application will be given a read FilePermission  
// and JndiPermission for all files and directories in its document root.  
grant {  
    // Required for JNDI lookup of named JDBC DataSource's and  
    // javamail named MimePart DataSource used to send mail  
    permission java.util.PropertyPermission "java.home", "read";  
    permission java.util.PropertyPermission "java.naming.*", "read";  
    permission java.util.PropertyPermission "javax.sql.*", "read";  
    ...  
};  
// The permission granted to your JDBC driver  
// grant codeBase "jar:file:${catalina.home}/webapps/examples/WEB-INF/lib/driver.jar!/" \{  
//     permission java.net.SocketPermission "dbhost.mycompany.com:5432", "connect";  
// \};
```

Example from apache-tomcat-6.0.18/conf/catalina.policy

## Apple sandbox named generic rules

```
;; named - sandbox profile  
;; Copyright (c) 2006-2007 Apple Inc. All Rights reserved.  
;;  
;; WARNING: The sandbox rules in this file currently constitute  
;; Apple System Private Interface and are subject to change at any time and  
;; without notice. The contents of this file are also auto-generated and not  
;; user editable; it may be overwritten at any time.  
;;  
(version 1)  
(debug deny)  
  
(import "bsd.sb")  
  
(deny default)  
(allow process*)  
(deny signal)  
(allow sysctl-read)  
(allow network*)
```

## Apple sandbox named specific rules

```
;; Allow named-specific files
(allow file-write* file-read-data file-read-metadata
  (regex "^(/private)?/var/run/named\\pid$"
    "^/Library/Logs/named\\.log$"))

