nedelja informatike 3

Autostoperski vodič kroz softverske ranjivosti

David Davidović

Matematička gimnazija
Nedelja informatike[3]

15. decembar 2016.

Softverska ranjivost (vulnerability)

- Greška u ideji ili realizaciji softvera
- *Ranjivost* jer je moguće naterati softver na ponašanje za koje nije predviđen
- Nepredviđeno ponašanje uključuje:
  - Nemogućnost normalnog funkcionisanja (*denial of service*)
  - Odavanje informacija (*information disclosure*)
  - Dodeljivanje nepredviđenih privilegija (*privilege escalation*)
  - Izvršavanje proizvoljnog koda (*code execution*)
  - …i mnogo, mnogo više!
- Nadalje: **eksploatacija** (ili *exploitation*) je izazivanje nepredviđenog ponašanja softvera korišćenjem neke ranjivosti

Tipovi ranjivosti

- **Lokalni** ili **udaljeni**? Da li Perica može da eksploatiše ranjivost čak i ako nema fizički pristup uređaju na kome se nalazi ranjiv softver?
- **Feature** ili **bug**? Da li je dobro implementirana ideja dovela do ranjivosti, ili je dobra ideja pogrešno implementirana?
- **Razni dodatni uslovi.** Dva bitna primera:
  - Da li je za eksploataciju neophodna prethodna autorizacija na sistemu? Ako da, kakva? Da li, na primer, samo administrator može eksploatisati ranjivost, ili to može bilo koji korisnik?
  - Sa kolikom uspešnošću se eksploatacija može izvršiti? Da li softver mora biti u nekom specifičnom stanju da bi to uspelo? Da li je ranjivost osetljiva na tajming ili druge male promene u uslovima?

Zašto o svemu ovome?

Yahoo reveals new hack where 'unauthorized third party' stole data from more than 1B accounts

KEN YEUNG DECEMBER 14, 2016 2:03 PM

TAGS: YAHOO, YAHOO NEWS, YAHOO SECURITY

CNET › Security › Sony hack leaked 47,000 Social Security numbers, celebrity data

Sony hack leaked 47,000 Social Security numbers, celebrity data

Documents leaked online include the personal information, salaries and home addresses for employees and freelancers who worked at the studio, a data security analyst finds.

Zašto o svemu ovome? (2)

NSA Said to Have Used Heartbleed Bug, Exposing Consumers

Michael Riley
April 12, 2014, 6:00 AM GMT+2

Hackers of cheaters' site Ashley Madison threaten to expose user profiles

Those who broke into the site are reportedly threatening to post stolen user information every day the "discreet" affair network remains online.

Životna priča jedne softverske ranjivosti

- Tim koji radi na softveru ima malo vremena, nema članova koji su u mogućnosti da sagledaju bezbednosne implikacije rešenja odabranih prilikom dizajna ili realizacije softvera, ili ih jednostavno ne zanima preterano bezbednost onoga na čemu rade
- Neki granični slučaj, na prvi pogled sasvim nevin, se previdi ili će “biti rešen kasnije”™
- Ovaj granični slučaj primeti osoba sa veštinom da prepozna bezbednosne implikacije naizgled bezopasnih bagova
  - *bolja varijanta*: ova osoba ima moralne principe
  - *lošija varijanta*: ova osoba nema moralne principe ili je spremna da zažmuri na jedno oko za nekoliko stotina hiljada dolara

Životna priča jedne softverske ranjivosti (2)

- **U boljoj varijanti**, tim koji radi na softveru biva kontaktiran u tajnosti, nakon čega se greška ispravlja i u najmanjem mogućem roku plasira kao *bezbednosni apdejt*
- **U lošijoj varijanti**, tim koji radi na softveru nije zainteresovan za postojanje bezbednosnog propusta, preti dobrim Samarićanima koji ih kontaktiraju tužbama, ili čak ni ne postoji
- **U najgoroj varijanti**, osoba koja je otkrila ranjivost prodaje informacije na crnoj berzi bezbednosnih propusta, gde agencije od tri slova (NSA, CIA, MI6) ili zlonamerni hakeri dolaze u posed svega što im je potrebno da iskoriste ranjivost za svoje potrebe

