Stockholms universitet

Program Security, Malware, and its Mitigation

Jesper Bergman
PhD Candidate
(jesperbe@dsv.su.se)

December 14, 2020

Version: 0.1

What is this?

- The course is an introduction to information security.
- Student count is circa 200.
- Everyone has their own education, background, knowledge, experience, motivation etc.

I will try to make it understandable for as many of you as possible.

Some of you might already know some of this stuff. I am going over it anyway.

Who am I?

- I am a PhD student and a teacher at the department.

What is the Agenda for Today?

- Chapter three in the book (pp. 131-231)
- Not all 100 pages are relevant.
- Learning outcome:
  - What is software security?
  - What are software vulnerabilities?
  - What is malware or malicious software?
  - What countermeasures are there to these threats?

Computers

Let us start from ground level, and then gradually, walk upstairs.

Oxford dictionary says:

- Computer: "An electronic device for storing and processing data, typically in binary form, according to instructions given to it in a variable program." [1]

Computer Programming

- Computer program: "A series of coded software instructions to control the operation of a computer or other machine." [2]

Computer Input and Computer Output

If we choose to see a computer as a black box, which it often times is with closed platforms and closed source code, we have (1) input, (2) unknown operations, and (3) output.

Modern Computers and Computer programs

In a modern computer, you have numerous attack surfaces (potential entries into the system).

- Most computers and operating systems are considered secure if they are not connected to the internet, the peripheral devices and the number of installed programs are reduced to a minimum.

Modern Computers and Computer programs (2)

How attractive is a default installation of, for example, Windows 10 or Ubuntu desktop with only a keyboard and a mouse?

It is rather secure. We have measurements of that in form of software certificates (which we at the end of this lecture).

Modern Computers and Computer programs (3)

To be able to use a modern computer, we would want software and some sort of network connection.

- How to keep a *usable* computer secure?

Now we are moving into the subject matter that chapter three concerns.

Useful Computers

Generally speaking, we want:

- Computer that have connection to the Internet.
- Software that does something meaningful with our input and output.

Software Security

Frankly put, everything is software. So, software needs to be secure.

Software Security (2)

Software, or programs, are the instructions executed in the processing unit, as mentioned in the beginning.

- Software runs on an operating system (which is also software).
- Software is sometimes run by an "engine", like Java (which is also software).
- Software is sometimes compiled by a "compiler", like C (which is also software).
- Software does sometimes use input from other devices, which require "drivers" (which are also software) to work properly. For example a network interface card or a USB socket.
- Software sometimes communicates with other software through, for example, the operating system.

Software Security (3)

Summary: a lot of software (or *computer programs*).

- There are many sources of error in a system like a computer.
- "The chain is not stronger than its weakest link."

Ideally, all software should to be secure.

Programming Oversights

Now that we gone through how computers and software work, we should be on the same page and ready to dive into chapter three (starting on page 131.).

Programming Oversights (2)

A simple program is not very vulnerable.

```c
#include <stdio.h>

int main() {
    printf("HelloWorld!");
    return 0;
}
```

Programming Oversights (4)

"A simple program is not very vulnerable."

- Depends on the programmer. A bad programmer can mess things up in very few lines of code.
- Depends on the programming language.
- Depends on the surrounding software (like the operating system).
- What is "simple"? Simple as in small or simple as in minimalistic?

We will get back to these questions later on.

Programming Oversights (5)

The book includes a number of programming "oversights" in section 3.1 that concern the security of a software program:

- 3.1.1: **Buffer overflow**
- 3.1.2: Incomplete mediation
- 3.1.3: Time-of-check, time-of-use errors
- 3.1.4: Undocumented Access Point
- 3.1.5: Off-by-one errors
- 3.1.6: Integer overflow
- 3.1.7: Unterminated null-terminated string
- 3.1.8: Parameter length, type, and number
- 3.1.9: Unsafe utility program
- 3.1.10: Race conditions

We will not go through them all.

3.1.1: Buffer Overflow Vulnerability

- The most important vulnerability to know about.
- A simple, yet effective, vulnerability to exploit.
- If a programmer for some reason fails to correctly handle the memory management, it becomes an attack surface.
- In a nutshell, the attack crashes the memory and runs arbitrary code in it.

