SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINARSKI RAD

WEBASSEMBLY

Emanuel Njegovec

Voditelj:

Zagreb, travanj, 2025.

WebAssembly

Emanuel Njegovec

Sažetak

Unesite sažetak na hrvatskom.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Ključne riječi: prva ključna riječ; druga ključna riječ; treća ključna riječ

Sadržaj

1. Uvod u WebAssembly

WebAssembly je jedan od najnovijih jezika za razvoj aplikacija na web-u. Predstavio ga je 2017. tim inženjera vodećih svjetskih web-preglednika kao što su Google Chrome, Microsoft Edge, Mozilla Firefox i Safari s ciljem stvaranja(?) jezika koji bi omogućio razvoj bržih, manjih, sigurnijih i energetski efikasnijih web-aplikacija u odnosu na trenutni industrijski standard, Javascript. WebAssembly je jezik niske razine sličan asembleru, kompaktnog i prenosivog bytecode-a. S obzirom da je sintaksom nalik strojnom jeziku, WebAssembly kod je rezultat prevođenja drugih, viših jezika kao što su C/C++, Rust ili Typescript koristeći kompajlere kao što su Emscripten ili Rustc. Ovo omogućava developerima prebacivanje postojećih web-aplikacija direktno u WebAssembly. Kao i Javascript, namijenjen je izvršavanju na klijentskoj strani u web-pregledniku. Prijašnji pokušaji za razvojem bržeg i efikasnijeg jezika za prevođenje i izvođenje u web okruženju (Microsoft ActiveX, Google Native Client, asm.js) nisu postigli rezultate kakvi se očekuju od strojnih jezika.

Za razliku of Javascript-a koji se parsira i kompajlira/prevodi tek u trenutku pokretanja, WebAssembly se unaprijed kompilira iz viših jezika u skup visoko optimiziranih Wasm binarnih datoteka koje se zatim izvršavaju. WebAssembly je dizajniran na način da podržava streaming i paralelno dekodiranje i kompilaciju što omogućava web-preglednicima procesiranje i izvršavanje koda prije nego dospije cijeli modul te tako smanjuje vrijeme učitavanja web-stranice.

Logički gledano, WebAssembly modul je odvojen u nekoliko sekcija: import sekcija koja deklarira funkcije i druge resurse iz host okruženja, export sekcija u kojoj su popisane funkcije i entiteti koji se izvoze host okruženju, podatkovna sekcija za inicijalizaciju memorijskog prostora. Glavna memorija je velik linearan niz bajtova koji u potpunosti odvojen od izvršnog stoga. WebAssembly radi unutar virtualnog stroja baziranog

na stogu, a funkcije rade s podacima na implicitnom stogu operanada. Kontrola toka je strukturirana koristeći konstrukte iz viših jezika kao što su loop, if/else za razliku od jednostavnih skokova.

Kako bi komunicirao sa svojim izvršnim okruženjem (IO, network access), WebAssembly ovisi o svojem host okruženju. Unutar web-preglednika ovo se najčešće izvodi koristeći neki Javascript engine kao što su V8 ili SpiderMonkey. Ovo zahtijeva određenu razinu JS koda za instanciranje i povezivanje wasm modula. WebAssembly binarni kod je moguće i prevesti u tekstualni format koristeći WebAssembly Binary Tool.

Prenosivost i efikasnost WebAssembly-a su doveli do drugih primjena, neke od kojih su poslužiteljske aplikacije (Node.js), nezavisni runtime-ovi te čak i ugradbeni sustavi. WebAssembly System Interface (WASI) je razvijen radi uspostave interakcije direktno s operacijskim sustavima. Obzirom da se wasm moduli izvršavaju unutar sandbox okruženja i imaju ograničen pristup resursima host sustava, cilj WASI-a je pružiti standardizirani skup API-a (I/O, networking) na razini sustava koji se mogu uvesti u WebAssembly modulima i izvršavati u raznim okruženjima operacijskih sustava na siguran i prenosiv način.

2. Primjena u stvarnom svijetu

Deset godina nakon što je prvi puta predstavljen, WebAssembly je svoju uporabu našao u raznim područjima na web-u i izvan, prvenstveno za optimizaciju resursno intenzivnih procesa, migraciju starijih nativnih aplikacija u web-aplikacije te omogućavanje drugim jezicima osim Javascripta izvršavanje na web-u. Njegova performantnost je razlog zašto se sve više koristi u zahtjevnim web-aplikacijama poput aplikacija za interaktivnu 3D vizualizaciju, audio i video obradu, video igara, alata za kompresiju podataka, procesiranje teksta i prirodnog jezika te PDF preglednicima. Velike tvrtke poput Google-a, eBay-a i Nortona koriste kako bi ubrzale svoje proizvode i usluge na web-u. Google je 2019. godine uspio prebaciti Earth, koji je do tada bio zasebna aplikacija, na sve web-preglednike koji podržavaju WebAssembly. Figma je jedan popularan alat za UI/UX dizajn čiji je web-editor inicijalno bio izgrađen u C++-u i kompajliran u asm.js. Nakon prebacivanja sa asm.js-a na Wasm, performanse Figme su se poboljšale tri puta, a najveća razlika je primjetna u vremenu učitavanja dokumenata u pregledniku. AutoCAD je program koji je stariji od web-a i sadrži veliku količinu C++ koda što znači da bi prebacivanje na web rezultiralo iznimno sporim programom. WebAssembly je ipak omogućio migraciju tako velikog i kompleksnog projekta u web-preglednik.

Osim unutar web-preglednika, WebAssembly je našao primjenu i u razvoju aplikacija na poslužiteljskoj strani i aplikacija u oblaku, mobilnih i desktop aplikacija te čak kod baza podataka. Cloudflare je svom alatu Workers dodao podršku za Wasm kako bi omogućio razvoj aplikacija u raznim jezicima. Notion je alat za produktivnost kojem su vrijeme navigacije u pregledniku njegovi developeri uspjeli ubrzat za preko 30% korištenjem WebAssembly implementacije SQLite3 baze podataka. Područje Interneta stvari (IoT) i mikrokontroleri predstavljaju još jednu potencijalnu primjenu WebAssembly-a kao apstrakcijski sloj niske razine za razvoj performantnih, malih i prenosivih programa.

3. Analiza performansi

WebAssembly je danas sve aktualniji i koristi se raznim web-aplikacijama te ga podržavaju svi veći i relevantni web-preglednici, no postavlja se pitanje koliko stvarno ubrzanje on pruža u odnosu na Javascript. Do danas je provedeno tek nekolicina istraživanja o performansama WebAssembly-a i one pružaju razne rezultate i zaključke. U originalnom članku koji predstavlja WebAssembly po prvi puta iznose se tvrdnje o njegovoj brzini i efikasnosti. Navodi se da C program preveden u Wasm je 34% brži u Google Chrome-u u odnosu na taj isti program preveden u Javascript. Također navode da mnogi benchmarkovi pokazuju da je brzina Wasm koda unutar 10% nativne brzine izvršavanja. Veličina Wasm binarnih modula je 62% manja od veličine asm.js koda te 82% manja od nativnog x86 koda.

De Macedo et al. u radu On the Runtime and Energy Performance of WebAssembly iz 2021. iznose svoje rezultate istraživanja o brzini izvođenja WebAssembly koda i njegovu energetsku intenzivnost. Njihova metodologija se sastojala od korištenja 10 programa za benchmark-anje napisanih u C-u i prevedenih u Wasm i Javascript. Koristili su tri različite veličine ulaznih podataka te mjerili potrošnju energije na procesoru te vrijeme izvršavanja. Svako mjerenje provedeno je 20 puta radi konzistentnosti rezultata. Postignuti rezultati pokazuju da Wasm jest energetski efikasniji i brži od Javascripta, no da su te razlike male, posebno za veće količine ulaznih podataka. Zaključuju da Wasm ima potencijala daleko premašiti Javascript u performansama s obzirom da je tek u relativno ranim fazama razvoja.

Nisu sva istraživanja u performanse WebAssembly-a pokazala poboljšanja u odnosu na Javascript. U radu iz 2019. godine po imenu Mind the Gap: Analyzing the Performance of WebAssembly vs. Native Code, Jangda et al., uvidjevši nedostatke prijašnjih istraživanja performansi Wasm-a, izrađuju ekstenziju BROWSIX-WASM koja omogućava

testiranje WebAssembly-a sa zahtjevnijim programima nego prije. BROWSIX-WASM je alat koji omogućava izvršavanje Unix programa kompajliranih u WebAssembly direktno u web-pregledniku. Nakon provođenja detaljnih testiranja i analiza rezultati pokazuju da je WebAssembly kod 1.43 puta sporiji od nativnog koda za Firefox i 1.92 puta za Chrome. Također kao zaključak iznose neke potencijalne uzroke ovog usporenja.

