ZI (2. ciklus)

Tema: OAuth2

Datum: 16/05/2025

Q: Koji su osnovni sudionici OAuth 2.0 protokola?
A: Vlasnik resursa, klijent, autorizacijski poslužitelj i resursni poslužitelj.

Q: Koji osnovni tokovi OAuth 2.0 se ne preporučuju za upotrebu u produkcijskom okruženju i zašto?
A: Tokovi:

1. Tok vjerodajnica vlasnika resursa jer korisnik daje lozinku izravno aplikaciji, što povećava rizik krađe vjerodajnica.
2. Implicitni tok jer pristupni token ide kroz URL i time je preizložen.

Q: Što je PKCE u OAuth 2.0 protokolu i što dodatno koristi u toku autorizacijskog koda?
A: PKCE je dodatni mehanizam koji javnim klijentima omogućuje sigurniji tok autorizacijskog koda. U toku autorizacijskog koda dodatno koristi dokazni niz i izazov.

Q: Navesti tri česte sigurnosne prijetnje za OAuth 2.0
A: Presretanje kodova ili tokena, Replay napadi, Phishing napadi, CSRF napadi.

Q: Navesti i objasniti dvije preporučene sigurnosne prakse vezane za OAuth 2.0 implementaciju
A: Preporučene prakse:

- Neka svi zahtjevi i odgovori između sudionika idu preko TLS-a radi zaštite od presretanja i krađe tokena.
- Korištenje state parametra za povezivanje zahtjeva i odgovora te sprječavanje lažnih zahtjeva.
- Ograničavanje privilegija pristupnih tokena radi smanjivanja štete u slučaju krađe/zlouporabe.

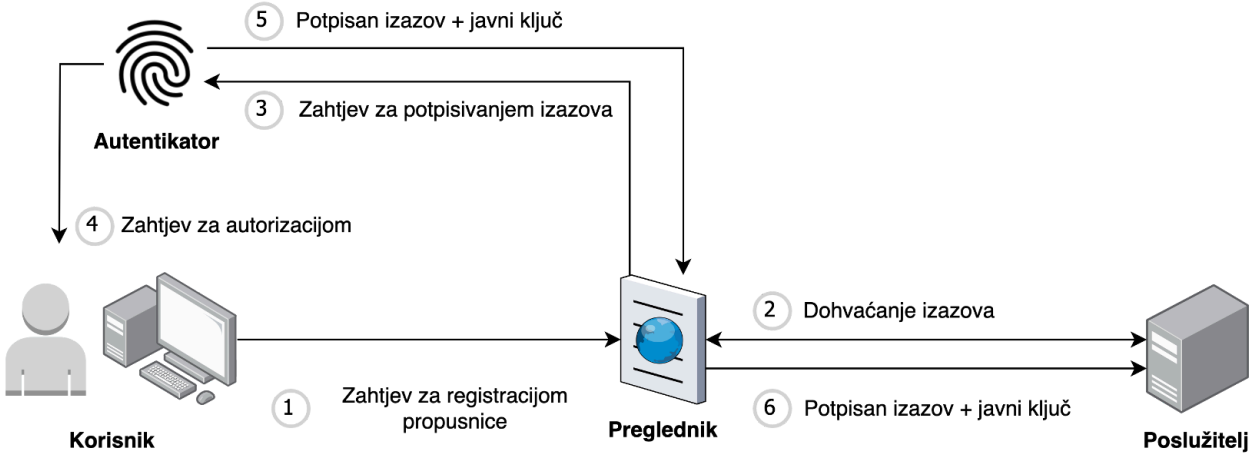
Tema: Passkeys

Datum: 16/05/2025

Q: Koje napade sprječava korištenje passkey tehnologije?

A: Sprječava “phishing” napade, “brute-force” pogađanje, curenje (data breach), “credential stuffing” napade.

Q: Opišite proces stvaranja i korištenja passkeya za prijavu na neku uslugu.
A:



Q: Što se događa ako korisnik izgubi uređaj na kojem je pohranjen passkey?
A: Specifikacija ne zahtjeva, niti definira mehanizme za povrat izgubljenog računa (eng. account recovery), ali brojne usluge pružaju takve mehanizme (npr. reset računa mailom, putem telefona, itd).

