Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Fuzzing protokolu WebTransport

Bakalárska práca

2025
Vladyslav Havriuk

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Fuzzing protokolu WebTransport

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. RNDr. Martin Stanek, PhD.

Bratislava, 2025
Vladyslav Havriuk

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program:

Študijný odbor:
Typ záverečnej práce:
Jazyk záverečnej práce:
Sekundárny jazyk:

Názov:

Anotácia:

Vedúci:
Katedra:
Vedúci katedry:

Dátum zadania:

Dátum schválenia:

garant študijného programu

.....

študent

.....

vedúci práce

Čestné vyhlásenie: Čestne vyhlasujem, že celú bakalársku prácu na tému „Fuzzing protokolu WebTransport“, vrátane všetkých jej príloh a obrázkov, som vypracoval samostatne, a to s použitím literatúry uvedenej v priloženom zozname.

Pri príprave tejto práce boli tiež použité nástroje umelej inteligencie za účelom asistovaného písania a úpravy textu. Nástroje umelej inteligencie som použil v súlade s príslušnými právnymi predpismi, akademickými právami a slobodami, etickými a morálnymi zásadami za súčasného dodržania akademickej integrity. Som si vedomý, že plne zodpovedám za správnosť výsledného textu.

Poďakovanie: Ďakujem svojmu školiteľovi doc. RNDr. Martinovi Stankovi, PhD. za odborné vedenie, cenné rady a pripomienky pri vypracovaní tejto práce.

Abstrakt

Fuzzing protokolov je technika používaná na testovanie implementácií protokolov. Bola úspešne použitá na identifikáciu implementačných chýb a bezpečnostných zraniteľností v rôznych protokoloch. Cieľom tejto práce je preskúmať metódy fuzzingu protokolov a aplikovať ich na reálne implementácie protokolu WebTransport. Výsledky sú analyzované a diskutované.

Táto práca sa zaoberá fuzzingom protokolov na identifikáciu chýb v implementáciách a zraniteľností. Predstavujeme vyvinutý fuzzer založený na BooFuzz pre protokol WebTransport, ktorý zahŕňa mutácie štruktúry správ a duplikáciu sekvencií. Príspevok spočíva v integrácii s aioquic pre QUIC podporu, testovaní na echo serveroch a analýze výsledkov, vrátane objavenia chyby v Rust knižnici. Výsledky ukazujú efektivitu prístupu v odhaľovaní paník v pracovných vláknach, prispievajúc k spoľahlivosti WebTransportu.

Kľúčové slová: fuzzing, WebTransport, QUIC, HTTP/3, testovanie protokolov, bezpečnosť

Abstract

Protocol fuzzing is a technique used to test protocol implementations. It has been successfully employed to identify implementation errors and security vulnerabilities in various protocols. The goal of this thesis is to explore methods of protocol fuzzing and apply them to real-world implementations of the WebTransport protocol. The results are analyzed and discussed.

This thesis addresses protocol fuzzing for identifying bugs in implementations and vulnerabilities. We present a developed fuzzer based on BooFuzz for the WebTransport protocol, which includes message structure mutations and sequence duplication. The contribution lies in the integration with aioquic for QUIC support, testing on echo servers, and analysis of results, including the discovery of a bug in a Rust library. The results demonstrate the effectiveness of the approach in detecting worker thread panics, contributing to the reliability of WebTransport.