OS X ‘goto fail’ bag

- Deo biblioteke koja se koristi na Apple OS X operativnom sistemu za komunikaciju putem SSL-a
  - **SSL** (sada **TLS**) — kriptografski siguran (za sada!) protokol za komunikaciju između dva sistema na nesigurnoj mreži (Internet)
  - SSL je **`s`** u `https://...`
- Ranjivost otkrivena 2014. na vrlo simpatičan način: Apple ju je identifikovao i izbacio bezbednosni apdejt za iOS, nakon čega su radoznali eksperti *reverse engineer*-ovali sam apdejt, otkrili koju ranjivost on pokriva, i zaključili da identičan propust postoji i na OS X-u, jer se koristi ista SSL biblioteka
- Sporna funkcija: `SSLVerifySignedServerKeyExchange`
  - Zvuči kao jako loše mesto za bag, zar ne?

OS X ‘goto fail’ bag — kod

```
1  static OSStatus
2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
3          SSLBuffer signedParams, uint8_t *signature, UInt16 signatureLen)
4  {
5      OSStatus        err;
6  // ...
7      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
8          goto fail;
9      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
10         goto fail;
11         goto fail;
12     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
13         goto fail;
14 // ...
15 fail:
16     SSLFreeBuffer(&signedHashes);
17     SSLFreeBuffer(&hashCtx);
18     return err;
19 }
```

- Vidite li grešku?

OS X ‘goto fail’ bag — kod

```
1  static OSStatus
2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa,
3          SSLBuffer signedParams, uint8_t *signature, UInt16 signatureLen)
4  {
5      OSStatus        err;
6  // ...
7      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
8          goto fail;
9      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
10         goto fail;
11         goto fail;
12     if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
13         goto fail;
14 // ...
15 fail:
16     SSLFreeBuffer(&signedHashes);
17     SSLFreeBuffer(&hashCtx);
18     return err;
19 }
```

- Vidite li grešku?

OS X ‘goto fail’ bag — analiza

- Slučajno duplirana naredba u izvornom kodu, kombinovana sa nedostatkom vitičastih zagrada oko `if` bloka, dovela je do *izuzetno neželjenih rezultata*!
- Nakon ove linije, izvršni tok programa bezuslovno skače na kraj metode, gde se vraća vrednost promenljive `err`
- Ova promenljiva sadrži povratnu vrednost funkcije pozvane unutar `if` bloka — ako je ova funkcija vratila povratnu vrednost “uspešno”, cela funkcija vraća tu povratnu vrednost
- **Rezultat**: prava provera serverskog potpisa se nikad ne izvrši, i ranjive verzije OS X-a omogućuju napadačima da ukradu identitet bilo kog servera koji koristi SSL!
- **Moguće primene**: krađa korisničkih imena i lozinki na onlajn servisima, serviranje kompromitovanog softvera prilikom pokušaja *download*-a...

Falsifikovanje Flickr API potpisa

- **Flickr** — sajt za deljenje slika — pruža integraciju drugih aplikacija sa svojim servisom pomoću API-ja. Ako korisnik to dozvoli, druga aplikacija može izvršiti određene akcije u korisnikovo ime (kao, na primer, današnje Facebook aplikacije)
- Svaka aplikacija koja želi da se integriše dobija svoj **API ključ** koji je identifikuje. Uz API ključ dobija se i **API tajna**, koju zna samo aplikacija
- API tajna se koristi kako bi se zahtevi koje napravi aplikacija u ime korisnika *kriptografski potpisali*, kao dokaz da je kod aplikacije taj koji je izdao zahtev, a ne uljez

Falsifikovanje Flickr API potpisa — metod

- Oblik zahteva:

  `api_key=`**`ključ`**`&perms=`**`privilegije`**`&parametar1=`**`vrednost1`**`...`

- Proces potpisivanja je sledeći:
  - Svi parametri (api_key, perms, parametar1...) se sortiraju po abecednom redu
  - Vrednosti i nazivi tih parametara se spajaju u jedan string:

    `api_key`**`ključ`**`parametar1`**`vrednost1`**`perms`**`privilegije`**
  - API tajna se dodaje na početak:

    `[tajna]``api_key`**`ključ`**`parametar1`**`vrednost1`**`perms`**`privilegije`**
  - Tako dobijen string se pušta kroz heš funkciju **MD5** i dobijeni izlaz je traženi kriptografski potpis!

