XI – Sigurnosni aspekti programiranja

S A D R Ž A J

11.1 – Osnovni pojmovi

- Sigurna aplikacija odnosi se na otpornost prema nekim vrstama napada
- Poslednjih godina statistika pokazuje da se 70% sigurnosnih propusta dešavaju na aplikativnom sloju.
- Siguran program je od vitalnog značaja za održavanje sigurnog sistema
- Princip zatvorenog koda ne predstavlja ozbiljnu prepreku za napadača: upotrebom disasemblera može se ponovo doći do osnovnog koda.
- Nezaštićen program je tempirana bomba – on funkcioniše dok su mu okolnosti naklonjene, tj. dok ne dođe do potencijalnog napada.
- Sigurnost se ne sme zasnivati na spoljnoj sigurnosti mreže na štetu unutrašnje sigurnosti programskog koda.
- Pravila „sigurnog programiranja“ podrazumevaju određena odstupanja od programske jednostavnosti a često i od njegove efikasnosti
- Argument za to je prilično ubedljiv: u slučaju pada sistema, do tada efikasan kod više nikome ne koristi
- Siguran programski kod štiti korisnika i od njega samog
- Neko može biti zloupotrebljen bez svog znanja i pristanka
- Sigurnost često rezultira umanjenjem efikasnosti i performansi sistema

11.1 – Osnovni pojmovi

- Standardne tehnike razvoja programskih paketa potpuno su neprikladne za kreiranje sigurnih aplikacija zbog svoje orijentisanosti na ispravno funkcionisanje programa i potpunog ignorisanja ostalog.
- Jedan od primera takvog pristupa je implementacija funkcionalnosti koja uključuje izvođenje neke akcije B ukoliko se pritisne taster A.
- Istovremeno se ne vodi računa o tome šta će se dogoditi ako korisnik pritisne taster C ili istovremenog pritiska kombinacije tastera C i D.
- Sigurnost programskog koda odnosi se na otpornost aplikacije na razne zloupotrebe napadača koje dovode do nedostatka funkcionalnosti
- U realnim situacijama postoji mnogo načina za uzrokovanje stanja uskraćivanja usluga neke aplikacije.
- Korisnik (napadač) može, namerno ili slučajno, uneti neodgovarajuće podatke u program te na taj način učiniti program nedostupnim
- Aplikacije reaguju različito na takve napade.
- Neke od njih se jednostavno sruše bez ikakvih poruka o grešci, neke se samo nepravilno ponašaju, a neke prouzrokuju stanje uskraćivanja resursa celog operativnog sistema.

11.1 – Osnovni pojmovi

- Program koji usled napada prouzrokuje stanje uskraćivanja usluga celog sistema smatra se potpuno neprihvatljivim sa gledišta sigurnosti.
- Razvoj sigurnih aplikacija prilično se razlikuje od razvoja običnih apl.
- Problemi koji se najčešće javljaju u programima (prepisivanje bafera) smatraju se i najozbiljnijim sigurnosnim propustima današnjice.
- U početku, programer je u celosti sam razvijao zadati program
- Danas, programiranje aplikacija koje izvršavaju kompleksne zadatke deli se među timovima programera koji pišu milione linija koda.
- Zato je potrebno, kod velikih programa, izraditi dokumente sa preciznim dizajnom koji prikazuju šta koji deo koda radi i kako se ponaša pri interakciji sa ostalim delovima programa.
- Kada programer završi pisanje određenog dela koda, ostali članovi tima moraju potpuno pregledati njegov dizajn i/ili programski kod.
- Osnovna načela razvoja programa diktiraju pisanje malih, nezavisnih jedinica koje se nazivaju ***programski moduli***.
- Svaki modul treba da se izoluje od štetnog uticaja ostalih modula, a to se može postići sakrivanjem delova programa (***enkapsulacijom***).

11.2 – Kategorije mogućih napada

1. **Subverzija aplikacije**
   - ✓ prouzrokuje izvršavanje neke nenameravane radnje
2. **Subverzija sistema i spoljnih aplikacija**
   - ✓ Iskorišćavanje sigurnosnih propusta aplikacije utiče na druge pokrenute aplikacije ili sistemske resurse.
   - ✓ Napad ne ugrožava samo pokrenutu aplikaciju nego se ta aplikacija često koristi i kao kanal za napad na druge sisteme, aplikacije pa i sam OS.
   - ✓ Napadi ovog tipa često uključuju i izvršavanje drugih aplikacija, kao što je komandni interpreter u OS UNIX, ili iskorišćavanje veze koju je druga aplikacija ostvarila sa mrežom.
3. **Prekidi funkcionalnosti**
   - ✓ Svaki oblik odbijanja usluga, u šta spada i grubo rušenje aplikacije

11.3 – Najčešće greške programera

- Razvoj sigurnih programa je proces koji zahteva opreznost i obazrivost pri svakom koraku i to na svim nivoima organizacije.
- Potrebne su stroge sigurnosne specifikacije, razvojni programeri sa mnogo iskustva i znanja na području sigurnosti kao i grupu za procenu kvaliteta(*Quality Assessment team*) za otkrivanje sigurnosnih problema
- **Najčešći problemi** koji čine većinu sigurnosnih propusta koji dovode do ranjivosti programa su:

1. **Pprepunjavanje bafera**
2. **Ranjivosti vezane za oblikovanje znakovnih nizova – *string*-ova**
3. **Autentifikacija**
4. **Autorizacija**
5. **Slabi kriptografski algoritmi**
6. **Problem kod simultanog korišćenja resursa**

Otkrivanjem i uklanjanjem ovih šest problema moguće je podići nivo sigurnosti aplikacija, a time i celog računara.

