0 Uwierzytelnienie i autoryzacja

Czym jest uwierzytelnianie?

Uwierzytelnianie to potwierdzenie, że jesteśmy tym, za kogo się podajemy, czyli inaczej to ujmując, potwierdzenie swojej tożsamości. Najczęściej możemy je spotkać podczas logowania do dowolnego systemu, gdzie:

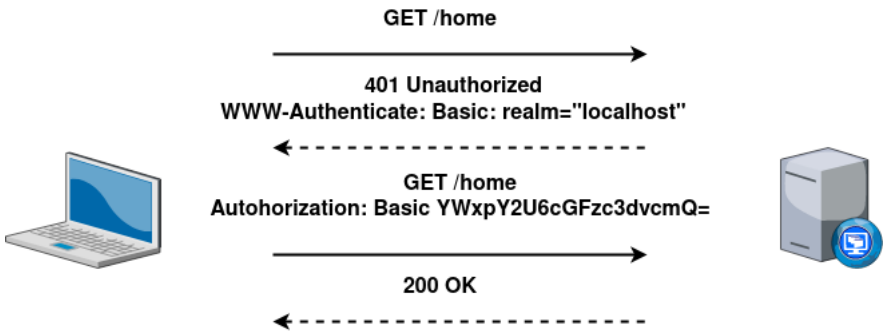
- Login jest wartością, która określa, za kogo podajemy się w danym systemie. Tak samo, jak w życiu jesteśmy identyfikowani przez Imię i Nazwisko, tak w danym systemie identyfikowani jesteśmy właśnie przez login (lub inny tego typu element, który jest indywidualny dla danego systemu – np. ID klienta w niektórych bankach itp.). Często również nasz login powiązany jest z naszym imieniem i nazwiskiem.
- Hasło to tajny ciąg znaków, który jest znany tylko przez nas. To właśnie hasło pozwala nam potwierdzić, że jesteśmy faktycznie tym, za kogo się podajemy, czyli w przypadku systemu potwierdzić, że konto identyfikowane za pomocą wpisanego przez nas loginu, należy faktycznie do nas.

Czym jest autoryzacja?

Autoryzacja to proces określania uprawnień danego podmiotu. Inaczej mówiąc autoryzacja pozwala na stwierdzenie czy dany podmiot (np. osoba) posiada dostęp do danego zasobu (np. folderu). **Autoryzacja występuje zawsze po uwierzytelnieniu**. Rozumiejąc oba pojęcia wydaje się to oczywiście bardzo logiczne – głównym celem autoryzacji jest kontrola dostępu, a nie można przecież określić, czy dany podmiot ma dostęp do danego obiektu, jeżeli nie wiedząc kim właściwie jest dany podmiot, a będąc bardziej precyzyjnym, czy jest tym za kogo się podaje.

Basic Authentication

Jedną z najprostszych metod ochrony dostępu jest uwierzytelnianie użytkowników za pomocą nazwy i hasła. Protokół HTTP obsługuje prostą metodę uwierzytelniania użytkowników nazwaną Basic Authentication. Jeżeli serwer HTTP wymaga uwierzytelnienia użytkownika, wtedy odpowiedź na żądanie HTTP zawiera kod zwrotny `401` (`Authorization Required`), oznaczający, że klient HTTP powinien zażądać od użytkownika wprowadzenia nazwy i hasła. W takiej sytuacji klient HTTP pobiera od użytkownika jego nazwę i hasło, łączy je w jeden łańcuch znakowy, koduje za pomocą algorytmu `Base64`, a następnie zapisuje w polu nagłówka `Authorization` ponowionego żądania HTTP. Jeżeli wynik weryfikacji nazwy/hasła użytkownika przez serwer HTTP jest negatywny, serwer HTTP ponownie przesyła kod zwrotny `401` (`Authorization Required`) - aż do skutku. Wprowadzone przez użytkownika dane uwierzytelniające są przez klienta HTTP automatycznie dołączane do kolejnych żądań HTTP. Należy zauważyć, że metoda HTTP `Basic Authentication` nie zapewnia ochrony przekazywanych danych uwierzytelniających. Kodowanie Base64 jest odwracalne, dzięki czemu ewentualny podsłuch połączenia HTTP umożliwia intruzowi przejęcie nazwy i hasła użytkownika. Pewniejsze metody uwierzytelniania użytkowników opierają się na formularzach HTML, których dane są przekazywane za pośrednictwem szyfrowanego protokołu HTTPS.



