

## **Sigurnost računalnih sustava**

# **Sigurnost programske podrške 2**

doc. dr. sc. Ante Đerek

doc. dr. sc. Stjepan Groš

izv. prof. dr. sc. Miljenko Mikuc

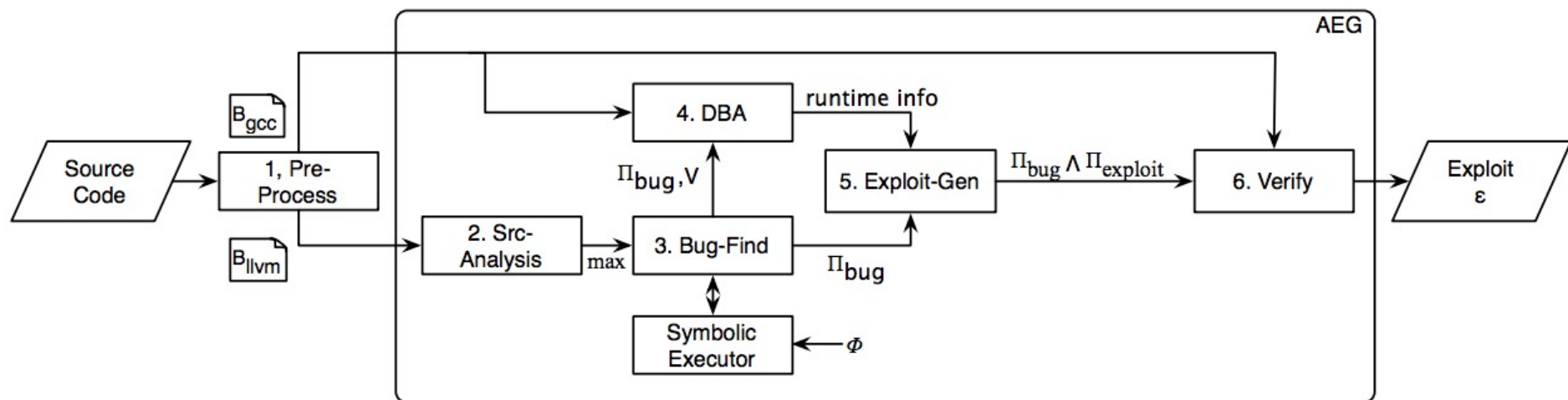
izv. prof. dr. sc. Marin Vuković

# Sigurni programi?

- kako izbjeći pogreške u programima?
- kako izbjeći loše posljedice pogrešaka u programima?
- što su sigurni programi?
  - različiti pogledi različitih korisnika
    - sukladnost zahtjevima?
    - dugo radi bez grešaka?
  - je li “sigurniji” program u kojem je u istom vremenu pronađeno i ispravljeno 100 grešaka ili onaj u kojem je pronađeno i ispravljeno 20?
- bug, error, fault, failure
- testiranje
  - penetrate & patch
  - neočekivano ponašanje - *fuzzing*

# Zašto su pogreške u programima „nezgodne“?

- Pogreška -> ranjivost! (ok, možda ne uvijek)
- Primjer: automatizirano pronalaženje pogrešaka i kreiranje vektora iskorištavanja (*exploit*)



<http://security.ece.cmu.edu/aeg/aeg-current.pdf>

# Taksonomija pogrešaka

- namjerne
  - zlonamjerne i nezlonamjerne
- nenamjerne
  - pogreške pri provjeri valjanosti
    - nepotpuno ili nedosljedno: provjera pristupa
  - pogreška u kontroliranom pristupu podacima
  - serijalizacija: pogreške u programskom toku
  - neadekvatna identifikacija ili autentifikacija
  - narušavanje graničnih uvjeta
    - pogreške u prvom ili posljednjem slučaju
  - logičke pogreške

# Nezlonamjerne pogreške

- preljevi spremnika
  - buffer overflows
  - najčešće ranjivosti
- nepotpuna provjera valjanosti ulaznih parametara
  - posebno na webu
- sinkronizacija provjere i pristupa
  - time-of-check to time-of-use
- najgore od svega: kombinacija navedenog

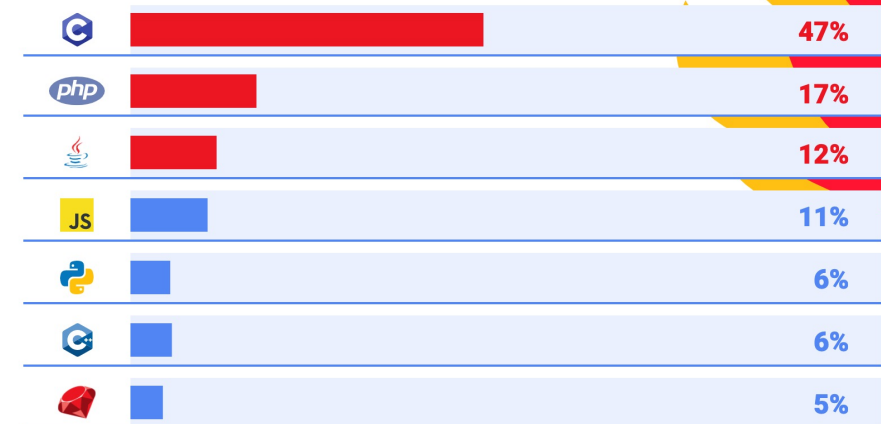
# Statička i dinamička analiza koda

- Statička analiza = analiza izvornog koda
  - Kako je sustav/rješenje izvedeno?
- Dinamička analiza = analiza binarnog (izvršnog) koda
  - Kako sustav/rješenje radi?
- Mnoštvo alata za jednu i drugu opciju
  - Rezultati upitni – uvijek se javljaju novi uzorci koje automatizirani alati ne mogu prepoznati (pogotovo kod statičke analize)

# Koji programski jezik odabrati?

- Koji je „najsigurniji“?
- Što uopće znači „najsigurniji“?
  - 1. slide!
- C nije nužno najnesigurniji nego je najduže prisutan!
- Ipak: možemo promatrati podršku za sigurnost u jezicima
  - Koliko je lako pogriješiti u pisanju koda?
  - Koliko nas programski jezik štiti od grešaka koje će rezultirati ranjivostima?

