Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/21BE12Sulicenko

ANALIZA BEZBEDNOSNIH NAPADA U NODEJS EKOSISTEMU

CYBERSECURITY ATTACK ANALYSIS IN THE NODEJS ECOSYSTEM

Boris Šuličenko, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIČKO I RAČUNARSKO INŽENJERSTVO

Kratak sadržaj – *U ovom radu opisano je kako rade NodeJS i NPM i koje su im mane. Opisani su neki od poznatih napada na NPM ekosistem i NodeJS kao platformu za izvršavanje JavaScript koda na serveru. Uz opise kako napadi mogu da se izvrše prikazane su i moguće mitigacije.*

Ključne reči: *NodeJS, NPM, izvršavanje napada, mehanizmi odbrane.*

Abstract – *This paper describes how NodeJS and NPM work and what their disadvantages are. Some of the most popular attacks on NPM and NodeJS as a platform for executing JavaScript code on servers are described here. Also, with descriptions of how attacks are executed, this paper shows how users can mitigate those attacks.*

Keywords: *NodeJS, NPM, executing attacks, defence mechanisms.*

1. UVOD

Veliki faktor u razvoju uspešnih *web* aplikacija igra uloga bezbednosti i otpornosti na različite napade. Samim tim autori aplikacija moraju biti upoznati sa tehnikama kako se napadi izvršavaju, ali i kako se od njih mogu odbraniti.

JavaScript je slabo tipizirani i interpretirani programski jezik visokog nivoa koji je 1995. godine kreirao Brendan Eich za potrebe *Netscape web* pretraživača. Zbog svoje jednostavnosti i efikasnosti stekao je veliku popularnost među programerima koji se bave razvojem *frontend* delova aplikacija. Iz želje da se *JavaScript* izvršava van *web* pretraživača je 2009. godine nastao *NodeJS*.

Zbog razvoja celog ekosistema napadačima je *NodeJS* postao zanimljiv koji zbog toga ulažu svoje resurse i vreme u istraživanje i smišljanje novih tehnika za probijanje bezbednosti aplikacija zasnovanih na ovoj tehnologiji od kojih su neke prikazane u ovom radu.

2. INTERNI RAD NODEJS-a

NodeJS je okruženje za izvršavanje *JavaScript* koda (eng. *runtime environment*) na serveru koji je 2009. godine kreirao Ryan Dahl [1].
Node Package Manager (skraćeno NPM) [2] je alat i registar za upravljanje zavisnostima u projektima koji se razvijaju korišćenjem *JavaScript* jezika.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nikola Luburić.

Da bismo shvatili zašto su neki od napada na *NodeJS* uspešni i kako oni zapravo rade i utiču na njega potrebno je da razumemo od kojih komponenti se *NodeJS* sastoji i kako one komuniciraju međusobno.
Tri glavne komponente su:

- *JavaScript* runtime – V8 *engine* [3]
- Petlja događaja (eng. *event loop*) – *libuv* biblioteka [4]
- Standardna biblioteka funkcija.

2.1. JavaScript runtime

NodeJS spada u okruženja koja rade na principu petlji događaja pa se sa visoke tačke gledišta skoro sva izvršavanja dešavaju u jednoj niti procesora.

Kao *runtime*, *NodeJS* koristi V8 *engine*. V8 je softver otvorenog koda i visokih performansi napisan u *C++*-u, a za potrebe razvoja *NodeJS*-a je adaptiran da može da radi izvan pretraživača. Njegova uloga je da vrši parsiranje, interpretiranje, kompajliranje i optimizaciju *JavaScript* koda koji korisnik želi da izvrši.

2.2. Petlja događaja

Prevedeni kod se izvršava u petlji događaja. U te svrhe *NodeJS* koristi biblioteku *libuv* koja je napisana u *C++*-u i služi kao apstrakcija nad operativnim sistemom. Pored toga, odgovorna je za asinhrono izvršavanje koda, pristup *file system*-u, mreži, itd.

