RANLJIVOSTI PRI IMPLEMENTACIJI STANDARDA WEBAUTHN

Sven Ulčar, Matevž Pesek
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
su23975@student.uni-lj.si, matevz.pesek@fri.uni-lj.si

Izvleček

Standard WebAuthn omogoča overjanje brez gesel z asimetrično kriptografijo in odpravlja več znanih ranljivosti tradicionalnega overjanja z gesli. Kljub formalno dokazani varnosti samega standarda, praktične implementacije pogosto vsebujejo strežniške napake, ki omogočijo obhod overjanja. Članek obravnava ranljivost nepravilne vezave poverilnice na uporabniški račun, ki napadalcu omogoča prevzem tujega računa z lastno veljavno poverilnico. Pregled štirih ranljivosti CVE iz let 2025 in 2026 pokaže, da se strežniške napake ponavljajo pri vezavi poverilnic, obravnavi kriptografskih izzivov in validaciji izvora. Razvita je bila namerno ranljiva demonstracijska aplikacija in njena popravljena različica: napad obide overjanje na ranljivi izvedbi, na popravljeni pa je zavrnjen. Primerjava štirih razširjenih knjižnic pokaže, da tri vezavo poverilnic prepuščajo aplikaciji, ena pa jo zahteva v vmesniku. Iz analize izhajajo smernice za razvijalce, med njimi stroga vezava poverilnice na uporabnika. Varnost sistemov WebAuthn ni samodejna posledica izbire knjižnice ali jezika, temveč zahteva eksplicitno strežniško preverjanje pripadnosti poverilnic.

Ključne besede: WebAuthn, FIDO2, vezava poverilnic, ranljivost overjanja, demonstracijska implementacija

VULNERABILITIES IN WEBAUTHN STANDARD IMPLEMENTATIONS

Abstract

The WebAuthn standard enables passwordless authentication using asymmetric cryptography and addresses several well-known vulnerabilities of traditional password-based authentication. Despite the formally proven security of the standard itself, practical implementations often contain server-side flaws that allow authentication bypass. This article addresses the vulnerability of incorrect credential binding to a user account, which allows an attacker to take over another user's account using their own valid credential. A review of four CVE vulnerabilities from 2025 and 2026 shows that server-side flaws recur in credential binding, cryptographic challenge handling, and origin validation. An intentionally vulnerable demonstration application and its patched version were developed: the attack bypasses authentication on the vulnerable version and is rejected on the patched one. A comparison of four widely-used libraries shows that three leave credential binding to the application, while one requires it at the interface level. The analysis yields developer guidelines, among them strict credential-to-user binding. The security of WebAuthn-based systems is not an automatic consequence of library or language choice, but requires explicit server-side verification of credential ownership.

Keywords: WebAuthn, FIDO2, credential binding, authentication vulnerability, demonstration implementation

1 UVOD

Overjanje z gesli ostaja najpogostejši način prijave v spletne aplikacije, hkrati pa je ena najšibkejših točk sodobnih informacijskih sistemov. Gesla delujejo kot deljena skrivnost (ang. *shared secret*) med uporabnikom in ponudnikom storitve. Takšna oblika je inherentno ranljiva za širok nabor groženj: napade z zvabljanjem (ang. *phishing*), napade s surovo silo in krajo poverilnic ob uhajanju podatkov iz ogroženih (ang. *compromised*) podatkovnih zbirk. Kraja poverilnic in napadi z zvabljanjem ostajajo med najpogostejšimi vzroki za zlorabe uporabniških računov. Kot odgovor na te izzive je konzorcij FIDO Alliance v sodelovanju z organizacijo W3C razvil standard WebAuthn (ang. *Web Authentication*) [7], ki predstavlja osrednji gradnik ogrodja FIDO2 [4] in uvaja model overjanja brez gesel, ki temelji na kriptografiji javnega ključa.

Namesto deljenih skrivnosti standard WebAuthn uporablja asimetrični kriptografski par ključev. Zasebni ključ pri tem ostane na uporabnikovi strani in se nikoli ne posreduje ponudniku storitve (ang. *Relying Party*, RP), pri katerem se registrira le pripadajoči javni ključ [7]. Takšna arhitektura zagotavlja odpornost proti napadom z zvabljanjem, napadom s ponavljanjem in napadom posrednika [9]. Tehnologija Passkey, ki temelji na načelih standarda FIDO2, uporabniško izkušnjo dodatno izboljša z možnostjo sinhronizacije poverilnic med napravami [8, 9].

Uvedba takšne tehnologije sama po sebi ne zagotavlja varne implementacije. Varnostne raziskave kažejo, da implementacije v praksi pogosto zaostajajo za teoretično varnostno ravnjo standarda [2, 14, 10]. Razkritja v platformah GitLab [5] in OneUptime [11] ter v razširjenih knjižnicah za PHP [12] in Javo [13] kažejo, da te napake niso omejene na posamezno implementacijo ali jezik, temveč se ponavljajo kot strukturni vzorec strežniških logičnih napak okoli sicer kriptografsko varnega standarda. Dejanska varnost sistemov, ki temeljijo na standardu WebAuthn, je tako odvisna ne le od zanesljivosti samega standarda, temveč tudi od pravilne implementacije strežniške logike.

Članek obravnava varnostne ranljivosti, ki izhajajo iz nepravilne strežniške implementacije standarda WebAuthn. Njegov glavni namen ni odkriti novih ranljivosti, temveč pokazati, da že dokumentirane strežniške napake tega standarda izvirajo iz ponavljajočega se vzorca ter jih je mogoče izolirati, ponovljivo demonstrirati in z majhnim posegom odpraviti. Osrednji prispevek članka je trojen: (1) ranljivost nepravilne vezave poverilnic na uporabnika, ki je v obstoječih delih obravnavana teoretično [6] ali razpršeno dokumentirana v posameznih zapisih CVE, je predstavljena kot minimalna referenčna demonstracija, v kateri sta napaka in njen popravek izolirana na en sam pogoj poizvedbe; (2) ta primer je umeščen v širši vzorec treh kategorij strežniških napak, ki jih potrjujejo nedavni zapisi CVE iz let 2025 in 2026 v tehnološko neodvisnih implementacijah; (3) primerjava štirih razširjenih knjižnic WebAuthn pokaže, ali vezavo poverilnice na uporabnika prepuščajo aplikaciji ali jo zahtevajo že v svojem vmesniku, iz česar izhajajo konkretne smernice za razvijalce. Glede na obstoječe raziskave [6, 14, 10] prispevek dodaja predvsem praktično, ponovljivo ponazoritev znane vrste ranljivosti in iz nje izpeljana priporočila za razvijalce.