## TOTAL REPORTED OPEN SOURCE VULNERABILITIES PER LANGUAGE



<https://www.whitesourcesoftware.com/most-secure-programming-languages/>

# Kako nas programski jezik „štiti” od grešaka?

- Razlike između
  - Što je programer htio napraviti
  - Što je prevoditelj (*compiler*) „shvatio” iz napisanog koda
  - Što na kraju radi izvršni kod
- Kako bi nas se moglo zaštititi (kao programere)
  - Minimizirati greške/bugove tijekom izvođenja – „ispadanje” programa
  - Programski jezik ne bi smio dozvoliti nedefiniranost – mogućnost različitih tumačenja – morao bi biti strog u definicijama objekata, tipova...
  - Kod se može pisati dobro (čitljivo, sigurno, jednoznačno...) ili loše (teško za analizu, bez provjera rubnih slučajeva, nedefinirano...)



# Sigurnost tipova podataka i programski jezici (1/3)

- Tip podatka definira razinu apstrakcije (npr. int vs float)
  - koje sve vrijednosti neki tip može poprimiti?
  - Koje sve operacije možemo raditi nad tim tipom podataka?
  - Kako ispravno inicijalizirati i deklarirati objekte/varijable tipa podataka?
- Problem:
  - koliko programski jezik pazi na način pisanja koda u smislu rukovanja tipovima podataka – možemo li napisati kod koji će biti besmislen/pogrešan/sklon ispadanju ali će se ispravno prevesti?
- Strogo tipiziran jezik (*strongly typed*)
  - Strog u definiranju i rukovanju tipovima podataka – „pazi umjesto nas”
  - Provjerava kod u nekom trenutku – najčešće prilikom prevođenja ili tijekom izvođenja (baca grešku)

# Sigurnost tipova podataka i programski jezici (2/3)

- Programski jezik C

```
int x=1;  
int y=3;  
int z=x/y;  
float z2 = x/y;
```

Ispis:

*z je: 0*

*z2 je: 2147483640*

*(problem je u ispisu i castu!)*

- Programski jezik Javascript

```
var x = 1;  
var y = 2;  
var z = 'Nije 3';  
var rez = z + y + x;
```

Ispis:

*Rez je: Nije 321*

- Programski jezik Java

```
short x = Short.MAX_VALUE;  
short y = x+1;  
System.out.println(y);
```

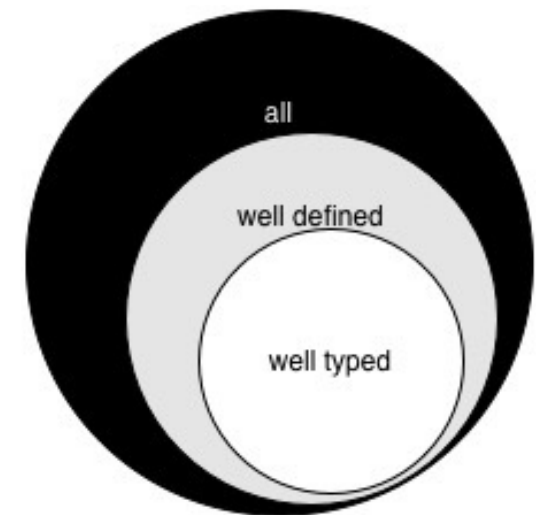
Ispis – greška:

*error: incompatible types: possible lossy*

*conversion from int to short*

# Sigurnost tipova podataka i programski jezici (3/3)

- Slabo tipizirani jezici – C, C++
  - Npr. C i buffer overflow - *well typed* ali nesiguran u ovom smislu
- Djelomično tipizirani jezici – Python, Ruby
  - (Vjerojatno) će javiti grešku tijekom izvođenja
- Strogo tipizirani jezici – Java , C#, Rust
  - Djelom zbog objektnosti i definicije klasa
  - Rukuju pogreškama (npr. *ArrayIndexOutOfBoundsException* ako pokušamo dohvatiti nešto što ne postoji – onda lako možemo „uloviti takvu grešku”)



<http://www.pl-enthusiast.net/2014/08/05/type-safety/>

# Neočekivano ponašanje i programski jezici (1/2)

- Možemo li biti sigurni da će se program izvoditi kako smo mi zamislili?
  - *What you see is not what you get?*
  - Mi vidimo/pišemo jedan kod a prevoditelj ga (možda) prevodi drukčije...
- Što se događa kod izvođenja programa?
  - Kombinacija izvornog koda, prevoditelja i funkcija specifičnih za operacijski sustav (upravljanje memorijom, driveri, programske knjižnice, *garbage collection*...)
- Cilj: spriječiti neželjeno/neočekivano ponašanje
  - Semantika programskog jezika

# Neočekivano ponašanje i programski jezici (2/2)

- Programski jezik C – primjer

```
{int x=0; printf("%d %d\n",x++,x++); }  
{int x=0; printf("%d %d\n",++x,++x); }  
{int x=0; printf("%d %d\n",x=100,x=200); }
```

Ispis (Debian):

```
1 0  
2 2  
100 100
```

Ispis 2 (Mac, FreeBSD):

```
0 1  
1 2  
100 200
```

```
char x=0;  
char arr[10] ;  
arr[10]=22;  
printf("%d\n", x) ;
```

Ispis:

```
0
```

Ali i:

```
*** stack smashing detected ***
```

- Još primjera

- Dijeljenje s 0, pointeri izvan granica, pomaci izvan granica...

