HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Izrada sigurnog koda

CCERT-PUBDOC-2007-12-212

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument, koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Sigurnosne ranjivosti programskih paketa su pogreške u programima koje napadači i zlonamjerni korisnici mogu iskoristiti za narušavanje integriteta računala. Uzroci ranjivosti su razni, a najčešći su oni vezani su uz sigurnost memorije, neodgovarajuću obradu ulaznih podataka, simultano korištenje zajedničkih resursa, dodjelu ovlasti ili pogreške grafičkog sučelja. Ranjivosti ili sigurnosni propusti se često javljaju zbog nepažnje programera. Napadač može zlouporabiti ranjivost aplikacije na mnogo načina, a najpoznatiji su pokretanje proizvoljnog programskog koda, neovlašten pristup podacima, napad uskraćivanja usluga (eng. Denial of Service - DoS), XSS (eng. Cross-Site Scripting) napad i još mnogi drugi.

Postoje mnogi alati za otkrivanje i, katkad, uklanjanje sigurnosnih ranjivosti. Iako oni mogu pomoći identificirati problem, još uvijek je potreban ljudski faktor. Sigurnosni propusti se javljaju u svim programima i operacijskim sustavima. Moguće je smanjiti vjerojatnost zlouporabe ranjivosti brižnim održavanjem sustava (npr. primjena nadogradnji i zakrpa), pažljivom instalacijom i pokretanjem programa (npr. upotreba vatrozida i kontrole pristupa) te stalnom ponovnom evaluacijom programskih paketa od strane programera tijekom ciklusa razvoja programa (eng. *Software Development Lifecycle* – SDL).

Jedna od tehnika razvoja aplikacija koja nastoji minimizirati vjerojatnost pojave ranjivosti je pisanje sigurnog koda. U slijedećim poglavljima govori se općenito o sigurnosti programskog koda, najčešćim pogreškama programera te ranjivostima koje su uzrokovane tim pogreškama. Također su objašnjeni principi sprečavanja problema, a na koncu je ukratko opisan primjer politike razvoja sigurnog koda u razvojnom procesu tvrtke Microsoft.

2. Općenito o sigurnosti programskog koda

Standardne tehnike razvoja programskih paketa potpuno su neprikladne za kreiranje sigurnih aplikacija zbog svoje orijentiranosti na ispravne funkcionalnosti programa i istovremenog potpunog ignoriranja ostalih. Jedan od primjera takvog pristupa je implementacija funkcionalnosti koja uključuje izvođenje neke akcije B ukoliko se pritisne tipka A. Istovremeno se ne vodi računa o tome što će se dogoditi ako korisnik pritisne tipku C, a pogotovo ne o rezultatu pritiska kombinacije tipaka C i D.

Sigurnost programskog koda odnosi se na otpornost aplikacije na razne zlouporabe napadača (korisnika) i u tom smislu na nedostatak funkcionalnosti.

U realnim situacijama postoji mnogo načina za uzrokovanje stanja uskraćivanja usluga neke aplikacije. Često korisnik (napadač) može, namjerno ili slučajno, unijeti neodgovarajuće podatke u program te na taj način učiniti program nedostupnim. Neke od namjernih akcija napadača uključuju podmetanje zlonamjerno oblikovanih podataka ranjivom programu. Aplikacije reagiraju različito na takve napade. Neke od njih se jednostavno sruše bez ikakvih poruka o pogrešci, neke se samo nepravilno ponašaju, a neke prouzrokuju stanje uskraćivanja resursa cijelog operacijskog sustava. Program koji uslijed napada prouzrokuje stanje uskraćivanja usluga cijelog sustava smatra se potpuno neprihvatljivim u području računalne sigurnosti. Zbog toga je razvoj sigurnih programskih paketa prilično različit od razvoja običnih aplikacija.

Problemi koji se najčešće javljaju u programima (npr. prepisivanje međuspremnika) smatraju se i najozbiljnijim sigurnosnim propustima današnjice. Točnije, najveći problem u računalnoj sigurnosti vezan je uz svojstvo najslabije karike. U počecima razvoja računarstva, programer je dobio zadatak i sam je razvio program koji rješava zadani problem. Danas se posao programiranja aplikacija koje izvršavaju kompleksne zadatke raspodjeljuje među timovima programera koji pišu milijune linija koda. Jedan programer u timu u jednom danu u prosjeku napiše svega 5 do 10 linija programskog koda. Za velike je programe zato potrebno izraditi dokumente s preciznim dizajnom koji prikazuje što koji dio koda radi i kako se ponaša pri interakciji s ostalim dijelovima programa. Dužnost svakog programera u timu je razviti programski kod bez pogrešaka (eng. *bug-free*). Zbog toga programeri moraju međusobno sudjelovati u revizijama te ponovnim evaluacijama dizajna i programskog koda aplikacije. Kada programer završi pisanje određenog dijela koda, ostali članovi tima moraju potpuno pregledati njegov dizajn i/ili programski kod. Osnovna načela razvoja programa diktiraju pisanje malih, samostojećih jedinica nazvanih modulima. Svaki se modul treba izolirati od štetnog utjecaja ostalih modula, a to se može postići skrivanjem dijelova programa (enkapsulacijom). Programiranje sigurnog koda uvjetuje da svaka akcija koju program (ili potprogram izvodi) mora biti sadržana u njegovim specifikacijama. Pri pisanju sigurnih programa potrebno je pridržavati se tri osnovna principa:

- skrivanja informacija (dijelova programskog koda – enkapsulacija),
- defanzivnog (robusnog) programiranja, te
- pretpostavljanja nemogućeg.