Falsifikovanje Flickr API potpisa — ranjivost

- Ono što stvaraoci ove šeme nisu uzeli u obzir je da je **MD5** funkcija ranjiva na *length extension* napad
- Kada imamo $h(S)$, gde je S neki string, a funkcija h predstavlja MD5 heš funkciju, možemo lako samo na osnovu njega izračunati i $h(SS')$ za bilo koji string S'!
- Potrebno je presresti bilo koji validan potpis za bilo koji validan zahtev napravljen pomoću nekog API ključa
- Šta se dešava ako pošaljemo sledeći zahtev?

  `a=pi_key`**`ključ`**`perms`**`privilegije`**`parametar1`**`vrednost1`**`...`

Falsifikovanje Flickr API potpisa — ranjivost (2)

- String koji dobijemo u procesu potpisivanja je isti kao i dosad, tako da isti potpis važi za njega:
  `[tajna]api_key`**`ključ`**`parametar1`**`vrednost1`**`perms`**`privilegije`**
  Nazovimo taj string S.
- Presreli smo potpis, dakle imamo $h(S)$
- Promenjeni zahtev koji šaljemo sadrži samo parametar po imenu `a`, koji za API ne znači ništa i koji će biti ignorisan
- Sada možemo na ovaj zahtev dodati proizvoljne parametre:
  `a=pi_key...&našParametar=`**`našaVrednost`**
- U ovom slučaju, S' je `našParametar`**`našaVrednost`**, a zbog MD5 ranjivosti možemo izračunati $h(SS')$, što je upravo validan potpis za naš novi zahtev, sa proizvoljnim parametrima

Falsifikovanje Flickr API potpisa — analiza

- Ranjivost starijeg datuma: davna 2009.
- Dozvoljava bilo kome ko može da presretne bilo koju komunikaciju korisnika i servera u kome se nalazi zahtev poslat od strane neke Flickr aplikacije da izdaje proizvoljne zahteve u ime te aplikacije
- Da stvar bude još gora, identična metoda za potpisivanje zahteva se koristila u desetinama drugih web sajtova koji su dozvoljavali integraciju sa drugim aplikacijama

Android Stagefright ranjivost

- **Stagefright** je naziv biblioteke koju Android operativni sistem koristi za baratanje multimedijalnim fajlovima, uglavnom audio i video
- U svakom trenutku u kome je potrebno učitati `.mp4`, `.mp3` ili neki drugi tip audio/video fajla, *Stagefright* je biblioteka koja će biti pozvana
- Konkretna ranjivost je uočena 2015. i broj ranjivih uređaja na tržištu je procenjen na **milijardu**!
- Ranjivost omogućava odavanje informacija i izvršavanje proizvoljnog koda u trenutku kada Android sistem parsuje `.mp4` fajl koji je pažljivo osmišljen da iskoristi ovu ranjivost
  - Ovo se može desiti čak i kada korisnik sa svog Android uređaja prosto poseti zlonamernu web stranu koja onda pokušava da reprodukuje zlonamerni video fajl

Android Stagefright ranjivost — kod

```
1  const void* data;
2  size_t size = 0;
3  uint64_t chunk_size = 0;
4  // ...  data, size i chunk_size se ucitavaju iz MP4 fajla ...
5
6  uint8_t* buffer = new (std::nothrow) uint8_t[size + chunk_size];
7  if (buffer == NULL) {
8      return ERROR_MALFORMED;
9  }
10
11 if (size > 0) {
12     mempcy(buffer, data, size);
13 }
```

- Ponovo — vidite li grešku?
- Značajno je teža za uočiti u odnosu na prethodne primere

Android Stagefright ranjivost — kod

```
1  const void* data;
2  size_t size = 0;
3  uint64_t chunk_size = 0;
4  // ...  data, size i chunk_size se ucitavaju iz MP4 fajla ...
5
6  uint8_t* buffer = new (std::nothrow) uint8_t[size + chunk_size];
7  if (buffer == NULL) {
8      return ERROR_MALFORMED;
9  }
10
11 if (size > 0) {
12     mempcy(buffer, data, size);
13 }
```

- Ponovo — vidite li grešku?
- Značajno je teža za uočiti u odnosu na prethodne primere