# Dakle, koji programski jezik odabrati?

- Strogo tipizirane, uz napomene:
  - Strogo tipizirane jezike teže je naučiti (manje toga je dopušteno, teži za pisanje)
  - Provjera tipova tijekom izvođenja može utjecati na performanse
- Koji otežava „neočekivano ponašanje” – dobra semantika
- Još važnih karakteristika
  - Kako upravlja memorijom?
    - Onemogućuje proizvoljni pristup memoriji
  - Što je s dretvama/nitima i sigurnošću?
    - Kako je uređena sinkronizacija, granični uvjeti?
  - Kontrola toka programa
    - Jasno je kako se program izvodi – npr. pozivi metoda i odgovori – nema neočekivanih poziva ili “neodgovorenih” poziva

# **Sigurnost računalnih sustava**

# **Sigurni razvoj programske podrške**

doc. dr. sc. Ante Đerek

doc. dr. sc. Stjepan Groš

izv. prof. dr. sc. Miljenko Mikuc

izv. prof. dr. sc. Marin Vuković

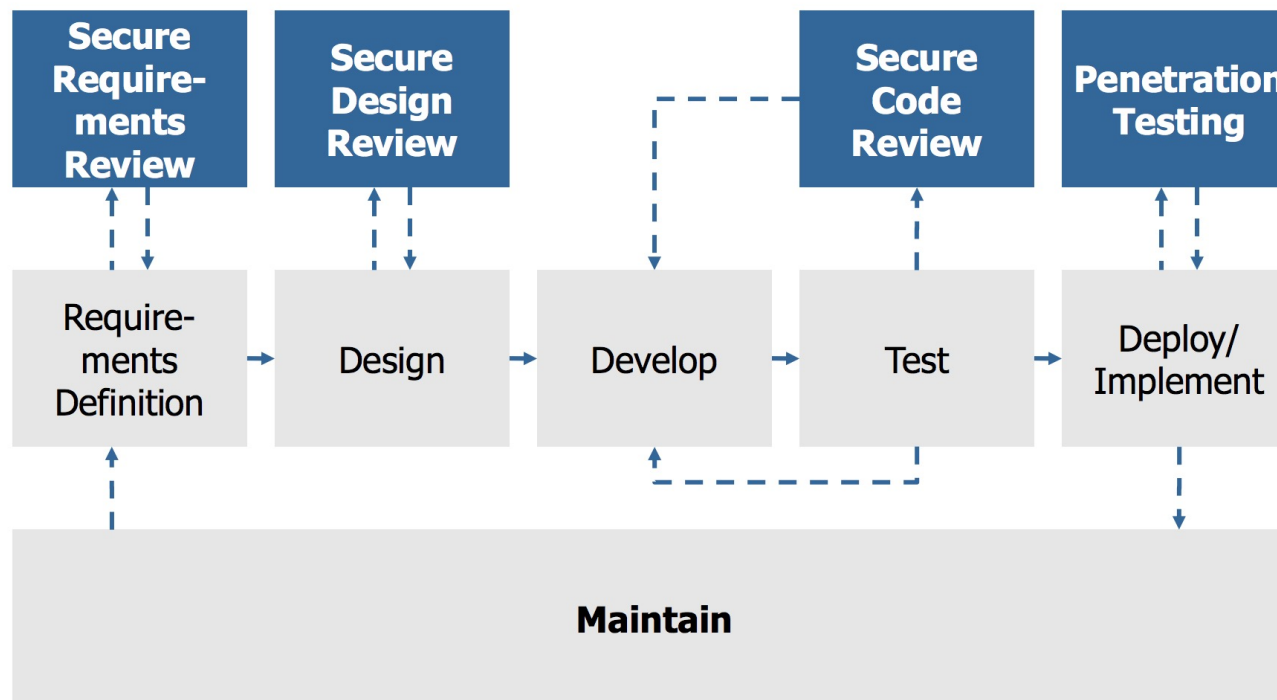
# Sigurnost i SDLC

- *Software Development LifeCycle*
  - *System Development LifeCycle*
  - ***Secure Development LifeCycle***
- 
- Cilj: uključivanje sigurnosti u dizajn sustava od faze skupljanja zahtjeva
    - ◆ *“Trošak otklanjanja ranjivosti tijekom faze dizajna manji je 30-60 puta nego tijekom faze produkcije”*
    - ◆ *NIST, IBM, and Gartner Group*