3. Najčešće pogreške programera

Razvoj sigurnih programa je proces koji zahtijeva opreznost i obazrivost pri svakom koraku te na svim razinama organizacije. Razvoj sigurnog programskog koda zahtijeva stroge sigurnosne specifikacije, razvojne programere s mnogo iskustva i znanja na području sigurnosti te skupinu za procjenu kvalitete (eng. *quality assessment* (QA) *team*) za otkrivanje sigurnosnih problema. Međutim kako bi se bilo moguće uhvatiti u koštac s ovim problemima potrebno je, metaforički rečeno, prvo naučiti hodati pa onda trčati. Nijedna tvrtka neće pokušati rješavati sve sigurnosne probleme odjednom. Iduća poglavlja opisuju pet problema koji čine većinu sigurnosnih ranjivosti programa. Otkrivanjem i uklanjanjem ovih pet problema moguće je povisiti razinu sigurnosti aplikacija, a time i cijelog računala.

3.1. Prepisivanje spremnika

Jedan od najčešće zlouporabljenih sigurnosnih propusta je prepisivanje spremnika. Ovdje neće biti detaljno objašnjeno kako napadač može iskoristiti propust prepisivanja spremnika za pokretanje

proizvoljnog programskog koda. Takve informacije moguće je saznati iz drugih izvora. Dovoljno je spomenuti da ukoliko postoji bilo kakva pogreška prepisivanja spremnika unutar programa, napadač ju može iskoristiti za potpuno preuzimanje nadzora nad ranjivim sustavom. Temeljni uzrok problema je uporaba statičnih varijabli fiksne veličine za pohranu ulaznih podataka. Zlonamjerni napadač može prepisati međuspremnike u koje se spremaju ulazni podaci te zloupotrijebiti spomenutu ranjivost.
Za izbjegavanje problema ključno je provjeravanje svih ulaznih podataka prije njihovog kopiranja u spremnik. Ako veličina ulaznih podataka prekoračuje veličinu spremnika, potrebno je zabilježiti iznimku i promijeniti tok programa. Implementacija ove provjere može biti dug i zamoran proces za svaki veći program. Na sreću, postoje mnogi alati koji automatski prepoznaju spomenutu situaciju. Međutim, takvi alati nisu magično rješenje problema. Njima je moguće otkriti propust, ali programer još uvijek mora pregledati rezultate i riješiti problem.
Pisanje koda za zlouporabu propusta prepisivanja spremnika smatra se crnom magijom. Potrebno je mnogo vještine i znanja o operacijskim sustavima, prevoditeljima programskog koda i strojnom kodu da bi se mogao napisati program koji iskorištava prepisivanje spremnika. Zbog toga mnogi ljudi sigurnosni rizik pogrešno ocjenjuju minimalnim. Pretraživanjem web stranica kao što su *SecurityFocus* i *Packetstorm*, lako je moguće pronaći gotove programe za zlouporabu propusta prepisivanja spremnika. Nevješti napadači mogu jednostavno preuzeti program, pokrenuti ga i kliknuti mišem na tipku te tako izvesti napad *Point and Click*. Ipak, uvijek ostaje pitanje kako autor programa za iskorištavanje ranjivosti pronađe i zloupotrijebi sigurnosni propust prepisivanja spremnika?

3.2. Ranjivosti vezane uz oblikovanje znakovnih nizova

Ranjivosti vezane uz oblikovanje znakovnih nizova pripadaju novoj klasi sigurnosnih problema koji su otkriveni unazad nekoliko godina. Nepravilno oblikovanje znakovnih nizova pripada pogreškama konstrukata koje se koriste za formatiranje ulazno-izlaznih podataka u programskim jezicima C i C++. Takvi nizovi sadrže posebne tzv. „čuvare mjesta“ (eng. *placeholders*) (kao što su %s za nizove znakova, %d za cjelobrojne vrijednosti i td.) koji, ako se zlouporabe, pri unosu podataka mogu napadaču otkriti podatke o programskom stogu (eng. *call stack*) i korištenim varijablama. Posebno je opasna upotreba identifikatora %n jer ga napadač može iskoristiti za prepisivanje podataka u memoriji. Prepisivanje memorije omogućuje isto što i ranjivost prepisivanja spremnika, a to je pokretanje proizvoljnog programskog koda.
Temeljni uzrok propusta vezanih uz oblikovanje znakovnih nizova je upotreba funkcija s varijabilnim argumentima u programskim jezicima C i C++. Ovakvi se problemi mogu ukloniti validacijom ulaznih podataka i provjerom iznimaka programskog koda. Uz to, alati za automatsko testiranje koda mogu se koristiti za identifikaciju pogrešaka kao što je *printf(string)*.