3.1.1: Buffer Overflow (2)

The example on page 135 is rather good. Read it.

- A program for dialling was not programmed to handle absurd "telephone numbers".
- When entering a 100 digits long telephone number was entered, the program crashed.
- When crashing, arbitrary code can be run in the memory space - not only dialer's memory space, but in the operating system's memory space[1].
- If an attacker manages to get access to the memory space of the operating system, they can potentially attack the entire system.

[1] https://seclists.org/bugtraq/1999/Jul/244

3.1.1: Buffer Overflow (3)

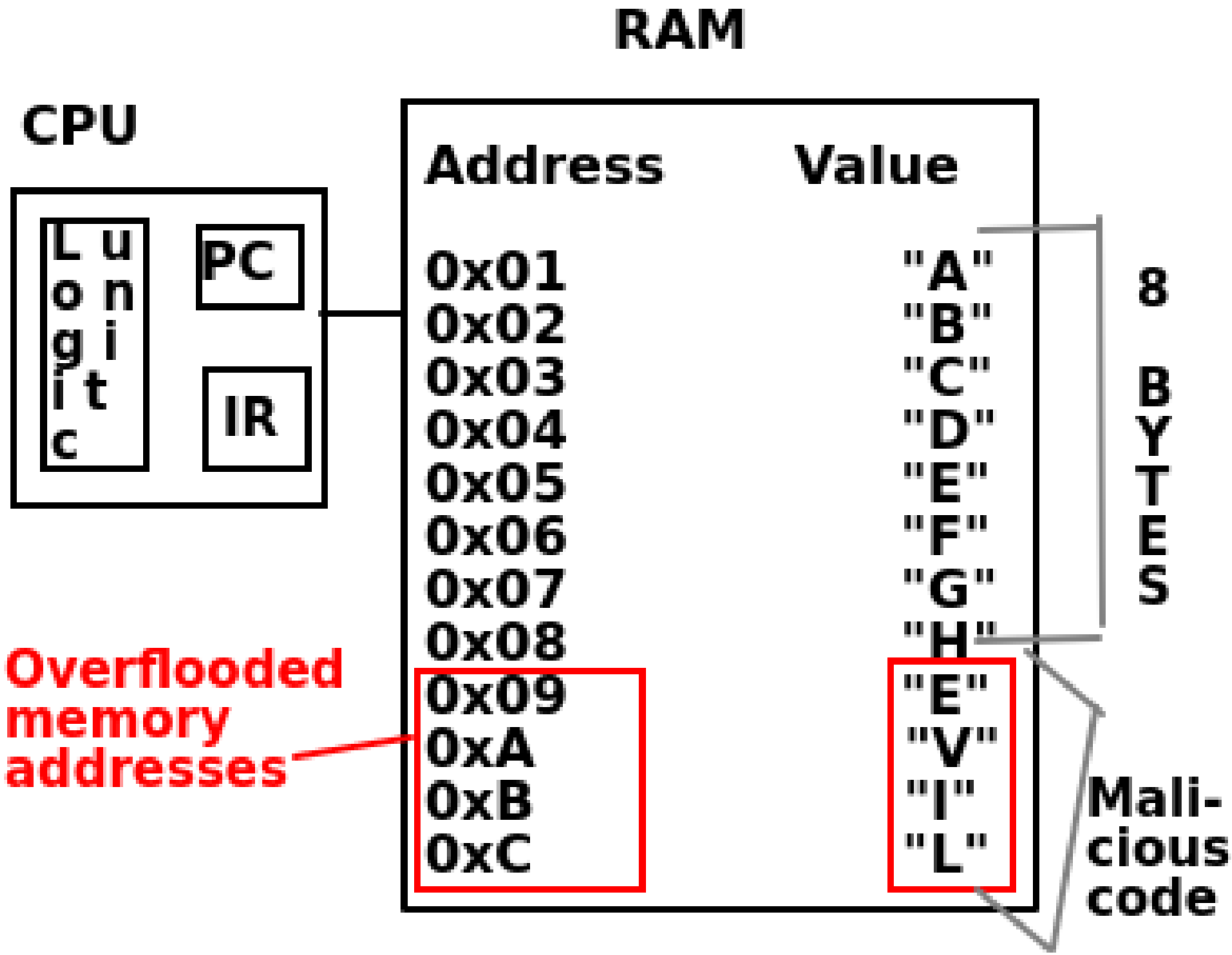
"A simple program is not very vulnerable."

```
#include<stdio.h>

int main(int argc, char *argv[]){
    char buf[64];
    if (argc < 2){
            printf("Provide an argument");
                return(1);
    }
    strcpy(buf, argv[1]);
    return(0);
}
```