V poglavju 2 so predstavljeni teoretični okvir standarda WebAuthn, arhitektura ogrodja FIDO2 in obstoječe raziskave dokumentiranih ranljivosti. Poglavje 3 predstavi zasnovo demonstracije, poglavje 4 njeno izvedbo in rezultate, poglavje 5 interpretira ugotovitve in jih umesti v širši kontekst, zaključno poglavje 6 pa povzema ključno lekcijo in nakazuje smeri nadaljnjega dela.

2 TEORETIČNO OZADJE IN SORODNA DELA

2.1 Akterji in arhitektura WebAuthn

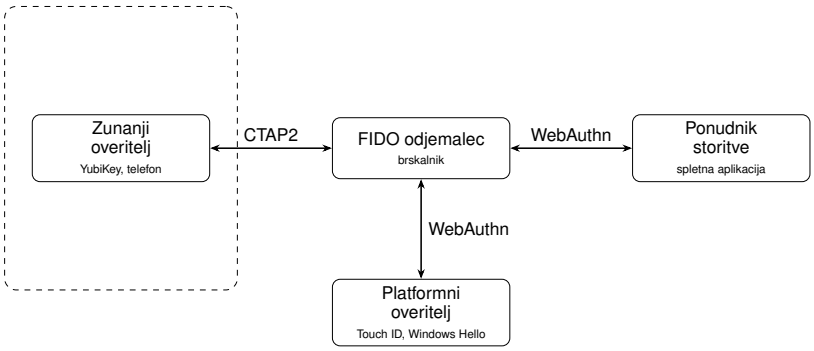
V arhitekturi WebAuthn nastopajo trije akterji: *odjemalec* (brskalnik), *overitelj* (strojna ali programska komponenta, ki hrani zasebne ključe) in *ponudnik storitve* (RP), spletna aplikacija, ki zahteva overjanje in hrani javne ključe svojih uporabnikov [7]. Vsak RP je enolično identificiran z domenskim imenom (`rpId`), na katero se vežejo poverilnice; poverilnica, registrirana za eno domeno, ne velja za drugo, kar onemogoča napade z zvabljanjem.

Ogrodje FIDO2 združuje dva komplementarna standarda: WebAuthn določa komunikacijo med odjemalcem in ponudnikom storitve [7], CTAP2 (ang. *Client to Authenticator Protocol*) pa komunikacijo med odjemalcem in zunanjim overiteljem [4]. Zunanji overitelj je lahko na primer ključ YubiKey ali telefon, platformni overitelj (npr. Touch ID ali Windows Hello) pa je skupaj z brskalnikom vgrajen v uporabnikovo napravo. Razmerje med komponentami prikazuje slika 1.

Varnost standarda WebAuthn in ogrodja FIDO2 je predmet aktivnega raziskovanja tako na teoretični kot na praktični ravni. Formalne analize, ki WebAuthn obravnavajo kot kriptografski protokol, so potrdile, da je ob ustreznih predpostavkah varen [3, 2], kar pomeni, da ranljivosti v praksi izhajajo pretežno iz implementacijskih napak. Pregled obstoječih raziskav v nadaljevanju je razdeljen na obravnavo strežniških implementacijskih ranljivosti in na ovire pri implementaciji ter uvajanju standarda v praksi.

2.2 Strežniške implementacijske ranljivosti

Dokumentirane strežniške ranljivosti standarda WebAuthn se združujejo v tri ponavljajoče se kategorije, ki jih predstavljamo v vrstnem redu koraka strežniškega preverjanja, v katerem nastopijo: napake pri validaciji izvora, pri obravnavi kriptografskih izzivov in pri vezavi poverilnic na uporabnika.



Slika 1: Arhitektura ogrodja FIDO2.

Prvo kategorijo predstavljajo napake pri validaciji izvora, ki je v standardu WebAuthn osrednja obramba pred napadi z zvabljanjem. CVE-2026-30964 [12] v knjižnici `web-auth/webauthn-lib` (PHP, do različice 5.2.4) izhaja iz tega, da knjižnica pri primerjanju vrednosti `clientDataJSON.origin` z vsakim konfiguriranim dovoljenim izvorom obe vrednosti reducira na ime gostitelja in tako tiho ignorira razlike v shemi in vratih. Poverilnica, registrirana za `https://login.example.com:8443`, je tako sprejeta tudi z izvora `https://login.example.com:9443`, kar omogoča obhod stroge politike izvora prek zmede vrat ali sheme na istem gostitelju.

Drugo kategorijo tvorijo napake pri obravnavi kriptografskih izzivov, ki naj bi zagotavljali enkratnost vsakega overjanja. CVE-2026-28787 [11] v platformi OneUptime (različice do 10.0.11) krši določilo specifikacije WebAuthn [7], ki zahteva, da strežnik generirani izziv shrani lokalno. OneUptime izziv vrne odjemalcu, ob preverjanju pa pričakovano vrednost prebere neposredno iz telesa odjemalske zahteve. Ker tako pričakovani izziv kot izziv v `clientDataJSON` izvirata iz istega prestreženega odgovora, se vedno ujemata, podpis pa ostane veljaven, saj ga je podpisal legitimen overitelj žrtve. Posledica je trajna možnost ponavljanja prestreženega podpisanega odgovora, kar dvofaktorsko zaščito popolnoma izniči.