Petlja događaja nije ništa drugo nego beskonačna *while* petlja koja unutar sebe izvršava kod u nekoliko faza. Gledano sa visokog nivoa apstrakcije ceo proces izvršavanja programa u *NodeJS*-u se izvršava na jednoj niti i samim tim se postavlja pitanje kako je moguće uslužiti više zahteva istovremeno?

Odgovor je u ne-blokirajućim I/O procesima ili nitima. To znači da proces koji je započeo neku akciju neće zauzeti 100% resursa i time onemogućiti korišćenje aplikacije drugim korisnicima. Proces ili nit će samo uspostaviti komunikaciju sa korisnikom i zatraženu akciju će smestiti u red za izvršavanje. Uz samu akciju korisnik treba da preda i funkciju povratnog poziva (eng. *callback function*) koju proces poziva nakon izvršenja zadatka. Ona se može posmatrati kao odgovor procesa samom korisniku.

Zadatak koji je dobijen od strane korisnika se prebacuje na *worker* nit koja se izvršava u pozadini. Nit će javiti glavnoj niti da je akcija izvršena i radnju obaveštavanja korisnika će prebaciti na nju.

Za ceo ovaj proces izvršavanja zadataka je zadužen *libuv* jer on sadrži *thread pool* i samim tim omogućava *multi-threading* način rada u *NodeJS*-u. *Libuv* je takođe zadužen i za upravljanje *worker* nitima.

2.3. Standardna biblioteka funkcija

Treća komponenta *NodeJS*-a je biblioteka standardnih funkcija koja pruža gotova rešenja za neke od najčešćih problema sa kojima programeri mogu da se susretnu prilikom razvoja aplikacija. Na primer rad sa događajima, *file system*-om, HTTP protokolom, procesima i nitima.

3. PREGLED PROBLEMA

Većina paketa je napisana od strane ljudi koji se bave razvojem softvera otvorenog koda iza kojih ne stoji velika kompanija koja može da ulaže sredstva u bezbednost tih paketa. Samim tim javlja se veliki broj ranjivosti koje napadači mogu da iskoriste.

3.1. Napadi na NPM ekosistem

Krhkost NPM ekosistema i zavisnosti među paketima se može videti na primeru kada je u martu 2016. godine autor paketa *left-pad* obrisao paket sa repozitorijuma što je dovelo do toga da veliki broj drugih paketa postane nedostupan, tj. neupotrebljiv dok se ne nađe i ne implementira zamena za taj paket [5].

U julu 2018. godine desilo da su kredencijali jednog od autora *eslint-scope* paketa omogućili napadaču da objavi malicioznu verziju paketa koja je slala lokalne fajlove na udaljeni server.

NPM ekosistem ima veliki broj slabih tačaka, a neke od njih su [6]:

- Velika površina napada - prosečan NPM paket koristi oko 80 drugih paketa u stablu zavisnosti i ima oko 40 programera koji ga održavaju ili imaju pristup njemu.
- Popularni paketi direktno i indirektno utiču na više od 100000 paketa.
- Postoje nalozi korisnika koji imaju pristup i do 100000 paketa, na direktan i indirektan način.
- Skoro 40% paketa zavise od drugih paketa gde je poznata bar jedna ranjivost.

Modeli pretnje NPM ekosistema su kreiranje malicioznih paketa, eksploatisanje koda koji se više ne održava, preuzimanje kredencijala za upravljanje paketom, itd.

Neke od osnovnih stvari koje se mogu učiniti da se spreče napadi su podizanje svesti ljudi koji učestvuju u izradi paketa o stvarima koje im se mogu desiti i kako da se zaštite od njih. NPM *cli* pruža alat *audit* [7] koji služi za skeniranje paketa u projektu naspram baze podataka ranjivih paketa. Ukoliko se pronađe neki ranjiv paket obavestiće korisnika o riziku pri njegovom korišćenju.
Mitigacije koje NPM kao organizacija može da ponudi su provera paketa pre objavljivanja, obuka i provera autora pre dobijanja dozvole za objavljivanje paketa. Ove mitigacije su teške za realizaciju jer zahtevaju veliku količinu resursa.