3.1.1: Buffer Overflow (4)

Let us depict what happened in that C code (simplified).



3.1.1: Buffer Overflow (5) - How is it Possible?

(1) Due to the programmers and programming languages.

- Memory management support in low level programming language Assembler/C++/C.

(2) Due to the construction of computers.

- The CPU always points to next instruction in memory via the motherboard; no error checking, just execution.

Why use these languages then? Performance, freedom, control, interaction with hardware.

3.1.1: Buffer Overflow (6) - Countermeasures.

Write more secure code. For example:

- Double check lengths of input.
- Make sure to catch out-of-bounds errors.
- Run with the least possible privilege.

3.1.3: Incomplete Mediation

Basically, this means poor checking of for example input data.

A fundamental principle of security is to minimise, restrict, and validate any input data.

3.1.3: Incomplete Mediation (2)

A simple example in PHP:

Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php",

```
<form method="post" action="test_form.php">
```

So far, so good.

However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

https://www.w3schools.com/php/php_form_validation.asp

3.1.3: Incomplete Mediation (3)

Incomplete mediation mitigation:

- Validate all input. Why allow 100 digit phone number?
- Wash input from special characters.
- In the case of PHP, escape characters by converting them to HTML characters.

3.1.4: Time-of-check and time-of-use errors

This flaw has goes under the same theme of poor checking.

If there is a delay between check and action and the programmer has not taken this into account: problem.

3.1.10: Race Conditions

Who gets there first?

Same theme as before: undesirable timing of events (unsynchronised). E.g. two requests of a certain resource; no one ones gets it, or maybe both get it (overbooking/overselling). See example on page 163.

3.1.X: Software Vulnerabilities

This is not mentioned in chapter three in the book. Please read it carefully.

When vulnerabilities are discovered, they are often patched and published.

National Computer Emergency Response Teams (CERTs), around the world, like JP-CERT, CERT-US, and CERT-SE, usually publishes information about the latest critical vulnerabilities.

Let us look at what a critical vulnerability and its implications: CVE-2020-1350
`https://nvd.nist.gov/vuln/detail/CVE-2020-1350`

3.1.X: Software Vulnerabilities (2)

This is not mentioned in chapter three in the book. Please read it carefully.

Published vulnerabilities are available in different databases. For example:

- `https://cve.mitre.org/` (non-profit organisation)
- `https://nvd.nist.gov` (US government)
- `https://vuldb.com/` (community driven)

Generally, these databases provide the following detailed:

- Identification number CVE-year-ID
- Description of operation.
- CVSS: Common Vulnerability Score System. Scale: 0-10.
- Attack vector(s)
- URL to more details.

3.2 - Malicious Software (Malware)

Another huge topic that we will have to cover in a few slides. The outline is as follows (same as in the book):

- Definitions
- History
- Infection vectors
- Propagation and activation
- Modus operandi
- Mitigations or countermeasures

3.2 - Malicious Software Categories

The big ones according to Pfleeger et al. (p. 170):

- Virus - malicous code that spreads itself into other programs
- Trojan - "code that contains unexpected, undocumented, additional functionality"
- RAT - trojan that once run provies remote access to the victim system.
- Worm - malicious software that spreads itself through a network.
- Spyware - malicious spying, information stealing software.
- Rootkit - malware running with highest privileges.
- Backdoor - code that enables for unauthorised access to a program or system.

Toolkit - programs (false positives) like Netbus (lab 1).

3.2 - Malicious Software History

A bit of history does not hurt.

See page 175 for a summary of notable malware.