11.3 - Prepisivanje bafera

- Jedan od najčešće zloupotrebljavanih sigurnosnih propusta
- Ako postoji bilo kakva greška prepisivanja bafera unutar programa, napadač je može iskoristiti za potpuno preuzimanje kontrole sistemom.
- Osnovni uzrok problema je upotreba statičnih promenljivih fiksne veličine za čuvanje ulaz.podataka tako da zlonamerni napadač može prepisati bafere u kojima se čuvaju ulazni podaci te ih zloupotrebiti
- Potrebno je proveravanje svih ulaznih podataka pre njihovog kopiranja u bafere: ako veličina ulaznih podataka prekoračuje veličinu bafera, treba zabeležiti izuzetak i promeniti tok programa
- Implementacija ove provere može biti dug i zamoran proces ali postoje mnogi alati koji automatski prepoznaju spomenutu situaciju.
- Oni samo otkrivaju propust,ali programer još uvek mora rešiti problem
- Potrebno je mnogo veštine i znanja o OS, kompajlerima i mašinskom kodu da bi se mogao napisati program koji iskorišćava ovaj propust.
- Zbog toga mnogi ljudi sigurnosni rizik pogrešno ocenjuju minimalnim.
- Postoje gotovi programi (sajt ***SecurityFocus*** i ***Packetstorm***), sa kojim je moguće zloupotrebiti propust prepisivanja bafera.

11.3 – Prepisivanje bafera

- Do prekoračenja bafera dolazi ukoliko se prilikom upisa podataka u bafer nedovoljno ili uopšte ne vodi računa o veličini tog bafera, pa se podaci upišu na susedne memorijske lokacije.

Primer: 8 B bafer ***A*** (niz karaktera) i celobrojna vrednost ***B*** veličine 2 B

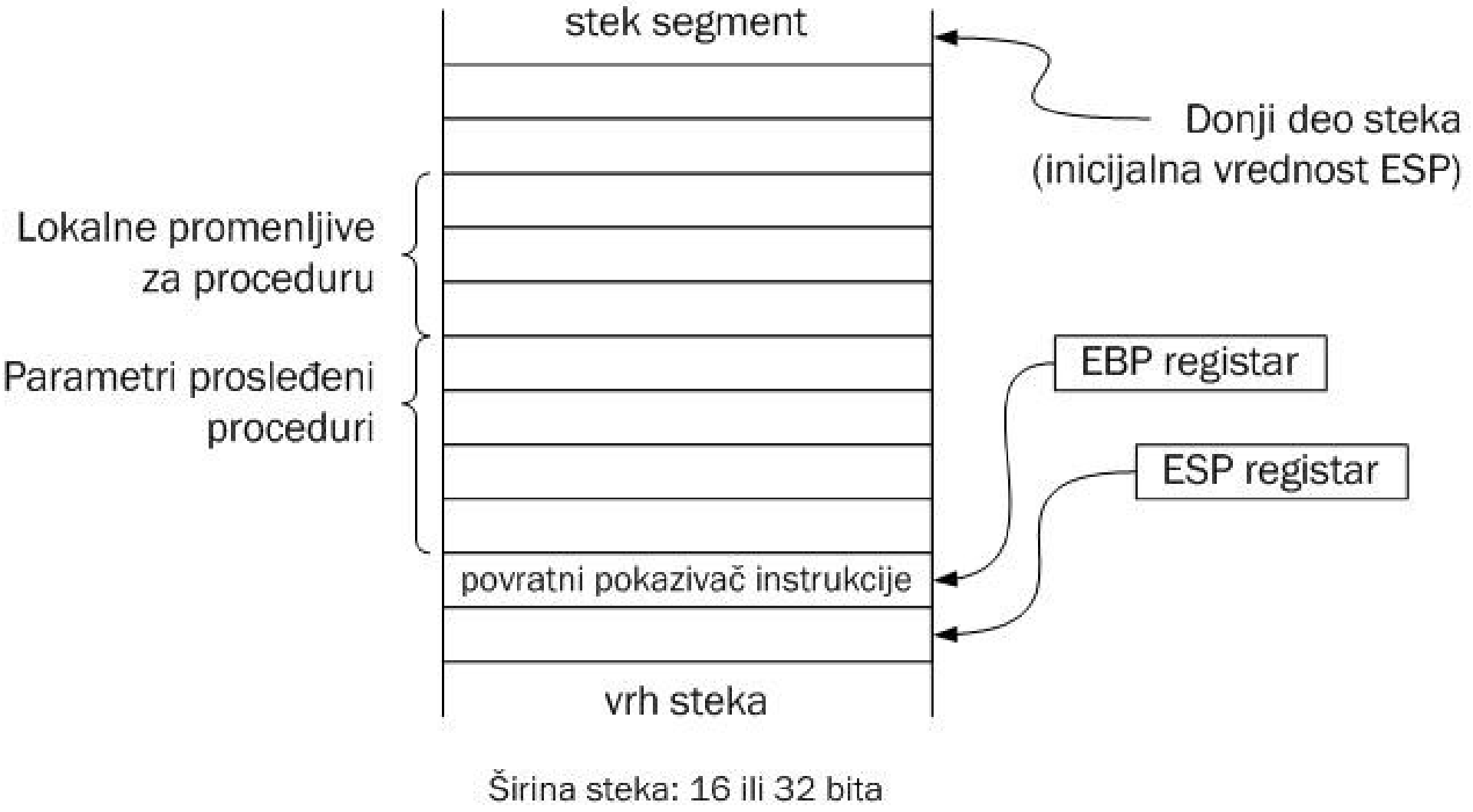
- U početnom stanju, ***A*** sadrži sve nule, a ***B*** broj 3