3.2. Primeri napada

Napad na paket *event-stream*, desio se tako što je napadač dobio *maintainer* pristup kodu uz pomoć socijalnog inženjeringa nakon čega je dodao malicioznu zavisnost u ovaj projekat - *flatmap-stream* paket koji je bez znanja krajnjih korisnika krao *Bitcoin*-e. Ovaj maliciozni kod je opstao dva ipo meseca u produkciji i bio je skinut skoro osam miliona puta [8].

Početkom 2022. godine *mainteiner* paketa *colors* i *FakerJS* je objavio verzije u kojima je namerno narušio rad paketa. Kao rezultat došlo je do *Denial of Service* napada na bilo kom serveru koji koristi *NodeJS* i ove pakete. U tom momentu *colors* je imao 20 miliona skidanja nedeljno i koristio se u 4 miliona projekata, a *FakerJS* oko 2 miliona skidanja nedeljno. Mitigacije za ovakvu vrstu napada mogu biti: vraćanje paketa da koristi prethodnu, stabilnu verziju, korišćenje alternativnih paketa sa kojima se može ostvariti isti učinak i pre korišćenja paketa i njegovog nadograđivanja pregledati da li nova verzija donosi neke štetne promene [9].

Ne moraju svi napadi na pakete da budu destruktivne prirode. Primer ovakvog napada je korišćenje paketa *peacenotwar* koji je namenjen u svrhe protesta protiv rata u Ukrajini i koji na standardni izlaz ispisuje aktuelne vesti o ratu. Najveći uticaj se desio na korisnike *vue-cli* paketa i *Unity game engine*-a gde je preko indirektnih zavisnosti u *node-ipc* paket ubačen *peacenotwar* [10].

4. ANALIZA NAPADA

Ovo poglavlje sadrži detaljne analize napada na NPM i *NodeJS*, kao i tehnike i metode koje se mogu primeniti za njihovu realizaciju. Pored opisa kako se napad izvršava prikazane su i tehnike uz pomoć kojih se korisnici mogu odbraniti ili preventirati sam napad.

4.1. *Typosquatting* napad

Typosquatting napad radi na principu tako što napadač za ime paketa odabere nešto što veoma liči na ime već nekog poznatog paketa i postavi ga na NPM. Oslanja se na principe socijalnog inženjeringa i na to da korisnik neće izvršiti proveru paketa pre njegovog instaliranja, već će se osloniti na svoje sećanje da dobije ime paketa. *Typosquatting* napad se takođe oslanja na to da korisnik napravi grešku u kucanju prilikom instaliranja paketa gde može greškom da instalira maliciozni paket koji je namerno tako nazvan [11].

Jedan od popularnijih primera ovog napada se desio na paket *crossenv* [12]. Ovo ime paketa je slično sa *cross-env* koji je predstavljao metu napada. Maliciozni paket je obuhvatio sve funkcionalnosti originalnog paketa, tako da se razlika u korišćenju nije mogla ni primetiti. Jedina razlika je bila ta što je maliciozni paket slao podatke o varijablama okruženja na udaljeni server tj. napadaču.

Neke od mitigacija koje se mogu iskoristiti da bi se zaštitili su korišćenje *–ignore-scripts* argumenta komandne linije za sprečavanje izvršavanja proizvoljnih komandi, poverljive informacije skladištiti van varijabli okruženja i koristiti alate poput *npq*-a [13] za skeniranje zavisnosti pre instalacije.