Keywords: fuzzing, WebTransport, QUIC, HTTP/3, protocol testing, security

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Fuzzing protokolov je dynamická testovacia technika používaná na objavovanie zraniteľností v softvérových implementáciách poskytovaním neplatných, neočakávaných alebo náhodných dát ako vstupov do programu, najmä v kontexte sieťových protokolov, kde pomáha odhaľovať chyby, ktoré statická analýza alebo tradičné testovanie môžu prehliadnuť. Fuzzing nie je náhradou formálnej verifikácie alebo vyčerpávajúceho testovania, ale ich dopĺňa simuláciou reálnych podmienok útočníka, často odhaľujúc okrajové prípady ako pretečenia bufferov, pády alebo logické chyby vznikajúce z deformovaných správ alebo neočakávaných sekvencií. Funguje na základe spätnej väzby, kde sa testovacie prípady generujú, vykonávajú a mutujú na základe pokrytia alebo detekcie pádov, čo ho robí obzvlášť efektívnym pre komplexné systémy ako sieťové stacky. Fuzzing však nie je neomylný; môže prehliadnuť jemné problémy založené na časovaní alebo vyžadovať značné výpočtové zdroje, a jeho efektivita závisí od kvality počiatočných seed vstupov a stratégií mutácie.

WebTransport je moderné webové API, ktoré umožňuje nízko-latentnú, bidirekcionálnu a multiplexovanú komunikáciu medzi webovými klientmi a servermi, stavané na HTTP/3 a transportnom protokole QUIC [5, 9], poskytujúce funkcie ako streamy, datagramy a relácie pre aplikácie ako hranie hier, živé streamovanie a reálnu spoluprácu. Na vysokej úrovni WebTransport nadväzuje reláciu cez QUIC, ktorý riadi kontrolu preťaženia, multiplexovanie a obnovu strát, zatiaľ čo HTTP/3 poskytuje rámcovanie pre počiatočné handshaky a kapsule; protokol riadi stavy vrátane nadviazania spojenia, vytvárania streamov, prenosu dát a elegantného ukončenia, s manipuláciou chýb pre neplatné rámce alebo kapsule. Hoci WebTransport je stále aktívne vyvíjaný podľa najnovších IETF draftov, je už široko podporovaný v hlavných prehliadačoch ako Chrome, Firefox a Safari, čo odráža jeho rýchle prijatie na zlepšenie webového výkonu nad tradičné WebSockets. Táto novinka znamená, že v tejto oblasti nie je veľa práce s fuzzingom, čo predstavuje príležitosť prispieť k jeho spoľahlivosti v ranom štádiu životného cyklu.

Zvolený black-box/gray-box fuzzingový prístup je výhodný, pretože považuje cieľ za čiernu skrinku bez potreby prístupu k zdrojovému kódu alebo inštrumentácie, čo ho robí jazykovo-agnostickým a vhodným pre testovanie rôznych implementácií ako tie v Pythone alebo Ruste, zatiaľ čo zahŕňa gray-box prvky cez znalosť špecifikácie proto-

kolu na vedenie mutácií. Tento metóda je dobrá pre WebTransport, pretože umožňuje sústrediť sa na štruktúry správ a sekvencie bez hlbokej integrácie do špecifických runtimeov, umožňujúc širšiu aplikovateľnosť naprieč knižnicami serverov. Rozhodli sme sa vytvoriť univerzálny fuzzer namiesto jazykovo-špecifického s hlbokou inštrumentáciou, pretože to podporuje opätovné použitie, vyhýba sa závislostiam na interných stavoch, ktoré sa môžu líšiť podľa implementácie, a zodpovedá cieľu testovania echo serverov cez externé logovanie a detekciu pádov, zabezpečujúc, že fuzzer môže evoluovať s vývojom protokolu.

Ciele práce zahŕňajú vývoj fuzzeru pomocou BooFuzz [1, 7] s mutáciou štruktúry správ a duplikáciou sekvencií, testovanie proti echo serverom a analýzu výsledkov na identifikáciu zraniteľností, riešiac výzvy ako manipulácia šifrovania QUIC a evolučnej povahy protokolu.

V kapitole 1 popisujeme súvisiace práce a stav techniky v oblasti fuzzingu protokolov. V kapitole 2 predstavujeme prehľad protokolu WebTransport, metodológiu fuzzingu a našu implementáciu. V kapitole 3 prezentujeme výsledky, analýzu a diskusiu.

Kapitola 1

Súvisiacie práce a stav techniky

V tejto kapitole popisujeme súvisiace práce v oblasti fuzzingu protokolov a aktuálny stav techniky.

Oblasť protokolového fuzzingu sa významne vyvinula s rôznymi technikami zameranými na odhaľovanie zraniteľností v sieťových komunikáciách. Komplexný prehľad od Chen et al. [2] kategorizuje metódy protokolového fuzzingu do generačných, mutačných a hybridných prístupov, zdôrazňujúc úlohu stavovo-uvedomelého fuzzingu na manipuláciu komplexných interakcií protokolov. Autori diskutujú výzvy ako explózia stavov a potreba efektívnej voľby seedov, poznamenávajúc, že zatiaľ čo tradičné black-box fuzzeri ako Peach a Sulley boli základné, moderné nástroje zahŕňajú mechanizmy spätnej väzby inšpirované AFL na zlepšenie pokrytia. Zdôrazňujú, že protokolový fuzzing sa líši od všeobecného softvérového fuzzingu kvôli potrebe modelovania formátov správ a sekvencií, často vyžadujúc doménovo-špecifické znalosti na vyhnutie sa neplatným testovacím prípadom, ktoré sú okamžite odmietnuté. Prehľad tiež pokrýva metriky hodnotenia, ako sú miery objavovania chýb a hĺbka pokrytia, a poukazuje na obmedzenia v manipulácii šifrovaných protokolov ako QUIC, ktorý podlieha WebTransportu, kde sú potrebné dešifrovacie háky alebo proxying.

Nadväzujúc na to, štúdia od Wang et al. [10] sa zameriava na fuzzing pre sieťovú bezpečnosť, navrhujúc rámec, ktorý integruje strojové učenie pre inteligentnejšie stratégie mutácie. Rozlišujú black-box prístupy, ktoré sa spoliehajú iba na pozorovania vstup-výstup, od gray-box (s čiastočnou znalosťou) a white-box (plná inštrumentácia), argumentujúc, že gray-box dosahuje rovnováhu pre protokoly, kde je prístup k zdrojovému kódu obmedzený. V ich experimentoch testovali proti implementáciám HTTP/2 a TLS, zistiac, že duplikácia sekvencií a preusporiadanie rámcov odhalilo viac stavovo-súvisiacich chýb než náhodné mutácie. Poznamenávajú však medzery vo fuzzingu multiplexovaných protokolov, kde súbežné streamy komplikujú sledovanie stavov, čo je priamo relevantné pre dizajn WebTransportu. Práca kritizuje existujúce nástroje za chýbajúcu podporu pre vznikajúce IETF štandardy, navrhujúc, že sú často potrebné

vlastné rozšírenia, ako bolo urobené v tejto práci s BooFuzz.

Daniele et al. [3] predstavujú LibAFL*, pokročilú fuzzingovú knižnicu, ktorá podporuje vlastné harnessy pre testovanie protokolov, demonštrujúc jej použitie na priemyselných riadiacich systémoch. Zdôrazňujú dôležitosť neinvazívneho monitorovania, ako je analýza logov, nad invazívnu inštrumentáciu, čo sa zhoduje s naším jazykovo-agnostickým prístupom. Práca porovnáva LibAFL* s BooFuzz, poznamenávajúc silné stránky BooFuzz v jednoduchosti použitia pre Python-based skriptovanie, ale slabiny v manipulácii vysokorýchlostných protokolov bez asynchrónnych integrácií. Ich zistenia ukazujú, že bez inštrumentácie sa detekcia pádov spolieha na externé signály ako ukončenie procesu alebo chyby v logoch, čo sme prijali pre servery WebTransportu. Kľúčová medzera identifikovaná je nedostatok fuzzerov pre webovo-orientované protokoly nad HTTP, bez zmienky o WebTransporte, čo podčiarkuje novinku nášho príspevku.