A	A	A	A	A	A	A	A	B	B
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**3**

- Ako programer pokuša da u bafer ***A*** upiše niz karaktera „**Sigurnost**“ praćen oznakom za kraj niza „**\0**“, doći će do prekoračenja bafera i prepisivanja vrednosti podatka u baferu ***B***
- Do prekoračenja, očigledno dolazi zbog greške programera (osim ako programer nije imao nameru da u podatak B upiše karakter „t“ praćen oznakom za kraj niza).

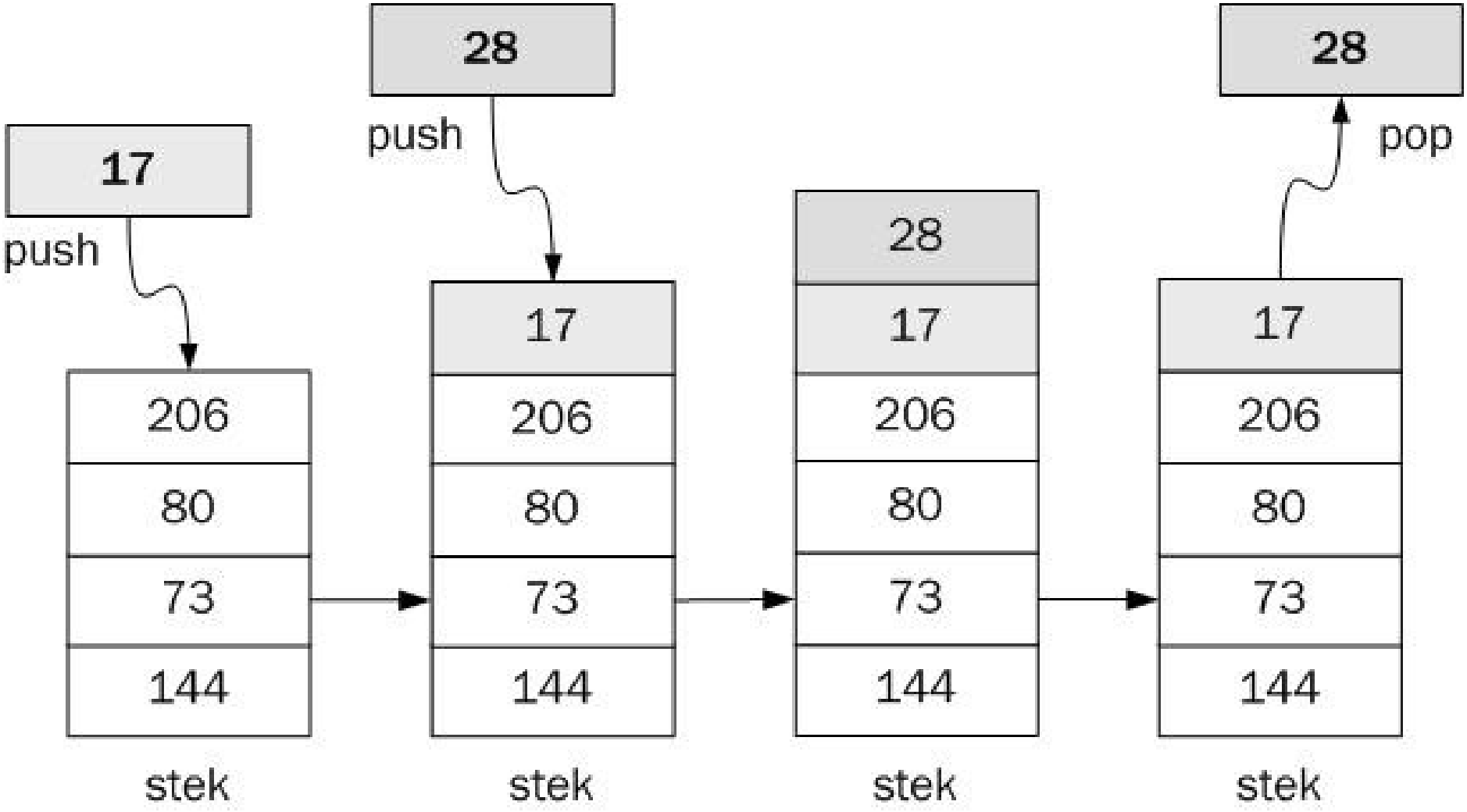
A	A	A	A	A	A	A	A	B	B
S	**i**	**g**	**u**	**r**	**n**	**o**	**s**	**t**	**0**