Q: Može li se jedan passkey koristiti za više usluga? Objasnite.
A: Ne, passkey je kriptografski vezan uz domenu usluge za koju je stvoren i ne može biti korišten ni za jednu drugu.

Q: Zašto je uveden WebAuthn standard i što omogućuje?
A: Uveden je kako bi se omogućio standardizirani pristup autentifikacije bez lozinki na webu. Omogućuje povećanu sigurnost i interoperabilnost između OS-ova, preglednika i web usluga.

Tema: Obfuskacija koda

Datum: 16/05/2025

Q: Objasnite što je obfuskacija kôda?
A: Obfuskacija kôda je tehnika kojom se programski kôd namjerno čini teže razumljivim ljudima, iako zadržava istu funkcionalnost i ponašanje. Cilj je da kod i dalje radi isto, ali da ga je puno teže analizirati, razumjeti ili prepraviti.

Q: Objasnite koje su glavne svrhe obfuskacije kôda?

A: Glavna svrha obfuskacije je zaštita softvera od obrnutog inženjerstva, gdje napadači iz izvršnog koda dolaze do izvornog. Obfuskacija povećava vrijeme i trošak obrnutog inženjerstva. Time se štitimo od krađe algoritama, štitimo intelektualno vlasništvo i otežavamo otkrivanje ranjivosti u sustavu.

Q: Kako obfuskacija može otežati reverzno inženjerstvo?
A: Reverzno inženjerstvo omogućuje napadačima da rekonstruiraju izvorni kôd iz izvršne datoteke, ali obfuskacija taj proces čini puno težim. Obfuskacija može zaustaviti neke dekompajlere. Ako dekompilacija uspješno prođe generirani kod opet može biti presložen za razumjet.

Q: Što je neprozirni predikat (opaque predicate) i zašto je bitan u kontekstu obfuskacije kôda?
A: Neprozirni predikat je metoda obfuskacije. Čini ga logički uvjet koji je uvijek istinit (ili uvijek neistinit), ali se izvana to ne može lako znati. Napadač ne može sa sigurnošću reći koje se grane koda izvršavaju, pa mora analizirati više mogućih puteva. To značajno povećava vrijeme potrebno za analizu izvođenja.

Q: Što je bogus insertion i zašto je bitan u kontekstu obfuskacije kôda?
A: Bogus insertion je metoda obfuskacije. Označava umetanje koda koji nema stvarnu funkciju - ili se nikad ne izvršava ili nema utjecaja na rad programa. Ova tehnika dodatno zbunjuje analitičara jer povećava količinu koda koji se mora pregledati.

Tema: Statička analiza koda

Datum: 23/05/2025

Q: Što je statička analiza koda?
A: Statička analiza koda je metoda detektiranja pogrešaka u računalnim programima koja se provodi ispitivanjem koda bez izvršavanja programa.
BITNO: **statička** – prije pokretanja, **automatizirana** – ne izvode je ljudi

Q: Navedite tri razlike između statičke i dinamičke analize koda
A: (odaberi tri)

Vrsta analize koda:	**Statička**	**Dinamička**
1. Definicija	Analiza koda bez izvršavanja	Analiza koda izvršavanjem u stvarnom ili simuliranom okruženju
2. Kada se izvodi	Prije izvršavanja koda	Dok se program izvršava
3. Glavni fokus	Kvaliteta koda, sigurnosne ranjivosti	Problemi s performansama,

	i pridržavanje standarda kodiranja	pogreške tijekom izvođenja
4. Potrebni alati	Alati za statičku analizu (npr. SonarQube, Checkmarx, TypeScript)	Profiler, debugeri i drugi alati za praćenje izvođenja

Q: Što je potrebno napraviti kako bi se stvorio novi detektor za alat FindBugs?
A: Potrebno je napisati novu klasu detektora po zadanima pravilima (nasljedi klasu, implementiraj sušelje) i uključiti je u .xml konfiguracijsku datoteku.