Energetska efikasnost je jedna mjera koja je posebno bitno kod prenosivih uređaja, između ostalog i pametnih mobitela. Iz tog razloga u svom radu Comparing the Energy Efficiency of WebAssembly and Javascript in Web Applications on Android Mobile Devices iz 2022. godine, van Hasselt et al. provode istraživanje s namjerom otkrivanja troše li Wasm web-aplikacije zaista manje energije na mobilnim uređajima. Postavili su dva istraživačka pitanja: kako korištenje Wasm-a utječe ne potrošnju energije te koliko odabir web-preglednika na Android-u utječe na energetsku potrošnju Wasm-a i Javascripta. Eksperiment je postavljen tako da se na mobilnom uređaju po 30 puta učitavaju i pokreću web-stranice (Wasm ili Javascript) prvo na jednom pregledniku (Chrome), a zatim na drugom (Firefox). Rezultati pokazuju da je prosječna energetska potrošnja WebAssembly-a (31.9 J) na oba preglednika više nego duplo manja od potrošnje Javascripta (81.7 J). Također pronašli su da je energetska potrošnja Chrome-a veća od potrošnje Firefox-a, no da je razlika veća kad se koristi Javascript nego kad se koristi WebAssembly. Kao zaključak predlažu da developeri web-aplikacija funkcionalnosti koje se cijelo vrijeme vrte u pozadini implementiraju koristeći WebAssembly.

IoT (internet stvari) je jedno područje primjene gdje potencijali WebAssembly-a mogu doći do velikog izražaja. Upravo iz tog razloga Oliveira et al. u radu Analysis of WebAssembly as a Strategy to Improve Javascript Performance on IoT Environments istražuju kakve učinke primjena Wasm-a ima na tipičnom uređaju u tom području, Raspberry Pi.

Wang u svom radu Empowering Web Applications with WebAssembly: Are We There Yet? postavlja pitanje kako web-preglednici optimiziraju izvršavanje WebAssembly-a u odnosu na Javascript i kako to utječe na performanse.

4. Sigurnosna razmatranja

WebAssembly je danas još vrlo nov jezik te je s toga bitno postaviti pitanje kakav je sa gledišta sigurnosti i potencijalnih ranjivosti. Wasm je izgrađen djelomično s ciljem povećane sigurnosti te stoga neke od njegovih sigurnosnih značajki uključuju: sandbox okruženje koje izolira Wasm runtime od host okruženja i osigurava da moduli moraju koristiti određeni API za dohvaćanje vanjskih resursa, posebno odvojeni memorijski prostor koji onemogućava programima i modulima koruptiranje izvršnog okruženja, skakanje na nedopuštene memorijske lokacije ili izvođenje bilo kakvog neodređenog ponašanja.

Unatoč ovim značajkama, Hilbig et al. i Lehmann et al. u svojim radovima proveli sigurnosnu analizu velikog skupa Wasm binarnih datoteka iz raznih izvora poput repozitorija koda, menadžera paketa i web-stranica s ciljem utvrđivanja njihovih ranjivosti. Njihova istraživanju ukazuje na nekoliko aspekata koji predstavljaju potencijalnu ranjivost:

- korištenje područja linearne memorije, tzv. neupravljani stog, za spremanje neprimitivnih podataka. Taj stog je moguće iskoristiti za napade preljeva stoga kako bi se pristupilo vanjskim podacima. Njihovo istraživanje je pokazalo da 65% Wasm binarnih datoteka i u prosjeku 44% sadržanih funkcija koristi taj neupravljani stog.

- korištenje potencijalno nesigurnih alokatora memorije. S obzirom na nisku razinu memorijske organizacije WebAssembly-a, kako bi se izbjegla fragmentacija i ponovo iskorištavanje memorijskog prostora dealociranih objekata, u uporabi su česti vanjski alokatori umjesto alokatora koje pružaju npr. Emscripten, Clang i Rust kompajleri. 38% binarnih datoteka koristi neki vanjski memorijski alokator.

- WebAssembly moduli mogu interagirati sa vanjskim svijetom koristeću funkcije uvezene iz host okruženja. Ako je neki Wasm binarni kod kompromitiran preko

npr. prijašnje dvije ranjivosti, moguće je izvršiti maliciozni kod na samom sustavu koji izvršava taj kod preko uvezenih API poziva. Analizirajući uvezene funkcionalnosti u Wasm binarnom kodu poput funkcija za izvršavanje koda, mrežni pristup, čitanje i pisanje iz datoteka, interakcija s DOM-om i dinamičko povezivanje, otkriveno je da njih oko 21% koristi neku od navednih funkcija.