3.3. Autentikacija

Autentikacija je najkritičnija komponenta bilo kojeg sigurnosnog sustava. Ukoliko je autentikacija korisnika nepravilno izvedena, sve ostale sigurnosne funkcionalnosti, kao što su kriptiranje, provjere ispravnosti i pouzdanosti informacija (eng. *audit*) te autorizacija, postaju beskorisne. Najčešća pogreška kod autentikacije je uporaba slabih autentikacijskih uvjerenja (eng. *authentication credentials*). Napadač ih može iskoristiti za tzv. *brute force* napad (npr. za otkrivanje zaporki). Osim toga, potrebni su strogi predlošci za provjeru zaporki koji minimiziraju mogućnost njihovog pogađanja. Danas je uobičajena preporuka koristiti zaporke koje sadrže alfanumeričke i posebne znakove, a koje su istovremeno dulje od 7 znakova.
Mnogi programi, a posebno Web aplikacije, koriste autentikatore za identifikaciju korisnika nakon prijave. Autentikatori su kao "ulaznice" koje se izdaju nakon što korisnik preda autentikacijsko uvjerenje. "Ulaznice" se mogu iskoristiti za provjeru autentičnosti umjesto zaporke i korisničkog imena. Kod Web aplikacija , autentikatori su obično pohranjeni u tzv. *cookie* datotekama. Tijekom kreiranja aplikacije važno je osigurati otpornost autentikatora na *brute force* i *prediction* napade.

3.4. Autorizacija

Autorizacija je proces dozvole ili zabrane pristupa pojedinom resursu temeljem učinjene identifikacije i autentikacije. Ranjivosti vezane uz autorizaciju su uobičajen problem mnogih aplikacija. Česte pogreške su:

- Nepravilna izvedba autorizacije - nakon prve uspješne autentikacije, sučelje će dozvoliti pristup resursu sve dok postoji ta autentikacija. Ovakav slučaj se obično javlja kod korištenja identifikatora. Pretpostavlja se da je identifikator nemoguće pogoditi ili izmijeniti.
- Nedovoljna provjera unesenih podataka - HTTP *cookie* datoteka primjer je datoteke za pohranu podataka koje korisnik upisuje u Web aplikaciju. Ukoliko se autorizacija aplikacije temelji na podacima u *cookie* datoteci, postoji mogućnost da se temelji na falsificiranim podacima. Falsificirani mogu biti ili korisničko ime ili zahtijevani resurs.
- Pogreške u pretvaranju podataka - aplikacije često donose autorizacijske odluke na temelju autentikacijskog uvjerenja i zahtijevanog resursa. Mnoge sigurnosne ranjivosti vezane su uz pogreške pretvaranja imena resursa u standardan oblik. Na primjer, iako je aplikacija zabranila pristup datoteci \secure\secret.txt, ona može dozvoliti pristup resursu \public\...\secure\secret.txt na temelju imena direktorija. *Unicode* i *hex* kodiranje znakova pripadaju u istu kategoriju.

3.5. Kriptografija

Ukoliko se neki podaci kriptiraju unutar aplikacije, ti se podaci smatraju vrlo osjetljivima. Rijetki su programeri upoznati sa svom matematikom kriptografskih algoritama. Pogreška u proizvoljno dizajniranim kriptografskim algoritmima ili njihovoj implementaciji može potkopati sigurnost cijele aplikacije.

3.6. Problemi u simultanom korištenju resursa

Ovaj problem demonstrira primjer programa koji automatski broji koliko je ljudi ušlo na nogometni stadion. Neka su ulazna vrata s rampom spojena na računalo i neka mu šalju signal svaki puta kada netko prođe kroz rampu. Za svaku rampu pokrenut je proces koji prati signal. Svaki put kada proces primi signal, čita globalnu varijablu "Ulaz", uvećava ju za 1 i ponovno zapiše. Na taj način više procesa broji ljude koji su ušli na stadion. Pretpostavi li se istovremeni prolazak dviju osoba kroz različite rampe, slijed događaja može izgledati ovako:

1. Proces A prima signal sa rampe A.
2. Proces B prima signal sa rampe B.
3. Proces A čita varijablu Ulaz=1000.
4. Proces B čita varijablu Ulaz=1000.
5. Proces A uvećava varijablu Ulaz za 1 pa slijedi Ulaz=1001.
6. Proces B uvećava varijablu Ulaz za 1 pa slijedi Ulaz=1001.
7. Proces A piše Ulaz=1001.
8. Proces B piše Ulaz=1001.

Proces B je pročitao varijablu "Ulaz" prije nego što ju je proces A uspio uvećati i ponovno zapisati. Rezultat toga je čitanje iste vrijednosti varijable "Ulaz". Nakon što proces A uveća varijablu "Ulaz" i ponovno ju zapiše, proces B prepiše njenu vrijednost istom. Zbog tzv. *race condition* stanja koje nastaje, jedan od dva čovjeka nije pribrojen. Kako postoji mogućnost stvaranja dugih redova na svakoj ulaznoj rampi, spomenuto stanje javit će se mnogo puta prije nego što svi ljudi uđu na stadion. Nepošteni prodavač ulaznica može si prisvojiti nekoliko njih bez bojazni da će biti uhvaćen. S gledišta sigurnosti programa postoji nekoliko načina na koje se može zlouporabiti simultano izvođenje procesa ili *race condition* stanje. Ukoliko program zapisuje privremene datoteke ili privremeno otpušta ovlasti za datoteke i direktorije kako bi izvršio administratorsku operaciju, napadač može stvoriti *race condition* stanje pažljivim odabirom vremena napada. Na primjer, napadač može iskoristiti slučaj kada program provjerava status datoteke prije nego što počne pisati u nju. Između provjere datoteke i