Q: Navedite dva primjera greške u kodu koje statička analiza detektira.
A: Nekoliko primjera grešaka u kodu koje statička analiza detektira:

- Beskonačna rekurzivna petlja
  - `public String someFunction() { return this.someFunction(); }`
- Ignoriranje povratne vrijednosti
  - `s.toLowerCase();`
- Stvaranje iznimke bez *bacanja*
  - `try { … } catch (Exception e) { new IOException(…); }`
- Korištenje krive relacijske ili Booleanove operacije
  - `if (name != null || name.length > 0) { … }`

Q: U kojoj fazi CI/CD-a se izvodi statička analiza i zašto?
A: Odmah nakon commit-a, a unutar CI/CD-a odmah nakon build-a (izgradnje) kako bi se greške otkrile što ranije, jer kasnije mogu skupo koštati kada uđu u produkciju.

Tema: Statička analiza koda: Programski jezik C

Datum: 23/05/2025

Q: Što su to nedefinirana ponašanja i kako utječu na sigurnost programa?
A: Nedefinirana ponašanja se događaju kada se program nađe u izvanrednom stanju za koji specifikacija programskog jezika nema definiranu strategiju rezolucije. Daljnje ponašanje programa tako ovisi o okolini izvođenja te je vrlo često nepredvidivo. Mnoge ranjivosti proizlaze iz mogućeg iskorištavanja nedefiniranih ponašanja.

Q: Navedite nekoliko čestih pogrešaka koje uzrokuju nedefiniranim ponašanjima u C programima.
A: Buffer overflow, Integer overflow, Out-of-bounds indexing, Null-pointer dereference…

Q: Objasnite tehniku semantičke statičke analize.
A: Semantička statička analiza obuhvaća dizanje koda u apstraktnu reprezentaciju (kao npr. AST - abstract syntax tree) kako bi se mogla analizirati kontrola i protok podataka u programu. Nad

tom apstraktnom reprezentacijom se onda rade brojne analize kako bi se ustanovile ranjivosti u izvornom programu.

Q: Koja je korist korištenja više alata sa statičku analizu na istom kodu?
A: Alata za statičku analizu ima mnogo i svaki na svoj način provodi analizu. Neki alat tako može biti puno bolji (ili lošiji) od nekog drugog alata za specifične tipove ranjivosti. Kombiniranjem više alata širimo skup ranjivosti koje možemo efektivno detektirati.

Q: Što je to UBSAN i kako se razlikuje od alata za statičku analizu?
A: UndefinedBehaviorSanitizer (UBSAN) je program integriran u C prevoditelje koji služi za spriječavanje nedefiniranih ponašanja. Za razliku od alata za statičku analizu, UBSAN djeluje za vrijeme prevođenja uvodom provjera za zadana nedefinirana ponašanja u strojni kod, tako da program stane u trenutku kada se nađe u stanju koje izaziva nedefinirana ponašanja.

Tema: Dinamička analiza koda: Fuzzing

Datum: 23/05/2025

Q: Objasnite razliku između black-box, white-box i grey-box fuzzing pristupa
A: Ovisi o udio koda koji tester/napadač ima na raspolaganju.

- White-box -> sav kod i struktura dostupna
- Grey-box -> neki dijelovi koda/strukture dostupni
- Black-box -> Nema pristup izvornom kodu/strukturi

Q: Navedite i opišite najmanje tri vrste tehnika generiranja ulaznih podataka u fuzzingu
A: Vrste tehnika generiranja ulaznih podataka u fuzzingu:

- **Mutacijsko** (modificiramo postojeće validne ulaze)
- **Generativno** (stvara ulazne podatke prema specifikaciji)
- **Gramatičko** (koristi formalnu gramatiku za generiranje ulaza, idealan za jezike i strukturirane protokole) stvaranje ulaznih podataka
- (ima još i **hibridno**, to valjda samo sebe objašnjava)

Q: Navedite barem 2 uspješna primjera korištenja fuzzinga u industriji te ih ukratko opišite
A:
Google Project Zero

- Elitni tim sigurnosnih istraživača
- Koriste fuzzing za pronalazak zero-day ranjivosti
- Otkrili tisuće sigurnosnih propusta u kritičnom softveru
- Jedan od najuspješnijih primjera primjene fuzzinga

Microsoft Security Development Lifecycle (SDL) dio CI/CD-a

- Fuzzing kao obavezni dio razvoja softvera

- Implementirano za sve Microsoft proizvode
- Značajno smanjenje sigurnosnih incidenata

Q: Opišite neke od (barem 3) glavnih izazova moderne fuzzing metodologije i kako se adresiraju

A: Glavni izazovi moderne fuzzing tehnologije:

Logičke barijere:

- **Problem**: Fuzzing teško otkriva semantičke greške (npr. pogrešnu poslovnu logiku).
- **Rješenje**: Hibridni pristupi (kombinacija fuzzinga sa statičkom analizom ili ručnim testiranjem).

