ZADANIE Č.3

I-UPB

Vladimír Hlačina
Fakulta elektrotechniky a informatiky
Slovenská technická univerzita
xhlacina@stuba.sk

Rudolf Bachorík
Fakulta elektrotechniky a informatiky
Slovenská technická univerzita
xbachorik@stuba.sk

Jakub Zigo
Fakulta elektrotechniky a informatiky
Slovenská technická univerzita
xzigo@stuba.sk

Igor Kuzmin
Fakulta elektrotechniky a informatiky
Slovenská technická univerzita
xkuzmini@stuba.sk

6. November 2023

ABSTRAKT

Cieľom zadania je rozšíriť existujúcu webovú aplikáciu z predchádzajúceho zadania. Jadrom zadania je oboznámiť sa s bezpečnou správou používateľských hesiel v databáze.

Informácia ku prácam na projekte

Hlavnú časť implementácie realizoval kolega Rudolf Bachorík. Ďalej sa na implementácii FE podieľal kolega Jakub Zigo. Spracovanie kompletnej dokumentácie realizoval Vladimír Hlačina. Časť overenia hesiel na FE implementoval Igor Kuzmin.

1 Základ pre prihlasovanie a registráciu

V tejto časti popíšeme základy pre prihlasovanie a registráciu. Prvým krokom bolo navrhnutie databázovej štruktúry pre používateľov systému. Túto časť zadania sme si predpripravili ešte počas predošlého zadania, čo nám uľahčilo prácu pri implementácii. Tabuľku `users` sme doplnili o 3 pridané stĺpce. Konkrétne sa jedná o stĺpce `token, timestamp` a `public_key`. Stĺpec `token` slúži na ukladanie session tokenu, ktorý je potrebný pre autentifikáciu používateľa. `timestamp` tvorí základ pre overenie platnosti session tokenu. Posledný stĺpec `public_key` slúži na ukladanie verejného kľúča používateľa, ktorý je potrebný pre šifrovanie a dešifrovanie dát. Ďalším krokom bolo implementovať bezpečné ukladanie hesla do databázy. V poslednom zadaní sme implementovali uloženie hesla do databázy vo forme hashu, čo však dostatočne nespĺňa bezpečnostné štandardy. Vylepšenie tejto metódy počíva v použití tzv. saltu. Nasledujúci kód zobrazuje implementáciu tejto metódy [1].

```
1       bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt(5)).decode('utf-8')
```

Salt je náhodná hodnota, ktorá sa pridáva k heslu pred jeho hashovaním pomocou knižnice `bcrypt` [1]. Hlavnou výhodou bezpečnosti saltovania spočíva v tom, že salt je jedinečný pre každé heslo. Týmto spôsobom sa zamedzí možnosti použitia tzv. rainbow tabuliek, ktoré obsahujú vopred vypočítané hashe pre rôzne kombinácie hesiel a saltov.

1.1 Registrácia

Registrácia je realizovaná pomocou formulára, ktorý obsahuje 3 polia. Prvým poľom je pole pre zadanie mena používateľa. Ďalším je pole pre zadanie emailu používateľa. Posledným poľom je pole pre zadanie hesla používateľa.

```
1        data = json.loads(str(request.data, 'utf-8'))
2        data['email'] = str(decryptRSA(data['email']), 'utf-8')
3        data['password'] = str(decryptRSA(data['password']), 'utf-8')
4        data['name'] = str(decryptRSA(data['name']), 'utf-8')
```

Na vyššie uvedenom kóde môžeme vidieť prvý krok registrácie. V tomto prípade sa jedná o dešifrovanie dát pomocou privátneho kľúča používateľa. V tomto prípade počítame so splením požiadaviek overenia jednotlicýh polí na FE, ktorému sa budeme venovať v ďalšej časti 3. Následne overíme či sa heslo nenachádza v databáze bežne používaných hesiel. Podrobnostiam sa rovnako budeme venovať v ďalšej časti dokumentácie 4. Následne sa overí či daný email existuje v databáze a v prípade, že neexistuje vytvorí sa nový záznam.

1.2 Prihlásenie

Prihlásenie je rovnako realizované pomocou formulára, ktorý obsahuje email a heslo. Po zadaní mena a hesla sa odošlú na server zašifrované pomocou public RSA kľúča. Na úspešné prihásenie musí email spĺňať nasledovné podmienky. Prvou podmienkou je, overenie brut-force útoku 2. Následne sa email a heslo odšifrujú pomocou privátneho RSA kľúča servera. Ďalej sa overí či užívateľ existuje a skontroluje sa správnosť hesla. Po úspešnom prihlásení je pre používateľa obrazený admin panel s možnosťou nastavenia nastaviť privátneho a verejného RSA kľúč 5. Ako bonus sme sa rozhodli implementovať tzv. session token pre jednotlivých používateľov. Token pozostáva z 32 náhodných znakov, ktoré sa zašifruje pomocou `AES256`, kde kľúčom je hash hesla a inicializačnný vektor je hash emailu (`SHA256`). Následne sa tento zašifrovaný token pošle späť klientovi, ktorý si ho pomocou svojich prihlasovacích údajov rozšifruje. Token má životnosť 10 minút, po uplynutí tejto doby sa musí používateľ znova prihlásiť ak nevykonal žiadnu požiadavku na sever.