Tretjo kategorijo predstavljajo napake, pri katerih strežnik sprejme prijavo, ne da bi vezavo poverilnice na predstavljenega uporabnika pravilno preveril. Guan et al. [6] so jo opisali kot napade ponovne vezave overitelja (ang. *authenticator rebinding attacks*), pri katerih napadalec svojo poverilnico poveže s tujim računom, kadar strežnik tega razmerja ne preverja dosledno. Yadav in Seamons [14] sta isto kategorijo opisala na odjemalski strani: napadalec, ki preko ogrožene razširitve brskalnika ali napada XSS izvaja kodo v žrtvinem brskalniku, lahko med registracijo žrtvin javni ključ zamenja z napadalčevim (ang. *misbinding attack*), kar prav tako poveže napadalčevo poverilnico z žrtvinim računom, vendar prek napada s posrednikom v brskalniku in ne prek strežniške logične napake. Dve realizaciji v produkcijskih sistemih kažeta različni obliki iste pomanjkljivosti na strežniški strani. CVE-2025-26788 [13] v strežniku StrongKey FIDO Server (Java, različice 4.10.0 do 4.15.0) izhaja iz iskanja poverilnice zgolj po `credential_id` brez vezave na `user_id`. Napadalec lahko v zaključno zahtevo vstavi lasten `credential_id` in se prijavi kot žrtev. CVE-2026-0723 [5] v platformi GitLab (Ruby) izhaja iz drugačnega vzorca: validacijska funkcija ob neuspehu (npr. ko se `id` v odgovoru overitelja ne ujema z `rawId`) sicer vrne neuspeh, a klicna koda vrnjene vrednosti ne upošteva, zato se vse nadaljnje preveritve podpisa, izziva in izvora preskočijo, strežnik pa kljub temu zaključi prijavo z uspehom. Skupna značilnost obeh primerov je, da je kriptografska verifikacija podpisa izvedena ločeno od preverjanja, ali poverilnica pripada uporabniku, ki se predstavlja. Prav ta kategorija, nepravilna strežniška vezava poverilnice na uporabnika, je predmet praktične demonstracije v osrednjem delu članka.

2.3 Ovire pri implementaciji in uvajanju

Lassak et al. [10] so na konferenci USENIX Security 2024 z intervjuji z 28 vodji informacijske varnosti ugotovili, da je kompleksnost implementacije standarda FIDO2 primarna ovira za sprejetje v praksi in hkrati vir varnostnih napak. Hölbl in Kompara [9] sta v slovenskem prostoru predstavila tehnologijo Passkey in opozorila, da specifikacija WebAuthn ne predpisuje protokola za sinhronizacijo ali varnostno kopiranje zasebnih ključev, zato to odgovornost prevzemajo ponudniki platform, kar v praksi ustvarja kompromis med uporabnostjo sinhroniziranih ključev in varnostjo ključev, vezanih na napravo.

Predstavljeni članek nadgrajuje navedena dela s poudarkom na strežniški implementacijski ranljivosti vezave poverilnic ter jo demonstrira s praktično aplikacijo in primerja z dokumentiranimi primeri iz prakse.

3 METODOLOGIJA

Za praktično demonstracijo ranljivosti vezave poverilnic je bila razvita namerno ranljiva demonstracijska spletna aplikacija[1]. Cilj demonstracije je pokazati, da lahko napadalec pridobi dostop do administratorskega računa brez posedovanja njegovih poverilnic in doseže zaščiteno administratorsko stran. Aplikacija je zasnovana kot minimalna referenčna implementacija, ki izolira eno samo napako, nepravilno vezavo poverilnic, in jo naredi ponovljivo ter preverljivo.

Izbira te ranljivosti za demonstracijo temelji na njeni dokumentirani prisotnosti v produkcijskih implementacijah (poglavje 2) in na izoliranem popravku, ki omogoča kontroliran preizkus ranljive in popravljene različice z eno samo spremenljivko.

3.1 Merila uspešnosti

Doseganje cilja presojamo po dveh vnaprej določenih merilih. Demonstracija ranljivosti je uspešna, če napadalec, ki nima administratorjevih poverilnic, z lastno veljavno poverilnico pridobi sejo uporabnika `admin` in dostop do zaščitene strani `/admin`. Merilo pravilnosti popravka je, da isti napad po eni dodatni omejitvi v iskalnem pogoju poizvedbe ne uspe več, medtem ko legitimna registracija in prijava ostaneta nespremenjeni. Prvo merilo potrjuje prisotnost ranljivosti, drugo pa, da jo obravnavani popravek odpravi, ne da bi spremenil pričakovano delovanje sistema za legitimne uporabnike.

3.2 Postavitev okolja in tehnološki sklad

Aplikacija je implementirana v jeziku Python 3.12 s spletnim ogrodjem Flask 3.1.0. Za strežniško implementacijo standarda WebAuthn se uporablja knjižnica `py_webauthn` 2.7.0, ki pokriva generiranje opcij za registracijo in overjanje ter verifikacijo podpisanih odgovorov. Podatkovni sloj temelji na SQLite3, ki ne zahteva zunanjega strežnika. Izvajanje poteka v enem samem kontejnerju Docker, dosegljivem na naslovu `http://localhost:8080`.

Podatkovna baza vsebuje tabeli `users` in `credentials`. Tabela `users` hrani le identifikator (`id`) in uporabniško ime (`username`); gesel ne shranjuje, saj overjanje poteka izključno prek poverilnic WebAuthn. Tabela `credentials` hrani identifikator poverilnice (`credential_id`), javni ključ (`public_key`), števec podpisov (`sign_count`) in tuji ključ `user_id`, ki poverilnico veže na lastnika. Ob inicializaciji se ustvari uporabnik `admin` z navidezno poverilnico z neveljavnim javnim ključem, kar onemogoča legitimno prijavo in sili napadalca k izkoriščanju ranljivosti.

3.3 Zaporedje korakov demonstracije

Demonstracija je zasnovana v naslednjih korakih:

1. **Postavitev okolja.** Zagon ranljive različice aplikacije v kontejnerju Docker in inicializacija baze z administratorskim računom, ki ima navidezno (neuporabno) poverilnico.
2. **Registracija napadalca.** Napadalec ustvari lasten račun prek strani `/register` in si zapomni `credential_id` svoje nove poverilnice.
3. **Izvedba napada na ranljivo različico.** Napadalec na prijavni strani prepiše klic `navigator.credentials.get()` in v polje `allowCredentials` vstavi lastni `credential_id`; nato se prijavi z uporabniškim imenom `admin`.
4. **Preverjanje uspeha.** Preveri se, ali je napadalec pridobil sejo uporabnika `admin` in dostop do zaščitene administratorske strani `/admin`.
5. **Popravek.** Razširi se poizvedba SQL pri iskanju poverilnice, tako da preverja še `user_id`.
6. **Ponovitev napada na popravljeni različici.** Isti postopek kot v koraku 3 se izvede še enkrat in preveri se, da prijava ne uspe.