Dubinski izazovi:

- **Problem**: Checksum provjere, "magic bytes", autentikacijski mehanizmi blokiraju nevažeće ulaze.
- **Rješenje**: Format-aware fuzzing (npr. Peach Fuzzer) koji modificira samo ne-kritične dijelove ulaza.

Strukturni izazovi:

- **Problem**: Složene arhitekture (mikroservisi, JIT kompilacija) otežavaju praćenje izvršavanja.
- **Rješenje**: Alati s podrškom za distribuirano praćenje (npr. ClusterFuzz) i hardverski feedback (npr. honggfuzz s Intel PT).

Organizacijski problemi:

- **Problem**: Integracija u CI/CD, nedostatak resursa i znanja.
- **Rješenje**: Automatizacija kroz CI/CD pipeline (npr. OSS-Fuzz za open-source). Edukacija razvojnih timova.

Vremensko upravljanje:

- **Problem**: Nemogućnost određivanja optimalnog trajanja fuzzing kampanje.
- **Rješenje**: Coverage-guided fuzzing (npr. AFL++) koji prioritizira putanje s novim pokrivenostima.

Regulatorni aspekti:

- **Problem**: Usklađenost s GDPR-om i industrijskim standardima.
- **Rješenje**: Dokumentiranje procesa i korištenje certificiranih alata (npr. Defensics).

Q: Navedite najmanje tri popularna fuzzing alata te im opišite svrhu tj. domenu u kojoj se koriste

A: Popularni fuzzing alati i njihova svrha:

- **Defensics (Synopsys)** - mrežni protokoli
- **AFL (American Fuzzy Lop)** - najpopularniji grey-box fuzzer, executable/binary datoteke
- **libFuzzer** - jedinični testovi, najlakši za korištenje

Tema: Pentest koda

Datum: 06/06/2025

Q: Što je penetracijsko testiranje i koji je cilj ovog postupka?

A: Penetracijsko testiranje je kibernetička vještina u kojoj stručnjak za kibernetičku sigurnost ili tim stručnjaka pokušava pronaći ranjivost sustava i iskoristiti ih. Cilj je pronalazak ranjivosti sustava.

Q: Koji tipovi penetracijskog testiranja postoje i ukratko ih opišite?

A: Tipovi testiranja:

- **White box** testiranje je tehnika testiranja programskog rješenja koja uključuje testiranje unutarnje strukture programskog rješenja (ukratko koda) uz to da su testeru poznate sve informacije o sustavu.
- **Black box** testiranje je tehnika testiranja programskog rješenja ili sustava koja uključuje testiranje samo vanjske strukture rješenja, tj. jedine informacije s kojima tester raspolaže su one koje dobiva kao povratne informacije od sustava.
- **Grey box** testiranje je kombinacija White box i Black box tehnike.

Q: Koji su koraci penetracijskog testiranja koda, navedite ih?

A: Koraci penetracijskog testiranja koda su:

1. **Planiranje i priprema**
2. **Izrada modela prijetnji**
3. **Izrada plana testiranja**
4. **Provođenje testova**
5. **Automatizirano i ručno analiziranje koda**
6. **Izvještavanje**
7. **Ponovno provođenje cijelog postupka**

Q: Što uključuje definiranje sigurnih granica koda u procesu izrade modela prijetnji?

A: Definiranje sigurnosnih granica koda podrazumijeva definiranje područja visokog rizika u kodu. To su dijelovi koda koji se bave autentifikacijom, provjeravanjem korisničkog unosa, deserijalizacijom podataka, šifriranjem, pospremanjem podataka, …

Q: Koje kategorije alata za penetracijsko testiranje koda postoje i navedite po 2 za svaku kategoriju.