11.3 - Prepisivanje skladišta-stack



Širina steka: 16 ili 32 bita

11.3 - Prepisivanje skladišta-stack



11.3 – Oblikovanje znakovnih nizova

- Pripadaju novoj klasi sigurnosnih problema koji su skoro otkriveni.
- Nepravilno oblikovanje znakovnih nizova pripada greškama konstruktora koje se koriste za formatiranje ul./iz. podataka (C i C++).
- Takvi nizovi sadrže posebne tzv. „čuvare mesta“ (***placeholders***) (kao što su **%s** za nizove znakova, **%d** za celobrojne vrednosti i td.) koji, ako se zloupotrebe, pri unosu podataka mogu napadaču otkriti podatke o programskom steku (***call stack***) i korišćenim promenljivama.
- Posebno je opasna upotreba identifikatora **%n** jer ga napadač može iskoristiti za prepisivanje podataka u memoriji, isto kao i kod prepisivanja skladišta, pa može da pokrene proizvoljni programski kod
- Osnovni uzrok propusta vezanih uz oblikovanje znakovnih nizova je upotreba funkcija sa promenljivim argumentima u programskim jezicima C i C++.
- Ovakvi se problemi mogu ukloniti validacijom ulaznih podataka i proverom izuzetaka programskog koda.
- Alati za automatsko testiranje koda mogu se koristiti za identifikaciju grešaka kao što je printf (*string*).

11.3 – Autentifikacija

- Autentifikacija je najkritičnija komponenta bilo kog sigurnosn. sistema
- Ukoliko je autentifikacija korisnika nepravilno izvedena, sve ostale sigurnosne funkcionalnosti, kao što su šifrovanje, provere ispravnosti i pouzdanosti informacija (*audit*) te autorizacija, postaju beskorisne.
- Najčešća greška kod autentifikacije je upotreba slabih autentifikacijskih uverenja (*authentication credentials*).
- Napadač ih može iskoristiti za ***brute force*** napad za otkrivanje lozinki
- Potrebni su strogi kriterijumi za proveru lozinki koji minimiziraju mogućnost njihovog pogađanja.
- Preporuka je da koristimo lozinke koje sadrže alfanumeričke i posebne znakove, a koje su istovremeno minimalne dužine od 8 znakova.
- Mnogi programi, a posebno Web aplikacije, koriste autentifikatore za identifikaciju korisnika nakon prijave umesto korisni.imena i lozinke.
- Kod Web aplikacija, oni su obično snimljeni u tzv. *cookie* datotekama.
- Tokom kreiranja aplikacije važno je osigurati otpornost autentifikatora na ***brute force*** i ***prediction*** napade.

11.3 – Autorizacija

- Autorizacija je proces dozvole ili zabrane pristupa pojedinom resursu na osnovu učinjene identifikacije i autentifikacije.
- Ranjivosti vezane uz autorizaciju su uobičajeni problemi mnogih aplikacija koje prouzrokuju sledeće greške:

1. **Nepravilna izvedba autorizacije** - nakon prve uspešne autentifikacije, sistem će dozvoliti pristup resursu sve dok postoji ta autentifikacija. Ovakav slučaj se obično javlja kod korišćenja identifikatora. Pretpostavlja se da je identifikator nemoguće pogoditi ili izmeniti.
2. **Nedovoljna provera unesenih podataka** - HTTP *cookie* datoteka je primer datoteke za čuvanje podataka koje korisnik upisuje u Web aplikaciju. Ukoliko se autorizacija aplikacije zasniva na podacima iz *cookie* datoteke, postoji mogućnost da se zasniva na falsifikovanim podacima. Falsifikati mogu biti ili korisničko ime ili zahtevani resurs.