pisanja u datoteku postoji vremenski period kojeg napadač može zlouporabiti za napad. Problem ima naziv "vrijeme provjere-vrijeme korištenja" (eng. *time of check-time of use*) problem.
Ostali slučajevi simultanog korištenja resursa podložni zlouporabi uključuju korištenje dijeljenih podataka ili neku drugu metodu komunikacije među procesima. Ako napadač može mijenjati podatke nakon što su upisani, a prije nego što ih se pročita, tada može onemogućiti izvođenje operacija programa, mijenjati podatke i sl. Nesigurno rukovanje dretvama (eng. *threads*) u višedretvenim programima može rezultirati oštećenjem podataka. Ukoliko napadač uspije manipulirati programom tako da izazove interakciju dviju takvih dretvi, može izvesti napad uskraćivanja usluga (eng. *Denial of Service* – DoS). U nekim slučajevima napadač može iskoristiti *race condition* stanje za prepisivanje međuspremnika na gomili (eng. *heap*). Međuspremnik se u tom slučaju prepisuje podacima iz procesa koji koristi više podataka od procesa koji je zauzeo memoriju za spremnik. Kao što je rečeno u poglavlju o propustu prepisivanja spremnika, napadač može zloupotrijebiti takvu ranjivost za pokretanje programskog koda po volji. Programski kod niske razine koji sadrži tzv. „upravljače“ (eng. *handlers*) signala, posebno je ranjiv na takve napade.

4. Principi sprečavanja problema

4.1. Prikrivanje informacija (enkapsulacija)

Modul obično specificira i implementira apstrakciju. Specifikacije modula opisuju ponašanje i svojstva apstrakcije, a implementacija sadrži konkretnu realizaciju u obliku programskog koda. Efikasno programiranje uključuje upotrebu već postojećeg programskog koda, biblioteka ili komponenti. Svaki dobro dizajniran modul treba enkapsulirati (grupirati i sakriti) podatke s oznakom *private/public* i pripadni programski kod. Štoviše, modul treba pružati dobro dizajnirana sučelja kojima se može pristupiti njegovim podacima i mijenjati ih. Sve nedokumentirane, neodređene opcije koda, nuspojave ili definicije mogu poslužiti kao skriveni kanal za curenje podataka. Podaci mogu postati oštećeni i izmijenjeni te kao takvi postaju sigurnosni rizik. Ideja enkapsulacije izravna je posljedica "principa najmanje ovlasti" (eng. *Principle of Least Privilege*) i ona čini temelj integriteta i sigurnosti programskog koda. Tijekom cijelog procesa stvaranja specifikacije, dizajna, implementacije, testiranja i održavanja aplikacije, potrebno je imati na umu da se jednostavnost isplati.

4.2. Defenzivno programiranje

Defenzivno se programiranje temelji na ideji da dani program nakon pokretanja ne smije ovisiti ni o čemu što je korisnik sam stvorio. Svaki put kada korisnik (napadač) pokrene program, programer mora pretpostaviti da ga korisnik može srušiti bilo namjerno bilo nenamjerno unosnom neodgovarajućih podataka. Dakle, potrebno je u programski kod umetnuti što više naredbi "assert" za provjeru koda i presretanje posebno oblikovanih podataka prilikom njihovog prolaska iz jedne funkcije (modula) u drugu funkciju (modul). Ni u kojem slučaju ne bi se smio primjenjivati pristup *garbage-in garbage-out*, odnosno pretakanje neodgovarajućih podataka iz jednog dijela programa u drugi. To je, u većini slučajeva, vrlo pogubno. S druge strane, programer ne bi smio zloupotrijebiti ugrađena svojstva određenih programskih jezika, kao što su, primjerice, pokazivači (eng. *pointers*). Kod njih je, specifično, vrlo opasno ponovno zauzimanje memorije koje može rezultirati pojavom tzv. "visećih" pokazivača (eng. *dangling pointers*). Ukoliko funkcija vraća pokazivač, javlja se opasnost od neovlaštenog pristupa podacima programa (ranjivost curenja podataka). Isto vrijedi i za nizove (polja). Još jedna potencijalna ranjivost vezana je uz rukovanje pogreškama koje vraćaju funkcije. Ako se one ne provjeravaju, vrijednosti koje vraća funkcija ne bi se smjele koristiti dalje u programu jer mogu imati razoran učinak. Općenito je dobra preporuka izbjegavanja upotrebe misterioznog koda ili nekih posebnih svojstava i opcija nepoznatih većini programera. Tehnika defenzivnog programiranja često se naziva i robusnim programiranjem (eng. *robust programming*).