An informal overview of some infamous malware by Mikko Hypponen:
`https://www.youtube.com/watch?v=cf3zxHuSM2Y`

3.2 - Malware Infection Vectors

1. Entrance via email and URLs.
2. Unpatched vulnerabilities in VPN, RDP, Citrix, and other web services servers.
    - WannaCry exploited the unpatched vulnerability EthernalBlue (patch was released two months earlier).
    - Vulnerabilities in software are discovered every day.

Spam and phishing emails are very trend sensitive. Greta Thunberg, Covid-19, US election to mention have been common topics during 2020.

3.2 - Malware Infection Vectors (2)

Malware may enter a system via email, URLs, or vulnerable services running on the system. Pfleeger et al. also mention the following entry points:

- Via other executables.
- Via email attachments.
- Embedded in document files (email attachments)
- Autorun files.
- Storage media, like USB drives.

When these are opened, they run into operation and the malware is active.

3.2 - Malware Operations

Once it has entered the system and been executed, the show is on. The following procedures might follow.

- Privilege escalation through buffer overflow (admin/root access).
- Boot sector infection - first thing that is run when booting from a HD. A weakness in computers still today (see figure on page 188).
- Memory based malware - stays in the memory, hence difficult to detect.

3.2 - Malware Modus Operandi

Now that we know the methods. What is the purpose of the malware? What is the goal?

- Money (criminals) - e.g. ransomware or banking trojan.
- Power (nation state actors) - e.g. sabotage or espionage.

3.2 - Malware Modus Operandi (2)

A few examples of malicious software from the aforementioned categories.

(Organised) Criminals' malware:

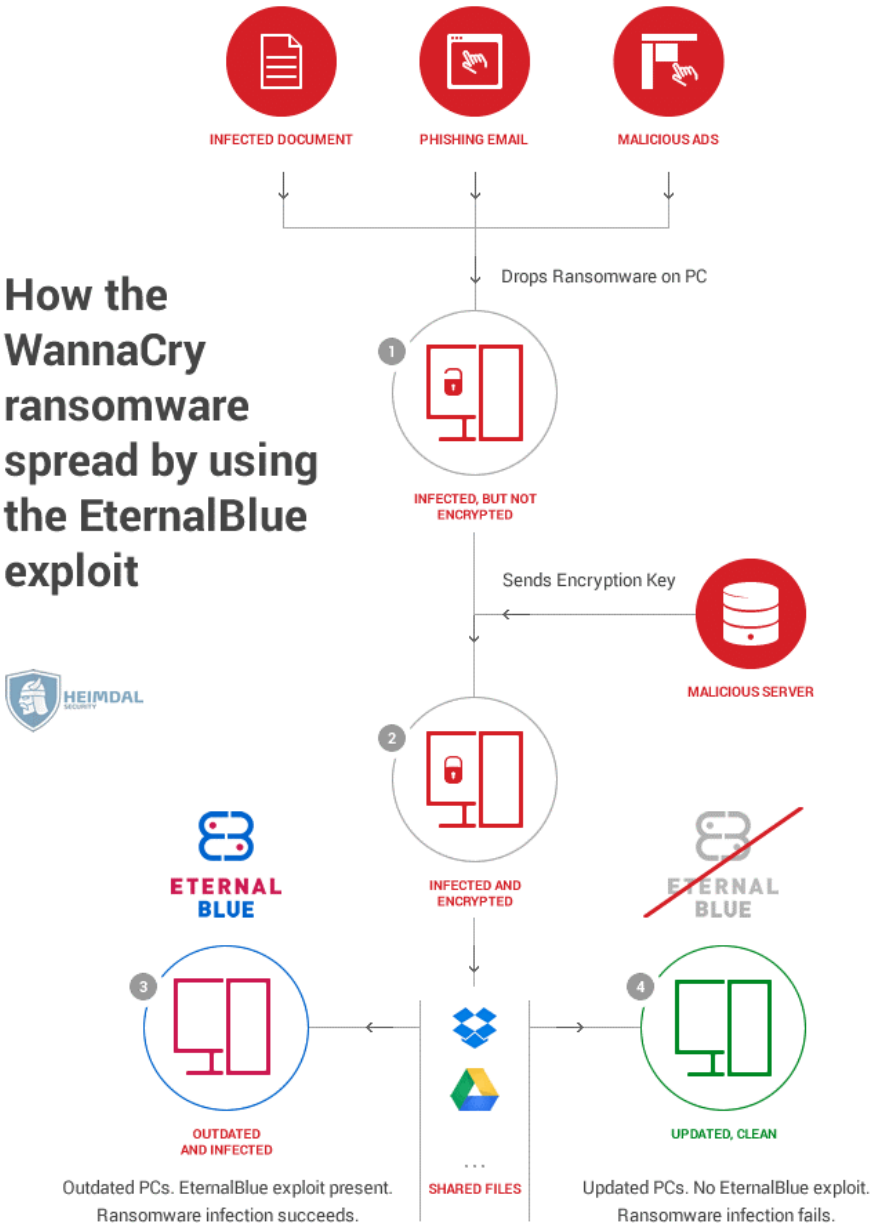
- Zeus - a banking trojan, stealing credentials in secret.
- Ryuk, WannaCry, CryptoLocker, Petya - ransomware, decrypting files for ransom.