Android Stagefright ranjivost — analiza

- `chunk_size` i `size`, kao i ceo `data` bafer, se čitaju iz MP4 fajla, dakle pod "našom" su kontrolom
- Prilikom alokacije niza dužine `size + chunk_size` može doći do prekoračenja zbog sabiranja, ako odaberemo prave vrednosti za `size` i `chunk_size` parametre
- Prekoračenje `unsigned` celih brojeva u C++-u: vraćanje u 0!
- Ako je `size` ogroman broj koji prilikom sabiranja sa `chunk_size` daje mali broj zbog prekoračenja…
- …Stagefright će alocirati mali niz ali će u njega upisati `size` bajtova, zahvaljujući pozivu `memcpy`

Android Stagefright ranjivost — analiza (2)

- Pisanje van granica niza koji smo alocirali je vrlo često *game over* za softver, jer se van granica tog niza nalazi sasvim nepovezana *heap* memorija koja je alocirana za nešto sasvim drugo
- Ovakav tip ranjivosti se naziva **prekoračenje bafera**, ili **buffer overflow**
- Sama eksploatacija nije ovde gotova, jer ne znamo šta se nalazi u memorijskim lokacijama u koje će Stagefright greškom upisati bajtove iz fajla
- Za uspešnu eksploataciju potrebno je, kroz vrlo pametne manipulacije nad MP4 fajlom, naterati Stagefright kod koji ga parsuje da dovede *heap* na neko predvidivo stanje
- Takođe se moraju zaobići druge bezbednosne mere koje Android poseduje kako bi sprečio eskploataciju ranjivosti

Android Stagefright ranjivost — analiza (3)

- Nažalost, nemamo vremena za opisivanje ovih tehnika s obzirom na to da su vrlo kompleksne, i potrebno je mnogo veštine kako bi se osmislile, pa i objasnile

- Tim koji je otkrio ovu ranjivost je uspeo da napravi maliciozan web sajt koji, nakon što ga poseti korisnik sa ranjivim uređajem, servira maliciozan MP4 fajl i dobija neograničen pristup celom uređaju

JDRUŠTVO

- *Podsetnik*: **JDRUŠTVO** je zadatak na početnom testu za Nedelju informatike, u kome je cilj eksploatisati ranjivosti u web aplikaciji u cilju dobijanja pristupa poverljivim informacijama
- Interesovanje za zadatak je bilo sjajno i svi učesnici su se maksimalno potrudili — hvala!
- Sajt *Z Biotech* u zadatku je odličan primer nekompetentno implementiranog komada softvera punog ranjivosti
- Nećemo se mnogo zadržavati na ovome, s obzirom na to da su kompletne instrukcije za eksploataciju dostupne na sajtu Nedelje informatike, u okviru rešenja za prijemni test

JDRUŠTVO — kod

```php
<?php
// ...
$id = $_GET['id']; // Ovo dolazi direktno od korisnika!
$res = $mysqli->query("SELECT * FROM users WHERE id = " . $id);
// ...
```

- Gde je greška?

JDRUŠTVO — kod

```php
<?php
// ...
$id = $_GET['id']; // Ovo dolazi direktno od korisnika!
$res = $mysqli->query("SELECT * FROM users WHERE id = " . $id);
// ...
```

- Gde je greška?

JDRUŠTVO — analiza

- Naravno: `id` parametar, koji korisnik može da promeni u bilo šta, se direktno ubacuje u SQL upit
- Ne proverava se da li je taj parametar broj (kao što bi trebalo da bude)
- Zbog ovoga, `id` parametar se lako može manipulisati da se na bilo koji način nadoveže na SQL upit u koji se ubacuje i u okviru njega izvrši proizvoljan SQL kod
- Za zainteresovane: mnogo više detalja u rešenjima testa, dostupnim na sajtu!