2 Implementácia anti-brute force opatrení

V tejto časti popíšeme implementáciu anti-brute force opatrení. Prvým krokom bolo navrhnutie databázovej štruktúry pre tento systém. V našej implementácií sme sa rozhodli o pridanie ďalšej tabuľky do databázy.

ip	**login_attempts**	**timestamp**
VARCHAR(64) NOT NULL	INT NOT NULL	DECIMAL(20, 5)

Tab. 1: Tabuľka štruktúry databázy pre brute-force

Tabuľka 1 popisuje štruktúru spomínanej tabuľky. Táto tabuľka obsahuje 3 stĺpce. Stĺpec `ip` slúži na ukladanie IP adresy používateľa, ktorý sa pokúša prihlásiť do systému. Stĺpec `login_attempts` slúži na ukladanie počtu pokusov o prihlásenie. `timestamp` slúži na nastavenie 10 minútového timera, ktorý sa nastaví po 3 neúspešných pokusoch o prihlásenie. Tento timer zamedzí možnosti nekonečného pokusu o prihlásenie. Po uplynutí 10 minút sa počet pokusov o prihlásenie neresetuje. Ďalšie neúspešné pokusy o prihlásenie znova nastavia timer na 10 minút. Pokusy o prihlihlásenie sú resetované po pol hodine. Jednou alternatívnych možností bolo použitie knižnice `flask-limiter` [2], táto knižnica poskytuje nastavenie limitovania počtu požiadaviek na server. Z dôvodu nedostatku času by sme na nasledovnom zadaní chceli túto knižnicu implementovať a nahradiť ňou doterajšie riešeni.

3 Overovanie zložitosti hesla

Overenie zložitosti hesiel sme implementovali na strane FE. Prvým krokom bola implementácia overenia emailu používateľa. Pre overenie správnosti formátu sme použili štandardný Regulárny výraz, ktorý je uvedený v nasledovnej časti kódu.

```
1       const emailRegex = /^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+[A-Za-z]2,$/;
```

Ďalším krokom bolo overenie zložitosti hesla. Pre overenie zložitosti hesla sme použili základné parametre, ktoré by bezpečné heslo malo spĺňať. Nasledujúca časť kódu zobrazuje chybové hlášky, ktoré sa zobrazia v prípade, že používateľ zadá neplatné heslo.

```
1       const errorMessages = {
2           length: 'Heslo musí obsahovať viac ako 12 znakov.',
3           number: 'Heslo musí obsahovať aspoň jedne čislo.',
4           letterCase: 'Heslo musí obsahovať veľké aj malé písmená.',
5           specialCharacter: 'Heslo musí obsahovať aspoň jeden špeciálny znak.'
6       }
```

Posledným krokom bolo overenie mena používateľa. V tomto prípade nebolo potrebné použitie regulárneho výrazu, preto sme sa rozhodli o overenie prázdnosti poľa pre zadanie hesla. V prípade prázdnosti poľa sa zobrazí chybová hláška, čo platí aj pre vyššie uvedené polia.

4 Bežne používané heslá

Táto sekcia popisuje spôsob overenie bežne používaných hesiel. Dôležitosť použitia známych hesiel potvrdzuje aj fakt existencie známych hashov a saltov pre tieto heslá. V našej implementácií sme sa rozhodli použiť verejne dostupné heslá [3]. Tieto heslá sme uložili do súboru `10-million-password-list-top-1000000.txt`. Súbor obsahuje set milióna hesiel. Overenie prebieha na strane servera kde sa overí či sa zadané heslo nenachádza v súbore. V prípade, že sa heslo nachádza v súbore, zobrazí chybová hláška a používateľ musí zvoliť iné heslo.

5 Nahranie verejného/privátneho kľúča

Nahrávanie verejného a privátneho kľúča je funkcionalita povolená len pre prihlásených používateľov. Táto funkcionalita je realizovaná pomocou formulára, ktorý obsahuje 2 polia a 2 tlačidlá. Jednotlivé polia obsahujú placeholder, ktorý hovorí o tom, či sa jedná o verejný alebo privátny kľúč. Validita kľúčov sa overuje pomocou overenia či sú jednotlivé kľúče nastavené vo formáte `.pem`. Verejný kľúč je následne uložený v databáze a server ho používa na šifrovanie. Pri posielaní verejného kľúča na uloženie do databázy sa posiela aj token pomocou ktorého overujeme či je daný používateľ naozaj prihlásený. Privátny kľúč je uložený v rámci kódu čo zabezpečuje, že sa k nemu nedá pristupovať z vonkajšieho prostredia. Pre dodatočné zabezpečenie je možné aplikovanie obfuskácie kódu. Dešifrovanie na strane klienta sme implementovali už v predošlom zadaní. V tomto prípade sme sa rozhodli o implementovanie overenia riešenia, ktoré spočíva v odoslaní textovej správy, zašifrovanej pomocou verejného kľúča na server. Server túto správu dešifruje pomocou privátneho kľúča a následne ju zašifruje pomocou nahraného verejného kľúča. Táto správa je následne odoslaná späť klientovi, ktorý ju dešifruje pomocou nahraného privátneho kľúča.

V prípade, že sa správa zhoduje s pôvodnou správou, je overenie úspešné.

Literatúra

[1] bcrypt: A library to help you hash passwords. `https://www.npmjs.com/package/bcrypt`.

[2] Flask-limiter: Limited drops at the time. `https://flask-limiter.readthedocs.io/en/stable/`.

[3] Seclists: Collection of multiple types of lists used during security assessments. `https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/`.