Nation state malware:

- Stuxnet - worm/trojan, damaging Iranian nuclear centrifuges.
- The Dukes - trojans, stealing information and credentials in secret.

3.2 - Malware Infection Process (an example)

https://heimdalsecurity.com/blog/wannacry-ransomware/



3.2 - Malware Infection Process (a recent example)

Ryuk attack kill chain

Initial Compromise	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command And Control	Exfiltration	Impact
Spear phish	Buer Loader	Task scheduler	Cobalt Strike	SystemBC	Cobalt Strike	Cobalt Strike	Cobalt Strike	SharpHound	Cobalt Strike	Cobalt Strike	Ryuk
	WMI			GMER		Command line	RDP		SystemBC	SystemBC	
	PowerShell					BloodHound					
	cmd.exe										
	MAL/Inject-GQ										
	Cobalt Strike Beacon										

SOPHOSlabs

https://news.sophos.com/en-us/2020/10/14/inside-a-new-ryuk-ransomware-attack/

3.3 - Countermeasures (to software attacks)

The best countermeasure is for programmers to code secure software and for users to not install malicious software.

We have a course called "Software Security" (SoSec) on the subject.

In the book, the following countermeasures are suggested:

- Users: User vigilance (*carefulness*) and malware detectors (pp. 197-203).
- Developers: Proper development techniques and software testing (pp. 203-216).
- Security Design Process: Saltzer & Schroeder's famous design principles (important to know) (pp. 216-224).