[1] Izvorna koda: `https://github.com/svenko99/WebAuthn-CTF`

3.4 Metodologija primerjave knjižnic

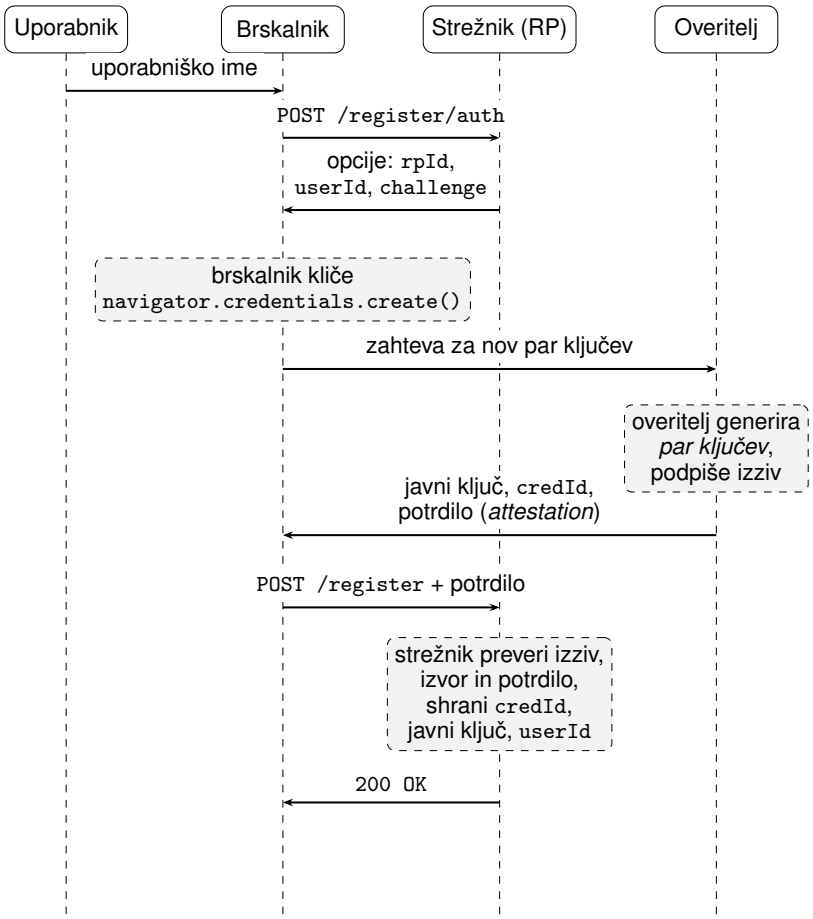
Poleg demonstracije smo opravili primerjavo strežniških knjižnic za standard WebAuthn, da bi ugotovili, ali je izolirana napaka vezave poverilnic značilnost posamezne knjižnice ali širša lastnost načina, kako knjižnice zasnujejo svoj vmesnik. V primerjavo smo vključili štiri razširjene odprtokodne knjižnice, po eno iz štirih jezikovnih okolij, med njimi tudi tisto, uporabljeno v demonstraciji: `py_webauthn` 2.7.0 (Python, različica iz demonstracije), `@simplewebauthn/server` 13.3.2 (TypeScript), `web-auth/webauthn-lib` 5.3.5 (PHP) in Yubicov `java-webauthn-server` 2.9.0 (Java). Merilo izbora sta bila razširjenost knjižnice in pokritost različnih jezikov, da ugotovitev ni vezana na eno samo okolje.

Za vsako knjižnico smo pregledali javni vmesnik in uradno dokumentacijo tistega dela, ki izvede overjanje, torej funkcije oziroma vmesnika za iskanje in verifikacijo poverilnice ob prijavi. Primerjali smo po enem samem merilu, ki neposredno ustreza demonstrirani napaki: ali vmesnik med obveznimi vhodi zahteva identiteto uporabnika, ali pa vezavo poverilnice na uporabnika v celoti prepušča klicni aplikacijski kodi. Rezultati primerjave so predstavljeni v razdelku 4.9.

4 IZVEDBA IN REZULTATI

4.1 Tok registracije

Registracija sledi standardnemu toku WebAuthn, prikazanemu na sliki 2: odjemalec strežniku posreduje uporabniško ime, strežnik vrne opcije z naključnim izzivom, brskalnik prek vmesnika `navigator.credentials.create()` pridobi od overitelja nov par ključev in podpisano potrdilo o registraciji (ang. *attestation*), strežnik pa preveri ujemanje izziva in izvora ter shrani javni ključ.

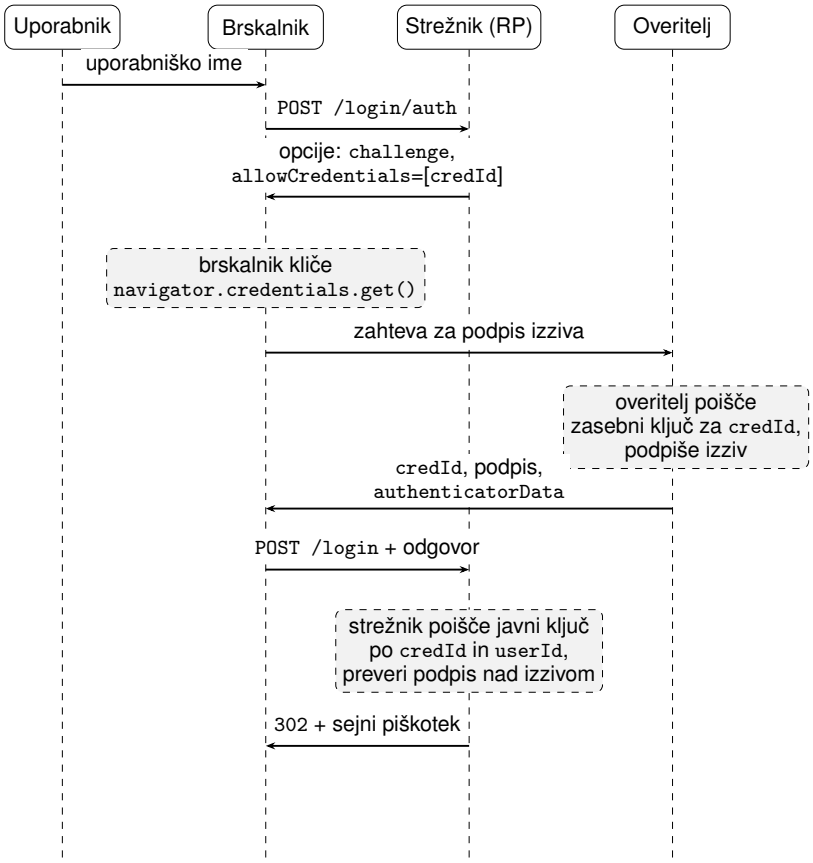


Slika 2: Potek registracije WebAuthn.

