Sistematična analiza napada POODLE na SSL 3.0 z AES-CBC in ocena protiukrepov

Anja Klančar, Matevž pesek
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
ak01265@student.uni-lj.si, matevz.pesek@fri.uni-lj.si

Izvleček

V pričujočem članku obravnavamo kriptografski napad POODLE (Padding Oracle On Downgraded Legacy Encryption), ki izkorišča ranljivost protokola SSL 3.0 pri uporabi šifriranja AES v načinu CBC skupaj s pristopom overi-nato-šifriraj. Najprej predstavimo teoretično ozadje napada, vključno z vplivom podlage, veriženja blokov in razločevanja med različnimi tipi napak pri dešifriranju. Nato prikažemo, kako lahko napadalec z opazovanjem odzivov strežnika postopoma razkriva podatke brez poznavanja skrivnega ključa. V praktičnem delu opišemo implementacijo AES-CBC v programskem jeziku Java, izvedbo napada POODLE ter analizo pogojev, ki omogočajo njegovo uspešnost. Posebej se osredotočimo na dve obrambni strategiji: poenotenje odzivov ob napaki, pri katerem strežnik ne razkriva vrste napake, in uporabo pristopa šifriraj-nato-overi, ki prepreči napad že pred dešifriranjem podatkov. Rezultati pokažejo, da je POODLE posledica neustrezne kombinacije kriptografskih mehanizmov in da je varnost takšnih sistemov močno odvisna tudi od pravilne implementacije protiukrepov.

Ključne besede: kriptografija, kriptografski napad, POODLE, SSL

SYSTEMATIC ANALYSIS OF THE POODLE ATTACK ON SSL 3.0 WITH AES-CBC AND EVALUATION OF COUNTERMEASURES

Abstract

In this paper, we examine the POODLE (Padding Oracle On Downgraded Legacy Encryption) cryptographic attack, which exploits a vulnerability in the SSL 3.0 protocol when AES encryption in CBC mode is used together with the MAC-then-encrypt approach. We first present the theoretical background of the attack, including the role of padding, block chaining, and the distinction between different types of decryption errors. We then show how an attacker can gradually recover data by observing server responses without knowing the secret key. In the practical part, we describe an implementation of AES-CBC in the Java programming language, the execution of the POODLE attack, and an analysis of the conditions that enable its success. Particular attention is given to two defense strategies: unifying error responses so that the server does not reveal the type of error, and using the encrypt-then-MAC approach, which prevents the attack before any data is decrypted. The results show that POODLE is a consequence of an inappropriate combination of cryptographic mechanisms and that the security of such systems also depends strongly on the correct implementation of countermeasures.

Keywords: cryptography, cryptographic attack, POODLE, SSL

1 UVOD

Kriptografija je temeljni gradnik sodobnih informacijskih sistemov, saj zagotavlja varno komunikacijo, zaščito podatkov in ohranja zaupanje v digitalnih okoljih. Poleg varovanja zaupnosti omogoča tudi preverjanje pristnosti, celovitosti in nezanikljivosti informacij, kar je ključno pri prenosu občutljivih podatkov v kritičnih aplikacijah, kot so bančništvo, elektronsko poslovanje in varna spletna komunikacija.
Cilji kriptografije so običajno opredeljeni v štirih kategorijah: zaupnost, celovitost, avtentikacija in nezanikljivost. Zaupnost zagotavlja, da imajo dostop do podatkov zgolj avtorizirane osebe, kar se doseže s šifriranjem. Celovitost pomeni, da se podatki med prenosom ne spremenijo, kar preverimo s pomočjo zgoščevalnih funkcij. Avtentikacija zagotavlja, da je pošiljatelj sporočila resničen, kar se v praksi izvaja s kodo za preverjanje sporočila (MAC), ki obenem prispeva k ohranjanju celovitosti. Nezanikljivost pa zagotavlja, da pošiljatelj ne more zanikati svoje udeležbe pri komunikaciji, kar se dosega z digitalnimi podpisi [1].

Med protokoli, ki udejanjajo te varnostne lastnosti, je ključnega pomena protokol SSL (Secure Socket Layer), namenjen vzpostavitvi varnega prenosa informacij prek omrežij. Protokol je sestavljen iz dveh slojev: *Record* plasti, ki skrbi za zaupnost, avtentičnost in zaščito pred napadi s ponovitvijo, ter *Handshake* protokola, ki omogoča izmenjavo ključev, inicializacijo komunikacije in uskladitev kriptografskega stanja med odjemalcem in strežnikom [2].

Kasneje zastarela najnaprednejša različica SSL protokola je bila različica SSL 3.0, ki je nadomestila različico SSL 2.0 zaradi več varnostnih pomanjkljivosti, kot so uporaba 40-bitnih ključev za avtentikacijo v nekaterih načinih, šibek mehanizem MAC in pomanjkljiva avtentikacija polja dolžine podloge, ki je omogočala manipulacijo z bajti na koncu sporočila. Kot naslednik SSL 3.0 je bil vzpostavljen protokol TLS 1.0 (Transport Layer Security), ki je uvedel pomembne izboljšave, med drugim uporabo varnejšega HMAC. Vendar pa ostaja ranljivost v obliki t.i. "downgrade attack", kjer lahko napadalec prisili odjemalca in strežnik, da komunikacijo vzpostavita prek starejše, manj varne različice SSL 3.0, kar predstavlja osnovo za izvedbo POODLE napada [3]. Najnovejša različica protokola TLS je različica TLS 1.3, ki je danes najbolj priporočljiva za uporabo. V primerjavi s predhodnimi različicami je prinesel veliko izboljšav, kot so izboljšana varnost, hitrejše delovanje in večjo fleksibilnost [4].