- **Greške u pretvaranju podataka** - aplikacije često donose autorizacijske odluke na osnovu autentifikacijskog uverenja i zahtevanog resursa. Na primer, iako je aplikacija zabranila pristup fajlu \secure\secret.txt, ona može dozvoliti pristup public\...\secure\secret.txt

11.3 – Kriptografija

- Ukoliko se neki podaci šifriraju unutar aplikacije, ti se podaci smatraju vrlo osetljivima.
- Retki su programeri koji detaljno poznaju svu matematičku složenost kriptografskih algoritama koje koriste.
- Greška u proizvoljno dizajniranim kriptografskim algoritmima ili njihovoj implementaciji može potkopati sigurnost cele aplikacije.
- Slabi ili dobro poznati kriptografski algoritmi mogu da daju lažnu sliku o sigurnosti kriptovanih informacija
- Pronalaženjem kriptografskog algoritma potencijalni napadač lako dolazi do „sigurnih" podataka a da vlasnik nije svestan toga
- Treba primenjivati složene algoritme kriptovanja koji će otežati njihovo otkrivanje i dekripciju potencijalnim napadačima

11.3 Problemi u simultanom korišćenju resursa

- Korišćenje zajedničkog resursa od dva ili više korisnika ili procesa može dovesti do problema poznatog kao ***race condition***.

Primer: *Imamo dva procesa, P0 i P1, koji žele da privremeno sačuvaju neku vrednost na istoj memorijskoj lokaciji.*

1. P0 proverava da li je memorijska lokacija A slobodna. Slobodna je. P0 je obavešten da je lokacija A slobodna;
2. P1 proverava da li je memorijska lokacija A slobodna. Slobodna je. P1 je obavešten da je lokacija A slobodna;
3. Proces P0 upisuje podatak na lokaciju A;
4. Proces P1 upisuje svoj podatak na lokaciju A;
5. Proces P0 čita podatak sa lokacije A – POGREŠAN.

- S gledišta sigurnosti programa postoji nekoliko načina na koje se može zloupotrebiti simultano izvođenje procesa ili ***race condition*** stanje
- Između provere datoteke i pisanja u datoteku postoji vremenski period kojeg napadač može zloupotrebiti za napad.
- Problem ima naziv "vreme provere-vreme korišćenja" (*time of check-time of use*) problem.

11.3 Problemi u simultanom korišćenju resursa

- Ostali slučajevi simultanog korišćenja resursa podložni zloupotrebi uključuju korišćenje deljenih podataka ili neku drugu metodu komunikacije među procesima.
- Napadač može menjati podatke nakon što su upisani, a pre nego što budu pročitani, i na taj način može uticati na tok izvršenja programa pa u krajnjem slučaju i da onemogućiti njegovo izvršavanje.
- Nesigurno rukovanje nitima (***threads***) u višenitnim programima može rezultirati oštećenjem podataka.
- Ukoliko napadač uspe manipulisati programom tako da izazove interakciju dveju takvih niti, može izvesti ***DoS*** napad.
- U nekim slučajevima napadač može iskoristiti ***race condition*** stanje za prepunjavanje bafera (*heap*).
- Bafer se u tom slučaju prepisuje podacima iz procesa koji koristi više podataka od procesa koji je zauzeo memoriju za skladište.
- Napadač može zloupotrebiti takvu ranjivost za pokretanje programskog koda po volji ubacivanjem tog koda u napadnuti računar tj. prepisivanjem originalnih podataka koji se nalaze u baferu.

11.4 Principi sprečavanja problema

1. **Prikrivanje informacija (Enkapsulacija)**

- ✓ Modul obično specificira i implementira apstrakciju.
- ✓ Specifikacije modula opisuju ponašanje i svojstva apstrakcije, a implementacija sadrži konkretnu realizaciju u obliku programsk.koda
- ✓ Efikasno programiranje uključuje upotrebu već postojećeg programskog koda, biblioteka ili komponenti.
- ✓ Svaki dobro dizajnirani modul treba da enkapsulira (grupiše i sakrije) podatke s oznakom private/public i pripadajući programski kod
- ✓ Šta više, modul treba da pruži dobro dizajnirane interfejse kojima se može pristupati njegovim podacima i menjati ih.
- ✓ Sve nedokumentovane, neodređene opcije koda, nuspojave ili definicije mogu poslužiti kao skriveni kanali za curenje podataka.
- ✓ Podaci mogu postati oštećeni i izmenjeni te kao takvi postaju sigurnosni rizik.
- ✓ Ideja enkapsulacije direktna je posledica "principa najmanje ovlašćenja" (*Principle of Least Privilege*) i ona čini osnovu integriteta i sigurnosti programskog koda.