Rysunek 1: Przykład działania `Basic Authentication`

Session-cookie Authentication

Uwierzytelnianie za pomocą ciasteczka (cookie) w protokole HTTP to mechanizm, który umożliwia serwerom kontrolowanie dostępu użytkowników do zasobów na stronie poprzez przypisywanie im unikalnych identyfikatorów sesji w postaci ciasteczek. Ciasteczka to niewielkie fragmenty danych przechowywane w przeglądarce użytkownika, które są przesyłane wraz z każdym żądaniem HTTP (w nagłówku) do serwera. Mechanizm ten wprowadza pewną formę stanu w środowisku bezstanowym protokołu HTTP, co umożliwia trwałe przechowywanie informacji o sesji użytkownika.

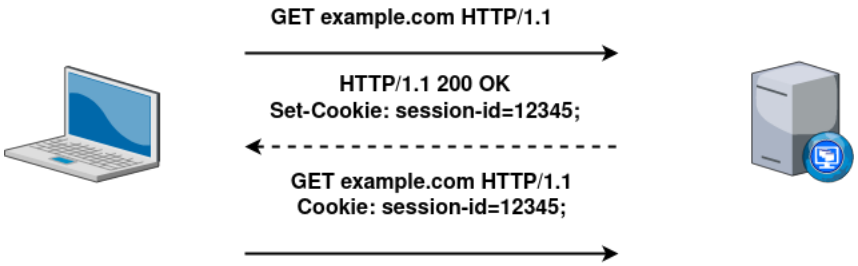
Proces uwierzytelniania za pomocą ciasteczka w protokole HTTP rozpoczyna się od momentu, gdy użytkownik próbuje uzyskać dostęp do chronionych zasobów na stronie internetowej. W odpowiedzi na to żądanie, serwer internetowy wysyła stronę logowania, która zawiera formularz, w którym użytkownik wprowadza swoje dane uwierzytelniające, takie jak nazwa użytkownika i hasło.

Po wypełnieniu formularza, użytkownik wysyła go na serwer, zawierając wprowadzone dane uwierzytelniające. Serwer następnie sprawdza podane dane w bazie danych użytkowników. Jeśli dane są poprawne, serwer generuje unikalny identyfikator sesji, który może być używany do identyfikacji użytkownika podczas tej sesji.

Ten identyfikator sesji jest umieszczany w formie ciasteczka i przesyłany w nagłówku odpowiedzi HTTP z powrotem do przeglądarki użytkownika. Przeglądarka odbiera ciasteczko i przechowuje je lokalnie, co pozwala na późniejsze wykorzystanie.

Każde kolejne żądanie HTTP wysłane przez użytkownika do serwera automatycznie dołącza ciasteczko w nagłówkach żądania. Serwer, odbierając takie żądanie, sprawdza zawartość ciasteczka. Jeśli jest ono poprawne i ważne, serwer identyfikuje sesję użytkownika, co pozwala na dostęp do chronionych zasobów.

Ważne jest, aby zabezpieczyć ciasteczka przed potencjalnymi atakami, takimi jak kradzież sesji (session hijacking) lub wstrzykiwanie skryptów (XSS). Sesje użytkownika mogą być zakończone poprzez wylogowanie się lub automatycznie po upływie pewnego czasu bez aktywności. W przypadku wylogowania, serwer usuwa identyfikator sesji lub ustawia ciasteczko na nieprawidłową wartość, uniemożliwiając dalszy dostęp do chronionych zasobów.



Rysunek 2: Przykład działania `Session-cookie Authentication`

Token-based Authentication

Uwierzytelnianie za pomocą tokenów to proces, w którym użytkownik udowadnia swoją tożsamość przy użyciu specjalnego tokena, który jest bezpiecznym i samozawierającym się mechanizmem uwierzytelniania. JWT (JSON Web Token) jest popularnym standardem do tworzenia takich tokenów, a proces uwierzytelniania na ich podstawie jest powszechnie stosowany w aplikacjach internetowych. Proces uwierzytelniania opiera się na generowaniu, przesyłaniu i weryfikacji tokenów JWT. Dzięki temu, aplikacje mogą bezpiecznie kontrolować dostęp do swoich zasobów.