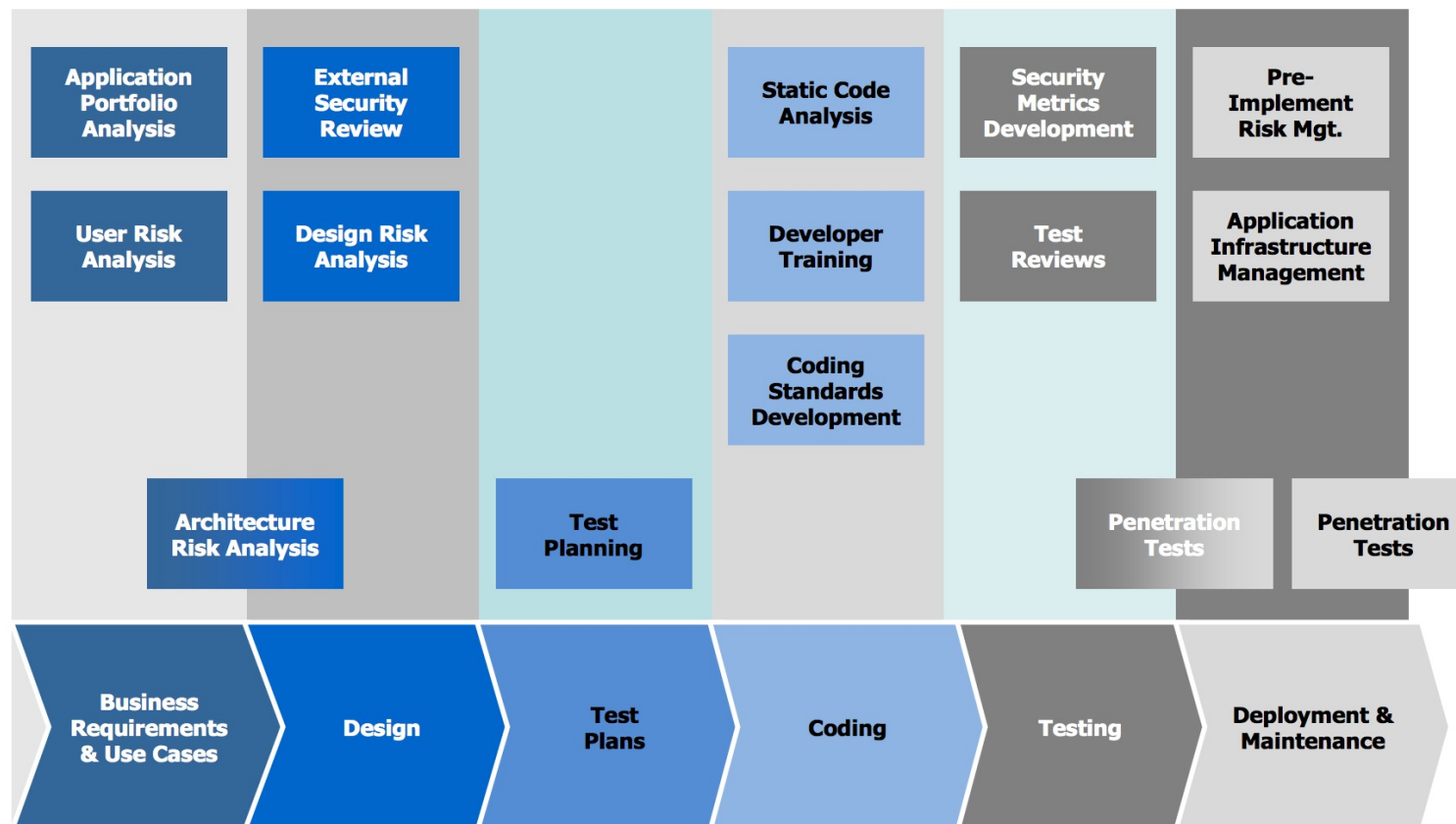
# “Životni ciklus zahtjeva”

*(Ili kako bi razvoj trebao izgledati u kontekstu sigurnosti)*



[https://www.owasp.org/images/7/76/Jim\\_Manico\\_\(Hamburg\)\\_-\\_Securing\\_the\\_SDLC.pdf](https://www.owasp.org/images/7/76/Jim_Manico_(Hamburg)_-_Securing_the_SDLC.pdf)

# Sigurnost i životni ciklus programskog proizvoda



[https://www.owasp.org/images/7/76/Jim\\_Manico\\_\(Hamburg\)\\_-\\_Securing\\_the\\_SDLC.pdf](https://www.owasp.org/images/7/76/Jim_Manico_(Hamburg)_-_Securing_the_SDLC.pdf)

# Koraci prije implementacije sustava

- Što moramo osigurati?
  - ◆ Osnovni i dodatni sigurnosni zahtjevi
- Profili napadača
  - ◆ Protiv koga se borimo?
  - ◆ Tko je najvjerojatniji napadač?
- Identifikacija i klasifikacija entiteta u sustavu
  - ◆ Što sve moramo štititi?
- Postavljanje arhitekture
  - ◆ Kako ćemo osigurati osnovne sigurnosne zahtjeve kroz dizajn i podjelu ovlasti u okviru komponenata sustava

# Smjernice za siguran dizajn

Minimizacija prostora za napad (*Minimize attack surface area*)

Definiranje sigurnih početnih postavki  
(*Establish secure defaults*)

Princip najmanjih prava  
(*Principle of Least privilege*)

Princip obrane u dubinu  
(*Principle of Defense in depth*)

Sigurno ispadanje (*Fail securely*)

Ne vjerujte vanjskim uslugama  
(*Don't trust services*)

Razdvajanje zaduženja  
(*Separation of duties*)

Izbjegavajte sigurnost prikrivanjem  
(*Avoid security by obscurity*)

Jednostavna sigurnost  
(*Keep security simple*)

Ispravne sigurnosne zakrpe  
(*Fix security issues correctly*)

# Minimizacija prostora za napad (1/2)

- *Minimize attack surface area*
- Primjer: pretraga na stranici koja je podložna na napad umetanjem koda
  - Što ako je vidljiva svima?
  - Što ako je vidljiva samo ulogiranim korisnicima?
  - Što ako se upiti validiraju (...)?
  - Što ako je uopće nema?
- Svaka funkcionalnost potencijalno otvara sigurnosne propuste
  - Princip minimizacije jest da ne dodajemo funkcionalnosti koje nisu doista potrebne čime ćemo efektivno smanjiti sigurnosne rizike

# Minimizacija prostora za napad (2/2)

- Što je sve prostor za napad?
  - svi mogući putevi kojima podaci i naredbe ulaze ili izlaze iz sustava
  - kod koji štiti te puteve (šifrira, nadzire, autentificira...)
  - svi korisni podaci pohranjeni u sustavu
  - sav kod koji štiti te podatke (šifriranje, integritet...)
- Što je manje puteva i podataka u sustavu – prostor za napad je manji!

# Definiranje sigurnih početnih postavki (1/2)

- *Establish secure defaults*
- Primjer: Generirane lozinke kod registracije korisnika
  - Obično se želi olakšati korisnicima pa su lozinke jednostavnije
  - Uvesti *password policy* (znakovi, velika slova, duljina...)
  - Korisnicima omogućiti da na vlastitu odgovornost pojednostave lozinke

## Definiranje sigurnih početnih postavki (2/2)

- Iako se korisnicima želi olakšati, početne postavke trebale bi biti maksimalno sigurne
  - Ne koristiti jednu *default* lozinku (tipično za uređaje)
  - Slijede problemi kompleksnosti korištenja i konkurencije
  - Sigurnost je često “dosadna i komplicirana” (npr. Zigbee)
- Omogućiti korisnicima da na vlastitu odgovornost smanje razinu sigurnosti i olakšaju korištenje
  - Ovisno o namjeni
  - Bi li korisnici bili spremni smanjiti vlastitu sigurnost da su svjesni prijetnji?