V predstavljeni implementaciji opcije generira klic `generate_registration_options()` iz knjižnice `py_webauthn`, izziv pa se shrani v sejo do prejema potrdila. Verifikacijo prejetega potrdila izvede klic `verify_registration_response()`, ki preveri tako ujemanje izziva kot izvor. Ob uspešni verifikaciji se v tabelo `credentials` zapišejo `credential_id`, javni ključ, začetna vrednost `sign_count` in tuji ključ `user_id`.

4.2 Tok overjanja

Overjanje prav tako sledi standardnemu toku (slika 3). Za prijavljajočega uporabnika strežnik v polje `allowCredentials` vključi identifikatorje vseh njegovih registriranih poverilnic, brskalnik prek vmesnika `navigator.credentials.get()` zahteva od overitelja podpis izziva, strežnik pa preveri kriptografsko veljavnost prejetega podpisa.



Slika 3: Potek overjanja WebAuthn.

V predstavljeni implementaciji strežnik v sejo shrani izziv, ciljno uporabniško ime in identifikator uporabnika. Verifikacijo izvede klic `verify_authentication_response()` iz knjižnice `py_webauthn`, ki preveri zgolj kriptografsko veljavnost podpisa glede na podan javni ključ; zagotavljanje, da podan javni ključ pripada ciljnemu uporabniku, je odgovornost aplikacije in ni del API-ja knjižnice. Ob uspešni verifikaciji aplikacija v sejo zapiše identiteto ciljnega uporabnika in v bazi posodobi `sign_count`. Prav v koraku iskanja javnega ključa se nahaja ranljivost, opisana v naslednjem razdelku.

4.3 Namerno vgrajena ranljivost

Ranljivost se nahaja v koraku, kjer strežnik išče poverilnico v bazi s stavkom SQL iz Izpisa 1.

```
SELECT id, user_id, credential_id,
       public_key, sign_count
FROM credentials
WHERE credential_id = ?
```

Izpis 1: Iskalna poizvedba SQL ranljive različice. Pogoj `WHERE` preverja zgolj `credential_id`.

Pogoj `WHERE` preverja zgolj vrednost `credential_id`, ne pa tudi `user_id`. Poizvedba tako vrne katerokoli poverilnico z ujemajočim identifikatorjem, ne glede na to, kateremu uporabniku pripada. Popravljena različica razširi pogoj na `WHERE credential_id = ?  AND user_id = ?`.

S tem strežnik zavrne vsako poverilnico, ki ne pripada ciljnemu uporabniku, ne glede na veljavnost podpisa. Obe različici sta z vidika legitimnih uporabnikov funkcionalno enaki. Razlika se pokaže izključno pri poskusu zlorabe.

4.4 Potek napada

Napadalec ima dostop do brskalnika s podporo WebAuthn. Najprej registrira lasten račun na strani `/register`. Ko registracija uspe, strežnik v odgovoru HTTP vrne vrednost `credential_id` napadalčeve poverilnice, ki jo napadalec razbere iz zavihka *Network* v razvijalskih orodjih brskalnika (ang. *DevTools*). Nato se odjavi.

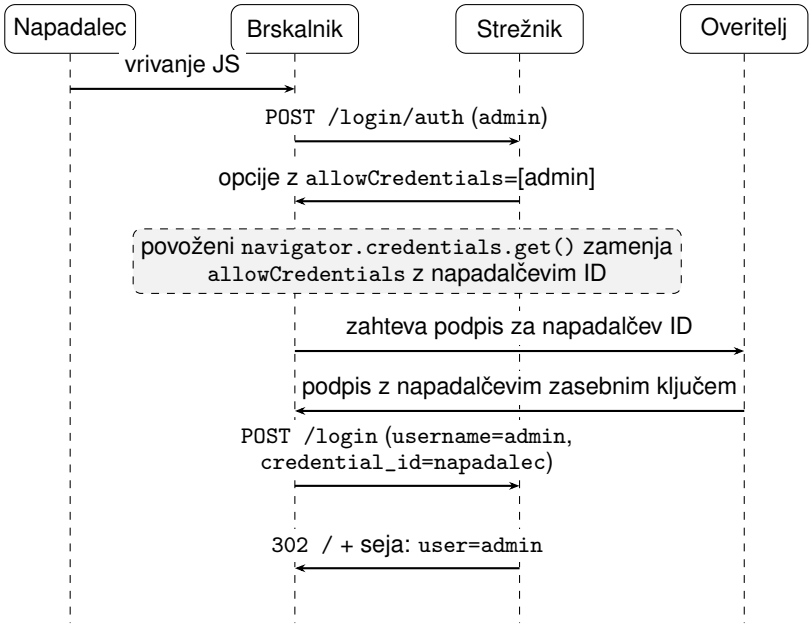
Napadalec nato odpre stran `/login` in v konzoli razvijalskih orodij pred sprožitvijo prijave prepiše klic `navigator.credentials.get()` s kodo iz Izpisa 2.

```
const originalGet = navigator.credentials.get;
navigator.credentials.get = async function (opts) {
  opts.publicKey.allowCredentials = [{
    type: "public-key",
    id: base64UrlToBuffer("NAPADALCEV_CREDENTIAL_ID")
  }];
  return originalGet.call(this, opts);
};
```

Izpis 2: Koda JavaScript, vrinjena v konzolo razvijalskih orodij, ki povozi klic `navigator.credentials.get()` in vstavi napadalčev `credential_id` v polje `allowCredentials`.

S tem napadalec v opcijah, ki jih strežnik vrne za ciljnega uporabnika, zamenja polje `allowCredentials` z lastnim `credential_id`.