4.2. *Supply chain* i *Dependency confusion* napad

Paketi koji se koriste kao zavisnosti se nalaze u centralnom registru koji može biti javni i privatni. Za krajnje korisnike taj registar predstavlja *supply chain*.
Supply chain attack je napad gde napadač pokušava da kompromituje izvorni kod paketa sa ciljem da ubaci maliciozni kod direktno u mesto gde se taj paket koristi ili indirektno kroz stablo zavisnosti.

Istraživač bezbednosti i etički haker Alex Birsan je napravio prototip napada na *supply chain* pod imenom *dependency confusion* tako što je uspeo da ubaci maliciozni kod u centralne repozitorijume kompanija poput *Apple*-a, *Microsoft*-a, *Uber*-a i *Yelp*-a. [14]

Dependency confusion napad za cilj ima da napravi zabunu prilikom instaliranja stabla zavisnosti na mašinama gde se projekat izvršava tako što se oslanja na greške da osoblje firme neće dobro podesiti mašinu da preuzima pakete sa privatnog registra. U takvom slučaju će se, podrazumevano, dobaviti maliciozni paket koji nosi isti naziv i koji je okačen na javni registar.

Mitigacije koje se mogu primeniti su korišćenje imenskog prostora (eng. *scoped namespace*) na NPM-u, kreirati ispravne konfiguracije, isključiti *install* i *uninstall* komande radi sprečavanja izvršavanja proizvoljnih skripti, dodatno istražiti pakete, sprečiti izloženost ranjivih informacija itd.

4.3. *Backdoor* napad

Backdoor je vrsta napada gde korisnici mogu da zaobiđu bezbednosne mere i dobiju pristup resursima kojima ne bi smeli da pristupaju na računaru, mreži ili aplikaciji, a u nekim slučajevima i pristup celim uređajima [15].

Svaki *backdoor* se sastoji iz dva dela: sam maliciozni kod koji se izvršava na zaraženoj mašini i komunikacioni kanal koji omogućava napadaču da ima pristup mašini. U *NodeJS*-u se može napraviti jednostavan backdoor korišćenjem ugrađenenog modula *child_process* i funkcije *exec* koja omogućava izvršavanje proizvoljnih komandi na mašini i *ExpressJS* biblioteke za pravljenje HTTP servera.

Načini na koje možemo izbeći *backdoor* napade su da koristimo poznate i dobro održavane pakete i da koristimo alate poput *npq* i *snyk* koji nas mogu upozoriti ako će instalacija paketa ubaciti neki maliciozni kod.

4.4. *Denial of Service* napad

Denial of Service (skraćeno - DoS) napadi za cilj imaju da onesposobe određeni servis na neki vremenski period ili trajno tako što će na njega slati veliku količinu zahteva i na taj način mu preopteretiti dostupne resurse ili naneti veliku novčanu štetu zbog upotrebe tih resursa.

Neki od principa zaštite su da se koriste asinhrone metode gde god je to moguće, da se ne čuva previše podataka u memoriji u jednom trenutku i da se ranjive metode, koje mogu da prouzrokuju DoS napad, dodatno zaštite [16].

4.5. *Prototype pollution* napad

Po definiciji *prototype pollution* napad je *denial of service* i *injection* napad gde napadač dobija kontrolu nad vrednostima *property*-a nekog objekta. Ovo omogućava napadaču da kontroliše samu biznis logiku aplikacije. Posledice ovog napada mogu biti nedostupnost servera, a u najgorem slučaju može da dođe i do *remote code execution* napada [17].

Da bismo bolje razumeli zašto je ovaj napad efektivan potrebno je objasniti kako prototipovi u *JavaScript*-u rade. Prototipovi predstavljaju mehanizam uz pomoć kojih *JavaScript* objekti mogu da nasleđuju osobine drugih objekata. Svaki *JavaScript* objekat u sebi ima ugrađeno jedno polje pod nazivom *__proto__* koje sadrži polja „roditeljskog“ objekta. Prilikom pristupanja polju objekta pretraga se prvo vrši u datom objektu, ako polje nije pronađeno u njemu pretraga se nastavlja u prototipu, zatim u prototipu prototipa i tako dalje dok se ne dođe do korenskog prototipa. Za izvršavanje ovog napada bitna je informacija da je *JavaScript* dinamički jezik što znači da je tokom izvršavanja programa moguće vršiti izmene nad prototipovima svih objekata [18].