# Princip najmanjih prava (1/2)

- *Principle of Least privilege*
- Primjer: Wordpress
  - Koji korisnik izvršava Wordpress?
  - Koje mu ovlasti trebaju nad datotekama?
  - Često je najlakše dati maksimalne ovlasti “jer tada sve radi” (studenti ali i iskusniji razvijatelji)

## Princip najmanjih prava (2/2)

- Korisnici / dijelovi sustava / servisi bi trebali imati samo ona prava na resurse koja im doista trebaju
  - Pri tome su resursi disk, memorija, mreža, integrirani servisi ali i procesor
- Da bi sve ovo bilo moguće, funkcionalnosti sustava moraju biti precizno raspodijeljene
  - ◆ kako bi se svakom npr. procesu mogla dati odgovarajuća ovlast

# Princip obrane u dubinu (1/2)

- *Principle of Defense in Depth*
- Primjer 1: Dvorac (...)



<http://www.medievalwall.com/architecture/fortress-medvedgrad/>

- Primjer 2: Ranjivo administratorsko sučelje
  - ◆ Npr. kada je korisnik ulogiran kao administrator onda putem forme za dodavanje novog korisnika može napraviti napad umetanjem
  - ◆ Ali: rizik da anonimni korisnik napravi napad umetanjem je minimalan ako je to sučelje osigurano na ostalim razinama – primjerice da se napadač ne može prijaviti kao administrator

## Princip obrane u dubinu (2/2)

- Kako bi spriječili ranjivost naizgled je dovoljno staviti jedan mehanizam kontrole
  - ◆ Princip obrane u dubinu zapravo govori da uvijek treba biti više mehanizama obrane
- Dobra praksa - više neovisnih mehanizama koji štite isti dio sustava
  - ◆ Sprječava ili barem smanjuje utjecaj napada
- Sjetimo se principa sigurnih početnih postavki
  - ◆ Previše sigurnosti može učiniti sustav prekompleksnim za korisnike -> paziti na to!
  - ◆ Problem uporabljivosti – inače problem kod povećane sigurnosti
- Ali, očekujemo i da sigurnost bude jednostavna...

# Sigurno ispadanje (1/2)

- *Fail securely*
- Primjer: Elektronska brava
  - ◆ Što se događa kada nema napajanja?
- Svaki sustav će najvjerojatnije nekada ispasti
  - ◆ zbog greške ili napada
  - ◆ važno je što će sustav izvesti u tom trenutku

# Sigurno ispadanje (2/2)

- Što radi kod?

```
isAdmin = true;
try {
    codeWhichMayFail();
    isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex)
{
    log.write(ex.toString());
}
```

[https://www.owasp.org/index.php/Fail\\_securely](https://www.owasp.org/index.php/Fail_securely)

- Dodatni termini

- ◆ *Fail Open* ili *Fail Close*?
- ◆ Primjer s elektronskom bravom (prije)
- ◆ Što je s nuklearnim oružjem?



# Ne vjerujte vanjskim uslugama

- *Don't trust services*
- Primjer: Vanjski CRM
  - ◆ Podaci korisnika se sigurnim kanalom šalju u vanjski CRM koji ima sigurnosne ranjivosti
  - ◆ Podaci CRM-a se vraćaju u naš sustav – mogu sadržavati umetanja, prelijevanja spremnika i slično – uvijek potrebna validacija
- Ipak, uvijek će se koristiti neke vanjske usluge na koje nećete imati utjecaj
  - ◆ Provjera i zaštita podataka u oba smjera!

# Razdvajanje zaduženja (1/2)

- *Separation of Duties*
- Ključno sa gledišta kontrole prevara
- Primjer 1: Naručivanje artikala
  - ◆ Je li u redu da ista osoba odlučuje, naručuje i preuzima artikle?
- Primjer 2: Web trgovina
  - ◆ Administrator bi trebao imati poseban pogled ali ne i imati ovlast obavljanja akcija kao druge razine korisnika (npr. kupnja)
  - ◆ Problem porecivosti!
  - ◆ Nema preklapanja između funkcionalnosti (pa tako ne mora biti preklapanja ni između ovlasti)



## Razdvajanje zaduženja (2/2)

- Web – tipičan problem korisnika s ovlasti administratora
  - ◆ Administratori nikada ne bi smjeli imati ulogu običnih korisnika
- Rješenje problema:
  - ◆ Razmisliti koliko štete može počinuti jedan korisnik?
    - ◆ (Odnosno napadač koji otme sjednicu korisniku)
  - ◆ Za kritičnije radnje potreban “potpis”/ akcija više korisnika
  - ◆ Zahtjeva raspodjelu procesa u sigurnosnom smislu

# Izbjegavajte sigurnost prikrivanjem

- *Avoid security by obscurity*
- Primjer: izvorni kod Linuxa
  - ◆ Cijeli izvorni kod je dostupan a Linux je i dalje siguran
- Tajne treba čuvati ali činjenica da je npr. Izvorni kod tajan ne smije biti jedina linija obrane
  - ◆ Kod bi trebao biti siguran i onda kada je javno dostupan
  - ◆ Pravilno implementirani mehanizmi su ti koji ostvaruju sigurnost!