11.4 Principi sprečavanja problema

2. **Defenzivno programiranje**

- ✓ Zasniva se na ideji da dati program nakon pokretanja ne sme zavisiti ni od čega što je korisnik sam stvorio.
- ✓ Svaki put kada korisnik (napadač) pokrene program, programer mora pretpostaviti da ga korisnik može srušiti bilo namerno bilo nenamerno.
- ✓ Potrebno je u programski kod umetnuti što više naredbi "**assert**" za proveru koda i presretanje posebno oblikovanih podataka prilikom njihovog prolaska iz jedne funkcije/modula u drugu funkciju/modul.
- ✓ Nikako se ne bi se smeo primenjivati pristup ***garbage-in garbage-out***, pretakanje neodgovarajućih podataka iz jednog u drugi deo programa
- ✓ S druge strane, programer ne bi smeo zloupotrebiti ugrađena svojstva određenih programskih jezika, kao što su, na pr. pokazivači (*pointers*).
- ✓ Kod njih je, specifično, vrlo opasno ponovno zauzimanje memorije koje može rezultirati pojavom „visećih„ pokazivača(*dangling pointers*)
- ✓ Ukoliko funkcija vraća pokazivač, javlja se opasnost od neovlašćenog pristupa podacima programa (ranjivost curenja podataka).
- ✓ Naziva se još i robusnim programiranjem (*robust programming*).

11.4 Principi sprečavanja problema

3. **Pretpostavljanje nemogućeg**

- ✓ Programeri obično prvo napišu program pa ga testiraju kako bi potvrdili njegov ispravan rad.
- ✓ Tada, ukoliko ih pronađu, ispravljaju pronađene greške u kodu i ponovno testiraju aplikaciju.
- ✓ Ovaj proces ne vodi nužno potpuno ispravnom kodu jer se ne može dokazati odsustvo grešaka.
- ✓ U većini situacija ovaj metod daje prilično dobre rezultate.
- ✓ Potpuna provera ispravnosti nekog programa za sada prekoračuje ljudske mogućnosti, tako da su trenutno najbolje rešenja što temeljitije testiranje i ugradnja provere grešaka na svim mestima, pa čak i tamo gde je mogućnost njihove pojave samo teoretska.
- ✓ Opisanim pristupom može se uočiti mnogo grešaka, od kojih neke mogu ozbiljno ugroziti sistem.
- ✓ Ni u kojem slučaju ne sme se, dakle, dozvoliti nekontrolisano propagiranje odbačenih podataka kroz sistem.

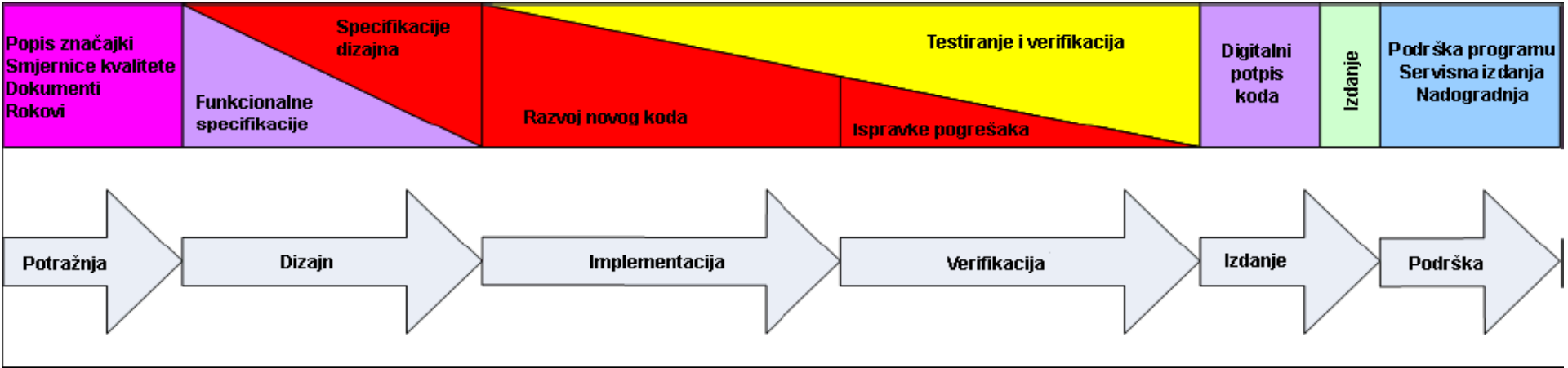
11.5 – Razvojni put aplikacije

- ***Trustworthy Computing Security Development Lifecycle*** **-** životni ciklus razvoja softvera sa osvrtom na sigurnost je proces razvoja softvera razvijen u Microsoftu.
- „**The Security Development Lifecycle**“ – predstavlja koncept razvoja softvera koji je postao poznat kao **SDL standard**.
- Ovaj koncept vodi **kroz sve faze** razvoja softvera uključujući i obrazovanje i usmeravanje projektanata i programera, projektovanje, razvoj, testiranje, isporuku (objavljivanje) i održavanje, tj. proces posle objavljivanja (**post-release**).
- Koncept podrazumeva **analizu** mogućih napada i izloženosti i pre početka projektovanja aplikacije kao i procenu rizika.
- **SDL** navodi **listu zabranjenih API funkcija** koje mogu prouzrokovati sigurnosne probleme i koje treba vremenom **ukloniti iz nasleđenog koda**