U sklopu istraživanja za ovaj rad napravljena je demonstracija koja prikazuje kako može da se napadne *web* server koji koristi *ExpressJS* biblioteku u pozadini. Prikazana su dva načina napada. Prvi način koristi propust u funkciji *merge* iz popularne *lodash* [19] biblioteke i dovodi do *denial of service*-a. Treba napomenuti da je ovaj propust popravljen u novijim verzijama biblioteke, ali može biti dostupan u drugim bibliotekama koje implementiraju isti algoritam jer ovakva funkcija predstavlja najčešći način za realizaciju *prototype pollution*-a. Pored spajanja običnih atributa ukoliko ova funkcija nije ispravno implementirana izvršiće i spajanje prototipova. Ukoliko se u aplikaciji radi *merge* operacija nad objektom koji predstavlja korisnički unos i proizvoljnog objekta iz aplikacije može doći do napada na server. Na listingu 4.1 prikazan je pojednostavljeni *endpoint* i primer malicioznog podatka koji se šalje na njega.

```
// Primer ExpressJS endpoint-a sa POST metodom
app.post("/", (req, res) => {
  merge({}, req.body)
  res.send("OK")
})

// Maliciozni korisnički ulaz
const payload = {
  "__proto__": {
    "toString": 123,
    "valueOf": "It works"
  }
}
```

Listing 4.1 *ExpressJS endpoint* sa POST metodom i malicioznim korisničkim ulazom

Pošto se metode *toString* i *valueOf* često implicitno koriste prilikom izvršavanja koda njihovo ponašanje je sada izmenjeno i samim tim dolazi do nedostupnosti servera zbog učestalih grešaka na najjednostavnijim *JavaScript* operacijama.

Druga vrsta napada za rezultat ima izvršavanje proizvoljnih komandi na napadnutom računaru. Do njegovog izvršavanja može doći ukoliko se negde u kodu nepažljivo koristi *child_process* modul i *exec* funkcija. Primer ovakvog koda se može videti na listingu 4.2 gde se dinamički, bez prethodnih provera, izvršavaju proizvoljne komande. Takođe je prikazan i maliciozni korisnički ulaz koji uz prethodno korišćenje *merge* funkcije može da doda malicioznu komandu na listu komandi za izvršavanje.

```
// Primer ExpressJS endpoint-a sa POST metodom
app.post("/", (req, res) => {
  const commands = {
    script1: "ls",
    script2: "touch test.js"
```

```
  }

  for (const key in commands) {
    childProcess.execSync(commands[key], { stdio: "inherit" })
  }

  res.send("OK")
})

// Maliciozni korisnički ulaz
const payload = {
  "__proto__": {
    "my command": "echo You are attacked!"
  }
}
```

Listing 4.2 Izvršavanje komandi korišćenjem *exec* funkcije iz *child_process* modula i maliciozni korisnički ulaz

Neki od načina odbrane su da se koriste bezbedne verzije *merge* metoda, da se objekti kreiraju bez prototipa sa *Object.create(null)* metodom i korišćenje *Object.freeze()* metode koja sprečava menjanje atributa objekata.

5. ZAKLJUČAK

Prilikom kreiranja aplikacija bezbednost treba da se shvati ozbiljno i da se primenjuju odgovarajuće tehnike zaštite jer u suprotnom posledice napada mogu biti ozbiljne.

Da bi napadač mogao da osmisli novi napad, a i autor aplikacije da se zaštiti od potencijalnih napada bitno je da se razume interni rad tehnologije u kojoj je aplikacija realizovana. U ovom radu je opisano kako *NodeJS* interno funkcioniše i opisane su njegove slabe tačke kroz primere napada.