Još primera

- Primeri ranjivosti o kojima smo govorili nisu odabrani na osnovu kritičnosti same ranjivosti, već zanimljivosti i lakoći shvatanja
- Ovo nisu ni izbliza najgori softverski propusti koji su se desili u novijoj istoriji računarstva
- Nepoznat je broj ranjivosti (a i njihove cene) kojima se trguje na crnim berzama, između hakera sa zaista zlim namerama, a kojih stvaraoci softvera nisu svesni
- Čak su i ranjivosti koje su odgovorno prijavljene i rešene pod velikim rizikom, jer u periodu između izbacivanja zvaničnog bezbednosnog apdejta i trenutka kada ga korisnici softvera instaliraju postoji velika mogućnost da zlonamerni akteri iskoriste priliku za eksploataciju

Još primera — nekompletna lista

- *Heartbleed* je ranjivost koja je dozvolila napadačima da pročitaju sadržaje memorije skoro svih servera koji podržavaju HTTPS protokol
- *Shellshock* je omogućio neverovatno lako izvršavanje proizvoljnog koda na velikom broju servera koji koriste određenu konfiguraciju instaliranog softvera
- *Linux kernel* `vmsplice()` *ranjivost* je omogućila bilo kom kodu koji se izvršava na Linux sistemima sa ranjivim verzijama kernela da dobije potpun administratorski pristup celom sistemu
- *Adobe Flash* je svoju samrt proteklih godina obeležio tolikim brojem nekompetentnih ranjivosti da su i najzagriženiji teoretičari zavere priznali da je malo verovatno da su one uvođene namerno, u tajnosti

Kako da izbegnem ovo u svom kodu?

- Nikako.
- Ljudi pišu kod, i ljudi će uvek praviti nemarne greške
- Nijedan sistem nije apsolutno siguran
- Ali, sa vedrije strane, bar postoje načini da se verovatnoća za ovako brutalnim bezbednosnim propustima smanji

Testiranje

- Automatizovano testiranje softvera je u 2016. godini apsolutno neophodno — nema izgovora
- *Unit testing*, *integration testing*, *system testing*
- Postoji veliki broj knjiga koje govore o važnosti postojanja testova za sve netrivijalne kompenente sistema, kao i sa savetima koji se tiču pisanja korisnijih i boljih testova
- Čak se i cele metodologije razvoja softvera (konkretno, *test-driven development* ili TDD) okreću oko koncepta automatizovanog testiranja

Testiranje — kako bi pomoglo?

- `goto fail` bag u OS X-u o kome smo govorili bi bio uhvaćen čak i rudimentarnim testiranjem sporne metode
- Ispravno testiranje bi uhvatilo propust u kodu za *Z Biotech* sajt
- Android Stagefright ranjivost je mogla da se ustanovi ranije korišćenjem *fuzz* testiranja, koje u poslednje vreme uzima maha
  - U *fuzz* testiranju, specijalan alat testira softver na nasumičnim i skoro-nasumičnim (ali struktuiranim) ulazom, u nadi da će stohastički pronaći situacije u kojima dolazi do nepredviđenog ponašanja

Odbrana u dubinu (defense in depth)

- Mantra: **oslanjanje na jedan sloj bezbednosti je izuzetno glupo!**
- *Idealan* sistem sa n nivoa bezbednosti je i dalje siguran čak i kada je probijena svaka moguća kombinacija od $n - 1$ nivoa
- Idealan sistem ne postoji, ali ka njemu treba težiti
- Jedan nivo bezbednosti može biti kompromitovan iz velikog broja razloga — kompromitovati taj nivo i jedan ispod je već dosta teže
- *Primer* — server konfigurisan na potpuno siguran način i dalje treba biti iza restriktivnog *firewall*-a
- Srodan koncept je *princip minimalne privilegije*: svaka komponenta treba da ima minimalne moguće privilegije koje su joj neophodne da obavlja svoj posao, i nikada više od toga

Odbrana u dubinu — kako bi pomogla?

- Ukoliko bi *Z Biotech* sajt držao poverljive kompanijske dokumente u odvojenoj bazi sa posebnom pristupnom šifrom i na posebnoj mreži, ranjivost ne bi bila eksploatabilna u tom konkretnom slučaju

- Android Stagefright ranjivost je dobar primer (nažalost neuspele) odabrane u dubinu: kako bi se došlo do eksploatacije koja može da načini štetu korisniku i ozbiljno ugrozi njegovu bezbednost, bilo je potrebno jako mnogo veštine i poznavanja čitavog sistema kako bi se zaobišle dodatne predostrožnosti ugrađene u Android
  - Za neki drugi tip ranjivosti, ovi slojevi bezbednosti bi možda i potpuno sprečili eksploataciju