V tem članku je opisan kriptografski napad POODLE (Padding Oracle On Downgraded Legacy Encryption), ki temelji na ranljivosti protokola SSL 3.0 (Secure Socket Layer). Protokol uporablja AES (Advanced Encryption Standard) šifriranje v CBC (Cipher Block Chaining) načinu, kar samo po sebi ni nevarno, problem pa postane, če se ta način uporablja skupaj z načinom overi-nato-šifriraj. Problem z overi-nato-šifriraj je, da mora strežnik sporočilo najprej dešifrirati in šele nato lahko preveri overitveno značko. Zaradi takšnega delovanja nam strežnik v primeru napake pri dešifriranju pove tip napake, do katere je prišlo. Za napad sta pomembna dva tipa napak: napaka podloge in napaka značke. Nadaljnje predstavimo dve rešitvi preprečitve napada. Prva predstavljena rešitev je da ne glede na tip napake strežnik vrne napako značke, kar mora biti v praksi pravilno implementirano, saj drugače odpre vrata za časovni napad — torej, če je v resnici napaka podloge, je napaka hitreje vrnjena, kot pa če je napaka značke. V drugi rešitvi pa je predstavljena implementacija z načinom šifriraj-nato-overi, pri katerem se mora najprej preveriti značka, preden se dešifriranje sploh začne in bi se zato spreminjanje tajnopisa pravočasno ugotovilo, preden bi bil napad lahko izveden.

2 PREGLED PODROČJA

Zaradi razvoja informacijske in komunikacijske tehnologije je le ta prisotna povsod — v podjetjih, šolah, državnih inštitucijah, po domovih in drugi infrastrukturi. Zaradi njene razširjenosti so vse bolj aktualni napadi na infrastrukturo in podatke, imenovani kibernetski napadi, zaščita pred temi napadi pa se imenuje kibernetska varnost. Kibernetski napadi se izvajajo z namenom pridobivanja finančne koristi, vohunjenja, posebne oblike vojskovanja in druge [5].

Kibernetske napade lahko ločimo tudi med ciljno usmerjenimi napadi (angl. targeted attacks) in neciljnimi napadi (angl. untargeted attacks). Pri ciljno usmerjenih napadih si napadalec izbere specifičen cilj (specifično podjetje, sistem ipd.) in ga napade. Takšni napadi so ponavadi večja grožnja kot neciljni napadi, saj je napad narejen specifično za to tarčo in njene ranljivosti. Pri neciljnih napadih gre za napade, ki poskušajo ciljati čim več naprav ali uporabnikov hkrati. Za distribucijo večinokrat uporabljajo inter-

net. Primera takšnih napadov sta na primer ribarjenje (angl. phishing) in izsiljevalska programska oprema (angl. ransomware). [6]

Uspešni napadi imajo lahko katastrofalne posledice. Na državni ravni so to lahko na primer grožnja nacionalni varnosti, škoda gospodarskim in političnim odnosom države in škoda nacionalnemu gospodarstvu [7]. Kibernetske napade, ki so povezani s tehnologijo šifriranja, opredelimo kot kriptografske napade [8]. Kriptografske napade lahko razdelimo na več vrst:

Napadi na razpoložljivost (angl. Avalibility attacks) so napadi, katerih cilj je onemogočiti dostop do podatkov. Najpogostejši izmed njih je napad DoS (Denial-of-service).

Napadi na celovitost (angl. Integrity attacks) so napadi s ciljem uničenja podatkov. Te vrste napadov je veliko težje zaznavati kot napade razpoložljivosti, saj so velikokrat bolj prefinjeni.

Napadi na zaupnost (angl. Confidentiality attacks) so napadi, ki poskušajo ponarediti ali ukrasti zaupne informacije. Ti tipi napadov pogosto vključujejo tudi uporabo drugih dveh vrst napadov. [9]

POODLE napad spada v kategorijo napadov na celovitost, saj omogoča branje zaupnih informacij, do katerih napadalec ne bi smel imeti dostopa. Obstaja več kriptografskih napadov, ki so v nekaterih pogledih podobni POODLE, ki ga naslavljamo v pričujočem članku. Dva od teh sta CRIME-napad in Bleichenbacherjev napad. CRIME izkorišča ranljivost istega protokola kot POODLE, torej SSL 3.0, Bleichenbacherjev napad pa spada v isto skupino napadov, torej Padding Oracle attacks, za katere je značilno, da za napad izkoriščajo podlogo zakriptiranih podatkov.

2.1 Crime

Ta napad izkorišča lastnosti kompresijske metode DEFLATE, ki je uporabljena v TLS do verzije 1.2 in v SSL 3.0. Ta metoda med drugim podatke skrči, tako da zamenja instance enakih nizov s kazalcem, ki kaže iz prvega enakega niza na drugega in z dolžino niza. Po kompresiji podatki niso vidni, je pa vidna velikost skompresirane poizvedbe. To pomeni, da bo končna poizvedba manjša, če je veliko nizov enakih, kot pa če so si vsi med sabo različni.
Odjemalčeve poizvedbe so lahko videti na primer takole:

```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100
101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8
```

Napadalec začne s poizvedbo, v kateri nastavi piškotek na 0 in opazuje velikost skompresirane poizvedbe:

```
POST /secretcookie=0 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100
101 Firefox/14.0.1
```

Piškotek povečuje za 1, dokler ne opazi, da je velikost skompresirane poizvedbe manjša kot prej. Ko se to zgodi, pomeni, da je bil prvi bajt piškotka pravilno ugotovljen, saj je bil ponavljajoči niz daljši. V našem primeru bi to bil bajt z vrednostjo 7. Torej ko je bil bajt nastavljen na 0, je bil ponavljajoči niz samo: secretcookie=, ko pa smo bajt nastavili na 7, pa je ponavljajoči niz postal: secretcookie=7 in je bila zato velikost končne skompresirane poizvedbe manjša. [10] [11]

2.2 Bleichenbacherjev napad

Bleichenbacherjev napad, prav tako kot POODLE, izkorišča podlaganje sporočila. Izvede se ga lahko pri nesimetrični šifri RSA, ki uporablja standard podlaganja PKCS#1, ki je prikazan na sliki 1.

Gradniki napada so: množica intervalov Mi, kriptopis c, izbrano število si, javni ključ, sestavljen iz vrednosti N in e, ter skrivni ključ, ki ga napadalec ne pozna, sestavljen iz d in N.

RSA definira zakriptirano sporočilo kot

$$c = m^e \mod N,$$

dekriptirano sporočilo pa kot:

$$m = c^d \mod N$$

Napad se lahko izvede, ker vemo, v katerem intervalu je m. Če označimo število

$$0x00 \quad 0x00 \quad 0x01 \quad \underbrace{0x00 \ldots 0x00}_{(k-2)}$$

z B kjer je k = dolžina sporočila + dolžina podloge, dobimo, da m obstaja med 2B in 3B − 1, saj je

$$2B = 0x00 \quad 0x00 \quad 0x02 \quad \underbrace{0x00 \ldots 0x00}_{(k-2)}$$

in

$$3B = 0x00 \quad 0x00 \quad 0x03 \quad \underbrace{0x00 \ldots 0x00}_{(k-2)}$$

Napadalec začne z izbiro števila s_0. S tem številom izračuna vrednost

$$s_i^e \cdot c \bmod N$$

in jo pošlje strežniku. Če strežnik odgovori, da sporočilo ustreza formatu PKCS#1, dobimo trditev:

$$2B \le s_i \cdot m \mod N \le 3B - 1$$

Če to pretvorimo v drugačno obliko, dobimo neenačbo:

$$2B \le s_i \cdot m - N \cdot r \le 3B - 1,$$

za vsak r, ki je celo število, kar smo dobili s pretvorbo formule za ostanek. Iz tega zdaj lahko izpeljemo naslednjo neenačbo:

$$\frac{2B + N \cdot r}{s_i} \le m \le \frac{3B - 1 + N \cdot r}{s_i},$$

torej smo dobili prvi interval za m. Iz prejšnje neenačbe lahko prav tako izpeljemo tole:

$$\frac{-3B + 1 + s_i \cdot m}{N} \le r \le \frac{-2B + s_i \cdot m}{N}$$

, pri čemer nastavimo vrednost m na levi strani na 2B, saj vemo, da ne more biti manjši od tega, in pa na desni na 3B − 1, saj vemo, da ne more biti večji od tega. Zdaj imamo vse potrebno, da izračunamo r. Iteriramo po vseh vrednostih, ki so znotraj intervala, kjer je r mogoč in vsako vstavimo v enačbo, kjer je izražen m. Vsakič, ko to naredimo dobimo nov interval za m. Nov interval se doda v interval množice intervalov.

Ko imamo prvi interval za m, se vrnemo na začetek, kjer napadalec pošilja izbrana števila. Ko je število pravilno, ponovimo celoten postopek, le da na koncu, preden dodamo nov interval v množico, preverimo, če ima neničelni presek s prejšnjo množico intervalov. Če je presek prazen, intervala ne dodamo, sicer pa ga dodamo. Na koncu upoštevamo novo množico intervalov in staro zavržemo.

Ko imamo v množici le še en interval oblike [a, a], vemo, da je a = m, torej smo našli dekriptirano sporočilo.

Ta napad si deli nekaj značilnosti s POODLE, na primer izkoriščanje podloge in ugibanje števil, v drugih pogledih pa se od njega precej razlikuje. [12][13]

0x00	0x02	Naključni bajti (podloga)	0x00	Čistopis

Slika 1: **Oblika sporočila s podlogo po standardu**

2.3 Asimetrična kriptografija

Da lahko začnemo kriptirati podatke z simetričnimi algoritmi kot je AES, najprej potrebujemo način dogovora o skrivnem ključu, ki ga bomo uporabljali pri šifriranju in dešifriranju. Eden od teh načinov je asimetrična kriptografija. Pri asimetrični kriptografiji imamo dva ključa: javni in skrivni ključ. Dva možna algoritma takšnega dogovora sta RSA in eliptične krivulje.

RSA je najširše uporabljena shema javne enkripcije ključa. Problem te metode je, da so potrebna zelo velika števila, da je varna. Priporočena dolžina ključa je 2048 bitov, kar je lahko za manjše, manj zmogljive naprave, problem. Da se temu problemu izognemo, lahko uporabimo eliptične krivulje.

Eliptična krivulja je definirana z enačbo

$$y^2 = x^3 + ax + b$$

in je simetrična glede na x os. Največja prednost, ki jo ima uporaba eliptičnih krivulj pred RSA je, da je osnovna operacija pri eliptičnih krivuljah seštevanje točk (angl. point addition), ki pa je zelo draga, kar pomeni, da imamo lahko veliko manjše ključe kot pri RSA. Če imamo enačbo $kP = E$ (kP kjer pomeni k-kratno seštevanje točke P samo s sabo) in poznamo vrednosti P in E, je iz tega zelo težko izračunati n. Najboljši algoritem za tak izračun, ki ga trenutno poznamo je Pollard's rho. Ta algoritem ima časovno kompleksnost $0(\sqrt{n})$, kar pomeni, da bi pri 256 bitnemu ključu k, potrebovali približno 10^{38} korakov, da bi ga ugotovili, kar pa je za današnje računalnike veliko preveč. Niso pa vse krivulje varne in jih moramo zato previdno izbrati. Na tak način lahko potem modificiramo na primer protokol Diffie-Hellman in izračunamo deljeno skrivnost. [14][15]

Protokol Diffie-Hellman z eliptičnimi krivuljami temelji na problemu diskretnega logaritma na eliptični krivulji, medtem ko navadni Diffie-Hellman temelji na problemu diskretnega logaritma. Ko dve strani (Bob in Alice) želita vzpostaviti varno komunikacijo prek javnega kanala, si morata izmenjati skupno skrivnost (angl. shared secret) tako, da je nihče ne more prestreči. Bob si zamisli število n, za katero velja $1 \leq n \leq red\ krivulje$. Izračuna $B = nP$, kjer je P javno znana točka na krivulji. Alice si zamisli število k, za katero veljajo enake omejitve kot n, in izračuna $A = kP$. Nato si izmenjata B in A. Zdaj lahko oba izračunata skupno skrivnost, Bob izračuna $A = kP$, Alice pa kB. Nekdo, ki bi izmenjavi prisluškoval, skrivnosti ne bi mogel izračunati, saj ne bi poznal vsaj ene skrivne vrednosti (n ali k). [16]

2.4 Pogoja za POODLE

Man-in-the-middle je napad, pri katerem napadalec pride med strežnik in odjemalca. Ko to uspešno naredi, lahko vidi šifrirana sporočila med njima, ne vidi pa čistopisov.

Za POODLE napad je to potrebno, saj napadalec potrebuje dostop do strežnika, ki pozna skrivni ključ in zna dešifrirati tajnopise, prav tako pa mora napadalec prestreči sporočila med odjemalcem in strežnikom. Napad se najpogosteje izvede z metodo "Spoofing". Poznamo več vrst, kot so: IP spoofing, ARP spoofing in DNS spoofing.

Pri IP spoofingu se napadalec predstavlja kot legitimna spletna stran, kar doseže s spreminjanjem IP paketov. Pri ARP spoofingu gre za povezovanje napadalčevega MAC naslova z IP naslovom legitimnega uporabnika. Ko odjemalec zdaj poskuša poslati sporočilo legitimnemu uporabniku, ga bo v resnici poslal napadalcu. Pri DNS spoofingu napadalec spremeni zapise na DNS strežniku in tako uporabnika preusmeri na svojo spletno stran, namesto na legitimno. [17]

V protoklih, ki so naprednejši od SSL 3.0 (npr. TLS 1.0), je potrebno strežnik prepričati, da za povezavo uporablja SSL 3.0, kjer bomo lahko izkoristili ranljivost POODLE.

To se doseže z izkoriščanjem **"downgrade dance"**. Downgrade dance je implementacija na nekaterih spletnih brskalnikih za lažjo kompatibilnost s starejšimi strežniki. Sproži se, če handshake ni uspel, saj odjemalec predvideva, da strežnik ne pozna protokola, s katerim odjemalec poskuša komunicirati. Nato odjemalec poskusi z nižjim protokolom in postopek ponovi. Napadalec torej v fazi handshake povzroči, da se ta ne izvede uspešno in to počne, dokler ne pride do protokola SSL 3.0, v katerem lahko izkoristi POODLE ranljivost [18].

3 SESTAVNI DELI NAPADA

3.1 AES-CBC

AES je simetrična šifra, ki se lahko uporablja v več načinih. Način, ki je pomemben za ta napad, je CBC (Cipher Block Chaining). CBC je način, v katerem AES deluje kot blokovna šifra. Ključna lastnost blokovne šifre je, da čistopis razreže na 16 bajtne bloke,

kar je potrebno pri šifriranju in dešifriranju. Pri šifriranju se nad vsakim blokom najprej izvede operacija xor (ekskluzivna disjunkcija) s prejšnjim šifriranim blokom, prvi blok pa za to operacijo uporabi IV (Initialization Vector). Vsak blok se torej pred šifriranjem pripravi takole: **encBlock[i-1] ^ ptBlock[i]**. [19] [20] Postopek dešifriranja je prikazan na sliki 2.

3.2 Podlaganje

Ker uporabljamo blokovno šifro, je potrebno zagotoviti, da je vsak blok dolg 16B, saj ne moremo izvesti xor operacije nad dvema blokoma različnih dolžin. To se doseže z dodajanjem podloge (angl. padding). V naši implementaciji uporabljamo standard PKCS#7, saj je najpogostejši standard podlaganja v AES-CBC.

Podloga po tem standardu se generira glede na to, koliko bajtov v bloku manjka do dolžine 16. Preostanek bajtov zapolni z bajti z vrednostjo števila mest, ki še manjkajo. Torej, če imamo blok, ki je dolg 13B, bodo zadnji trije bajti 0x03 0x03 0x03. Če je dolžina sporočila deljiva s 16 in ga lahko razrežemo na enako velike bloke, je treba dodati na konec še en blok dolžine 16B, ki bo imel vse bajte nastavljene na vrednost 0x10 (desetiško 16) [21].

3.3 Overitvena značka

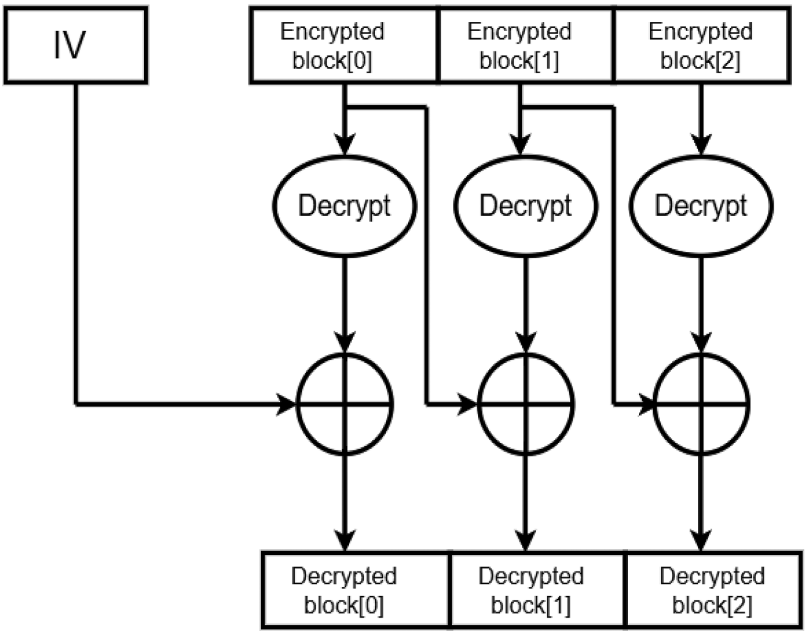
Za zagotavljanje celovitosti in avtentičnosti sporočila, se izračuna overitvena značka sporočila. To značko se nato priključi sporočilu, da jo lahko po dekripciji ponovno izračunamo in preverimo, če je enaka podani. Če je značka enaka, smo lahko prepričani, da se sporočilo ni spremenilo, v nasprotnem primeru pa avtentičnost sporočila ni več zagotovljena in ga je treba zavreči .

V naši implementaciji uporabljamo HMAC z zgoščevalno funkcijo SHA-256. Značko se izračuna pred kriptiranjem in pridruži čistopisu. Po koncu šifriranja se značko iz sporočila ponovno izračuna in primerja s tisto, ki je pritrjena na čistopis [21].

3.4 Overi-nato-šifriraj

Način overi-nato-šifriraj opisuje v kakšnem vrstnem redu se zgodi overitev sporočila. Najprej se izračuna značka iz čistopisa in se k njemu priključi. Nato sledi dodajanje podloge, ki se izračuna glede na dolžino čistopisa in značke skupaj. Vse skupaj se nato zašifrira.

Zaradi uporabe tega načina, se mora celotno sporočilo najprej dešifrirati, preveriti pravilnost podloge in šele nato preveriti veljavnost značke. To je še posebej problem pri blokovni šifri, saj imajo šifrirani bloki vpliv drug na drugega. To pomeni, da lahko spremenimo tajnopis in s tem dobimo informacije o čistopisu preden se značka preveri in strežnik sporočilo zaradi nepravilnosti zavrže. [21]



Slika 2: **Dešifriranje blokov pri AES-CBC**

Slika 3: **Struktura sporočila pri uporabi načina Overi-nato-šifriraj**

3.5 Operacija xor

Operacija xor je bitna operacija, ki vrne 0, če imata operanda oba enako vrednost, in ena, če imata različni. Operacija je komutativna in asociativna, kar je ključnega pomena pri POODLE napadu. Če torej izvedemo xor nad dvema bajtoma z isto vrednostjo, dobimo bajt 0x00, ki pa ne bo imel nobenega vpliva na katerokoli drugo vrednost v računu. Če pa operacijo izvedemo nad dvema različnima bajtoma, pa kot rezultat dobimo mesta kjer se razlikujeta (npr. **01100101 ^ 10010100 = 11110001**, saj se bajta razlikujeta na indeksih 0, 4, 5, 6 in 7, kjer je v rezultatu vrednost 1 in sta enaka na indeksih 1, 2, in 3, kjer je rezultat 0).

4 METODOLOGIJA

V programskem jeziku Java smo implementirali šifro AES-CBC, nad njo izvedli POODLE napad ter jo popravili tako, da napad ni več mogoč.

Pomembnejši knjižnici, ki smo ju uporabili pri implementaciji, sta crypto in security. Pri prvi smo uporabili Mac za overjanje sporočila in spec.SecretKeySpec, pri drugi pa SecureRandom za generiranje naključnega ključa in pa Inicializacijskega vektorja (IV).

4.1 Implementacija AES-CBC

Implementirali smo veriženje blokov po standardu CBC. Poleg tega smo implementirali tudi AES šifro s

- standardnimi operacijami:
- substitucija bajtov (angl. Byte substitution),
- zamik vrstic (angl. Shift rows),
- mešanje stolpcev (angl. Mix columns) in
- dodajanje rundnega ključa (angl. add Round key).

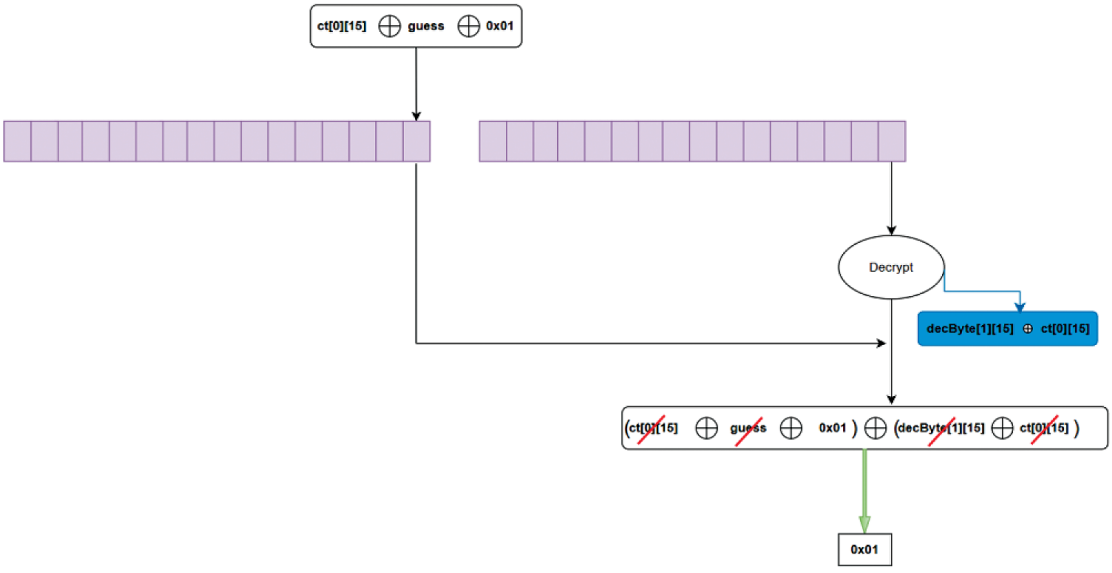
Matriki za substitucijo bajtov smo ročno zakodirali v program, enako smo naredili tudi pri matrikah za mešanje stolpcev in konstante rund.

Za overjanje sporočila smo uporabili HMAC z zgoščevalno funkcijo SHA-256 iz knjižnice crypto.

4.2 IPOODLE

Pri tem napadu ima napadalec na voljo kriptopis in strežnik, ki ga zna dešifrirati, a čistopisa ne pokaže. Čistopis je zakriptiran s šifro AES-CBC, narejen pa po načinu Overi-nato-šifriraj. Kot je opisano v poglavju AES-CBC, se nad vsakim blokom po šifriranju izvede xor s prejšnjim zakriptiranim blokom. Posledično se vsaka sprememba v določenem bloku odrazi tudi v naslednjem. Operacija XOR vrne vrednost 0, kadar sta oba bita enaka, pri tem pa velja, da se vsak bit, ki se xora z 0, ohrani (npr. **0x00 ^ 0x04 = 0x04**).

Napadalec začne s spreminjanjem zadnjega bajta predzadnjega bloka. Nad tem bajtom izvede xor z vrednostjo med 0 in 256 (0 vključeno) in z bajtom



Slika 4: **Dekriptiranje bloka pri POODLE napadu, če je zadnji bajt pravilno ugotovljen**

v vrednosti 0x01. Vrednost med 0 in 256 je vrednost, za katero napadalec ugiba, da je pravilna vrednost zadnjega bajta naslednjega bloka, 0x01 pa je vrednost podloge, ki se prilega poziciji mesta spremembe. Če je uganjena vrednost pravilna, bo strežnik vrnil napako podloge, saj se napadalčev bajt xor bajt naslednjega bloka ne bo izračunal v 0. Če pa je uganjena vrednost napačna, bo vrnil napako značke, saj je bila vrednost podloge pravilna (0x01 za zadnji bajt).

Ko je bajt pravilno ugotovljen, napadalec ponovi postopek na predzadnjem bajtu predzadnjega bloka, z razliko v vrednosti podloge, ki jo zdaj nastavi na 0x02. Popraviti je tudi potrebno prejšnji ugotovljeni bajt, da bo vrednost podloge tudi tam prišla 0x02. To lahko naredimo tako, da izvedemo xor med spremenjeno vrednostjo (**originalByte ^ guess ^ 0x01**) in (**changedByte ^ 0x01 ^ 0x02**), kjer se bosta bajta z vrednostjo 0x01 izničila, kar pomeni, da dobimo **originalByte ^ guess ^ 0x02**.

Ugotavljanje zadnjega bajta predzadnjega bloka je prikazano na sliki 4.

Ko napadalec ugotovi celoten blok, začne reševati blok pred tem. Vrednost podloge spet nastavi na 0x01, vrednosti prej spremenjenega bloka pa nastavi nazaj na originalne in shrani uganjene vrednosti v nov seznam. Ko spremeni vrednost bajta in je pripravljen na pošiljanje strežniku na dešifriranje, mora zadnji blok odstraniti, saj mora zdaj biti pravilna podloga na predzadnjem bloku in ne na zadnjem. Ker je lahko podložen le zadnji blok, mora biti blok, katerega vrednosti ugibamo, zadnji.

Težava nastane, ko je dešifriranih dovolj blokov, da ostanejo samo še trije. To je problem, saj so za pravilno dekripcijo potrebni vsaj štirje bloki. Prvi blok je vedno IV, naslednji bloki so besedilo, ki je bilo zakriptirano (število blokov je odvisno od dolžine besedila), naslednja dva bloka sta overitvena značka (v naši implementaciji uporabljamo HMAC s SHA-256, ki izračuna zgoščeno vrednost v dolžini 256 bitov, kar je 32 bajtov, torej 2 * 16) in na koncu še podloga, da dobimo dolžino deljivo s 16. Težavo se odpravi tako, da pred prvi blok dodamo (4 - numberOfRemainingBlocks) * 16B z vrednostjo 0x00. Ker imajo bajti vrednost 0, ne bodo imeli nobenega vpliva na vrednosti naslednjih blokov, bodo pa poskrbeli, da pri dekriptiranju ne bo prišlo do napake zaradi napačne dolžine.

Ko ugotovimo vrednosti vseh blokov razen prvega, je napad končan, saj je prvi blok IV, ki pa ni del sporočila, zato lahko ostane zakriptiran. Na koncu dobimo celoten dešifriran tajnopis. [21] [22]

5 ANALIZA

Implementirali smo dve rešitvi za preprečitev napada.
1. rešitev: Ena od možnosti preprečitve napada je, da imamo namesto dveh tipov napak, samo enega. Torej, če je podloga ali značka napačna, strežnik vedno vrne napako značke. Implementacija te rešitve je zelo pomembna, saj lahko napačna implementacija dopusti možnost časovnega napada. Pri časovnem napadu napadalec meri čas od trenutka, ko pošlje kriptopis strežniku, do trenutka, ko mu ta vrne napako. Če je čas med tema dvema dogodkoma manjši, lahko sklepa, da je bila podloga napačna, če pa je čas daljši, pa sklepa, da je bila podloga pravilna. Napačna implementacija te rešitve je lahko na primer takšna:

```
private static void checkPadding(byte[] pt) {
    int paddingLength = pt[pt.length - 1];
    if (paddingLength <= 0 || paddingLength > 16)
        throw new Report.InvalidTag();
    for (int i = pt.length - paddingLength; i < pt.length; i++) {
        if (pt[i] != paddingLength)
            throw new Report.InvalidTag();
    }
}
```

V prikazani implementaciji se napaka podloge sproži takoj, ko bajt ne ustreza podlogi, kar je veliko hitreje, kot če bi preverjali celotno podlogo in za tem še značko, zato je trivialno ugotoviti pravilnost podloge.
Pravilna implementacija izgleda takole:

```
private static void checkPadding(byte[] pt) {
    int paddingLength = pt[pt.length - 1];
    if (paddingLength <= 0 || paddingLength > 16)
        throwError = true;
    for (int i = pt.length - paddingLength; i < pt.length; i++) {
        if (pt[i] != paddingLength)
            throwError = true;
    }
}

private static void verifyTag(byte[] pt, byte[] tag, byte[] key) thr
ows Exception {
    final byte[] calculatedTag = createTag(pt, key);

    for (int i = 0; i < calculatedTag.length; i++) {
        if ((calculatedTag[i] ^ tag[i]) != 0)
            throwError = true;
    }
    if (throwError)
        throw new Report.InvalidTag();
}
```

static boolean throwError = false;
Pri preverjanju značke je potrebna posebna previdnost glede načina izvedbe, saj je preverjanje možno imple-

```
if (!Arrays.equals(tag, calculatedTag))
    throwError = true;

namesto:

for (int i = 0; i < calculatedTag.length; i++) {
    if ((calculatedTag[i] ^ tag[i]) != 0)
        throwError = true;
}
```

mentirati tudi na naslednji način:
Pri prvi rešitvi bi namreč lahko prišlo do časovnega napada.
2. rešitev: Še ena možna rešitev za obrambo pred POODLE napadom je, da namesto načina overi-nato-šifriraj uporabimo način šifriraj-nato-overi. Če uporabimo ta način, se vedno najprej preveri značka in šele nato tajnopis dešifriramo, kar bi preprečilo, da sploh pridemo do napake podloge, ker bi bilo že takoj ugotovljeno, da je bil tajnopis spremenjen, in bi ga zato strežnik zavrnil.

Na sliki 5 je prikazano, kateri deli kriptopisa so šifrirani pri uporabi načina Šifriraj-nato-overi. Dela, pobarvana z vijolično (čistopis in podloga), sta zakriptirana, medtem ko je del, pobarvan z zeleno

Slika 5: **Struktura sporočila pri uporabi načina Šifriraj-nato-overi**

(značka), v obliki čistopisa.

Časovna zahtevnost POODLE

Pri POODLE napadu za dešifriranje čistopisa potrebujemo največ 256 poskusov za dešifriranje enega bajta. Ker ugotavljamo vsak bajt posebej in dešifriranje enega bajta ne vpliva na dešifriranje drugega, se s povečevanjem dolžine sporočila potreben čas za izvedbo napada povečuje linearno, torej .

6 ZAKLJUČEK

V tem članku smo pokazali POODLE napad na šifro AES-CBC. Ugotovili smo, da sta kritični točki v SSL 3.0, ki naredita ta napad mogoč, način overi-nato-šifriraj in način CBC. S spremembo katere koli od teh dveh točk, napad ne bi bil več mogoč. Pokazali smo, da lahko dešifriramo celoten tajnopis, brez izjem, ne da bi poznali skrivni ključ.

Pred napadom smo se zavarovali na dva načina: strežnik je vedno vrnil napako značke in z uporabo načina šifriraj-nato-overi pri kriptiranju. Pri prvi rešitvi smo morali biti pazljivi na implementacijo, saj bi v primeru nepazljive izvedbe bil omogočen časovni napad, tako pri preverjanju pravilnosti značke kot tudi pri času vrnjene napake. Pri drugi rešitvi smo pokazali, da v načinu šifriraj-nato-overi napad ni mogoč, saj je najprej preverjena veljavnost značke, šele potem pa se začne postopek dekripcije.

Z napadom na šifro AES-CBC, ki uporablja način overi-nato-šifriraj smo uspeli dešifrirati celoten tajnopis. Pokazali smo, da ni potrebno veliko informacij za izvedbo napada. Vse kar potrebujemo je le sporočilo o tipu napake s strani strežnika (značke/podloge). Po popravkih načina šifriranja smo se napada tudi obranili, hkrati pa pazili, da implementacija ni odprla vrat za časovni napad.

Analiza je pokazala, da je ranljivost v SSL 3.0 kritična, saj napadalcu omogoča popolno dešifriranje podatkov, če pridobi dostop do strežnika za dekripcijo, kar je mogoče z izvedbo napada man-in-the-middle. Ker man-in-the-middle ni eden od enostavnejših napadov za izvesti, je tudi težko dobiti ustrezne pogoje za izvedbo POODLE, a ko enkrat te pogoje imamo, je napad trivialen.

Izpostavili smo, da je pri implementaciji zaščite potrebna posebna previdnost, saj lahko neustrezna izvedba povzroči nove ranljivosti. Pokazali smo, da je napad možen tudi pri uporabi protokolov, novejših od SSL 3.0, ali pri nepravilni implementaciji predlaganih rešitev.

Naše nadaljnje raziskave se osredotočajo na analizo različnih variacij POODLE napada na sodobnejše protokole, kot je TLS 1.3, ter na razvoj učinkovitih metod njihove preprečitve. Pravtako bi lahko raziskali podobne napade, zlasti tiste, ki izkoriščajo nepravilno ali nepopolno implementacijo zaščitnih ukrepov ter razviti mehanizme, ki bi večino tovrstnih napadov preprečile.

LITERATURA

[1] M. Ubaidullah and Q. Makki, “A Review on Symmetric Key Encryption Techniques in Cryptography,” International Journal of Computer Applications, vol. 147, no. 10, pp. 43–48, Aug. 2016, doi: 10.5120/ijca2016911203.

[2] D. Wagner and B. Schneier, “Analysis of the SSL 3.0 Protocol.”

[3] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, “Cryptographically verified implementations for TLS,” in Proceedings of the 15th ACM conference on Computer and communications security, Alexandria Virginia USA: ACM, Oct. 2008, pp. 459–468. doi: 10.1145/1455770.1455828.

[4] J. Zhou, W. Fu, W. Hu, Z. Sun, T. He, and Z. Zhang, “Challenges and Advances in Analyzing TLS 1.3-Encrypted Traffic: A Comprehensive Survey,” Electronics, vol. 13, no. 20, p. 4000, Oct. 2024, doi: 10.3390/electronics13204000.

[5] E. A. Fischer, “Cybersecurity Issues and Challenges.”

[6] J. M. Biju, N. Gopal, and A. J. Prakash, “CYBER ATTACKS AND ITS DIFFERENT TYPES,” vol. 6, no. 3, 2019.

[7] Y. Li and Q. Liu, “A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments,” Energy Reports, vol. 7, pp. 8176–8186, Nov. 2021, doi: 10.1016/j.egyr.2021.08.126.

[8] S. Gohwong, “The State of the Art of Cryptography-Based Cyber-Attacks,” Asian Crime and Society Review, vol. 6, no. 2, pp. 24–34, Jul. 2019, Accessed: Sep. 10, 2025. [Online]. Available: https://so02.tci-thaijo.org/index.php/IJCLSI/article/view/242595

[9] W. Duo, M. Zhou, and A. Abusorrah, “A Survey of Cyber Attacks on Cyber Physical Systems: Recent Advances and Challenges,” IEEE/CAA Journal of Automatica Sinica, vol. 9, no. 5, pp. 784–800, May 2022, doi: 10.1109/JAS.2022.105548.

[10] Y. Gluck, N. Harris, and A. Ngel, “BREACH: REVIVING THE CRIME ATTACK.”

[11] P. G. Sarkar and S. Fitzgerald, “ATTACKS ON SSL A COMPREHENSIVE STUDY OF BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 BIASES.”

[12] D. Bleichenbacher, “Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1,” in Advances in Cryptology — CRYPTO ’98, H. Krawczyk, Ed., Berlin, Heidelberg: Springer, 1998, pp. 1–12. doi: 10.1007/BFb0055716.

[13] H. Bock, J. Somorovsky, and C. Young, “Return Of Bleichenbacher’s Oracle Threat (ROBOT).”

[14] E. Bach, “Toward a theory of Pollard’s rho method,” Information and Computation, vol. 90, no. 2, pp. 139–155, Feb. 1991, doi: 10.1016/0890-5401(91)90001-I.

[15] V. Kapoor, V. S. Abraham, and R. Singh, “Elliptic curve cryptography,” Ubiquity, vol. 2008, no. May, pp. 1–8, May 2008, doi: 10.1145/1386853.1378356.

[16] R. Haakegaard and J. Lang, “The Elliptic Curve Diffie-Hellman (ECDH).”

[17] A. Mallik, "MAN-IN-THE-MIDDLE-ATTACK: UNDERSTANDING IN SIMPLE WORDS."
[18] E. S. Alashwali and K. Rasmussen, "What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS," in Security and Privacy in Communication Networks, vol. 255, R. Beyah, B. Chang, Y. Li, and S. Zhu, Eds., Cham: Springer International Publishing, 2018, pp. 468–487. doi: 10.1007/978-3-030-01704-0_27.
[19] National Institute of Standards and Technology (US), "Advanced Encryption Standard (AES)," National Institute of Standards; Technology (U.S.), Washington, D.C., NIST FIPS 197-upd1, May 2023. doi: 10.6028/NIST.FIPS.197-upd1.
[20] J. Daemen, "The Rijndael Block Cipher."
[21] D. Jelenc, "Varnost podatkov." Accessed: Apr. 21, 2025. [Online]. Available: https://varnost-podatkov.lem.im/6-ae-kdfs.html?print-pdf#/sec-title-slide
[22] B. Möller, T. Duong, and K. Kotowicz, "This POODLE Bites: Exploiting The SSL 3.0 Fallback."

▪

Anja Klančar je študentka na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Zanimajo jo področja kriptografije, kibernetske varnosti in umetne inteligence.

▪

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.