# Jednostavna sigurnost (1/2)

- *Keep security simple*
- Primjer 2: Pretjerana i neatomatizirana obfuskacija koda

## Jednostavna sigurnost (2/2)

- Često se sigurnost pokušava uvesti dodavanjem provjera i povećavanjem složenosti sustava
  - ◆ Posebno u smislu arhitekture
  - ◆ Posebno kada se ugrađuje u već postojeći sustav
  - ◆ “Gorak sustav”
- Činjenica je da takav pristup (uz sigurnost) uvodi i veću vjerojatnost za greške i ranjivosti
- Zaključak: sve riješiti na najjednostavniji mogući način
- Ipak, razmisliti o obrani u dubinu i ne zanemariti više razina obrane

# Ispravne sigurnosne zakrpe

- *Fix security issues correctly*
- Primjer 1: Uočena je elevacija prava u segmentu aplikacije
  - ◆ Najlakše je izravno popraviti taj segment i zaboraviti na sve
  - ◆ Ali, treba razmisliti koristi li se isti kod još negdje
  - ◆ Nakon popravka treba testirati funkcionalnost (jer popravci često uvedu nove pogreške!)
- Ne žuriti s popravkom nego pokušati doći do stvarnog uzroka
- Ako su korišteni uzorci dizajna onda je vjerojatno da je greška propagirala po sustavu (ili više njih)

## **Sigurnost računalnih sustava**

# **Sigurnost operacijskih sustava**

doc. dr. sc. Ante Đerek

doc. dr. sc. Stjepan Groš

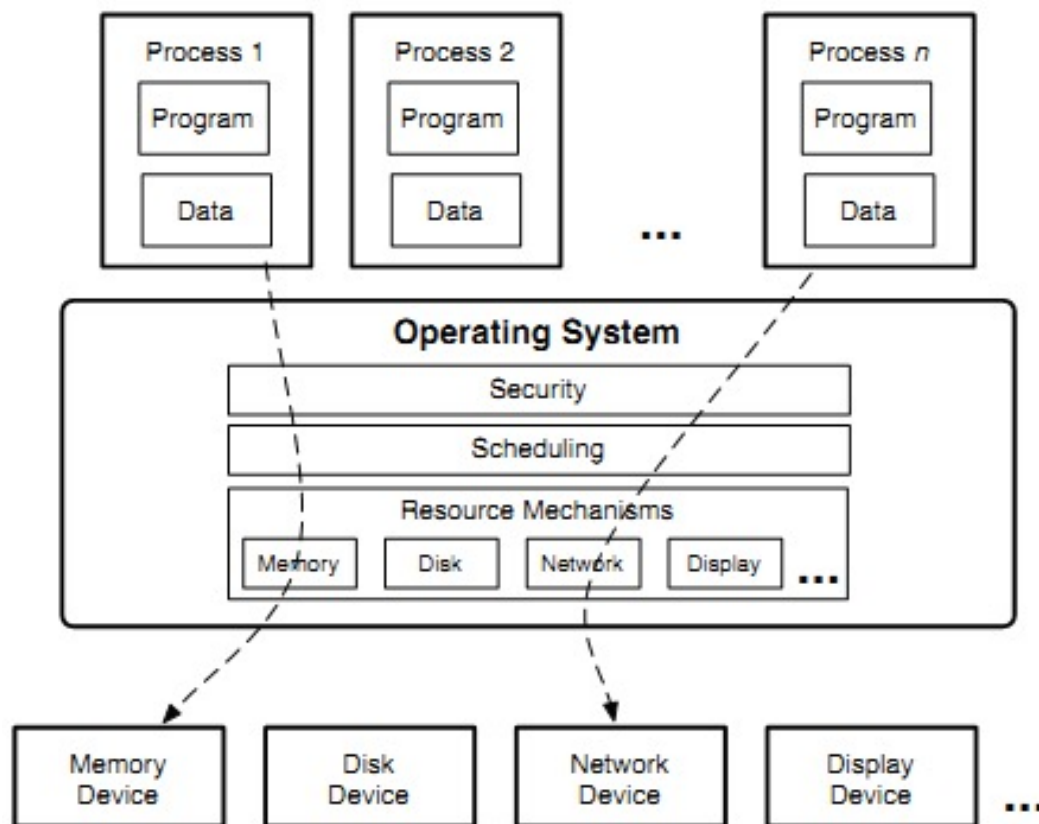
izv. prof. dr. sc. Miljenko Mikuc

izv. prof. dr. sc. Marin Vuković

# Operacijski sustav...

- operacijski sustav - program ili skup programa koji
  - povezuju i objedinjuju sve sklopovske dijelove računala
  - omogućuju njihovu djelotvornu uporabu
- podloga svim ostalim programima koji se izvode na računalu
  - olakšava i pojednostavnjuje njihovo izvođenje
  - identifikacija, autentifikacija, pristup datotekama, komunikacija procesa, vanjske jedinice, memorija, mreža, biblioteke
- višekorisnički rad
  - kako da korisnici/procesi ne ometaju jedan drugog?
- osnovne metode: razdvajanje i dijeljenje

# Uloga operacijskog sustava



“Operating system security”, Trent Jaeger



# Razdvajanje

- osnova zaštite – objekti jednog korisnika nisu vidljivi drugom korisniku
  - fizičko razdvajanje
  - vremensko razdvajanje
  - logičko razdvajanje
  - kriptografsko razdvajanje
- jednostavnost implementacije vs. pružena sigurnost
  - učinkovitost resursa
- *sandboxing*
- osim razdvajanja, želimo i dijeljenje resursa (*sharing*)
  - npr. poziv iste funkcije iz dva procesa

# Dijeljenje

- različite razine
  - bez zaštite
  - izolacija: procesi ne “vide” jedan drugog
  - “dijeli sve” ili “ne dijeli ništa” – *public/private*
  - dijeljenje uz ograničenje pristupa – kontrola pristupa
  - dijeljenje prema mogućnosti – dinamičko pravo pristupa
  - ograničenje korištenja objekta – kontrolira pristup i ono što se s objektom radi
- složenost implementacije vs. prikladnost zaštite
- pristup podacima na više razina: bit, byte, word, polje, zapis, datoteka, disk (*volume*) – granularnost

# Ovlasti – primjer Linux/macOS (1/4)

- Ovlasti korisnika i grupa – jednostavno a izuzetno važno za osiguravanje sustava
- Česte greške? Previše ovlasti
  - Uvijek imati na umu princip najmanjih nužnih ovlasti!
- Pitanje: tko sve smije čitati/pisati/pokretati ovu datoteku/direktorij?