Nikada bezbednost kroz tajnovitost

- **Bezbednost kroz tajnovitost** je oslanjanje na tajnovitost implementacije i dizajna nekog sistema kao primarnu bezbednosnu meru
- Ovo je **neopisivo loš princip**, jer sistem može imati ranjivosti koje samo čistom srećom nisu još otkrivene od strane potencijalnih zlonamernih napadača!
- *Kirhofsov princip* (Kerchkoffs' principle): koncizno rečeno, *sistem mora biti bezbedan čak i kada sve informacije o njemu, sem kriptografskih ključeva, padnu u ruke neprijatelja.*
- Bezbednost kroz tajnovitost se oslanja na pretpostavku da nijedan napadač neće biti dovoljno motivisan da se dovoljno posveti razumevanju tajnog sistema kako bi otkrio njegove mane. Ovo samo znači da taj napadač zasad nije naišao, i da uvek postoji opasna šansa da hoće.

Nikada sopstveni kripto

- **Nikada ne implementirajte svoju kriptografiju** (never roll your own crypto) je "nepisano pravilo" koje je motivisano činjenicom da je izuzetno teško implementirati kriptografske operacije, ili osmisliti kriptografske protokole, na bezbedan način
- Neočekivane vrste napada su moguće, a fundamentalne kriptografske funkcije su najslabija karika sistema — ukoliko nisu bezbedne, bezbednost celog sistema pada u vodu
- Postoje već osmišljene kriptografske funkcije i protokoli koji su preživeli godine pokušaja razbijanja od strane akademika u oblasti kriptografije
- Postoje već testirane implementacije tih kriptografskih funkcija i protokola od strane kompetentnih inženjera, koje će gotovo uvek biti sigurnije od onog što laik može napraviti

Bezbedniji programski jezici

- C i C++ su jezici koji ne garantuju memorijsku bezbednost i sadrže mnogo “nagaznih mina” u formi nedefinisanog ponašanja i iznenađujuće semantike
- Iako moguće, jako je teško pisati bezbedan C/C++ (posebno C) kod
- Na sreću, postoji mnogo manje razloga za korišćenjem ovih jezika u današnje vreme, posebno u osetljivim delovima arhitekture
- Većina softverskih ranjivosti koje su danas velika pretnja za bezbednost osoba, kompanija pa čak i država, se ne bi ni dogodila da su za softver korišćeni bezbedniji jezici

Bezbedniji programski jezici — alternative

- Ne postoji ultimativni programski jezik — sve zavisi od konkretnog problema, uslova i preference autora

- Ipak, evo nekih predloga:
  - **Rust** za *low-level* sistemsko i serversko programiranje
  - **C#** za mnoge zadatke za koje se danas koristi C++
  - **Scala**, **Haskell** ili **Python** za zadatke višeg nivoa koji zahtevaju veću ekspresivnu moć na nivou jezika

Specifičnost softverske bezbednosti

- Za skoro sve stvari unutar razvoja softvera, moguće je na ličnom iskustvu naučiti “pravi način”
  - Tačnost programa se može objektivno testirati
  - Performanse programa se mogu objektivno testirati
  - Pristupi dizajnu softvera se mogu, takođe, testirati
- Kada je bezbednost u pitanju, situacija je drugačija
- Bezbednost se ne može lako objektivno testirati — nemoguće je kvantifikovati bezbednost sistema ili dokazati da je sistem bezbedan
- Zbog ovoga, “nepisana pravila” i “dobre prakse” je bitnije usvojiti i poštovati bez obzira na pređašnje iskustvo

Odgovornost softverskog inženjerstva

- Kako softver zauzima bitniju ulogu u našem životu, oblast softverskog inženjerstva postaje sve bliža tradicionalnom inženjerstvu
- Odgovornost prema korisnicima sistema, pažljivo postupanje sa podacima i osvrt na moguće rizike
- *Move fast and break things* i slični slogani, iako fundamentalno neotuđivi svakom motivisanom programeru, se moraju sagledati i iz do sada slabo akcentovane perspektive odgovornosti inženjera
- Previdi, propusti ili prosta nepažnja od strane softverskih inženjera mogu imati dalekosežne posledice po ljude koji od njihovog softvera zavise
- *Do your thing, have fun*, ali imajte i ovo na umu!

Kraj

- Hvala na pažnji!
- Hvala na interesovanju za JDRUŠTVO!
- Pitanja?