A: Kategorije i neki pripadni alati su:

1. Alati za **statičku analizu koda**: Semgrep, SonarQube, CodeQL, FindBugs (obsolete) / SpotBugs, Checkmarx
2. Alati za **dinamičku analizu koda**: OWASP ZAP, Buri Suite, ThreadSanitizer, Valgrind, AFL++, LibFuzzer, Seeker

---

Tema: Revizija koda

Datum: 06/06/2025

Q: Što je proces revizije koda?

A: Proces revizije koda je postupak analize izmjene izvornog koda koja se treba isporučiti u službeni izvorni kod. U procesu revizije sudjeluju sudionici projekta, proučavaju izvorni kod te kada se ispune svi uvjeti (norme, pisana pravila, definicija zadatka), tada potpisuju izmjenu koja se potom smije isporučiti.

Q: Koje su dobre prakse revizije koda?

A: Ispravno uočavanje propusta, poznavanje normi, pravila i definicije zadatka, dobro razumijevanje cijelog izvornog koda, jasnoća i eksplicitnost u komunikaciji s kolegama (kroz komentare).

Q: Kako sudionici revizije potvrđuju ispravnost iteracije izmjene?

A: Sudionici potvrđuju ispravnost iteracije ovisno o implementaciji sustava za reviziju koda koji se koristi. To može biti kroz potpise, ocijene ili bodove. Ocjenjivati mogu svi ili samo jedan sudionik, ovisno o pravilima.

Q: Nabrojati izazove računalne sigurnosti u procesu revizije koda.

A: Nedostatak aktivnog rada na sigurnosti, manjak poznavanja pojma i svijesti o računalnoj sigurnosti, nedostatak motivacije, vremena, uporaba vanjskih komponenti te ljudski faktori (umor, zasićenost poslom, stres i slično).

Q: Nabrojati poboljšanja računalne sigurnosti u procesu revizije koda.

A: Motivirati sudionike revizije koda za fokus na sigurnost te podizanje opće svijesti, edukacija i pomoć od strane stručnjaka.

Tema: Bounty programi

Datum: 06/06/2025

Q: Što je bug bounty program i koja je njegova osnovna svrha?

A: Bug bounty program je inicijativa u kojoj tvrtke ili organizacije nagrađuju pojedince za pronalazak i prijavu sigurnosnih ranjivosti. Osnovna svrha je otkriti i otkloniti sigurnosne propuste prije nego što ih iskoriste zlonamjerni napadači.

Q: Što znači pojam "white-hat hacker" u kontekstu bug bounty programa?

A: White-hat hacker je etički haker koji s dopuštenjem organizacije traži sigurnosne propuste kako bi ih prijavio i pomogao u njihovom otklanjanju.

Q: Navedite barem dvije prednosti korištenja bug bounty programa.
A: Prednosti bug bounty programa su veći broj sigurnosnih istraživača, brže otkrivanje ranjivosti, niži troškovi u odnosu na tradicionalno testiranje te dodatna motivacija za etičke hakere.

Q: Nabroji 4 osnovna koraka u procesu javnih bug bounty programa.
A: Prijava ranjivosti -> Evaluacija -> Potvrda i klasifikacija -> Isplata i objava

Q: Kako se vrednuju prijavljene ranjivosti u bug bounty programima?
A: Ranjivosti se vrednuju prema njihovoj težini i potencijalnom utjecaju na sigurnost, često koristeći standard poput CVSS (Common Vulnerability Scoring System).

Tema: Objava ranjivosti

Datum: 13/06/2025

Q: Što je CVD?
A: CVD je koordinirani proces više dionika u kojem se prikupljaju informacije o ranjivostima od njihovih otkrivača, koordinira dijeljenje tih informacija između relevantnih dionika i javno objavljuje postojanje ranjivosti i mjere za njihovo ublažavanje.