11.5 – Razvojni put aplikacije

Standardni Microsoft proces razvoja softvera obuhvata **sledeće faze:**

1. **analiza zahteva** (spisak zahteva, smernice za obezbeđivanje kvaliteta, dokumentovanje arhitekture projekta i formiranje preliminarnog rasporeda)
2. **projektovanje** (specifikacije strukture i funkcionalnosti)
3. **razvoj** (razvoj novog koda uz povremene provere i testiranje koda)
4. **testiranje** (provera koda i ispravka grešaka)
5. **objavljivanje** (potpisivanje koda i isporuka gotovog proizvoda)
6. **održavanje i servisiranje** (tehnička podrška krajnjim korisnicima, sigurnosne zakrpe i servisni paketi)



11.6 – Zaštita softvera

- Na osnovu stepena zakonske zaštite softvera, u praksi se teško otkrivaju nelegalne radnje i još veći problem je dokazivanje na sudu.
- Zbog ovoga, proizvodačima softvera je lakše i jeftinije da pirateriju sprečavaju ugradnjom tehnoloških zaštitnih mera u svoj softver.
- Metode zaštite softvera čine softverske i hardverske metode i one se uglavnom sastoje od provere identiteta korisnika.
- Korisnik ima određen hardverski uređaj koji se spaja sa računarom, ili neki ključ ili datoteku za registrovanje koja se od njega traži prilikom instaliranja programa, podaci se čuvaju šifrovani.
- Postoje sledeće metode zaštite:

1. **Zaštita CD-a od kopiranja** - svodi se na proveravanje ispravnosti CD/DVD-a u CD/DVD čitaču.
   - ✓ Tako se sprečava kopiranje i pokretanje programa sa čvrstog diska računara, ali je nemoguće razlikovati originalni od ilegalno kopiranog CD-a.
   - ✓ Bolje rešenje je šifrovanje podataka, mada to onda remeti jednostavnost upotrebe.

11.6 – Zaštita softvera

2. **Ključ za registraciju** – ugrađena je programska funkcija koja od korisnika traži da se unese određeni ključ za registraciju.

- ✓ Ta vrednost je uvek ista i ne zavisi od parametara kao što su korisnički podaci ili podaci o korisnikovom računaru.
- ✓ Ovde je dovoljno da se jednom pronađe registracioni ključ, objavi se na Internetu, i svako može besplatno registrovati taj softver
- ✓ Način probijanja zaštite zasnovane na ključu za registraciju je prepravka dela koda za proveru ispravnosti ključa tako da se potpuno zaobilazi provera pa program normalno nastavlja rad.
- ✓ Da bi se sprečilo takav proboj zaštite, treba koristiti registracioni ključ za dešifrovanje delova programskog koda.
- ✓ Ako je deo koda šifrovan pomoću registracionog ključa, onda ga je moguće dešifrovati samo pomoću tog istog ključa.
- ✓ Bolji način zaštite je generisanje registracionog ključa pomoću parametara koji se sakupljaju iz podataka o korisniku.
- ✓ Ovakav program generiše ključ iz korisničkih podataka i upotrebljava podatke kao što je korisničko ime, adresu, ime kompanije.

11.6 – Zaštita softvera

3. Autorizacija - funkcioniše na principu izazov – odgovor.

- ✓ Jedinstverni broj koji predstavlja izazov generiše se prilikom instaliranja programa pomoću serijskog broja isporučene kopije softvera i podataka o hardveru računara i konfiguraciji OS.
- ✓ Po završenoj instalaciji, program treba autorizovati.
- ✓ Generisani broj šalje se proizvođaču softvera koji korisniku vraća odgovor koji odgovara generisanom serijskom broju hardvera.

4. Zastita programa sa registracionom datotekom – isto je i što su ključevi za registraciju

- ✓ Prednost im je količina informacija koju je moguće staviti u njih.
- ✓ Registraciona datoteka ima informacije o korisniku, šifru za proveru identiteta korisnika, ključeve za dešifrovanje šifrovanih delova koda.
- ✓ Regis.datoteka je šifrovana tako da ju je nemoguće pročitati/promeniti
- ✓ U reg.datoteku se može upisati i deo izvršnog koda aplikacije, podaci o hardveru računara,pa je potrebna jedinstvena datoteka za svaki računar
- ✓ Provera ispravnosti datoteke ugrađuje se na više mesta u programu i informacije u datoteci se koriste za šitrovanje delova izvršnog koda.

11.6 – Zaštita softvera

5.Dongle

- ✓ To je uređaj koji se povezuje na računar preko nekog porta kako bi se obezbedilo sigurno pokretanje određene aplikacije.
- ✓ Stari tipovi *dongle*-a vezivali su se sa računarom preko paralelnog /serijskog priključka i pri pokretanju softvera, proveravalo se postoji li sam uređaj, i u slučaju da ga nema, pokretanje se prekidalo.
- ✓ Slabost ove provere je bila u napadu na deo programa koji proverava da li postoji ***dongle*** uređaj.
- ✓ Moderni *dongle* uređaji vezani su na računar preko USB porta.
- ✓ Za proveru postojanja, na uređaju je ključ bez koga se ne može pokrenuti aplikacija.
- ✓ Kod za proveru ispravnosti *dongle* uređaja šifruje se u aplikaciji, a može se šifrovati i sva komunikacija između aplikacije i *dongle* uređaja
- ✓ Ako je provera ključa ugrađena na više mesta u programu i ako su ti delovi koda šifrovani, teško je pronaći i skloniti sve provere iz aplikacije.