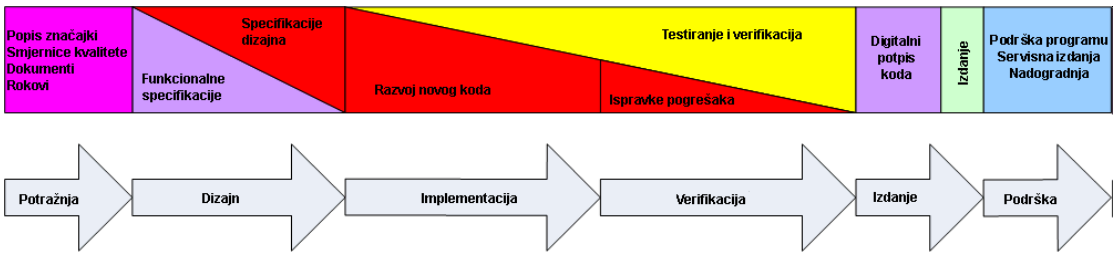
4.3. Pretpostavljanje nemogućeg

Programeri obično prvo napišu program pa ga testiraju kako bi potvrdili njegov ispravan rad. Tada, ukoliko ih pronađu, ispravljaju pogreške u kodu i ponovno testiraju. Ovaj proces ne vodi nužno potpuno ispravnom kodu jer se ne može dokazati odsutnost pogrešaka. Ipak, on daje prilično dobre

rezultate u većini situacija. Potpuna provjera ispravnosti nekog programa zasada prekoračuje ljudske mogućnosti, tako da su trenutno najbolje rješenja što temeljitije testiranje i ugradnja provjere pogrešaka na svim mjestima, pa čak i tamo gdje je mogućnost njihove pojave samo teoretska.
Opisanim pristupom može se uočiti mnogo pogrešaka, od kojih neke mogu ozbiljno ugroziti sustav. Ni u kojem slučaju ne smije se, dakle, dozvoliti nekontrolirano propagiranje odbačenih podataka kroz sustav.

5. Uključivanje mjera sigurnosti u razvojni proces tvrtke Microsoft

Slika 1. prikazuje općeniti razvojni proces tvrtke Microsoft. Ovakav je proces uobičajen za industriju.

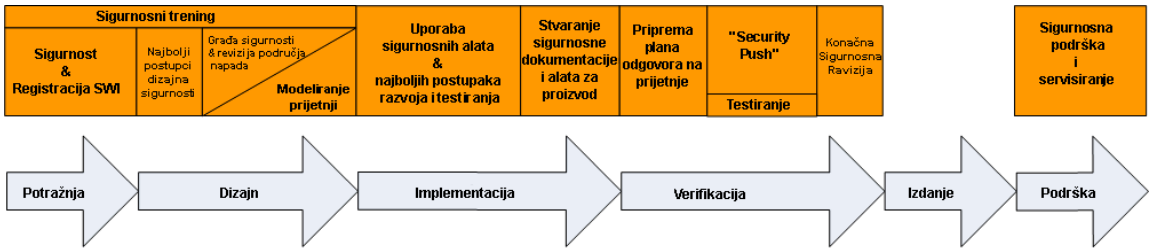


Slika 1. Uobičajen razvojni proces tvrtke Microsoft

Na slici se može uočiti pet ključnih točaka koje slijedom strelica sugeriraju razvojni proces po modelu slapa, međutim proces u stvarnosti slijedi spiralni model. Potrebe i dizajn često se revidiraju tokom implementacije u ovisnosti o promjenama na tržištu i događajima u razvoju programskih paketa. Nadalje, razvojni proces naglašava da u svakom trenutku postoji programski kod koji se može pokrenuti i izvoditi na računalu. Zbog toga je svaka ključna točka sa slike razložena na niz nadogradnji za testiranje i uporabu.
Iskustvo o sigurnosti programa iz realnog svijeta rezultiralo je visoko prioritetnim pravilima za izgradnju sigurnijih aplikacija. Tvrtka Microsoft ta pravila naziva SD^3+C – *Secure by Design, Secure by Default, Secure in Deployment, and Communications*. Slijedi kratki opis svakog pravila:

- Sigurnost po dizajnu (eng. *Secure by Design*): programski paket treba biti konstruiran, dizajniran i implementiran tako da štiti sebe i informacije koje obrađuje te treba pružati otpor napadačima.
- Zadana sigurnost (eng. *Secure by Default*): u stvarnosti, niti jedan programski paket ne pruža potpunu sigurnost. Zato programeri trebaju pretpostaviti da će se u programu otkriti sigurnosni propusti. Početne postavke aplikacije trebaju promovirati sigurnost. Na primjer, tvorničke postavke su pokretanje programa s minimalnim potrebnim ovlastima, servisi i svojstva koja nisu prijeko potrebna su isključeni ili dostupni malom broju korisnika.
- Sigurnost pri instalaciji (eng. *Secure in Deployment*): Svaki program treba imati prateće upute i alate koji olakšavaju upotrebu programa na siguran način. Instalacija nadogradnje i ispravljenih inačica treba biti jednostavna.
- Komunikacija (eng. *Communications*): Razvojni programeri moraju biti spremni na pojavu sigurnosnih ranjivosti u programskim paketima te otvoreno i odgovorno komunicirati s krajnjim korisnicima i/ili administratorima da im pomognu otkloniti sigurnosni problem (npr. izdavanje zakrpa, zaobilaženje problema i sl.)

Dok svaki element SD^3+C razvojnog procesa definira potrebne značajke programa, prva dva elementa – sigurnost po dizajnu i zadana sigurnost – najviše pridonose sigurnosti programa. Sigurnost po dizajnu diktira pravila za sprečavanje pojave ranjivosti, a zadana sigurnost zahtjeva minimalnu prvotnu izloženost programa napadima.
Slika 2. prikazuje sigurnosne mjere koje integriraju SD^3+C paradigmu u postojeći proces razvoja stvarajući sveukupan organizacijski proces.



Slika 2. Unapređenje razvojnog procesa uvođenjem mjera sigurnosti

5.1. Razvojni proces s uključenim mjerama sigurnosti

Bitno je spomenuti da je edukacijski program kritičan za uspješnost ciklusa razvoja sigurnosti (eng. *Security Development Lifecycle* – SDL). Studentima koju su tek diplomirali na području računalne znanosti i područjima vezanim uz računarstvo uglavnom manjka iskustva i vještine za trenutno uključenje u dizajn, razvoj i testiranje sigurnih programa. Čak i oni studenti koji su odslušali predmete vezane uz računalnu sigurnost susreli su se samo s kriptografskim algoritmima i modelima kontrole pristupa, a nisu se upoznali, primjerice, s pogreškama prepisivanja spremnika i pogreškama pretvorbe podataka. Općenito, programskim dizajnerima, inženjerima i testerima u industriji manjka prikladnih sigurnosnih vještina.

Organizacija koja želi razvijati sigurne programe mora preuzeti odgovornost edukacije svojih inženjera. Načini ostvarenja ovog izazova variraju u ovisnosti o veličini poduzeća i raspoloživim sredstvima.

5.1.1. Faza potražnje

Temeljno načelo razvoja sigurnog sustava je razmatranje sigurnosti od "dna prema vrhu". Dok mnogi proizvođači izdaju "iduće inačice" koje se nadograđuju na prethodna izdanja programa, faza potražnje i početno planiranje novog izdanja ili inačice nude najbolju priliku za gradnju sigurnih programa.

Tijekom faze potražnje, posao se podijeli između dvije osnovne skupine, razvojne skupine i skupine zadužene za sigurnost. Razvojna skupina ostvaruje kontakt sa središnjom skupinom zaduženom za sigurnost (skraćeno skupina za sigurnost) i podnosi zahtjev za dodjelom savjetnika za sigurnost (u tvrtki Microsoft u ciklusu razvoja sigurnosti nosi nadimak *security buddy*). Savjetnik za sigurnost je izvor znanja i vodič tijekom planiranja te pomaže razvojnoj skupini pregledom planova, svojim sugestijama i osiguravanjem da skupina za sigurnost planira svoje resurse u skladu s rokovima razvojne skupine. Osim toga, savjetuje razvojnu skupinu o ključnim točkama i izlaznim kriterijima određenim prema veličini projekta, složenosti i riziku. Savjetnik prati skupine za razvoj programa i za sigurnost od početka do završetka projekta i tzv. "Konačne sigurnosne revizije" (eng. *Final Securitiy Review*) te izlaska programskog paketa na tržište. Također, savjetuje upravu razvojne skupine o pozitivnom smjeru razvoja sigurnosnog elementa.

Faza potražnje razvojnoj skupini pruža priliku za razmatranje načina integracije sigurnosti u proces razvoja, identifikaciju ključnih ciljeva te optimizaciju sigurnosti da na druge načine, uz minimalne promjene planova i rokova. Kao dio procesa skupina treba proučiti kako će se sigurnosne komponente i sigurnosne mjere uklopiti i surađivati s nekim drugim programima. Sveukupno motrište razvojne skupine na sigurnosne ciljeve, izazove i planove treba biti sadržano u dokumentima s planovima. Planovi su podložni izmjenama tijekom procesa razvoja projekta te rano formiranje planova osigurava da niti jedan potrebni element nije propušten ili uočen u zadnji trenutak.

Svaka skupina za razvoj programa treba uzeti u obzir sigurnosne značajke kao integralni dio ove faze. Neke se sigurnosne potrebe identificiraju u ovisnosti o mogućim prijetnjama, a neke ovise o potrebama i zahtjevima korisnika. Sigurnosne se značajke moraju uskladiti sa standardima industrije i procesom dodjele certifikata kao što je *Common Criteria* certifikat. Razvojna skupina treba prepoznati spomenute sigurnosne potrebe kao dio uobičajenog procesa planiranja.

5.1.2. Faza dizajna

Faza dizajna identificira sveukupne potrebe i strukturu programskog paketa. Gledano iz perspektive sigurnosti, ključni elementi faze dizajna su:

- Definiranje sigurnosne arhitekture i smjernica dizajna: Definira se sveukupna struktura programa iz perspektive sigurnosti te se identificiraju komponente čiji je ispravan rad osnova sigurnosti (tzv. "provjerena baza"). Određuju se tehnike dizajna, kao što je uslojavanje, uporaba programskih jezika s dobro definiranim tipovima podataka, aplikacija s minimalnim ovlastima i minimizacija područja potencijalnih napada. (Uslojavanje se odnosi na organizaciju programskog paketa na dobro definirane komponente koje su izgrađene tako da se izbjegnu kružne ovisnosti među komponentama – viši slojevi mogu ovisiti o servisima nižih slojeva, ali obratna ovisnost nije dozvoljena.) Pojedinosti individualnih elemenata u arhitekturi detaljno se razrađuju u individualnim specifikacijama dizajna.
- Dokumentiranje elemenata područja potencijalnih napada: Uzimajući u obzir da niti jedan program neće biti potpuno siguran, važno je da su elementi koje će koristiti velika većina korisnika uočljivi te da korisnici imaju najmanje moguće ovlasti. Mjerenje područja potencijalnih napada pruža vrijedne parametre skupini za razvoj pri definiranju zadane sigurnosti te im omogućava identifikaciju dijelova programa podložnih napadu. Bitno je otkriti i ispitati svaku takvu instancu tijekom dizajna i implementacije tako da program izađe na tržište s početnim postavkama koje jamče najveću moguću sigurnost.
- Modeliranje prijetnji: Modele izrađuje skupina za razvoj na razini komponenti. Koristeći strukturne metode, skupina zadužena za komponente identificira sredstva kojima program mora rukovati i sučelja preko kojih se pristupa spomenutim sredstvima. Proces modeliranja prijetnji otkriva prijetnje koje mogu naškoditi svakom od sredstava te se procjenjuje rizik. Skupina zadužena za komponente kreira protumjere koje umanjuju rizik. Protumjere mogu biti enkripcija ili ispravan rad programa u smislu zaštite vlastitih sredstava. Na ovaj način razvojna skupina identificira potrebe sigurnosnih komponenti i područja programa koja zahtijevaju detaljne sigurnosne provjere i testiranja. Pri modeliranju prijetnji potrebno je koristiti alat za pohranjivanje modela prijetnji u strojnom obliku.
- Određivanje dodatnog kriterija za izlazak na tržište. Osnovni sigurnosni kriterij za tvorničke postavke definira se na organizacijskoj razini, a pojedine razvojne skupine, odnosno paketi mogu imati specifičan kriterij koji mora biti zadovoljen prije izlaska paketa na tržište. Na primjer, razvojna skupina koja radi ažuriranu inačicu programa u prodaji može pričekati neko vrijeme prije objave te nove inačice, iako postoje određene sigurnosne ranjivosti. (Najbolje je pogreške programa otkriti u razvojnoj fazi, a ne nakon izlaska programa na tržište.)

5.1.3. Faza implementacije

Tijekom faze implementacije, proizvodna skupina piše programski kod, testira ga i integrira u programski paket. Poduzimaju se koraci za uklanjanje sigurnosnih pogrešaka i sprečavanje njihovog nastanaka. Time se znatno smanjuje rizik pojave ranjivosti u konačnoj inačici.
Rezultati modeliranja prijetnji pružaju smjernice za implementacijsku fazu. Programeri posvećuju posebnu pažnju ispravnosti programskog koda te tako umanjuju vjerojatnost nastanka rizičnih propusta. Testeri programa pak osiguravaju uklanjanje takvih prijetnji.
Elementi ciklusa razvoja programskih paketa (SDL) koji se primjenjuju u fazi implementacije:

- Primjena standarda pisanja programskog koda i testiranja. Upotrebom standarda kodiranja moguće je ukloniti početne pogreške koje kasnije mogu izrasti u sigurnosne ranjivosti. Na primjer, uporabom sigurnijeg i dosljednijeg rukovanja nizovima znakova te konstruktima za rukovanje spremnicima moguće je izbjeći propuste prepisivanja spremnika. Standardi testiranja pomažu u otkrivanju potencijalnih sigurnosnih propusta.
- Primjena alata za testiranje nasumičnim upisom ulaznih podataka (eng. *fuzzing*). Nasumičan unos ulaznih podataka podrazumijeva strukturiran, ali neispravan unos podataka u već ugrađena programska sučelja (eng. *Application Programming Interfaces* – API) te mrežna sučelja kako bi se povećala vjerojatnost otkrivanja pogrešaka koje se mogu pretvoriti u sigurnosne ranjivosti.

- Primjena statičke analize pri pregledu programskog koda. Alatima je moguće uočiti neke pogreške programskog koda koje predstavljaju sigurnosni rizik, uključujući i pogreške prepisivanja spremnika, cjelobrojnih vrijednosti i neinicijaliziranih varijabli. Tvrtka Microsoft je mnogo uložila u razvoj takvih alata (dva alata koja se koriste dugi niz godina su PREfix i PREfast) i stalno poboljšava te alate kako se otkrivaju novi sigurnosni propusti.
- Revizije programskog koda. One nadopunjuju automatizirane alate i testove, a izrađuju ih razvojni programeri pregledom programskog koda. Oni otkrivaju i uklanjanju potencijalne sigurnosne ranjivosti.