(allow file-read-data file-read-metadata
  (regex "^(/private)?/etc/rndc\\.key$"
    "^(/private)?/etc/resolv\\.conf$"
    "^(/private)?/etc/named\\.conf$"
    "^(/private)?/var/named/"))
```

Example from /usr/share/sandbox på Mac OS X

Er der tilstrækkeligt med fokus på software i produktion

Kan en vilkårlig server nemt reetableres

Foretages rettelser direkte på produktionssystemer

Er der fall-back plan

Burde være god systemadministrator praksis

A real-life setup of an infrastructure from scratch can be daunting!

You need:

- Policies
- Procedures
- Incident Response

Running systems which require

- Configurations
- Settings
- Supporting infrastructure – networks
- Supporting infrastructure – logging, dashboarding, monitoring

Building something *secure* is **hard work!**

## Existing infrastructures

or even worse you inherited an infrastructure

Multiple networks, with different vendors, rules

Multiple generations of services, applications, technologies

Built over decades

Varying to no documentation

Organizational challenges

Ingrained culture – frozen in time

How do you get started improving security?



## Integrate or develop?

### Dont:

- Reinvent the wheel - too many times, unless you can maintain it afterwards
- Never invent cryptography yourself
- No copy paste of functionality, harder to maintain in the future

### Do:

- Integrate with existing solutions
- Use existing well-tested code: cryptography, authentication, hashing
- Centralize security in your code and organization

The CIS Controls™ are a prioritized set of actions that collectively form a defense-in-depth set of best practices that mitigate the most common attacks against systems and networks. The CIS Controls are developed by a community of IT experts who apply their first-hand experience as cyber defenders to create these globally accepted security best practices. The experts who develop the CIS Controls come from a wide range of sectors including retail, manufacturing, healthcare, education, government, defense, and others.

Source: <https://www.cisecurity.org/CIS-Controls-Version-7-1.pdf>

- **Offense informs defense:** Use knowledge of actual attacks that have compromised systems to provide the foundation to continually learn from these events to build effective, practical defenses. Include only those controls that can be shown to stop known real-world attacks.
  - **Prioritization:** Invest first in Controls that will provide the greatest risk reduction and protection against the most dangerous threat actors and that can be feasibly implemented in your computing environment. The CIS Implementation Groups discussed below are a great place for organizations to start identifying relevant Sub-Controls.
  - **Measurements and Metrics:** Establish common metrics to provide a shared language for executives, IT specialists, auditors, and security officials to measure the effectiveness of security measures within an organization so that required adjustments can be identified and implemented quickly.
  - **Continuous diagnostics and mitigation:** Carry out continuous measurement to test and validate the effectiveness of current security measures and to help drive the priority of next steps.
  - **Automation:** Automate defenses so that organizations can achieve reliable, scalable, and continuous measurements of their adherence to the Controls and related metrics.
- Source: CIS-Controls-Version-7-1.pdf

# Inventory and Control of Hardware Assets

CIS controls 1-6 are Basic, everyone must do them.

CIS Control 1:

Inventory and Control of Hardware Assets

Actively manage (inventory, track, and correct) all hardware devices on the network so that only authorized devices are given access, and unauthorized and unmanaged devices are found and prevented from gaining access.

What is connected to our networks?

What firmware do we need to install on hardware?

Where IS the hardware we own?

What hardware is still supported by vendor?

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

CIS Control 2:

Inventory and Control of Software Assets

Actively manage (inventory, track, and correct) all software on the network so that only authorized software is installed and can execute, and that all unauthorized and unmanaged software is found and prevented from installation or execution.

What licenses do we have? Paying too much?

What versions of software do we depend on?

What software needs to be phased out, upgraded?

What software do our employees need to support?

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

CIS Control 3:

Continuous Vulnerability Management

Continuously acquire, assess, and take action on new information in order to identify vulnerabilities, remediate, and minimize the window of opportunity for attackers.

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

# Controlled Use of Administrative Privileges

CIS Control 4:

Controlled Use of Administrative Privileges

The processes and tools used to track/control/prevent/correct the use, assignment, and configuration of administrative privileges on computers, networks, and applications.

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

## CIS Control 5:

Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers

Establish, implement, and actively manage (track, report on, correct) the security configuration of mobile devices, laptops, servers, and workstations using a rigorous configuration management and change control process in order to prevent attackers from exploiting vulnerable services and settings.

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf



# Maintenance, Monitoring and Analysis of Audit Logs

CIS Control 6:

Maintenance, Monitoring and Analysis of Audit Logs

Collect, manage, and analyze audit logs of events that could help detect, understand, or recover from an attack.

... and present it, use it daily, report it to management!

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

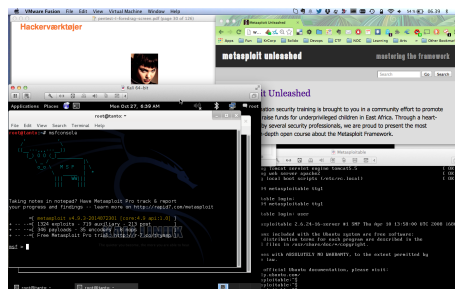
CIS Control 18:

Application Software Security

Manage the security life cycle of all in-house developed and acquired software in order to prevent, detect, and correct security weaknesses.

Source: Center for Internet Security CIS Controls 7.1 CIS-Controls-Version-7-1.pdf

# Our labs and training environment



- This course will use Linux
- We already created virtual machines
- Linux is free, and not as resource hungry
- We can have a hacker lab with multiple machines, on a single laptop
- Lots of free and open source tools run on Linux, save money!



Hacking looks like magic – especially buffer overflows

# Hacking is not magic



Hacking only demands ninja training and knowledge others don't have

It is like a puzzle, we need this, this and that. Make it happen in a repeatable way.

- Portscanning with Nmap is a critical task for any pentest
- We will now use some of the materials found in my Nmap Workshop and perform some Nmap scans  
<https://github.com/kramse/security-courses/tree/master/courses/pentest/nmap-workshop>
- We will NOT do all the exercises, I have copied the ones we will do into this course!
- But make sure you do the ones named on the following pages
- Nmap can output in XML and can be converted easily into HTML
- Nmap output can also be easily imported into a database, example Metasploit database
- Nmap, and other active tools produce network traffic which can often be captured and analyzed, so we will also reference a few example capture tools

Nmap is an important tool to get an overview of security in a network!



Now lets do the exercise

## Run small programs: Python, Shell script 20min

which is number **10** in the exercise PDF.



Now lets do the exercise

## Small programs with data types 15min

which is number **11** in the exercise PDF.





Now lets do the exercise

**Optional: Run parts of a Django tutorial 30min**

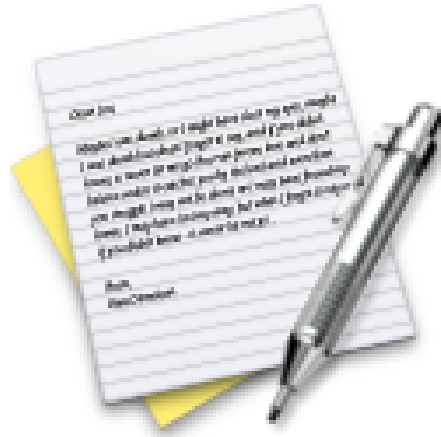
which is number **12** in the exercise PDF.



Now lets do the exercise

## Execute nmap TCP and UDP port scan 20 min

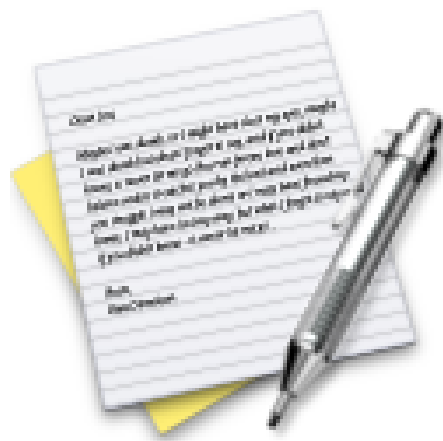
which is number **13** in the exercise PDF.



Now lets do the exercise

**Discover active systems ping and port sweep 15 min**

which is number **14** in the exercise PDF.



Now lets do the exercise

## Perform nmap OS detection

which is number **15** in the exercise PDF.



Now lets do the exercise

## Perform nmap service scan

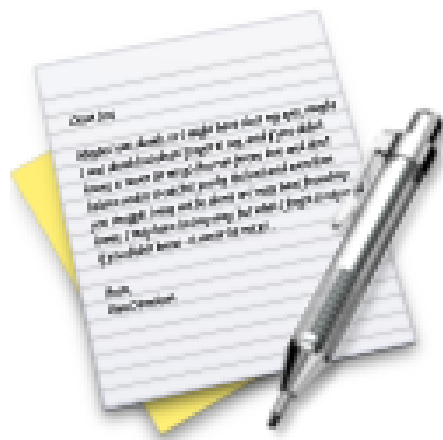
which is number **16** in the exercise PDF.



Now lets do the exercise

## Optional: Nmap full scan

which is number **17** in the exercise PDF.



Now lets do the exercise

## Reporting HTML

which is number **18** in the exercise PDF.



Now lets do the exercise

## Optional: Nmap Scripting Engine NSE scripts

which is number **20** in the exercise PDF.



## Setup the OWASP Juice Shop

If we have too much time, we will look into running the OWASP Juice Shop

This is an application which is modern AND designed to have security flaws.

Read more about this project at: <https://www2.owasp.org/www-project-juice-shop/>  
and <https://github.com/bkimminich/juice-shop>

It is recommended to buy the Pwning OWASP Juice Shop Official companion guide to the OWASP Juice Shop from <https://leanpub.com/juice-shop> - suggested price USD 5.99. Alternatively read online at <https://pwning.owasp-juice.shop/>

Sometimes the best method is running the Docker version

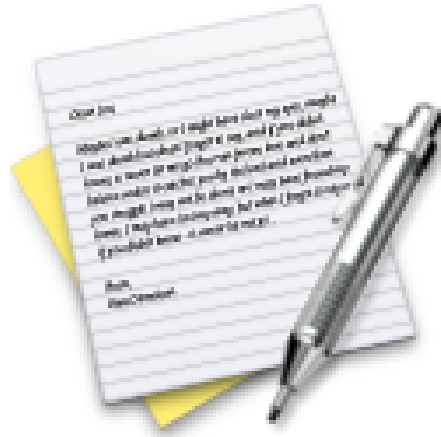
Later we will start hacking this awesome application



Now lets do the exercise

## Run OWASP Juice Shop 45 min

which is number **21** in the exercise PDF.



Now lets do the exercise

## Setup JuiceShop environment, app and proxy - up to 60min

which is number **22** in the exercise PDF.



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools