SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Diplomski studij

Razvoj komunikacijske programske podrške

Ak. God. 2021/22.

Sigurni razvoj programske podrške

Sigurnost i SDLC

Software Development LifeCycle

System Development LifeCycle

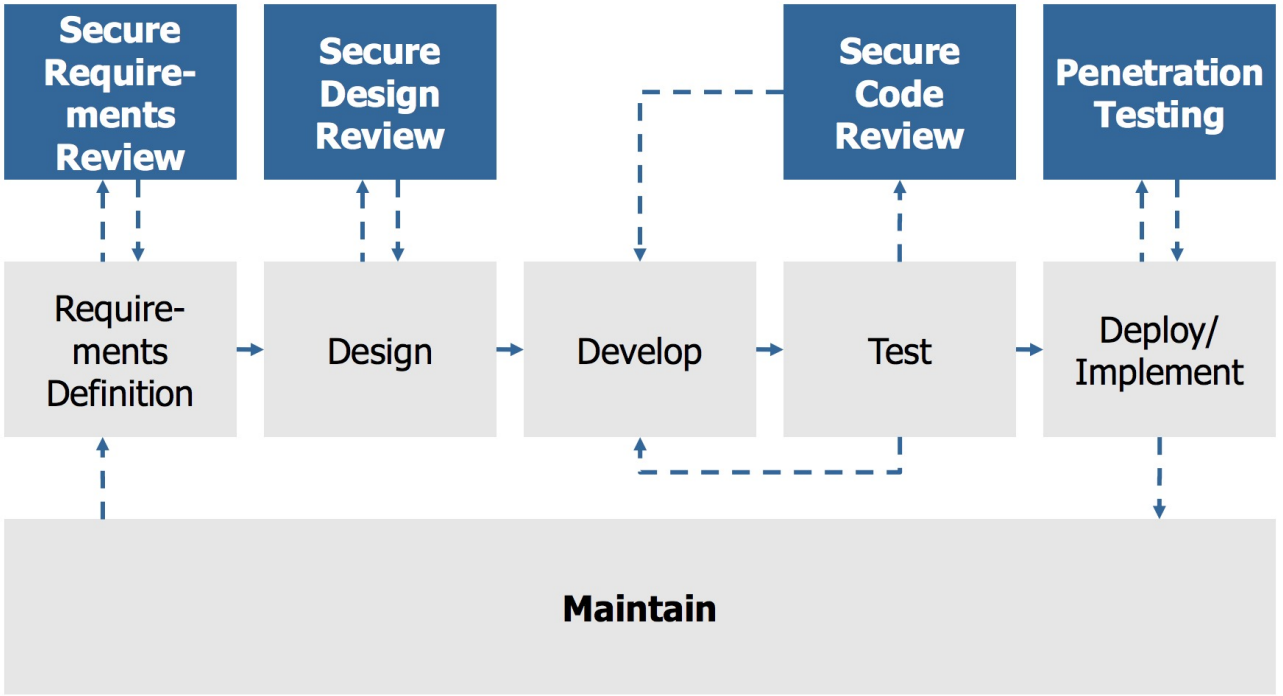
Secure Development LifeCycle

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*

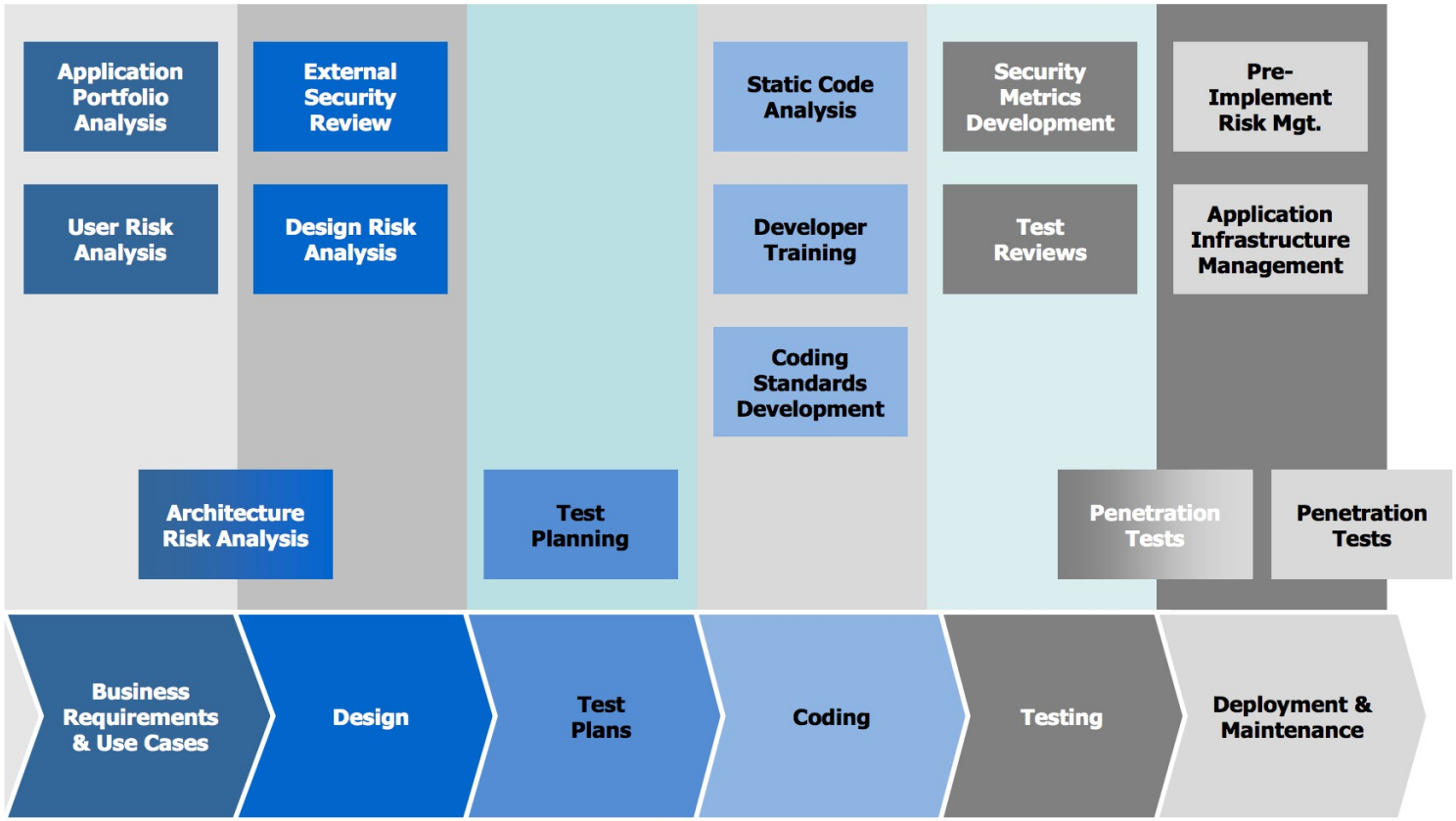
“Životni ciklus zahtjeva”

(Ili kako bi razvoj trebao izgledati u kontekstu sigurnosti)



https://www.owasp.org/images/7/76/Jim_Manico_(Hamburg)_-_Securiing_the_SDLC.pdf

Sigurnost i životni ciklus programskog proizvoda



https://www.owasp.org/images/7/76/Jim_Manico_(Hamburg)_-_Securiing_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 vidjljiva 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

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

Primjer 1: Dvorac (…)

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

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 počiniti 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 ostvarjuju 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)