5.1.4. Faza verifikacije

U fazi verifikacije program je funkcionalno potpun i izdaje se kao beta inačica koju testiraju korisnici. Tijekom beta testiranja razvojna skupina provodi dodatne sigurnosne revizije programskog koda, tzv. *security push*. Tvrtka Microsoft uvela je 2002. godine takve dodatne revizije tokom faze verifikacije operacijskog sustava Windows Server 2003 te kod još nekih programskih paketa. Dva su razloga za dodatne sigurnosne revizije:

- Životni ciklus spomenutih programskih paketa došao je do faze verifikacije kada je bilo prikladno provesti fokusirane revizije koda i testiranje.
- Dodatne revizije tijekom faze verifikacije osiguravaju ispravnost ciljne inačice programa te pružaju priliku za usporedbu koda ispravljenog tokom faze implementacije i prvotnog programskog koda (eng. *legacy code*).

Prvi razlog rezultat je slučajne odluke o provođenju dodatne sigurnosne revizije tijekom faze verifikacije. Kasnije se pokazalo da je to dobro za cjelokupni proces.

5.1.5. Faza puštanja proizvoda na tržište

U ovoj fazi programski se paket podvrgava Konačnoj sigurnosnoj reviziji ("KSR") (eng. *Final Security Review*). Cilj KSR-a je odgovoriti na jedno pitanje: "Je li ovaj programski paket spreman za korisnike sa stajališta sigurnosti?". KSR se provodi dva do šest mjeseci prije nego što je proizvod završen. Programski paket mora biti stabilan prije provođenja KSR-a, a predviđene preinake su minimalne i nevezane uz sigurnost.

KSR je nezavisna revizija programskog koda koju provodi središnja sigurnosna skupina. Sigurnosni savjetnik savjetuje razvojnu skupinu o opsežnosti KSR-a, a ona pak prosljeđuje potrebne informacije i sredstva sigurnosnoj skupini koja dovršava KSR. Spomenuti dokument započinje dovršavanjem upitnika koji je razvila razvojna skupina i razgovorom s članom sigurnosne skupine koji je dodijeljen KSR-u. Bilo koji KSR zahtijeva reviziju inicijalno identificiranih sigurnosnih pogrešaka koje ne ugrožavaju sveukupnu sigurnost aplikacije. KSR sadrži i izvještaj o tome kako se program nosi s novo otkrivenim ranjivostima. Za konačnu inačicu KSR-a potrebno je testirati moguće napade i po potrebi angažirati vanjske sigurnosne savjetnike kao dodatnu pomoć sigurnosnoj skupini.

KSR nije test koji aplikacija treba proći ili pasti, ili otkriti sve preostale ranjivosti, već predstavlja cjelokupnu sliku sigurnosnih komponenti i procjenjuje vjerojatnost s kojom će se program oduprijeti napadu kada izađe na tržište. Ako KSR otkrije uzorak preostalih ranjivosti, ispravno je ne samo ukloniti sigurnosne ranjivosti nego i ponoviti neku od prethodnih faza razvoja te poduzeti odgovarajuće akcije.

5.1.6. Faza podrške i servisiranja

Unatoč primjeni ciklusa razvoja programskih paketa, vrhunske metode razvoja ne generiraju programe bez sigurnosnih ranjivosti. Čak i kad bi bilo moguće ukloniti sve ranjivosti tijekom razvoja, uvijek će se naći inovativni napadači s novim metodama napada. Razvojne skupine moraju biti spremne odgovoriti na nove metode napada i izdati sigurnosne preporuke i zakrpe kada je to prikladno.

Cilj faze podrške je učiti iz prethodnih pogrešaka i iskoristiti informacije iz izvještaja o ranjivostima za otkrivanje novih sigurnosnih propusta. Tijekom ove faze razvojna skupina i sigurnosna skupina imaju priliku prilagoditi razvojni proces tako da se slične pogreške više ne javljaju.

6. Zaključak

U ovom dokumentu opisani su najčešći uzroci sigurnosnih ranjivosti programskih paketa, te mogući načini njihovog uklanjanja. Razvoj sigurnog koda uvjetovan je pažljivom organizacijom ciklusa razvoja sigurnosti programskih paketa, kao što je prikazano na primjeru tvrtke Microsoft. Nažalost, nikad nije moguće generirati potpuno siguran programski kod jer će napadači uvijek pronaći način za zaobilaženje sigurnosnih mjera i osmišljavanje nove tehnike napada. Ipak, ostvarenjem dobre podrške nakon izdavanja programa na tržište i konstantnim unaprjeđenjem programskog koda moguće je minimizirati ili čak spriječiti najčešće sigurnosne prijetnje.

7. Reference

[1] Prepisivanje spremnika , http://en.wikipedia.org/wiki/Buffer_overflow , kolovoz 2007.

[2] Propusti prepisivanja spremnika, http://www.cert.hr/filehandler.php?did=301, kolovoz 2007.

[3] Izgradnja sigurnih sustava, http://www.securityfocus.com/infocus/1596, lipanj 2002.

[4] Vodič za sigurno kodiranje, http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/SecureCodingGuide.pdf, svibanj 2006.

[5] M. G. Graff, K. R. van Wyk: *Secure Coding: Principles & Practices*, O'Reilly, 2003.

[6] F. Schindler: *Coping with Security in Programming*, Acta Polytechnica Hungarica, Vol.3 No.2, 2006.

[7] Ciklus razvoja sigurnosti programske potpore, http://msdn2.microsoft.com/en-us/library/ms995349.aspx, ožujak 2005.