# Ovlasti – primjer Linux/macOS (2/4)

- Podjela na tri skupine ovlasti
  - Vlasnik (owner) - **u**
  - Grupa korisnika (group, staff) - **g**
  - Svi ostali (other, all, everyone) – **o, a**
- Podjela na tri skupine akcija
  - Čitanje (read) - **r**
  - Pisanje (write) - **w**
  - Pokretanje (execute)\* - **x**
  - Bez ovlasti -

\* ako se radi o direktoriju, onda *execute* podrazumijeva dozvolu ulaska

# Ovlasti – primjer Linux/macOS (3/4)

- Neki primjeri...

Linux

```
lrwxrwxrwx 1 user user 18 Apr 10 22:35 .bash_profile -> /home/user/.bashrc
-rw-r--r-- 1 user user 2355 Apr 10 22:35 .bashrc
drwxr-xr-x 1 root root 4096 Apr 10 22:35 ..
```

MacOS

```
-rw----- 1 vukovic staff 771 Apr 17 2013 ssh_key
-rw-r--r-- 1 vukovic staff 623 Apr 17 2013 ssh_key.pub
```

- d – direktorij
- l – link, veza na neku datoteku

# Ovlasti – primjer Linux/macOS (4/4)

- Standardni način imenovanja: r,w,x za svaku ulogu
- Brojeve reference - oktavno
  - Brojevi predstavljaju r,w i x (r=4, w=2, x=1)
  - Npr.  $rwX = 4+2+1 = 7 \rightarrow rwxrwxrwx = 777$ 
    - svi mogu sve
  - Npr2.  $rw- = 4+2 = 6, r--=4 \rightarrow 644$ 
    - vlasnik može čitati i pisati (ne može pozvati kao program), a grupa i ostali mogu samo čitati
- Važne naredbe
  - `chmod 644 file.txt` -> mijenja ovlasti nad datotekom file.txt
  - `chown user:group file.txt` -> mijenja vlasnika / grupu datoteci

`rwxrwxrwx`  
owner group other

<code>rwX</code>	<code>r--</code>	<code>---</code>
<code>111</code>	<code>100</code>	<code>000</code>
<code>7</code>	<code>4</code>	<code>0</code>

<code>rw-</code>	<code>r--</code>	<code>r--</code>
<code>110</code>	<code>100</code>	<code>100</code>
<code>6</code>	<code>4</code>	<code>4</code>

# Standardni „*sandboxing*” kod operacijskih sustava

- Operacijski sustavi djelomično i rade sandboxing
  - samo se tu naziva „razdvajanje” (opisano prije nekoliko slideova)
- Memorija
  - *User space* (korisnički prostor)
    - dio memorije u kojem se izvode korisnički programi – sve što nije *kernel* (jezgra)
    - još razdvajanja: kako spriječiti da se *user* procesi međusobno ometaju, čitaju podatke...
    - Pristup jezgrenom dijelu isključivo kroz sistemske pozive
  - *Kernel space* (jezgreni prostor)
    - dio memorije u kojem se izvodi jezgra sustava
    - Ima pristup čitavoj memoriji
- Procesi i međusobno ometanje/čitanje...?
  - Jezgra raspoređuje procese (*kernel scheduler*) – vremenski (CPU) i logički (pristup memoriji)

# „*Sandboxing*” kao koncept

- Koncept kod kojeg se svaki proces/aplikacija izvodi u svojem „pješčaniku”
  - Za „izlazak iz pješčanika” treba posebne dozvole – npr. korištenje mrežne kartice
  - Princip najmanjih potrebnih prava (*least privilege*) – smije raditi/koristiti samo ono što stvarno treba
- Appleov standard na operacijskim sustavima MacOS i iOS
- Za ostale sustave postoje različita rješenja
  - Kontejneri, izolacija procesa...



# Za što se sve *sandboxing* koristi?

- Na ovom predavanju pričamo u kontekstu operacijskog sustava i mehanizama zaštite
- Inače se koncept sandboxinga koristi i za izolaciju sumnjivih programa, npr:
  - Analiza zloćudnog koda (malwarea)
  - Korištenje programa kojima ne vjerujete (ako ih već morate koristiti)
  - Laboratorijska vježba iz sigurnosti weba?
  - ...

# Apple i sandbox (1/2)

- Razvijatelji aplikacije moraju deklarirati na što njihova aplikacija „polaže prava” (*entitlement*)
  - Jedino tako mogu u službeni AppStore

`com.apple.security.network.client`

Network socket for connecting to other machines

For details, see [Enabling Network Access](#).

---

`com.apple.security.network.server`

Network socket for listening for incoming connections initiated by other machines





For details, see [Enabling Network Access](#).

<https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/EnablingAppSandbox.html>

- Primjer pristupa disku
  - Aplikacija može kreirati datoteku ali ima pristup isključivo njoj, ne i npr. ostatku direktorija (*com.apple.security.files.user-selected.read-write*)

# Apple i sandbox (2/2)

- Problem: što ako se aplikaciju izravno preuzme?
  - Više ne vrijede stroga pravila Applea i aplikacija ne mora biti u sandbox modu
- Sigurnosne dozvole
  - Dodatak na sandboxd, aplikacije mogu zatražiti pristup osjetljivijim funkcionalnostima a korisnik im to mora odobriti (poznato s iOSa i novijih Androida)