Napadalec nato v prijavni obrazec vnese uporabniško ime `admin` in nadaljuje. Brskalnik na podlagi zamenjanih opcij od overitelja zahteva podpis z napadalčevim zasebnim ključem. Strežnik prejme podpisani odgovor, poišče poverilnico zgolj po `credential_id` brez preverjanja lastništva, uspešno verificira podpis z napadalčevim javnim ključem in nastavi sejo na uporabnika `admin`. Napadalec s tem pridobi nepooblaščen dostop do administratorske strani. Celoten potek je shematsko prikazan na sliki 4.



Slika 4: Potek napada na ranljivo različico.

4.5 Zagon in izhodiščno stanje

Aplikacija je bila uspešno zagnana v kontejnerju Docker. Ob inicializaciji baze je bila za uporabnika `admin` ustvarjena navidezna poverilnica z identifikatorjem `DUvFhC3oiS3G8aO61d5hUMAehmI`, ki ne ustreza nobenemu fizičnemu ali platformnemu overitelju, dosegljivemu napadalcu. Poskus legitimne prijave v račun `admin` se zaključi že na strani odjemalca: ko brskalnik od overitelja zahteva podpis za enega izmed identifikatorjev v polju `allowCredentials`, overitelj ustreznega zasebnega ključa ne najde in vrne napako `NotAllowedError`. Strežnik podpisa nikoli ne prejme.

4.6 Pridobitev napadalčeve poverilnice

Napadalec se je pod uporabniškim imenom `napadalec` uspešno registriral prek strani `/register`. Med postopkom registracije brskalnik ustvari nov asimetrični par ključev, identifikator nove poverilnice pa odjemalec posreduje strežniku v telesu zahteve `POST /register` (Izpis 3). Napadalec to vrednost neposredno prebere iz svoje odhodne zahteve v zavihku omrežnega prometa razvijalskih orodij brskalnika in si jo zapomni za naslednji korak. V prikazanem zagonu je njegova vrednost `SMkiLVGPelqClYFjxkuqMrmhz0c`.

```
POST /register HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Cookie: session=eyJyZWciOnsiY2hhbGxlbmdlIjoibV9...

username=napadalec
&credential_id=SMkiLVGPelqClYFjxkuqMrmhz0c
&client_data_json=eyJ0eXBlIjoid2ViYXV0aG4uY3JlYX...
&attestation_object=o2NmbXRkbm9uZWdhdHRTdG10oGh...
```

Izpis 3: Zaključna zahteva registracije, iz katere napadalec prebere identifikator svoje nove poverilnice.

Po uspešni registraciji strežnik napadalca samodejno prijavi in preusmeri na začetno stran. Napadalec se nato odjavi prek `/logout`; v lokalni shrambi overitelja ostane le napadalčeva zasebna poverilnica.

4.7 Izvedba napada na ranljivo različico

Napadalec je nadaljeval na strani `/login`. Pred oddajo prijavnega obrazca je v konzolo razvijalskih orodij brskalnika vstavil kodo JavaScript iz Izpisa 2, ki povozi standardni vmesnik `navigator.credentials.get()`. Vstavljanje kode je bilo sprejeto brez napake; vsi nadaljnji klici prijavnega vmesnika v tem zavihku tečejo skozi povoženo različico. Postopek sledi shemi na sliki 4.

V prijavni obrazec je napadalec vnesel uporabniško ime `admin` in oddal zahtevo. Brskalnik je z zahtevo `POST /login/auth` strežniku posredoval ciljno uporabniško ime in prejel odgovor JSON z opcijami za podpis. V polju `allowCredentials` odgovora se je pojavil identifikator adminove navidezne poverilnice (Izpis 4). Strežnik je iskanje izvedel po uporabniškem imenu in s svojega vidika postopek opravil pravilno.

```
"allowCredentials": [
  { "type": "public-key",
    "id": "DUvFhC3oiS3G8aO61d5hUMAehmI" }
]
```

Izpis 4: Polje `allowCredentials` v odgovoru JSON zahteve `POST /login/auth` za uporabnika `admin`. Identifikator pripada navidezni poverilnici računa `admin`, ne napadalcu.

Pred posredovanjem opcij overitelju je povoženi `navigator.credentials.get()` polje `allowCredentials` prepisal z napadalčevim identifikatorjem (Izpis 5). Sprememba je bila izvedena v celoti znotraj brskalnika; strežnik o njej ni dobil nobenega obvestila.

```
"allowCredentials": [
  { "type": "public-key",
    "id": "SMkiLVGPelqClYFjxkuqMrmhz0c" }
]
```

Izpis 5: Stanje istega polja po izvedbi vrivanja JavaScript. Identifikator zdaj pripada napadalčevi poverilnici.

Overitelj je našel ujemajočo se napadalčevo zasebno poverilnico in vrnil podpis nad izzivom. Brskalnik je nato izvedel zaključno zahtevo prijave (Izpis 6). V telesu zahteve sta istočasno prisotna uporabniško ime `admin` in identifikator *napadalčeve* poverilnice. Te kombinacije legitimna implementacija ne bi smela sprejeti.

```
POST /login HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Cookie: session=.eJwlzU0PgiAYAOD_8p7rUG1Y3kyB6aGk...

username=admin
&credential_id=SMkiLVGPelqClYFjxkuqMrmhz0c
&authenticator_data=SZYN5YgOjGh0NBcPZHZgW4_krrm...
&client_data_json=eyJ0eXBlIjoid2ViYXV0aG4uZ2V0I...
&signature=MEUCIEDnfEWLcbBckCIy8JXfEd6Xyl2DeHlu...
```

Izpis 6: Zaključna zahteva prijave proti ranljivi različici. Polje `username` se sklicuje na uporabnika `admin`, polje `credential_id` pa na napadalčevo poverilnico.

Strežnik ranljive različice je poverilnico v bazi poiskal zgolj po identifikatorju, brez preverjanja, ali ta pripada uporabniku `admin`. Verifikacija podpisa je z napadalčevim javnim ključem uspela in strežnik je seji dodelil identiteto uporabnika `admin`. Odgovor s statusom `302` je preusmeril odjemalca na začetno stran in v sejnem piškotku zapisal novo identiteto (Izpis 7).

```
HTTP/1.1 302 FOUND
Location: /
Set-Cookie: session=eyJhbGVydCI6IlNpZ25lZCBpbiBh
            cyBhZG1pbi4iLCJ1c2VyIjp7ImlkIjoxLCJ1
            c2VybmFtZSI6ImFkbWluIn19...
```

Izpis 7: Odgovor ranljive različice. Sejni piškotek po dekodiranju vsebuje polje `user` z vrednostmi `{"id":1,"username":"admin"}`.