Fokus je bio na opisivanju nekih od poznatih napada koji su se desili u prethodnim godinama na NPM ekosistem i sam *NodeJS* kao platformu za izvršavanje *JavaScript* koda na serveru. Pored samih napada, dati su i mogući načini zaštite od napada sa kojima bi svaki autor *web* aplikacija trebao biti upoznat. Pored toga, prikazane su i ideje koje NPM kao organizacija može da primeni za bolju zaštitu.

6. LITERATURA

[1] Node.js [Na mreži] [Citirano 30 9 2022.] https://en.wikipedia.org/wiki/Node.js

[2] NPM [Na mreži] [Citirano 30 9 2022.] https://www.npmjs.com/

[3] V8 [Na mreži] [Citirano 30 9 2022.] https://v8.dev/

[4] Libuv [Na mreži] [Citirano 30 9 2022.] https://libuv.org/

[5] Serdar Yegulalp, *How one yanked JavaScript package wreaked havoc* [Na mreži] [Citirano 30 9 2022.] https://www.infoworld.com/article/3047177/how-one-yanked-javascript-package-wreaked-havoc.html

[6] Markus Zimmermann, Cristian-Alexandru Staicu, *Small World with High Risks: A Study of Security Threats in the npm Ecosystem*. 2019

[7] NPM Audit [Na mreži] [Citirano 30 9 2022.] https://docs.npmjs.com/cli/v8/commands/npm-audit

[8] Danny Grander, *Malicious code found in npm package event-stream downloaded 8 million times in the past 2.5 months* [Na mreži] [Citirano 30 9 2022.] https://snyk.io/blog/malicious-code-found-in-npm-package-event-stream/,

[9] Liran Tal, Assaf Ben Josef, *Open source maintainer pulls the plug on npm packages colors and faker, now what?* [Na mreži] [Citirano 30 9 2022.] https://snyk.io/blog/open-source-npm-packages-colors-faker/

[10] Liran Tal, *Alert: peacenotwar module sabotages npm developers in the node-ipc package to protest the invasion of Ukraine* [Na mreži] [Citirano 30 9 2022.] https://snyk.io/blog/peacenotwar-malicious-npm-node-ipc-package-vulnerability/

[11] Liran Tal, *What is typosquatting and how typosquatting attacks are responsible for malicious modules in npm* [Na mreži] [Citirano 30 9 2022.] https://snyk.io/blog/typosquatting-attacks/

[12] Snyk – crossenv [Na mreži] [Citirano 30 9 2022.] https://security.snyk.io/package/npm/crossenv

[13] Github – npq [Na mreži] [Citirano 30 9 2022.] https://github.com/lirantal/npq,

[14] Alex Birsan, *Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies* [Na mreži] [Citirano 30 9 2022.] https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

[15] Ulises Gascón, *What is a backdoor? Let's build one with Node.js* [Na mreži] [Citirano 30 9 2022.] https://snyk.io/blog/what-is-a-backdoor/

[16] Karl Düüna, *Secure Your Node.js Web Application, Keep Attackers Out and Users Happy*

[17] Prototype pollution [Na mreži] [Citirano 30 9 2022.] https://learn.snyk.io/lessons/prototype-pollution/javascript/#pgwwpvrchiwtb

[18] Object prototypes [Na mreži] [Citirano 30 9 2022.] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object_prototypes

[19] Lodash [Na mreži] [Citirano 30 9 2022.] https://www.npmjs.com/package/lodash

Kratka biografija:

Boris Šuličenko rođen je 22.8.1997. godine u Novom Sadu, Republika Srbija. Godine 2016. upisao je Fakultet tehničkih nauka u Novom Sadu smer Softversko inženjerstvo i informacione tehnologije. 2020. godine je diplomirao na osnovnim akademskim studijama i iste godine upisuje master akademske studije na istoimenom smeru.