Process Name	Threads	User	Memory	Sandbox
Google Chrome Helper (Rend...	22	vukovic	173,9 MB	Yes
Google Chrome Helper (GPU)	10	vukovic	829,1 MB	Yes
Google Chrome Helper (Rend...	21	vukovic	80,7 MB	Yes
WindowServer	10	_windowserver	633,0 MB	Yes
 Google Chrome	30	vukovic	576,0 MB	No
 Microsoft PowerPoint	33	vukovic	597,9 MB	Yes
Google Chrome Helper (Rend...	20	vukovic	165,2 MB	Yes
 Activity Monitor	4	vukovic	34,3 MB	No
screencapture	2	vukovic	3,2 MB	No
Google Chrome Helper (Rend...	17	vukovic	95,7 MB	Yes
kernel_task	210	root	437,0 MB	No
sysmond	3	root	6,6 MB	No
Google Chrome Helper (Rend...	18	vukovic	255,9 MB	Yes
coreaudiod	6	_coreaudiod	5,3 MB	Yes
 iTunes	18	vukovic	124,8 MB	No

# Linux i rješenja za sandboxing (1/2)

- Nekoliko ugrađenih mehanizama i puno dodatnih rješenja za neki (ili kompletni) sandboxing
  - <https://www.Opensourceforu.com/2016/07/many-approaches-sandboxing-linux/>
- *Namespace*
  - Izolacija procesa u smislu pristupa resursima
  - Resursi: datotečni sustav (*mnt*), procesi (*pid*), mreža (*net*), imena domena i *hostova* (*uts*), korisnici (*UIDs*), komunikacija između procesa (*ipc*)
  - Definira se čemu sve proces može pristupiti i kako „vidi” resurse

# Linux i rješenja za sandboxing (2/2)

- **Seccomp (*Secure Computing mode*)**
  - Proces se prebacuje u sigurni način rada u smislu da ne može pozivati sistemske pozive osim *exit()*, *sigreturn()*, *read()* i *write()*
  - Pri tome ima pristup samo onim resursima kojima je imao u trenutku poziva
  - Moguće ga je ugraditi u aplikacije (zašto to može biti nezgodno?) ili pokretati aplikacije u seccomp načinu rada
  - <https://man7.org/linux/man-pages/man2/seccomp.2.html>
- **Nadogradnja: seccomp-bpf (*Berkeley Packet Filter*)**
  - Rješava strogoću seccomp-a u smislu ograničavanja sistemskih poziva te omogućuje definiranje pravila po principu BPF pravila
- **Dodatno: Firejail, Docker**
  - <https://www.linux.org/threads/sandboxing-with-firejail.27106/>
  - [https://docs.docker.com/engine/security/trust/trust\\_sandbox/](https://docs.docker.com/engine/security/trust/trust_sandbox/)

# Windows i rješenja za sandboxing

- Tek od 2008. i Viste – *User Access Control* (UAC)
  - Sve aplikacije se standardno izvode pod običnim korisnikom i za sve kritičnije akcije trebaju administratorske ovlasti koje korisnik mora eksplicitno odobriti
- Windows 10 – Windows Sandbox
  - Sandbox u pravom smislu, no ne u smislu standardne zaštite operacijskog sustava nego isključivo za izoliranje sumnjivih programa
  - Potpuna izolacija procesa i brisanje svih podataka nakon korištenja
  - <https://techcommunity.microsoft.com/t5/windows-kernel-internals/windows-sandbox/ba-p/301849>

# Šifriranje datoteka/diska

- Još jedna mogućnost zaštite podataka - „*data at rest*”
- Treba li stvarno šifrirati čitav disk?
  - Primjer kuće: sef ili alarmni sustav?
- Različita rješenja na različitim operacijskim sustavima
  - Linux – dm-crypt, PGP
  - MacOS – FileVault
  - Windows – BitLocker
- Opcije: šifriranje pojedinačnih datoteka („ručno”) ili automatizirano šifriranje čitavog diska (WDE, FDE)
- Opcije 2: šifriranje diska i šifriranje *boot* diska

# Šifriranje čitavog (*boot*) diska - koncept

- Ako se šifrira čitav *boot* disk, kako se podiže operacijski sustav?
  - Tijekom podizanja sustava prvo se učitava operacijski sustav (*boot loader*)
  - No, ako je disk šifriran, prvo se mora pokrenuti sučelje u kojem korisnik „otključava” zapis operacijskog sustava (*pre-boot*) – tek nakon toga je moguće uobičajeno pokretanje čitavog sustava
- Kako se korisnik autentificira?
  - Lozinka, biometrija, pametna kartica, sigurno sklopovlje – TPM - HSM...
- Što se događa nakon toga?
  - Datoteke na disku ostaju šifrirane, a pristup datoteci od strane korisnika je dešifrira i drži u memoriji – pisanje datoteke inicira ponovno šifriranje



# Šifriranje čitavog diska – BitLocker

- Uveden od operacijskog sustava Windows Vista
- Postoji jedna nešifrirana particija diska s koje se podiže sustav (pre-boot)
- Tri načina rada
  - *Transparent operation mode* – ključ je pohranjen u sklopovlju (*Trusted Platform Module*)
  - *User authentication mode* - korisnik se mora autentificirati (prošli slide, lozinka, PIN – mogućnost sigurnosne kopije u oblaku)
  - *USB Key Mode* – korisnik ima USB na kojem se nalazi ključ za dešifriranje (ili čitač kartice)
- <https://www.anoopcnaair.com/bitlocker-unlocked-behind-the-scenes-windows-10/>

# Šifriranje čitavog diska – dm-crypt i LUKS

- dm-crypt
  - Šifriranje logičkih blokova – apstrakcija zbog koje može šifrirati različite strukture
  - Može šifrirati i boot diskove (na nekim Linux distribucijama - initrd)
  - Šifrirani disk / particiju mount-a kao datotečni sustav
  - Šifrirani ključ se pohranjuje u zaglavlju šifriranih podataka/diska
  - Naredbe cryptsetup i cryptmount
- LUKS (*Linux Unified Key Setup*)
  - Podrška dm-cryptu – upravljanje ključevima i postavljanjem svega
  - Veća razina sigurnosti - SALT

<https://wiki.archlinux.org/index.php/dm-crypt>

# Hvala!