Q: Nabroji glavne dionike CVD-a i za svakog navedi primjer.
A: Glavni dionici CVD-a su:

- **Istraživač** (finder) - pentester, istraživatelj, developer, korisnik
- **Prijavitelj** (reporter) - često je to sam istraživač
- **Proizvođač** (vendor) - razvijaju ili održavaju softver (npr. Microsoft)
- **Provoditelj zakrpe** (deployer) - bilo tko tko mora primjeniti patch, npr. korisnik, developer
- **Koordinator** (coordinator) - npr. CERT

Q: Nabroji i objasni korake CVD-a.
A: Koraci CVD-a su:

1. **Otkrivanje** - Istraživač pronalazi sigurnosnu ranjivost
2. **Prijava** - Ranjivost se prijavljuje odgovornom subjektu
3. **Verifikacija i trijaža** - Subjekt potvrđuje ranjivost i procjenjuje njezinu ozbiljnost
4. **Otklanjanje** - Razvija se i testira zakrpa ili drugo rješenje
5. **Objava** - Informacije o ranjivosti i zakrpi se javno objavljuju
6. **Promoviranje zakrpe** - Korisnike se potiče da implementiraju zakrpu što prije

Q: Nabroji politike objave ranjivosti i objasni ih.
A: Politike objave ranjivosti su:

- **Politika povjerljivosti** - Istraživač otkrije ranjivost, ali ne obavijesti ni proizvođača ni sigurnosne koordinatore
- **Potpuna objava** - Istraživač odluči odmah javno objaviti sve detalje o ranjivosti
- **Odgovorna objava** - Istraživač prvo obavještava proizvođača softvera i daje mu razumno vrijeme da razvije zakrpu

Q: Nabroji i objasni rizike prijave ranjivosti.
A: Rizici prijave ranjivosti su:
- **Operativni rizik** - Testiranje otkrivene ranjivosti može imati nepredviđene posljedice na sustav
- **Sigurnosni rizik** - Kvar mnogih sustava bi mogao rezultirati gubitkom života ili oštećenjem imovine ili okoliša
- **Pravni rizik** - Objavljivanje ranjivosti bez prethodnog savjetovanja s proizvođačem može dovesti do tužbi ili pravnih akcija

---

Tema: Sigurnost dobavnog lanca
Datum: 14/06/2025

Q: Navedite i opišite elemente od kojih se sastoji svaka poveznica u unutar dobavnog lanca.
A: Artefakti, procesi, sudionici i veze sa susjednim poveznicama ukoliko ih ima.
- **Artefakti** su resursi, među koje spadaju izvorni kôd, razvojna infrastruktura i ovisnosti.
- **Procesi** su akcije i koraci u kojima se koriste artefakti. Tu spada dohvaćanje ovisnosti, prevođenje programskog koda, testiranje, pakiranje, skeniranje ranjivosti itd.
- **Sudionici** su ljudi ili sustavi koji izvode procese nad artefaktima, primjerice programeri.

Q: Navedite i ukratko objasnite tri sigurnosna svojstva ključna za sigurnost dobavnog lanca.
A: Transparentnost, ispravnost i razdvojenost.
- **Transparentnost** je dostupnost znanja o cijelom dobavnom lancu, tj. mogućnost uvida u svaki korak procesa i vidljivost tko je što i kada napravio s artefaktima.
- **Ispravnost** je integritet procesa, artefakata i sudionika unutar dobavnog lanca. (Sigurnost da su svi artefakti autentični, neoštećeni i dolaze iz pouzdanih izvora.). To sprječava neovlaštene promjene.
- **Razdvojenost** je razdvajanje komponenti dobavnog lanca tako da se maknu nepotrebne veze i time smanji površina napada.

Q: Što je SBOM i koje sigurnosno svojstvo promovira?
A: SBOM (**Software Bill of Materials**) je inventar svih sastavnih komponenti softverskog proizvoda koji omogućuje praćenje metapodataka i povezivanje s drugim izvorima informacija. Promovira svojstvo **transparentnosti**.

Q: Nabrojite tri vektora napada na dobavni lanac, te za svaki ukratko opišite barem jednu strategiju zaštite.
A: Vektor ovisnosti, vektor build infrastrukture te ljudski vektor.
Vektor ovisnosti (odaberi jednu):

- Procjena komponenti putem sigurnosnih metrika – npr. korištenje alata OpenSSF Scorecard za procjenu sigurnosnih rizika i stanja ovisnosti, kako bi donijeli informiraniju odluku o odabiru ovisnosti.
- Automatizacija ažuriranja za ažuriranje zastarjelih ili ranjivih ovisnosti. To se može sastojati od: detekcija novih verzija ovisnosti, automatsko stvaranje pull requestova, zatim testiranje projekta s novim ovisnostima i automatski merge ako prođu testovi za taj pull request.
- Provjera digitalnih potpisa i povijesti izmjena (engl. commit) kako bi se otkrile sumnjive promjene.

Vektor build infrastrukture (odaberi jednu):

- Transparentnost build procesa, recimo zapisivanjem detaljnih build logova, smanjuje šansu da se napad na build infrastrukturu dogodi neopaženo. Ako dođe do kompromitacije, moguće je brzo utvrditi što se točno dogodilo, kako i tko je bio uključen.
- Izolacija build okoline korištenjem kontejnera ili virtualnih strojeva kako bi se spriječile neželjene međusobne interakcije.
- Reproducible builds koji omogućuju svakome da provjeri jesu li binarne datoteke (koje su produkt build procesa) zaista dobivene od poznatog izvornog koda, bez poremećaja i utjecaja neželjenih aktora.

Ljudski vektor (odaberi jednu):

- Obuka i osvješćivanje zaposlenika – redovite edukacije o sigurnosnim praksama i prepoznavanju napada, poticanje zaposlenika na odgovorno korištenje i dijeljenje informacija
- Ograničavanje privilegija i pristupa – princip najmanje privilegije
- Podrška ekosustavu otvorenog kôda. Poticanje zaposlenika na suradnju u projektima otvorenog koda, kako bi se smanjili rizici vezani uz nesiguran ili neodržavan kod.
- Sigurno upravljanje tajnama u kôdu (code secret management) unutar platformi kao što su GitHub i GitLab – da tajni API ključevi ili lozinke ne „procure" u javne repozitorije

Q: Čemu služi alat OpenSSF Scorecard i za koji vektor napada na dobavni lanac pruža zaštitu?
A: OpenSSF Scorecard služi za automatsku procjenu sigurnosti projekata otvorenog koda (open source). On daje ocjene na temelju različitih sigurnosnih kriterija, te tako olakšava procjenu ovisnosti i donošenje informiranih odluka. Pruža bolju zaštitu od napada na vektor ovisnosti.

Tema: Ponašanje programera

Datum: 14/06/2025

Q: Nabrojite barem 3 postojeća rješenja za problem sigurnosnih grešaka programera.
A: Npr. edukacija, bolji API design, dokumentacija, automatizacija procesa, revizija koda…

Q: Ukratko obrazložite što je to kultura u kontekstu tima razvojnih programera?
A: Kolektivna suma znanja, motivacija, stavova, ponašanja, radnih navika i rutina razvojnih programera.

Q: Koja je glavna prepreka razvoju sigurnog koda u agilnoj metodologiji razvoja?
A: Nedovoljno definiranog vremena za sigurnosni posao, prioritiziranje funkcionalnost iznad sigurnosti, problematično planiranje troška sigurnosnog posla.

Q: Što je disperzija odgovornosti u kontekstu razvoja sigurnog softvera?
A: Smanjen osjećaj individualne odgovornosti pojedinca, jer je za sigurnost odgovoran čitav tim ili tvrtka.

Q: Navedite barem 3 smjernice za dizajn upotrebljivih kriptografskih API-ja i njihovo sigurno korištenje.
A: Neke smjernice:

- Integracija kriptografije u postojeća sučelja
- API treba zadovoljavati sigurnosne i ne-sigurnosne potrebe
- API treba biti jednostavan za naučiti
- Nemoj narušiti programerovu paradigmu
- API treba biti jednostavan za korištenje
- API treba biti otporan na krivo korištenje
- Sigurne i nedvosmislene zadane postavke
- Pružiti način rada za testiranje
- Omogućiti lako održavanje i ažuriranje
- API treba komunicirati s krajnjim korisnikom

Nez dal Sistemski i operativni zapisi (logiranje) sada spada pod ZI