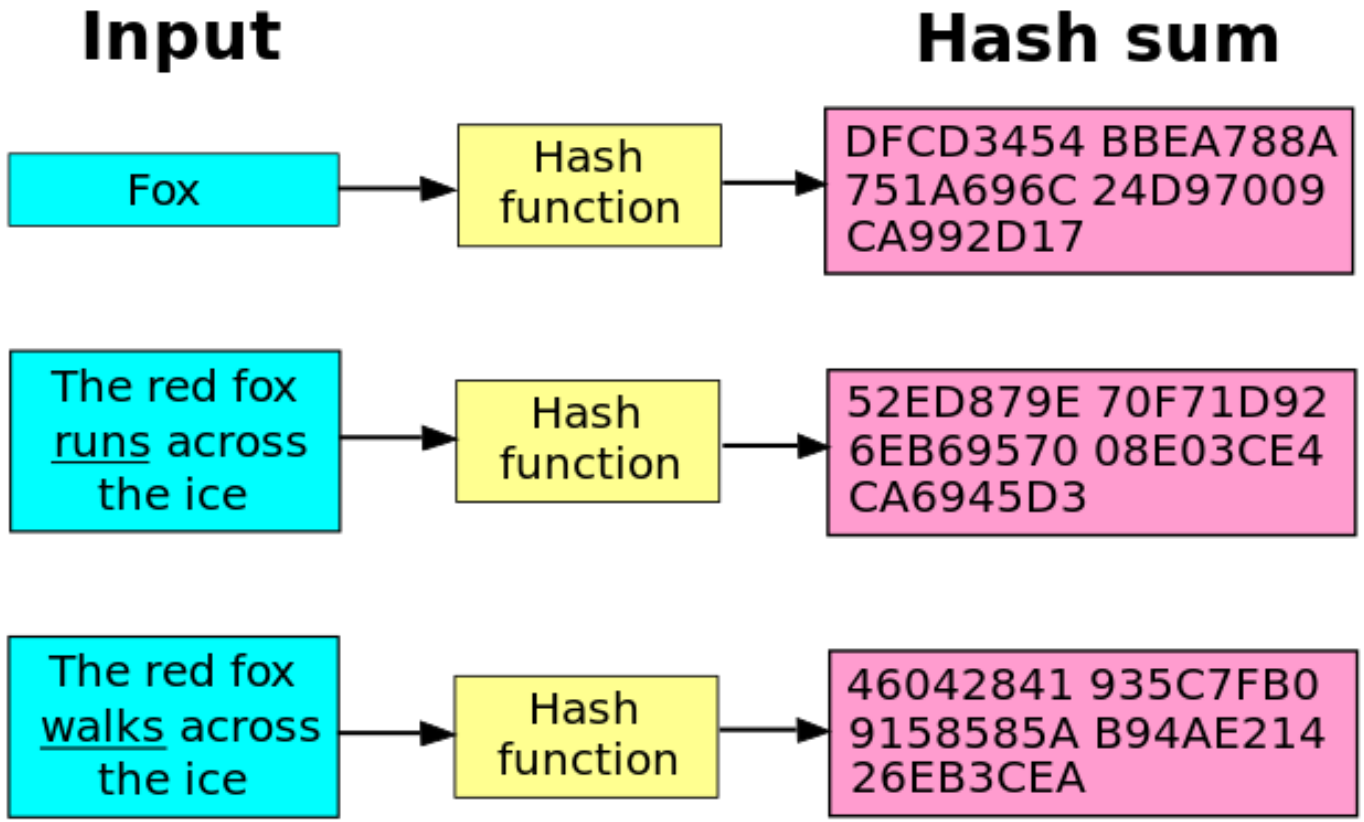
3.3 - Countermeasures for Users

Being careful (user vigilance) is an effective way of keeping safe and secure. In the book, the following recommendations are presented on pages 197 and 198:

- Use only "trusted" software
- Test software in an isolated environment (or let a trusted party do it for you, e.g. `https://virustotal.com` or `https://cuckoo.cert.ee`)
- Do not open potentially suspicious/malicious files.
- Only install secure software.
- Any website can be harmful.
- Backup and/or create restore points of your operating system.
- Backup executable system files.

3.3 - Countermeasures for Users: Malware Detectors

Thanks to hashing algorithms, it is easy to identify hitherto known malicious (or benevolent) software.



https://commons.wikimedia.org/wiki/File:Hash_function_long.svg

3.3 - Countermeasures for Users: Malware Detectors (2)

The hash sum will always be the same for a certain file. The hash sum for Stuxnet will always be the same.

Once malware has been registered in some sort of malware database as malicious, we can run our suspicious files against it, to see if we have downloaded malware.

This is a useful countermeasure for the individual user or entire nations and organisations, but it has an obvious limitation.

3.3 - Countermeasures for Developers

- Software Engineering Techniques (modularity, encapsulation, information hiding, mutual suspicion, confinement, simplicity, generic diversity (pp. 203-210)). Not very important, but skim through them.
- **Testing** is an important developer countermeasure though. More about that in the coming slides.

3.3 - Countermeasures for Developers: Testing

Software testing is a big; there are plenty of books in the topic. But generally, it is either open ("clear box") or closed ("black box") testing.

According to the book, testing is usually carried out in the following order (p. 211):

1. Unit testing
2. Integration testing
3. Functionality testing
4. Performance testing
5. Acceptance testing
6. Installation testing
7. (in case of bug fix: regression testing)

Fine. But, security wise then?

3.3 - Countermeasures for Developers: Testing (2)

Testing ensures that the software does what it should do (ideally). So security wise:

Software is secure as long as the requirement specification is secure.