11.6 – Zaštita softvera

6.FlexLM (*Flex Licence Manager*)

- ✓ Komercijalno rešenje nadzora nad sofrverskim licencama.
- ✓ Čini ga: zaštita izvršnog koda aplikacije, kontrola ispravnosti reg. datoteka na disku, softver za implementaciju na serverima koji nadziru korišćenje licenci u LAN-u, sistem za lako obnavljanje licenci i centralno administriranje celog sistema.
- ✓ Za zaštitu aplikacija, FlexLM koristi šifrovanje izvršnog koda aplikacije, šifrovanu registracionu datoteku koja zavisi od hardvera korisnikovog računara, kontrolu servera kojom se neovlašćenim korisnicima zabranjuje pokretanje mrežnih aplikacija, a u nekim verzijama primenjuje se i *dongle*.
- ✓ Sistem kontroliše korišćenje softvera i ispravnost licenci, a moguće je izabrati različite metode licenciranja softvera:

a. Node locked - pokreće se samo na računaru koji ima licencu

b. User based - može ga pokrenuti samo korisnik, vlasnik licence

c. Site licensing - mogu ga pokrenuti svi korisnici na određenoj lokaciji

11.6 – Zaštita softvera

d. ***Floating license*** - softver mogu pokrenuti svi korisnici u LAN-u, ali je ograničen broj korisnika koji mogu istovremeno raditi sa programom.

- ✓ Aplikacijski softver je zaštićen FlexLM sistemom.
- ✓ Softver ima ugrađene funkcije koje vrše proveru postojanja i ispravnosti registracione datoteke.
- ✓ Datoteka se mora nalaziti u određenom direktorijumu, a u datoteci je dat naziv softvera, podaci o vlasniku licence i vreme važenja licence
- ✓ U reg.datoteci zapisana je i putanja do servera za kontrolu licenci čiji je zadatak nadzor licenci u lokalnoj mreži.
- ✓ Aplikacija se povezuje s tim serverom i od njega saznaje lokaciju servera softverske kompanije koji nadgleda korišćenje licenci za sve mušterije tog proizvodača.
- ✓ FlexLM je softver koji koriste mnogi proizvodači softvera za zaštitu svojih aplikacija, ima mogućnosti da korisnik na svom računaru ima više aplikacija različitih proizvođača, a sve su zaštićene FlexLM
- ✓ Korisnik mora instalirati FlexLM softver samo jednom i taj sistem upravlja licencama za sve aplikacije za koje je odgovoran.

11.6 – Zaštita softvera

7. HASP - sistemi funkcionišu preko USB/serijskog priključka,

- ✓ Pri pokretanju **HASP** programa, ostvaruje se veza sa hardverskim priključkom gde se nalazi informacija potrebna za nastavak rada.
- ✓ HASP nudi i šifrovanje samog koda.
- ✓ Princip koji se koristi je šifrovanje podataka pomoću ključa koji se nalazi na pokretnom uređaju.
- ✓ Korišćenje se svodi na instalaciji upravljačkog programa, posle čega samo treba povezati uređaj na priključak računara.
- ✓ Za vreme obavljanja programa, zaštićeni softver šalje šifrovan niz znakova priključenom uređaju koji dešifruje poruku i vraća odgovor.

8. PC Guard – ima namenu za zaštitu softvera šifrovanjem izvršnog koda i detektovanjem pokušaja analize koda.

- ✓ Zaštita je primenjena direkrno na aplikacije (EXE, DLL), tako što se one umotavaju u sigurnosni omotač koji sadrži više nivoa šifrovanja i alate za sprečavanje reverznog inženjeringa.
- ✓ PC Guard ne koristi nikakav dodatni hardver, što olakšava njegovo korišćenje i namenjen je familiji OS Windows.

11.6 – Preporuke

- Osnovne smernice koje ne bi trebalo zanemariti pri projektovanju aplikacije i pisanju zaštićenog koda:

1. Svaki korisnik sistema treba da radi sa što manjim privilegijama - tako se smanjuje mogućnost štete, greške ili zloupotrbe prilikom napada.
2. Pošto je često neophodno ispitivati red po red koda, sistem zaštite treba da bude što jednostavniji.
3. Kod bi trebalo da bude izložen kritici kako bi bio temeljno ispitan, pa je poželjno da bude otvoren kod.
4. Davati što manje informacija o sebi.

Hvala na pažnji !!!

Pitanja

? ? ?