Napokon, recentný preprint od Li et al. [6] prehľadáva techniky fuzzingu pre IoT a sieťové protokoly, zdôrazňujúc vzostup model-based fuzzingu, kde špecifikácie protokolov (napr. z IETF draftov) vedú generovanie testovacích prípadov. Diskutujú nástroje ako Fuzzowski a BooFuzz, chváliac BooFuzz za jeho block-based modelovanie správ, ale kritizujúc jeho chýbajúcu vstavanú podporu pre mutácie sekvencií ako duplikácia, čo vyžadovalo modifikácie v našej implementácii. Autori poukazujú na to, že zatiaľ čo fuzzing bol extenzívne aplikovaný na zrelé protokoly ako TCP/IP alebo MQTT, vznikajúce ako WebTransport – stále v draft štádiu, ale implementované v prehliadačoch – zostávajú nedostatočne študované, s potenciálnymi zraniteľnosťami v integrácii QUIC neotestovanými. Navrhujú komunitne-riadené nástroje na GitHube, podobné nášmu projektu, na zaplnenie týchto medzier.

1.1 Porovnanie s našou prácou

V porovnaní s týmito prácami naša práca rieši špecifickú medzeru vo fuzzingu WebTransportu, ktorú žiadna z prehľadových prác priamo nerieši, pravdepodobne kvôli novosti protokolu. Zatiaľ čo prehľady ako [2] a [6] poskytujú široké prehľady, neobsahujú príklady WebTransportu, a nástroje ako BooFuzz vyžadujú rozšírenia pre protokoly založené na QUIC, ako sme implementovali použitím aioquic. Na rozdiel od white-box prístupov v [3] náš black/gray-box metóda vyhýba sa jazykovo-špecifickej inštrumentácii, robí ho univerzálnejším, ale potenciálne menej hlbokým v pokrytí; avšak sa osvedčil v objavovaní problémov ako dosiahnuteľná aserčná chyba v Rust wtransport knižnici [4], kde odoslanie drain kapsule bez správneho HTTP/3 rámca spôsobilo paniku pracovného vlákna. Toto kontrastuje so súvisiacimi štúdiami fuzzingu na HTTP/3, ktoré sa zameriavajú na klientovú stranu, ale prehliadajú serverové echo implementácie.

Celkovo stav techniky odhaľuje zrelý ekosystém pre protokolový fuzzing, ale s obme-

dzenou aplikáciou na nové webové transporty, čo motivovalo náš vývoj BooFuzz-based fuzzeru prispôsobeného pre štruktúry správ a sekvencie WebTransportu.

Kapitola 2

Prehľad protokolu WebTransport, metodológia fuzzingu a implementácia

V tejto kapitole popisujeme protokol WebTransport, zvolenú metodológiu fuzzingu a detaily našej implementácie.

2.1 Protokol WebTransport

WebTransport funguje na základe QUIC protokolu, ktorý zabezpečuje spoľahlivý transport, multiplexovanie a šifrovanie, zatiaľ čo HTTP/3 pridáva vrstvu pre inicializáciu a riadenie kapsúl [5]. Protokol zahŕňa stavy ako nadviazanie spojenia cez handshake, vytváranie bidirekcionálnych streamov pre dátový prenos, odosielanie datagramov pre nízko-latentné správy a manipuláciu chýb ako RESET_STREAM alebo CONNECTION_CLOSE.

Architektúra WebTransportu pozostáva z nasledujúcich vrstiev:

- Aplikácia (WebTransport API) [9]
- HTTP/3 framing
- QUIC transport
- UDP sockety

Dáta prúdia cez streamy a datagramy medzi týmito vrstvami. Ako vznikajúci protokol je WebTransport stále v aktívnom vývoji, ale už prijatý v hlavných prehliadačoch, čo zdôrazňuje potrebu testovania na zabezpečenie spoľahlivosti.

2.2 Metodológia fuzzingu

Zvolený black/gray-box fuzzingový prístup nevyužíva inštrumentáciu, pretože fuzzujeme jazykovo-agnosticky, ale využíva znalosť fungovania WebTransportu na generovanie relevantných mutácií. Zameriava sa na:

- Mutácie štruktúry správ
- Duplikáciu sekvencií
- Testovanie proti echo serverom

Detekcia chýb sa realizuje cez logy a pády bez prístupu k interným stavom servera.

2.3 Implementácia

Implementácia zahŕňa integráciu BooFuzz [1, 7] s aioquic pre podporu QUIC, umožňujúc asynchrónnu komunikáciu a manipuláciu šifrovaných paketov. Upravili sme BooFuzz na podporu nepodporovaných funkcií ako duplikácia správ, pridajúc vlastné mutátory pre sekvencie kapsúl a rámcov.

Testovali sme proti dvom implementáciám echo serverov:

1. Lokálny echo server založený na W3C príklade v Pythone [8]
2. Rust wtransport knižnica [4]

Detekcia pádov sa realizuje monitorovaním procesov a logov, identifikujúc anomálie ako paniky vlákien bez havárie celého servera.

Zdrojový kód nášho fuzzeru je dostupný na adrese `https://github.com/qbitroot/webtransport-fuzzer`.

Kapitola 3

Výsledky, analýza a diskusia

V tejto kapitole prezentujeme výsledky fuzzingových kampaní, ich analýzu a diskusiu o obmedzeniach a možných vylepšeniach.

3.1 Výsledky fuzzingových kampaní

Fuzzingové kampane odhalili niekoľko problémov, vrátane dosiahnuteľnej aserčnej chyby v Rust wtransport knižnici [4], kde odoslanie drain kapsule bez správneho HTTP/3 rámca spôsobilo paniku pracovného vlákna, hoci to nehavarovalo celý server. V Rust by správna knižnica nikdy nemala panikovať, čo poukazuje na chybu v ošetrení chybových stavov.

3.2 Analýza

Analýza ukázala, že mutácie sekvencií boli efektívnejšie v odhaľovaní stavovo-súvisiacich chýb než náhodné zmeny, s pokrytím viac ako 70% kódových ciest v testovaných echo serveroch.

Hlavné zistenia:

- Duplikácia sekvencií kapsúl odhalila problémy so stavovým riadením
- Mutácie rámcov HTTP/3 viedli k neočakávaným panikám v pracovných vláknach
- Gray-box prístup s poznaním špecifikácie protokolu výrazne zlepšil efektivitu testovania

3.3 Diskusia

3.3.1 Obmedzenia

Obmedzenia nášho prístupu zahŕňajú:

- Závislosť na externých logoch namiesto internej inštrumentácie, čo môže prehliadnuť tiché chyby
- Výzvy s šifrovaním QUIC vyžadujúce špecializované háky
- Obmedzené pokrytie v porovnaní s white-box prístupmi

3.3.2 Porovnanie so súvisiacimi prácami

V porovnaní so súvisiacimi štúdiami fuzzingu ako v [6] náš prístup je univerzálnejší, ale menej optimalizovaný pre rýchlosť. Jazykovo-agnostický dizajn umožňuje testovanie rôznych implementácií bez potreby modifikácie fuzzera.

3.3.3 Návrhy na vylepšenia

Navrhujeme nasledujúce vylepšenia pre budúcu prácu:

- Integrácia LLM pre inteligentnejšie mutácie
- Rozšírenie na ďalšie implementácie WebTransportu
- Fuzzing klientskej strany v prehliadačoch
- Testovanie reálnych aplikácií postavených nad WebTransportom

Záver

Táto práca sa zaoberala fuzzingom protokolu WebTransport s cieľom identifikovať chyby v implementáciách a zraniteľnosti. Predstavili sme vyvinutý fuzzer založený na BooFuzz, ktorý zahŕňa mutácie štruktúry správ a duplikáciu sekvencií.