Functionality is of course important; if the software breaks it is not complete, and hence not secure.

For security testing in particular, "penetration testing" is more spot on. More about that later.

3.3 - Countermeasures for Developers: Testing (3)

There are some limitations to software testing. You will find more in the book.

- Not possible to prove absence of problems.
- It is resource intense to do complete testing with competent testers.
- Black box vs. clear box testing both have limitations. Clearbox testing sometimes means altering the original code, which in itself might contain bugs.

Software testing done by certified professionals, can certify software to certain level of functionality and security assurance. This is known as "software assurance". In Sweden, FMV (The Swedish Defence Materiel Administration) does some software assurance certifications for those who pay for it.

3.3 - Countermeasures in Software Design Processes

In the book: "Countermeasure Specifically for Security" (p. 216).

- **Saltzer & Schroeder's design principles** are fundamental in information security.
- Penetration testing
- Proof of correctness
- Validation
- (Defensive programming)
- (Trustworthy computing initiative)
- (Design by contract)

Additionally, there are some countermeasures that do not work that are mentioned in the book as well.

3.3 - Saltzer & Schroeder's Design Principles

The secure design principle of:

- Least privilege.
- Economy of mechanism.
- Open design.
- Complete mediation.
- Permission based.
- Separation of privilege.
- Least common mechanism.
- Ease of use or psychological acceptance.

3.3 - Saltzer & Schroeder's Design Principles (2)

You should know these very very well for the exam.

3.3 - Software Design Process Countermeasures: Penetration testing

Penetration testing, or "pen test", is a common practice in information security.

Usually, it refers to IT consultants (or "ethical hackers", "white hat hackers", possibly "grey hat hackers") who, with permission, try to "hack" the infrastructure of the ordering organisation.

In software development, it is quite the same: try to break it to fix it.

A black box testing method that aims to identify vulnerabilities and flaws in the software.

3.3 - Other Software Design Process Countermeasures

- Proof of Program Correctness
- Validation

Also mentioned in the book, but not very important.

- Defensive programming
- Trustworthy Initiative
- Design by Contract

3.3 - Software Design Process Countermeasures: Proof of Program Correctness

Algorithms and programs can be formally verified **if** the translation from program code to logic is done correctly.

However, there is no way one program can determine whether an arbitrary program will halt when processing arbitrary input ("halting problem").

Hence, there is no program that program that can generally determine whether two programs are equivalent.

3.3 - Software Design Process Countermeasures: Proof of Program Correctness (2)

Worth mentioning Brookshear's view on proof of program correctness [3]:

Unfortunately, formal program verification techniques have not been refined to the point that they can be easily applied in general applications. The result is that in most cases today, software is 'verified' by testing it under various conditions - a process that is shaky at best. After all, verification by testing proves nothing more than that the program runs correctly for the cases under which it was tested. Ant additional conclusions are merely projections.

3.3 - Software Design Process Countermeasures: Validation

Software validation could mean different things depending on who you talk to.

- Requirements checking: Does the program fulfil the requirements specified?
- Design and code reviews: Inspect the code, does it fulfil requirements?
- System testing: Feed the software with arbitrary data to test security/functionality

3.3 - Countermeasures that do not work

According to the book, these countermeasures do not work.

- Penetrate-and-patch
- Security by obscurity
- Good code - bad code separator

3.3 - Countermeasures that do not work: Penetrate-and-patch

This is an old and outdated term, but anyway:

- Let the, preferably competent team, "tiger team" break the software.
- If they do not succeed, "everything is fine".
- If they find vulnerabilities - patch them.

According to the book, this is the source document for this term/method:
`https://web.archive.org/web/20120214190220/http://www.airpower.maxwell.af.mil/airchronicles/aureview/1979/jan-feb/schell.html`

I recommend reading this research paper by McGraw from 1998 instead [4]: `https://ieeexplore.ieee.org/abstract/document/666831`

3.3 - Countermeasures that do not work: Security by Obscurity

Simply the idea that obscurity will keep the system secure.

"No one will look in the Windows registry for the password anyway. Let us just put them there."

An analogue analogy would perhaps be: "No one would ever look in the exhaustpipe for the car keys. Let us just put them there."