Naknadna zahteva `GET /admin` je vrnila status `200` in prikazala zaščiteno administratorsko stran, ki za napadalca pred izvedbo napada ni bila dosegljiva.

4.8 Preverjanje popravljene različice

V popravljeni različici aplikacije je bila iskalnemu pogoju nad bazo poverilnic dodana vezava na identifikator uporabnika (Izpis 8). Drugi parameter popravljene poizvedbe je `user_id` ciljnega uporabnika, pridobljen iz seje. Napad je bil ponovljen z enako napadalčevo poverilnico in po enakem postopku.

```
- WHERE credential_id = ?
+ WHERE credential_id = ? AND user_id = ?
```

Izpis 8: Razlika med ranljivo in popravljeno različico iskalne poizvedbe v bazi poverilnic.

Iskanje napadalčeve poverilnice znotraj zapisov uporabnika `admin` je vrnilo prazen rezultat. Strežnik se je odzval s preusmeritvijo na `/login` in obvestilom `Invalid passkey credential.` (Izpis 9). Sejni piškotek polja `user` ni vseboval. Naknadna zahteva `GET /admin` je bila preusmerjena na začetno stran z obvestilom `Admin access only.`

```
HTTP/1.1 302 FOUND
Location: /login
Set-Cookie: session=.eJwljcsKgkAUQH8lZh2RQYXuipR
            SDF_Z5EauenGkcXyNlkX_ntHuLM7hvAlwbC
            XRyEkMwItsVkPX3XGcpS1mKGQBfEHmBHrJ...
```

Izpis 9: Odgovor popravljene različice na isti napad. Preusmeritev vodi na prijavno stran, sejni piškotek polja `user` ne vsebuje.

4.9 Rezultati primerjave knjižnic

Pregled po merilu iz razdelka 3.4 je pokazal jasno ločnico. V knjižnicah `py_webauthn` (Python), `@simplewebauthn/server` (TypeScript) in `web-auth/webauthn-lib` (PHP) knjižnica sama ne poišče poverilnice v bazi: integrator mora pravilen javni ključ poiskati sam in ga verifikacijski funkciji predati kot parameter. Identitete uporabnika ti vmesniki ne zahtevajo, zato je ujemanje med poverilnico in uporabnikom v celoti odgovornost klicne kode. Yubicova knjižnica `java-webauthn-server` (Java) je izjema: integrator mora implementirati vmesnik za iskanje poverilnic, ki kot obvezna parametra zahteva tako identifikator poverilnice kot identifikator uporabnika, in brez ujemanja po obeh poljih knjižnica poverilnice ne vrne. Med štirimi pregledanimi knjižnicami torej le ena vezavo poverilnice na uporabnika zahteva že v svojem vmesniku, preostale tri pa jo prepuščajo aplikaciji.

5 DISKUSIJA

Demonstracija je potrdila, da napadalec, ki ima v sistemu lasten registriran račun, prevzame administratorsko sejo zgolj z manipulacijo polja `allowCredentials` v brskalniku. Popravljena različica, ki pogoj SQL razširi z `user_id`, je isti napad zavrnila že pred kriptografsko verifikacijo. Rezultati se ujemajo s teoretičnim modelom napada ponovne vezave overitelja [6].

Hkrati je opisana ranljivost strukturno enaka tisti, ki jo dokumentira CVE-2025-26788 [13] v strežniku StrongKey FIDO Server. Ista napaka se pojavi v dveh tehnološko neodvisnih implementacijah: samostojnem

strežniku Java na eni strani in demonstracijski aplikaciji Python na knjižnici `py_webauthn` na drugi. Napaka torej ni omejena na posamezno knjižnico; k njej prispeva tudi zasnova vmesnikov, ki vezavo poverilnice na uporabnika prepuščajo aplikaciji, namesto da bi jo zahtevali. Demonstracija v tem obsegu ponazarja ponavljajoč se vzorec, ne le posamezne napake.

Primerjava knjižnic iz razdelka 4.9 to opažanje podpre in pojasni. Oblika vmesnika, kakršno ima Yubicov `java-webauthn-server` in ki vezavo poverilnice na uporabnika zahteva že med obveznimi vhodi, prenese odgovornost za pravilno vezavo iz aplikacijske kode v vmesnik knjižnice in tako napako, ki jo obravnava ta članek, oteži. Ker preostale tri pregledane knjižnice enako dovoljujejo pomanjkljivo vezavo, kakršno prikazuje demonstracija, ostaja tak pristop med njimi osamljen.

Onkraj obravnavane napake vezave je iz vseh treh kategorij strežniških napak mogoče izpeljati konkretne smernice, ki jih razvijalci integracije WebAuthn lahko preverijo pri reviziji strežniške kode. Po specifikaciji WebAuthn [7], §7.2, mora strežnik zagotoviti, da poverilnica, ki jo identificira `credential_id`, pripada identificiranemu uporabniku, kar v praksi pomeni kombinacijo polj `credential_id` in `user_id` v pogoju iskanja v bazi. Pri odkrivnih poverilnicah (ang. *discoverable credentials*), kjer uporabnik ob začetku tokova ni vnaprej znan, identiteto določa polje `userHandle` v odgovoru overitelja. Generirani izziv mora biti shranjen na strežniški strani in po prvi uporabi neveljaven (§13.4.3). Pričakovana vrednost se ne sme prebrati iz odjemalske zahteve. Validacija izvora (`clientDataJSON.origin`) mora preverjati celoten URL, vključno s shemo, gostiteljem in vrati, ne le imena gostitelja. Strežnik mora preveriti tudi `rpIdHash` v podatkih overitelja in zastavico prisotnosti uporabnika (UP, ang. *User Present*). Kadar je preverjanje uporabnika zahtevano, mora preveriti tudi zastavico preverjanja uporabnika (UV, ang. *User Verified*). Vrnjene vrednosti validacijskih funkcij se ne smejo spregledati, saj je izpustitev tega preverjanja že povzročila dokumentiran obhod verifikacije v produkcijski platformi (CVE-2026-0723 [5]). Števec podpisov (`sign_count`) naj se preverja kot monotono naraščajoč na ključih, vezanih na napravo (§6.1.1), kar omogoča zaznavo morebitnega kloniranja overitelja. Sinhronizirani ključi Passkey te varnostne funkcionalnosti ne podpirajo, ker veliki ponudniki platform pri njih vedno vračajo ničlo [1].