- iako WebAssembly ima strukturiranu kontrolu toka programa, indirektni pozivi funkcija, odnosno pozivi funkcija čije adrese nisu poznate u trenutku kompajliranja, mogu biti meta manipulacije gdje napadač može preusmjeriti poziv na funkciju u istoj klasi ekvivalencije funkcija istog tipa.

Također, jedna moguća ranjivost proizlazi iz inherentnih ranjivosti izvornog jezika i koda iz kojeg je Wasm kompajliran. Preljevi međuspremnika u memorijski nesigurnim jezicima kao što su C i C++ mogu biti izvor napada, npr. XSS (Cross site scripting) zbog prepisivanja podataka koji ne bi trebali biti dostupni novim podacima. Takvi memorijski nesigurni jezici čine dvije trećine izvornih jezika iz kojih su Wasm binarne datoteke kompajlirane.

Zbog svoje efikasnosti u odnosu na Javascript, WebAssembly se u jednom trenutku našao kao čest alat za tzv. cryptojacking, iskorištavanje resursa korisnika neke web-stranice za rudarenje kriptovaluta bez njihovog znanja i dopuštenja. Godinu dana nakon pojave WebAssembly-a, učestalost cryptojacking-a je porasla za preko 400% dok je godinu kasnije procijenjeno da pola web-stranica koje koriste WebAssembly ga upravo koriste u tu svrhu.

Kako bi se pokušalo zaštiti od gore navedenih ranjivosti, razni istraživači su predložili tehnike i metode za analizu Wasm binarnog koda. Dijele se na statičku analizu koda bez njegovog izvršavanja, dinamička analiza ponašanja tijekom izvršavanja koda te hibridna analiza koja kombinira prijašnje metode. Zahvaljujući njima omogućeno je npr. otkrivanje web-stranica koje rade cryptojacking i time je učestalost tog napada smanjeno na manje od 1%.

5. Zaključak

Iz opisanih analiza moguće je doći do nekoliko zaključaka o trenutnom stanju WebAssembly-a i istraživanjima o njemu. WebAssembly je stupio na scenu kao iznimno bitna tehnologija, inicijalno napravljena s ciljem poboljšanja performansi web-aplikacija kroz prenosivo, efikasno i sigurno runtime okruženje za druge jezike osim do tad standardnog Javascript-a. Istraživanja su generalno pokazala da u usporedbi sa Javascript-om uspijeva postići veću brzinu izvođenja, ali to uvelike ovisi o načinu testiranja i web-pregledniku. Jedno područje gdje Wasm daleko nadmašuje čisti Javascript je u potrošnji energije što ga čini savršenim za prenosive uređaje. Osim toga njegove performanse su dovele do primjene u raznim intenzivnijim aplikacijama te većim projektima koje je bilo potrebno prebaciti na web-okruženje. Nekoliko članaka iznosi da iz gledišta sigurnosti, WebAssembly ima još dosta prostora za napredak. Iako nudi razne sigurnosne prednosti, WebAssembly i dalje može lako pokupiti memorijske ranjivosti izvornih jezika iz kojih je kompajliran, a i sam sadrži određene ranjivosti koje bi napadači mogli iskoristiti za manipulaciju sustava koji izvršava Wasm kod. Ova istraživanja su ipak dovela do razvoja raznih metoda i alata za sigurnosnu analizu Wasm-a te su s toga neke prijetnje kao cryptojacking drastično smanjene.

Od svojih skromnih početaka, WebAssembly se razvija u zreli sustav zahvaljujući kompajlerima kao što su Emscripten, AssemblyScript i Clang koji omogućuju prevođenje raznih jezika, neki od kojih su C, C++, Rust, Go i Typescript, u kompaktan i efikasan strojni kod. Dostupnost sve većih WebAssembly binarnih datoteka koje se koriste u stvarnom svijetu dovodi do detaljnijih i opsežnijih istraživanja u karakteristike Wasm-a što je ključno u njegovom daljnjem razvoju kao temeljna tehnologija za aplikacije na web-u i izvan.

Literatura

[1] A. Hilbig, D. Lehmann, i M. Pradel, “An Empirical Study of Real-World WebAssembly Binaries: Security, Languages, Use Cases”, u *Proceedings of the Web Conference 2021*. Ljubljana Slovenia: ACM, travanj 2021., str. 2696–2708. https://doi.org/10.1145/3442381.3450138