Secure as long as it is hidden.

3.3 - Countermeasures that do not work: Good code - bad code Separator

It is not possible to separate any arbitrary programs.
- Why? Same as the unsolvable "halting problem" (HP).

Simply put, the HP states that **it is not possible to determine whether a certain program with certain input will halt or run forever**.

If you are interested, there are is a lot of information about the HP on the net.

3.3 - Countermeasures that do not work: Good code - bad code Separator (2)

No technology can automatically distinguish malicious extensions from benign code. For this reason, we have to reply on a combination of approaches, including human-intensive ones, to help us detect when we are going beyond the scope of the requirements and threatening the system's security.

Pfleeger et al. (2015, p. 212)

Software Assurance

This is not mentioned in the book, unfortunately, but it is good to know what software assurance is.

- There are certain assurances of software security in form of certificates.
- These assurances are based on testing done by competent people (a trusted party).
- Common criteria (CC) is the most common standard for certifying the functionality and security of certain software to a certain level. See: `https://www.commoncriteriaportal.org/`

Software Assurance (2)

The US-CERT's definition of SA: "Software assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner."

They have a good summary of software security and assurance here: `https://us-cert.cisa.gov/bsi/articles/knowledge/principles/foundations-software-assurance`

Military agencies, for example, would probably want to buy software that has a software assurance in form of a CC certificate.

Software Assurance (3)

An example CC certificate:

National Information Assurance Partnership

Common Criteria Certificate

is awarded to

Samsung Electronics Co., Ltd.

for

Samsung Knox File Encryption 1.0

The IT product identified in this certificate has been evaluated at an accredited testing laboratory using the Common Methodology for IT Security Evaluation (Version 3.1) for conformance to the Common Criteria for IT Security Evaluation (Version 3.1). This certificate applies only to the specific version and release of the product in its evaluated configuration. The product's functional and assurance security specifications are contained in its security target. The evaluation has been conducted in accordance with the provisions of the NIAP Common Criteria Evaluation and Validation Scheme and the conclusions of the testing laboratory in the evaluation technical report are consistent with the evidence adduced. This certificate is not an endorsement of the IT product by any agency of the U.S. Government and no warranty of the IT product is either expressed or implied.

Date Issued: 2019-12-09

Validation Report Number: CCEVS-VR-VID10994-2019

CCTL: Gossamer Security Solutions

Assurance Level: PP Compliant

Protection Profile Identifier:
PP-Module for File Encryption Version 1.0
Protection Profile for Application Software Version 1.3

Summary of this Lecture

- (3.1) Software vulnerabilities
- (3.2) Malware characteristics and mitigation
- (3.3) Saltzer & Schroeder's principles
- (3.x) Software assurance

Recommended Additional Literature

If you are interested, consider taking the courses Cyber Security, Software Security, and Network Security at the department.

Might be available online:
`https://www.su.se/english/library/`

For those who are interested, there are better books on the subject:

- *Secure Software Design* by Richardson and Thies, Jones & Barlett Learning, 2013.
- *Software Security Engineering* by Allen, Barnum, Ellison, McGraw, and Mead, Addison Wesley, 2008.
- *Computer Security - Principle and Practice by Stallings and Brown*, Pearson, 2018.

Bibliography

Oxford Dictionary

"Definition of Computer by Oxford Dictionary on Lexico.com also meaning of Computer" `https://www.lexico.com/definition/computer`

Accessed: 2020-12-01.

Oxford Dictionary

"Definition of Programme by Oxford Dictionary on Lexico.com also meaning of Programme"
`https://www.lexico.com/definition/programme`

Accessed: 2020-12-01.

G. Brookshear

Computer Science - An overview, Pearson, Ed. 5, Indiana, USA, 2005.

G. McGraw

"Testing for security during development: why we should scrap penetrate-and-patch", *IEEE Aerospace and Electronic Systems Magazine*, pp. 13-15, vol. 13, no. 4, IEEE, 1998.

https://ieeexplore.ieee.org/abstract/document/666831

Thank you for your Attention

Jesper Bergman
jesperbe@dsv.su.se

Department of Computer and Systems Sciences, DSV

Stockholm University