Uwierzytelnianie - I

Bezpieczeństwo systemów informatycznych

Marek Miśkiewicz

2026-01-04

PLAN

1. Wprowadzenie

2. Klasyczme metody uwerzytelniania

3. Perspektywa bezpieczeństwa

4. Analiza formalna protokołów uwierzytelniania

1. Wprowadzenie

Definicja uwierzytelniania

Definicja

- **Uwierzytelnianie (authentication)** – proces weryfikacji tożsamości podmiotu (użytkownika, urządzenia, procesu).
- **Autoryzacja (authorization)** – proces nadawania uprawnień po pozytywnym uwierzytelnieniu.
- Różnica:
  - *Uwierzytelnianie* odpowiada na pytanie **„kim jesteś?”**.
  - *Autoryzacja* odpowiada na pytanie **„co możesz zrobić?”**.

Rola uwierzytelniania w bezpieczeństwie

- Podstawowy element **triady CIA**:
  - Poufność (Confidentiality)
  - Integralność (Integrity)
  - Dostępność (Availability)
- Bez poprawnego uwierzytelniania:
  - poufność danych może być złamana,
  - integralność operacji zagrożona,
  - dostępność usług wykorzystana przez osoby nieuprawnione.
- Uwierzytelnianie to **brama wejściowa** do systemu – od jego jakości zależy bezpieczeństwo całej infrastruktury.

Powiązanie z kryptografią

- **Kryptografia symetryczna** – challenge–response, Kerberos.
- **Kryptografia asymetryczna** – podpisy cyfrowe, PKI, certyfikaty.
- **Funkcje skrótu** – haszowanie haseł, mechanizmy jednorazowych tokenów.
- Uwierzytelnianie korzysta z kryptografii, ale nie jest to samo co szyfrowanie.
- Cel: nie ukrycie treści, lecz **potwierdzenie tożsamości**.

2. Klasyczme metody uwerzytelniania

Hasła – podstawy

Hasło to najstarsza i wciąż najczęściej używana metoda uwierzytelniania. Opiera się na czynniku **“co wiesz”**. Użytkownik podaje tajne słowo lub ciąg znaków, a system sprawdza zgodność z przechowywanym wzorcem.

- Proste w implementacji i łatwe do zrozumienia.
- Najbardziej rozpowszechnione – praktycznie każdy system IT je wykorzystuje.
- Niestety, mają wiele ograniczeń, które czynią je podatnymi na ataki.

Przechowywanie haseł

Hasła nigdy nie powinny być przechowywane w postaci jawnej. Bezpieczne praktyki wymagają stosowania kryptograficznych funkcji skrótu oraz dodatkowych mechanizmów zabezpieczających.

- **Hashowanie**: SHA-256, bcrypt, Argon2 (rekomendacja NIST).
- **Sól (salt)**: unikalny ciąg znaków dodawany do każdego hasła → utrudnia ataki słownikowe.
- **Key stretching**: spowalnia obliczenia atakującego (PBKDF2, scrypt, Argon2).

Celem jest to, aby złamanie bazy haseł było kosztowne i czasochłonne, nawet jeśli wycieknie do atakującego.

Ataki na hasła

Hasła są atrakcyjnym celem ataków. Najczęściej spotykane scenariusze to:

- **Brute force** – próbowanie wszystkich możliwych kombinacji.
- **Ataki słownikowe** – próby najczęściej używanych haseł (np. „123456”).
- **Credential stuffing** – używanie haseł z wcześniejszych wycieków w innych serwisach.
- **Phishing** – skłonienie użytkownika, by sam podał swoje hasło.
- **Keylogging** – rejestracja naciśnięć klawiszy i przechwytywanie hasła.

Dobre praktyki

Zarówno użytkownicy, jak i administratorzy powinni stosować się do zasad bezpieczeństwa:

Użytkownicy:

- używanie menedżerów haseł,
- unikanie powtórzeń,
- stosowanie MFA.

Administratorzy:

- wymuszanie długości (>= 12 znaków),
- monitorowanie wycieków,
- blokady po nieudanych próbach logowania.

Ważne:
NIST (SP 800-63B) odradza wymuszanie częstych zmian haseł – prowadzi to do słabszych wyborów użytkowników.

Wady systemów opartych na hasłach

- Użytkownicy często tworzą słabe lub powtarzalne hasła.
- Trudność w zapamiętaniu wielu haseł → skłonność do używania jednego hasła w wielu systemach.
- Proces resetowania hasła to często najsłabsze ogniwo systemu.

Mimo że hasła są fundamentem uwierzytelniania, ich ograniczenia wymuszają rozwój metod bardziej zaawansowanych, jak MFA czy passwordless.

Challenge–Response

Challenge–Response – idea

Mechanizm **„wyzwanie–odpowiedź”** pozwala potwierdzić tożsamość bez przesyłania tajnego hasła wprost. Serwer wysyła losowe wyzwanie (*challenge*), a użytkownik oblicza odpowiedź (*response*) na podstawie sekretu.

- Chroni przed prostym podsłuchem.
- Każda sesja ma inne dane → trudniej wykorzystać przechwycone komunikaty.
- Wprowadza pojęcie **dynamicznych sekretów**.

Challenge–Response

Przykład działania

1. Serwer generuje losowy **nonce** (np. liczba losowa lub znacznik czasu).
2. Klient oblicza wartość funkcji `f(secret, nonce)`.
3. Serwer wykonuje to samo obliczenie.
4. Zgodność wyników = uwierzytelnienie.

i Adnotacja

Funkcja f może być:

- funkcją haszującą,
- operacją szyfrowania
- kluczem symetrycznym.

Challenge–Response

Zalety metody

- Sekret **nigdy nie opuszcza** urządzenia użytkownika.
- Odpowiedź ważna tylko w danym momencie (nonce).
- Ochrona przed **replay attack** (ponowne użycie komunikatu).
- Łatwy do zrozumienia i implementacji.

Challenge–Response

Ograniczenia

- Jeśli sekret jest słaby → podatność na ataki offline.
- Możliwość ataku **man-in-the-middle**, gdy kanał nie jest szyfrowany.
- Zależność od jakości funkcji kryptograficznej (hash, szyfrowanie).
- Brak ochrony przed phishingiem – użytkownik nadal może dać się oszukać.

i Adnotacja

W praktyce challenge–response najczęściej łączy się z dodatkowymi mechanizmami (TLS, certyfikaty).

Challenge–Response

Przykłady zastosowania

- **CHAP (Challenge-Handshake Authentication Protocol)** – używany w protokołach sieciowych (PPP).
- **RADIUS** – wykorzystuje wyzwania do uwierzytelniania klientów.
- **Systemy logowania offline** – np. generatory kodów jednorazowych.

Kerberos – idea i założenia

Kerberos to protokół uwierzytelniania oparty głównie na kryptografii symetrycznej i **biletach**. Centralnym elementem jest
KDC (Key Distribution Center)
z dwiema usługami:

- **AS** (Authentication Server)
- **TGS** (Ticket Granting Server).

Kerberos

Użytkownik po jednokrotnym zalogowaniu (SSO) uzyskuje dostęp do wielu usług poprzez bilety.

- Cel: silne uwierzytelnienie bez wysyłania hasła po sieci.
- Model zaufania: domena/realm i jeden KDC (lub klaster KDC).
- Kluczowa własność: **wzajemne uwierzytelnienie** klient<->serwer.

i Adnotacja

Terminologia: *principal* (tożsamość w Kerberosie), *realm* (domena), *ticket* (bilet), *authenticator* (dowód świeżości i tożsamości).

Architektura i klucze

Kerberos zarządza dwoma typami kluczy: długoterminowymi i sesyjnymi.

- **Klucze długoterminowe**
  - **`K_c`** – użytkownika (pochodny od hasła, string-to-key).
  - **`K_tgs`** – współdzielony przez TGS i KDC.
  - **`K_s`** – serwera usługi.
- **Klucze sesyjne**
  - **`K_c,tgs`** – dla klient<->TGS.
  - **`K_c,s`** – dla klient<->serwer usługi.

Bilet zawiera m.in.:
`client`, `server`, `K_c,s`, *timestamps*, *lifetime*, *flags*
i jest **szyfrowany kluczem serwera** (`K_s`), więc tylko serwer może go odczytać.

Wymiana z AS (AS-REQ / AS-REP)

Cel: uzyskać TGT (Ticket Granting Ticket), który pozwoli prosić o bilety do usług bez podawania hasła.

1. **AS-REQ:** klient → AS: `cname`, `realm`, żądane parametry, (zwykle) **pre-auth**.
2. **AS-REP:** AS → klient:
   - **TGT** zaszyfrowany `K_tgs` (nietykalny dla klienta),
   - `K_c,tgs` zaszyfrowany **kluczem klienta** `K_c`.

> **!** Ważne
>
> **Pre-Authentication (PA-ENC-TIMESTAMP):** klient szyfruje znacznik czasu kluczem z hasła. Chroni przed atakami offline na odpowiedź AS.

Wymiana z TGS (TGS-REQ / TGS-REP)

Cel: uzyskać **bilet do konkretnej usługi**.

1. **TGS-REQ:** klient → TGS:
   - TGT,
   - **authenticator** zaszyfrowany `K_c,tgs` (zawiera `cname` i **świeży time-stamp**),
   - nazwa docelowej usługi `sname`.
2. **TGS-REP:** TGS → klient:
   - **service ticket** zaszyfrowany `K_s`,
   - `K_c,s` zaszyfrowany `K_c,tgs`.

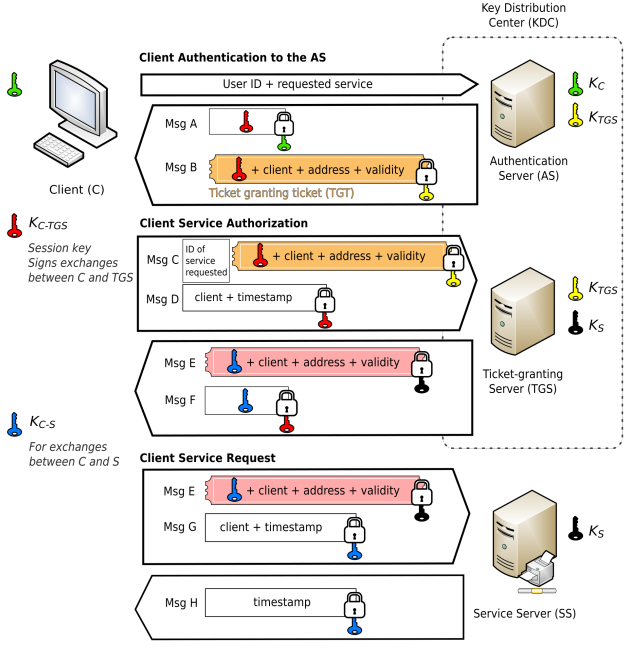
Wynik: klient ma bilet dla serwera s oraz klucz sesyjny `K_c,s`.

Użycie usługi (AP-REQ / AP-REP)

Cel: uwierzytelnienie klienta wobec serwera usługi i opcjonalne uwierzytelnienie serwera wobec klienta.

1. **AP-REQ:** klient → serwer s:
   - **service ticket** (szyfrowany `K_s`),
   - **authenticator** szyfrowany `K_c,s`.
2. **AP-REP (opcjonalny):** serwer → klient:
   - potwierdzenie i **mutual auth** z użyciem `K_c,s`.

Authenticator z aktualnym timestampem zapobiega **replay**. Bilet bez świeżego authenticatora nie wystarczy.



Kerberos - schemat działania (źródło: https://en.wikipedia.org/wiki/Kerberos_%28protocol%29)

Kerberos

Własności bezpieczeństwa

- **Brak przesyłania hasła**: hasło służy tylko do wyprowadzenia `K_c`.
- **Ochrona przed replay**: timestampy w authenticatorach + limity czasu (`clock skew`, *lifetime*).
- **Wzajemne uwierzytelnienie**: *AP-REP* potwierdza tożsamość serwera.
- **SSO**: TGT umożliwia uzyskiwanie kolejnych biletów bez ponownego logowania.

> **i** Adnotacja
>
> Zegary muszą być zsynchronizowane (NTP). Typowa tolerancja to kilka minut.

Flagi biletów i parametry

- **`forwardable`** – pozwala przekazać poświadczenia do dalszych usług.
- **`renewable`** – bilet można odświeżyć bez ponownego logowania.
- **`proxiable`** – delegacja uprawnień do pośrednika.
- **`lifetime`** – czas ważności biletu (krótkie TTL zmniejszają ryzyko nadużyć).

Dobór flag wpływa na bezpieczeństwo i wygodę w środowiskach z automatyzacją (np. joby, mikroserwisy).

Kerberos

Rozszerzenia i praktyka (AD, PKINIT, FAST)

- **Active Directory (AD):** Kerberos z rozszerzeniami (PAC, S4U, delegacje).
- **PKINIT:** logowanie do Kerberosa **z użyciem certyfikatu** zamiast hasła (X.509).
- **FAST:** tunel ochronny dla pre-auth (zmniejsza ryzyko ataków offline i MITM na fazę AS).

Zalecenia

W nowoczesnych wdrożeniach używaj tylko **AES** (wyłącz DES/RC4), włącz **pre-auth** i rozważ **PKINIT/FAST**.

Kerberos

Typowe ataki i błędy

- **Brak pre-auth** → AS-REP umożliwia **offline password guessing**.
- **Pass-the-ticket / Golden Ticket (AD)** – kradzież/TGT fałszowany po kompromitacji `krbtgt`.
- **Niesynchronizowany czas** → odrzucanie poprawnych authenticatorów.
- **Słabe hasła serwisów** → wyciek `K_s` = odczyt biletów.
- **Kompromitacja KDC** → kompromitacja całej domeny.

> **Ostrzeżenie**
>
> Kerberos centralizuje zaufanie. **Bezpieczeństwo KDC** jest krytyczne.

Kerberos

Dobre praktyki wdrożeniowe

- Wymuś **pre-auth**, używaj tylko **AES** i silnych *string-to-key*.
- Krótkie **lifetime** biletów, sensowne **renewable** i ograniczone **forwardable**.
- **NTP** w całej domenie.
- Ogranicz uprawnienia kont serwisowych, rotuj klucze, monitoruj anomalia TGT/TGS.
- W AD: chroń konto **`krbtgt`**, włącz **PAC hardening**, minimalizuj delegacje.

Kerberos w skrócie – przepływ

1. **AS**: AS-REQ <-> AS-REP → klient ma **TGT** i $K_{c,tgs}$.
2. **TGS**: TGS-REQ <-> TGS-REP → klient ma **service ticket** i $K_{c,s}$.
3. **Usługa**: AP-REQ <-> (AP-REP) → nawiązanie zaufanej sesji z $K_{c,s}$.

Efekt:

SSO, wzajemne uwierzytelnienie, brak ekspozycji hasła na sieć.

Wprowadzenie

Public Key Infrastructure (PKI) to zestaw mechanizmów, procedur i instytucji umożliwiających bezpieczne zarządzanie kluczami publicznymi i certyfikatami. Celem PKI jest dostarczenie czterech kluczowych własności bezpieczeństwa:

- **Uwierzytelnienie** – pewność, że podmiot jest tym, za kogo się podaje.
- **Poufność** – szyfrowanie przy użyciu klucza publicznego.
- **Integralność** – podpisy cyfrowe i hashe.
- **Niezaprzeczalność** – nadawca nie może wyprzeć się operacji wykonanej własnym kluczem prywatnym.

PKI jest fundamentem bezpieczeństwa Internetu (HTTPS, e-mail, podpisy elektroniczne).

Elementy PKI

- **Para kluczy**: publiczny i prywatny (asymetria).
- **Certyfikat cyfrowy** – dokument elektroniczny łączący klucz publiczny z tożsamością.
- **CA (Certification Authority)** – urząd certyfikacji, wystawia certyfikaty.
- **RA (Registration Authority)** – weryfikuje dane przed wystawieniem certyfikatu.
- **CRL (Certificate Revocation List)** i **OCSP** – mechanizmy unieważniania certyfikatów.
- **Repozytoria certyfikatów** – zbiory certyfikatów i CRL.
  Certyfikat to cyfrowy dowód tożsamości – jak paszport w świecie cyfrowym.

Proces wydania certyfikatu

1. Podmiot generuje parę kluczy (publiczny/prywatny).
2. Tworzy **CSR (Certificate Signing Request)** – zawiera klucz publiczny i dane identyfikacyjne.
3. RA/CA weryfikuje dane (np. czy domena rzeczywiście należy do wnioskującego).
4. CA podpisuje certyfikat swoim kluczem prywatnym.
5. Certyfikat staje się częścią **łańcucha zaufania**.

> **i** Adnotacja
>
> **CSR** = prośba o wystawienie certyfikatu, podpisana kluczem prywatnym podmiotu.

Łańcuch zaufania

Certyfikaty nie funkcjonują w izolacji – tworzą hierarchię.

- **Root CA** – najwyższy poziom, zaufany bezpośrednio przez systemy operacyjne/przeglądarki.
- **Intermediate CA** – pośrednie urzędy, wystawiają certyfikaty końcowe.
- **Certyfikat końcowy** – np. dla serwera HTTPS.

Użytkownik ufa serwerowi, ponieważ jego certyfikat jest podpisany przez CA, które znajduje się w liście zaufanych.

> ⚠ Ostrzeżenie
>
> To zaufanie jest **transytywne** – jeśli Root CA zostanie skompromitowany, cały łańcuch traci wiarygodność.

Struktura certyfikatu X.509

Certyfikat cyfrowy to uporządkowany dokument o określonej strukturze zgodnej ze standardem **X.509**.

Główne pola certyfikatu:

- **Version** – wersja formatu (v1, v2, v3).
- **Serial Number** – unikalny numer identyfikacyjny wystawiony przez CA.
- **Signature Algorithm** – algorytm użyty do podpisania (np. RSA-SHA256).
- **Issuer** – nazwa CA wystawiającego certyfikat (Distinguished Name).
- **Validity Period** – okres ważności: `Not Before` i `Not After`.
- **Subject** – dane podmiotu (właściciela certyfikatu): CN, O, OU, C.
- **Subject Public Key Info** – klucz publiczny + algorytm.
- **Extensions (v3)** – rozszerzenia, np.:
  - *Key Usage* – do czego można użyć klucza (podpis, szyfrowanie),
  - *Subject Alternative Name (SAN)* – alternatywne nazwy (domeny),
  - *Basic Constraints* – czy certyfikat może podpisywać inne (CA=TRUE).
- **Signature** – podpis cyfrowy CA nad całym certyfikatem.

Przykład pól certyfikatu {.example}

```
Subject: CN=example.com, O=Example Inc., C=US
Issuer: CN=Let's Encrypt Authority X3, O=Let's Encrypt, C=US
Validity: 2024-01-01 → 2024-04-01
Public Key: RSA 2048 bit
Extensions:
  - Key Usage: Digital Signature, Key Encipherment
  - Extended Key Usage: TLS Web Server Authentication
  - Subject Alternative Name: example.com, www.example.com
```

i Adnotacja

Certyfikat to **samo-opisujący się** dokument – zawiera wszystkie informacje potrzebne do weryfikacji tożsamości i klucza publicznego.

Zastosowania PKI

- **TLS/SSL (HTTPS)** – bezpieczna komunikacja w sieci.
- **Podpisy cyfrowe** – e-maile (S/MIME), kod oprogramowania.
- **VPN i sieci korporacyjne** – uwierzytelnianie użytkowników i urządzeń.
- **E-administracja** – podpis kwalifikowany, e-dowód.
- **IoT** – certyfikaty urządzeń.

PKI zapewnia spójny mechanizm tożsamości i uwierzytelniania w wielu dziedzinach.

Zalety i ograniczenia

Zalety

- Wysoki poziom bezpieczeństwa dzięki asymetrii (klucz prywatny nie musi być przekazywany).
- Uniwersalność i skalowalność – szerokie zastosowanie.
- Możliwość automatyzacji (np. Let's Encrypt).

Ograniczenia

- **Złożoność** wdrożenia i utrzymania.
- Zaufanie skupione w CA (kompromitacja CA = globalne zagrożenie).
- Problem z unieważnianiem (CRL, OCSP bywają opóźnione lub zawodne).
- Historia incydentów: DigiNotar (2011) → fałszywe certyfikaty dla *.google.com*.

3. Perspektywa bezpieczeństwa

Perspektywa bezpieczeństwa – wprowadzenie

Uwierzytelnianie to „brama wejściowa” do systemu. Nawet najlepsze mechanizmy kryptograficzne można obejść, jeśli atakujący znajdzie słabość w implementacji albo zmanipuluje użytkownika.

- Typowe ataki celują zarówno w technologię, jak i w **czynnik ludzki**.
- Dlatego analiza bezpieczeństwa musi obejmować:
  - ataki na protokoły i ich implementacje,
  - ataki socjotechniczne (phishing),
  - skuteczność mechanizmów obronnych.

Typowe ataki na systemy uwierzytelniania

- **Phishing** – fałszywe strony logowania, SMS-y (smishing), telefony (vishing).
- **Replay attack** – ponowne użycie poprawnych danych uwierzytelniających (np. tokenu).
- **Credential stuffing** – wykorzystanie wycieków haseł z innych serwisów.
- **MITM (Man-in-the-Middle)** – przechwycenie i modyfikacja komunikacji.
- **Brute force i słownikowe** – próby zgadywania haseł.
- **Ataki na biometrię** – fałszywe odciski palców, zdjęcia twarzy.

i Adnotacja

Wiele z tych ataków nie uderza w algorytmy kryptograficzne, ale w słabości implementacyjne i zachowania użytkowników.

Obrona i dobre praktyki

- **Hasła**
  - Bezpieczne funkcje haszujące (Argon2, bcrypt).
  - Monitorowanie wycieków, menedżery haseł.
- **Protokoły**
  - Szyfrowanie transportowe (TLS).
  - Nonce i znaczniki czasu (ochrona przed replay).
  - Certyfikaty i podpisy cyfrowe.
- **Organizacje**
  - MFA jako standard w systemach krytycznych.
  - Monitorowanie i alertowanie nietypowych logowań.
  - Szkolenia z rozpoznawania phishingu.

Case study 1 – Phishing (Google, 2017)

- Miliony użytkowników Gmaila otrzymały fałszywe zaproszenie do „Google Docs”.
- Link prowadził do podrobionej strony logowania Google.
- Użytkownicy sami podali dane napastnikom.
- **Czynnik ludzki**: ufność wobec pozornie wiarygodnej wiadomości.
- **Reakcja**: Google wprowadziło dodatkowe powiadomienia i mechanizmy wykrywania phishingu.

Źródła:

- https://www.wired.com/story/google-docs-scams-threat-phishing/
- https://www.lindabury.com/firm/insights/google-docs-incident-provides-real-life-example-dangers-phishing.html
- https://www.tripwire.com/state-of-security/google-docs-phishing-scam-win-security-awareness
- https://auth0.com/blog/all-you-need-to-know-about-the-google-docs-phishing-attack/

Case study 2 – Credential stuffing (LinkedIn, 2012 → 2016)

- Wycieka 6,5 mln haseł, później okazuje się, że ponad 100 mln rekordów.
- Dane trafiają do obiegu i są używane w innych serwisach.
- Użytkownicy często stosują **to samo hasło** w wielu miejscach.
- **Efekt domina**: włamanie do jednego serwisu ułatwia ataki na inne.
- **Wniosek**: MFA i monitorowanie wycieków są konieczne.

Źródła:

- Opis incydentu LinkedIn 2012/2016 w serwisie Have I Been Pwned (skala wycieku i konsekwencje)
- Oficjalny komunikat LinkedIn po wykryciu większego wycieku w 2016
- Wikipedia – szczegóły wycieku, liczba rekordów oraz efekt domina „credential stuffing”
- Analiza „efektu domina” (credential stuffing) na podstawie LinkedIn oraz innych incydentów
- Artykuł FreshSec: przegląd skutków nadużycia haseł LinkedIn i ataków bazujących na credential stuffing
- Omówienie problemu wtórnego wykorzystania skradzionych haseł przez cyberprzestępców – Law Society
- BBC – informacja o skali wycieku oraz odsprzedaży bazy haseł
- Enzoic – przykład efektu domina: przejęcie kont Marka Zuckerberga i innych przez credential stuffing na podstawie LinkedIn

Case study 3 – MITM w sieciach Wi-Fi

- Otwarta sieć Wi-Fi „Free_WiFi” w kawiarni.
- Użytkownicy logują się np. do bankowości.
- Napastnik przechwytuje i modyfikuje ruch.
- Obrona: TLS/HTTPS, weryfikacja certyfikatów, VPN.
- **Pytanie do studentów**: czy sprawdzacie, do jakiej sieci się łączycie?

Linki:

- https://littlebigtech.co.uk/using-public-wifi-for-banking-apps-heres-why-you-should-stop-immediately/
- https://www.startupdefense.io/cyberattacks/ssl-tls-mitm
- https://specopssoft.com/blog/man-in-the-middle-mitm-guide/

Wnioski z case study

- Nawet globalne firmy (Google, LinkedIn) i zwykłe sytuacje (kawiarnia) pokazują, że **atak najczęściej uderza w człowieka**.
- Technologia bez świadomości użytkownika jest niewystarczająca.
- W praktyce bezpieczeństwo to połączenie:
  - silnych protokołów,
  - dobrych procedur,
  - świadomych użytkowników.

4. Analiza formalna protokołów uwierzytelniania

Po co analizować protokoły formalnie?

Projektowanie protokołów uwierzytelniania jest trudne – pozornie poprawny protokół może zawierać subtelne błędy bezpieczeństwa.

Problemy w praktyce:

- **Błędy logiczne** – protokół działa, ale nie gwarantuje założonych własności.
- **Ataki nie uwzględnione przez projektantów** – np. replay, MITM, reflection attacks.
- **Błędy implementacyjne** – różnica między specyfikacją a kodem.

Ostrzeżenie

Historia kryptografii zna wiele przykładów protokołów, które wydawały się bezpieczne, a później znaleziono w nich luki (np. Needham-Schroeder Public Key Protocol).

Po co analizować protokoły formalnie?

Dlaczego formalizm?

- **Precyzja** – jednoznaczne określenie, co protokół ma gwarantować.
- **Systematyczność** – metodyczne sprawdzenie wszystkich możliwych scenariuszy.
- **Automatyzacja** – narzędzia mogą wykryć błędy, których człowiek by nie zauważył.
- **Dowód poprawności** – formalne potwierdzenie, że protokół spełnia wymagania bezpieczeństwa.

i Adnotacja

Analiza formalna nie zastępuje testów praktycznych, ale **uzupełnia** je, wykrywając błędy na poziomie logiki protokołu.

BAN Logic – podstawowa idea

BAN Logic (Burrows-Abadi-Needham, 1989) to jeden z pierwszych systemów formalnych do weryfikacji protokołów uwierzytelniania.

Cel:

Udowodnić, że po wykonaniu protokołu strony **wierzą** w określone fakty (np. świeżość klucza, tożsamość partnera).

Podstawowe pojęcia:

- **P believes X** – podmiot P wierzy w stwierdzenie X.
- **P sees X** – P otrzymał wiadomość zawierającą X.
- **P said X** – P kiedyś wysłał X.
- **fresh(X)** – X jest świeże (nie replay).
- **P controls X** – P ma autorytet nad X.

BAN Logic – przykład analizy protokołu

Prosty protokół uwierzytelniania:

```
1. A → B: Na                    (A wysyła nonce)
2. B → A: {Na, Nb, B}K_AB       (B odpowiada zaszyfrowane)
3. A → B: {Nb}K_AB              (A potwierdza)
```

gdzie K_AB to współdzielony klucz, Na, Nb to świeże liczby losowe (nonce).

Cel analizy:

Czy po zakończeniu protokołu **A wierzy, że komunikuje się z B**?

BAN Logic – przykład analizy protokołu

Założenia początkowe:

1. **A believes A ↔ B** – A wierzy, że K_AB jest dobrym kluczem współdzielonym z B.
2. **B believes A ↔ B** – B wierzy, że K_AB jest dobrym kluczem współdzielonym z A.
3. **A believes fresh(Na)** – A wierzy, że Na jest świeże.
4. **B believes fresh(Nb)** – B wierzy, że Nb jest świeże.

Krok 2 – co A otrzymuje:

A widzi: {Na, Nb, B}K_AB

Zastosowanie reguły odszyfrowywania:

- Jeśli A wierzy, że K_AB jest współdzielony tylko z B,
- i A widzi wiadomość zaszyfrowaną K_AB zawierającą Na,
- i A wierzy, że Na jest świeże,

To: A believes B said (Na, Nb, B)

A wierzy, że **B niedawno wysłał** tę wiadomość (zawiera świeże Na → nie replay).

BAN Logic – przykład analizy protokołu

Wnioskowanie:

Ponieważ:

- A rozpoznaje własne świeże Na w odpowiedzi,
- wiadomość jest zaszyfrowana kluczem K_AB, który tylko B zna,
- Nb jest świeże (nowe),

Wniosek: A believes B believes (A ↔ B)

A ma pewność, że komunikuje się z autentycznym B, a nie z atakującym.

Co BAN Logic wykrył:

- Protokół spełnia cel uwierzytelnienia A wobec B.
- Świeżość (Na, Nb) chroni przed replay attack.
- Współdzielony klucz gwarantuje, że tylko B mógł stworzyć odpowiedź.

BAN Logic – ograniczenia

Zalety:

- Pierwszy **formalny model** uwierzytelniania.
- Prostota i intuicyjność.
- Wykrył błędy w znanych protokołach (Needham-Schroeder).

Wady:

- Nie uwzględnia wszystkich typów ataków (np. reflection, type flaw).
- Idealizacja protokołów – uproszczenia mogą pominąć istotne szczegóły.
- Nie analizuje poufności ani integralności danych.

i Adnotacja

BAN Logic był punktem wyjścia – dzisiejsze narzędzia są znacznie bardziej zaawansowane.

Nowoczesne narzędzia weryfikacji

ProVerif

- Automatyczne narzędzie do weryfikacji protokołów kryptograficznych.
- Oparte na **logice Horna** i **rachunku π**.
- Analizuje:
  - uwierzytelnianie,
  - poufność,
  - anonimowość.
- Używane do analizy TLS, Signal Protocol, 5G AKA.

Zalety: pełna automatyzacja, obsługa złożonych protokołów.
Wady: może nie znaleźć dowodu (false negatives).

Nowoczesne narzędzia weryfikacji

Tamarin Prover

- Narzędzie oparte na **logice wielosortowej** i **teorii przepisywania**.
- Weryfikuje:
  - własności uwierzytelniania,
  - forward secrecy,
  - post-compromise security.
- Stosowane w analizie TLS 1.3, Noise Protocol, 5G.

Zalety: elastyczność, wsparcie dla skomplikowanych własności.
Wady: wymaga większej wiedzy od użytkownika, czasem potrzebna interakcja.

Nowoczesne narzędzia weryfikacji

Inne narzędzia:

- **Verifpal** – uproszczone narzędzie dla praktyków (mniej formalności).
- **CryptoVerif** – weryfikacja w modelu obliczeniowym (nie symbolicznym).
- **Scyther** – szybka analiza protokołów, GUI.

i Adnotacja

Wybór narzędzia zależy od:

- typu protokołu,
- wymaganych własności,
- poziomu automatyzacji.

Praktyczne zastosowania analizy formalnej

Przykłady sukcesów:

- **TLS 1.3** – formalnie zweryfikowany (ProVerif, Tamarin).
- **Signal Protocol** – dowód bezpieczeństwa end-to-end.
- **WireGuard VPN** – formalna analiza w Tamarin.
- **5G AKA** – wykryto i naprawiono luki dzięki ProVerif.

Korzyści:

- Wykrycie błędów **przed** wdrożeniem.
- Zwiększenie **zaufania** do protokołu.
- **Dokumentacja** założeń bezpieczeństwa.

Analiza formalna – podsumowanie

- Formalna weryfikacja protokołów uwierzytelniania to **niezbędny element** nowoczesnego projektowania bezpieczeństwa.
- BAN Logic był pionierem, ale dzisiejsze narzędzia (ProVerif, Tamarin) są znacznie potężniejsze.
- Analiza formalna **nie zastępuje** testów praktycznych, ale je **uzupełnia**.
- Kluczowe protokoły (TLS, Signal) przeszły formalną weryfikację – to standard w krytycznych systemach.

! Ważne

Zasada: Jeśli projektujesz protokół uwierzytelniania – zweryfikuj go formalnie **przed** wdrożeniem.