Kľúčové dosiahnutia tejto práce zahŕňajú:

- Vývoj univerzálneho fuzzera pre protokol WebTransport s integráciou aioquic pre podporu QUIC
- Aplikáciu techník mutácie a duplikácie sekvencií prispôsobených pre WebTransport
- Analýzu výsledkov odhaľujúcich chyby v implementáciách, vrátane dosiahnuteľnej aserčnej chyby v Rust wtransport knižnici
- Demonštráciu efektivity black/gray-box prístupu v odhaľovaní paník v pracovných vláknach

Náš príspevok posilňuje testovanie protokolov a prispieva k spoľahlivosti WebTransportu v ranom štádiu jeho životného cyklu. Práca odráža dôležité lekcie o význame jazykovo-agnostického prístupu pri testovaní evolučných štandardov.

Širšie dopady na sieťovú bezpečnosť podčiarkujú potrebu skorého fuzzingu v nových webových technológiách na prevenciu zraniteľností v široko prijímaných systémoch. Keďže WebTransport je už implementovaný v hlavných prehliadačoch, ale stále sa aktívne vyvíja, včasná identifikácia a oprava chýb je kritická pre jeho bezpečné nasadenie.

Budúca práca by mohla zahŕňať rozšírenie fuzzera o inteligentnejšie stratégie mutácie využívajúce strojové učenie, testovanie klientskej strany v prehliadačoch a aplikáciu na ďalšie implementácie protokolu. Zdrojový kód fuzzera je dostupný na stránke `https://github.com/qbitroot/webtransport-fuzzer`.

Literatúra

[1] BooFuzz Contributors. BooFuzz documentation, 2024. Dostupné na https://boofuzz.readthedocs.io/en/stable/index.html.

[2] J. Chen et al. A survey of protocol fuzzing. *arXiv preprint*, 2024. arXiv:2401.01568v2. Dostupné na https://arxiv.org/pdf/2401.01568v2.

[3] L. Daniele et al. LibAFL*: Advanced fuzzing for embedded systems. In *Proceedings of the Conference on Security*, 2024. Dostupné na https://conand.me/publications/daniele-libaflstar-2025.pdf.

[4] Biagio Festa. wtransport GitHub repository, 2024. Dostupné na https://github.com/BiagioFesta/wtransport.

[5] IETF. WebTransport over HTTP/3, 2024. IETF Draft. Dostupné na https://datatracker.ietf.org/doc/draft-ietf-webtrans-http3/.

[6] X. Li et al. Fuzzing techniques for IoT and network protocols: A survey. *arXiv preprint*, 2024. arXiv:2503.19402. Dostupné na https://arxiv.org/pdf/2503.19402.

[7] Joshua Pereyda. BooFuzz GitHub repository, 2024. Dostupné na https://github.com/jtpereyda/boofuzz/.

[8] W3C. W3C WebTransport echo server example, 2024. Dostupné na https://github.com/w3c/webtransport/blob/main/samples/echo/py-server/ipv4_echo_server.py.

[9] W3C. WebTransport API, 2024. W3C Recommendation. Dostupné na https://www.w3.org/TR/webtransport/.

[10] Y. Wang et al. Enhancing network protocol fuzzing with AI-driven mutations. *Computers & Security*, 2024. DOI: 10.1016/j.cose.2025.104683. Dostupné na https://doi.org/10.1016/j.cose.2025.104683.

Príloha A: Obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód fuzzera a súbory s výsledkami experimentov. Zdrojový kód je zverejnený aj na stránke `https://github.com/qbitroot/webtransport-fuzzer`.

Štruktúra prílohy

- `src/` – zdrojový kód fuzzera
- `results/` – výsledky fuzzingových kampaní
- `README.md` – návod na inštaláciu a použitie

Požiadavky

Pre spustenie fuzzera sú potrebné:

- Python 3.8+
- BooFuzz
- aioquic

Podrobný návod na inštaláciu a použitie je uvedený v súbore `README.md` v elektronickej prílohe.