Kljub temu da demonstracija kaže na širši vzorec, ima omejitve, ki jih moramo upoštevati pri posploševanju ugotovitev. Med te sodi obravnava zgolj ene kategorije strežniških napak v enem tehnološkem skladu in v lokalnem okolju brez sinhronizacije poverilnic prek infrastrukture Passkey. Posplošitev na druge sklade in produkcijska okolja terja samostojno preverjanje. Poleg tega demonstracija predpostavlja, da napadalec že ima v sistemu lasten registriran račun. Ta predpogoj je v praksi izpolnjen na storitvah z odprto registracijo, ne pa nujno povsod.

6 ZAKLJUČEK

V članku je obravnavana strežniška ranljivost vezave poverilnic v implementacijah standarda WebAuthn, pri kateri napadalec z manipulacijo polja `allowCredentials` ob prijavi prevzame tujo sejo z lastno veljavno poverilnico. Za ponovljivo preverjanje napada in njegove odprave je bila razvita demonstracijska implementacija, ki napako izolira v eni vrstici poizvedbe in omogoča neposredno primerjavo ranljive in popravljene različice. Glavna ugotovitev je, da varnosti overjanja s standardom WebAuthn ne zagotovi že izbira knjižnice ali jezika; zagotovi jo šele dosledna strežniška logika, ki vsako prejeto poverilnico izrecno preveri proti uporabniku, za katerega se zahteva dostop. Demonstrirani napad in njegova preprosta odprava potrjujeta, da je razlika med ranljivim in varnim sistemom pogosto en sam dodaten pogoj v poizvedbi. Brez njega je mogoče kriptografsko preverjanje podpisa obiti, saj strežnik sprejme veljaven podpis poverilnice, ki ne pripada ciljnemu uporabniku.

Med smiselne smeri nadaljnjega dela sodi razširitev demonstracije na druge kategorije implementacijskih ranljivosti, kot so pomanjkljiva validacija izvora (ang. *origin*), zanemarjeno preverjanje monotonega naraščanja `sign_count` kot indikatorja kloniranega overitelja, uporaba statičnih ali predolgih izzivov ter neustrezna obravnava potrdila o registraciji (ang. *attestation*). Drugo naravno nadaljevanje je obravnava odkrivnih poverilnic (ang. *discoverable credentials*), pri katerih strežnik uporabnika identificira izključno prek polja `userHandle` v odgovoru overitelja in se izpostavljenost napaki vezave bistveno spremeni. Vredna nadaljnje obravnave je tudi empirična analiza implementacij, ki temeljijo na knjižnici `py_webauthn`, glede dejanske prisotnosti enake napake v javno dostopnih projektih.

LITERATURA

[1] Apple Developer Forums. `signCount` behavior for synced passkeys (Apple engineer response on developer forum). `https://developer.apple.com/forums/thread/712079?answerId=723508022`, 2022.

[2] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable Security Analysis of FIDO2. In *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 125–156. Springer, 2021.

[3] Iness Ben Guirat and Harry Halpin. Formal Verification of the W3C Web Authentication Protocol. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security (HoTSoS)*, pages 1–10. ACM, 2018.

[4] FIDO Alliance. FIDO2 Specifications, 2024.

[5] GitLab Security. CVE-2026-0723: Limited authentication bypass for 2fa with WebAuthn and passkey authentication. `https://gitlab.com/gitlab-org/gitlab/-/work_items/585333`, 2026.

[6] Jingjing Guan, Hui Li, Haisong Ye, and Ziming Zhao. A Formal Analysis of the FIDO2 Protocols. In *Computer Security – ESORICS 2022, Part III*, volume 13556 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2022.

[7] Jeff Hodges, J.C. Jones, Michael B. Jones, Akshay Kumar, and Emil Lundberg. Web Authentication: An API for accessing Public Key Credentials Level 2. W3C Recommendation, April 2021.

[8] Matej Huš. Svet brez gesel. *Monitor*, 35(4), April 2025.

[9] Marko Hölbl and Marko Kompara. Alternativa geslom – FIDO2 in Passkey. In Luka Pavlič, Tina Beranič, and Marjan Heričko, editors, *OTS 2024 Sodobne informacijske tehnologije in storitve: Zbornik 27. konference*, Maribor, 2024. Univerzitetna založba Univerze v Mariboru.

[10] Leona Lassak, Elleen Pan, Blase Ur, and Maximilian Golla. Why Aren't We Using Passkeys? Obstacles Companies Face Deploying FIDO2 Passwordless Authentication. In *Proceedings of the 33rd USENIX Security Symposium*, pages 7231–7248, Philadelphia, PA, 2024. USENIX Association.

[11] NIST National Vulnerability Database. CVE-2026-28787: Server-side challenge state not persisted in OneUptime WebAuthn implementation. `https://nvd.nist.gov/vuln/detail/CVE-2026-28787`, 2026.

[12] NIST National Vulnerability Database. CVE-2026-30964: Origin validation bypass in `web-auth/webauthn-lib`. `https://nvd.nist.gov/vuln/detail/CVE-2026-30964`, 2026.

[13] Natalia Trojanowska-Korepta. CVE-2025-26788: Passkey authentication bypass in StrongKey FIDO Server. `https://www.securing.pl/en/cve-2025-26788-passkey-authentication-bypass-in-strongkey-fido-server/`, February 2025.

[14] Tarun Kumar Yadav and Kent Seamons. A Security and Usability Analysis of Local Attacks Against FIDO2. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2024.

Sven Ulčar je študent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Zanimata ga kibernetska varnost in sistemska infrastruktura.

Matevž Pesek je izredni profesor in raziskovalec na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je diplomiral (2012) in doktoriral (2018). Od leta 2009 je član Laboratorija za računalniško grafiko in multimedije. Od leta 2024 izvaja predmeta Varnost programov in Varnost sistemov, kjer se raziskovalno ukvarja s poučevanjem konceptov in organizacijo dogodkov s področja računalniške varnosti.