1. Klient loguje się, dostarczając dane uwierzytelniające (np. nazwa użytkownika i hasło).
2. Serwer uwierzytelnia klienta i generuje JWT.
3. JWT zawiera trzy części: nagłówek, ładunek (dane) i podpis.
4. Nagłówek zawiera informacje o typie tokenu i algorytmie podpisu.

5. Ładunek zawiera informacje o tożsamości użytkownika i uprawnieniach.
6. Podpis jest tworzony poprzez zastosowanie algorytmu kryptograficznego do nagłówka i ładunku przy użyciu klucza prywatnego.
7. Klient otrzymuje JWT i może go przechowywać.
8. Klient przesyła JWT w każdym żądaniu (w nagłówku) wysyłanym do serwera.
9. Serwer dekoduje JWT, weryfikuje podpis, i jeśli jest poprawny, uwierzytelnia klienta.

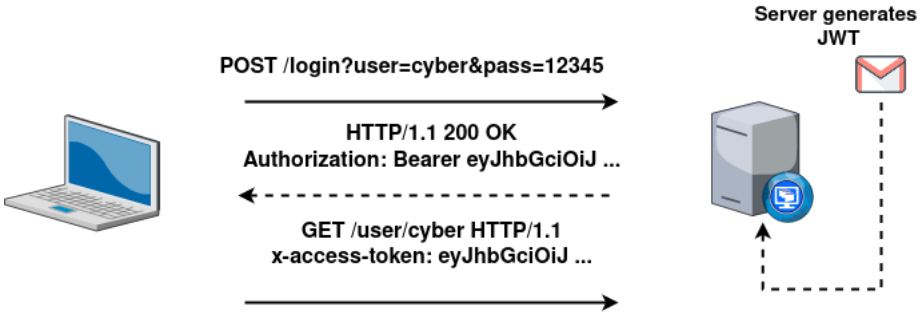
JSON Web Token (JWT) można wykorzystać na różne sposoby do obsługi uwierzytelniania i autoryzacji w aplikacjach internetowych. Dwa powszechne metody to przechowywanie JWT w pliku cookie oraz przesyłanie go w nagłówku autoryzacyjnym. Każda z tych metod ma swoje własne zalety i kwestie do rozważenia:

JWT w Pliku Cookie:

- JWT jest przechowywany jako plik cookie w przeglądarce użytkownika.
- Zalety:
  - Pliki cookie automatycznie są dołączane do każdego żądania HTTP, ułatwiając uwierzytelnianie w kolejnych żądaniach.
  - Przeglądarka zapewnia ochronę przed atakami CSRF, jeśli atrybuty `HttpOnly` i `Secure` są ustawione dla pliku cookie.
- Kwestie do rozważenia:
  - Pliki cookie mają ograniczenia co do rozmiaru (zazwyczaj około 4 KB), co może stanowić problem, jeśli JWT zawiera dużo danych.
  - Pliki cookie są narażone na ataki typu cross-site scripting (XSS), jeśli nie są odpowiednio zabezpieczone.
  - Należy starannie ustawić odpowiednie flagi (`HttpOnly`, `Secure`, `SameSite`) dla pliku cookie, aby poprawić bezpieczeństwo.

JWT w Nagłówku Autoryzacyjnym:

- JWT jest przesyłany w nagłówku `Authorization` żądania HTTP.
- Zalety:
  - To podejście jest bardziej wszechstronne i może być używane z różnymi typami klientów, w tym przeglądarkami i klientami spoza przeglądarki (np. aplikacje mobilne, interfejsy API).
  - JWT może być dołączany do niestandardowych nagłówków (np. token `Bearer`) i nie jest ograniczony przez rozmiar pliku cookie.
  - Pozwala na dokładniejszą kontrolę nad tym, jak token jest obsługiwany i zarządzany po stronie klienta.
- Kwestie do rozważenia:
  - Wymaga ręcznego dołączania tokenu do każdego żądania. Konieczne jest zautomatyzowanie dołączania tokenu do każdego wywołania interfejsu API.
  - Dodatkowe kwestie bezpieczeństwa są wymagane, aby zapewnić ochronę przed atakami CSRF, ponieważ przeglądarka nie obsługuje tego automatycznie.



Rysunek 3: Przykład działania `Token-based Authentication`

Linki

- https://security.stackexchange.com/questions/248195/what-are-the-advantages-of-using-jwt
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication
- https://cert.pl/hasla/
- https://cert.pl/uploads/2022/01/hasla/resources/wordlist_pl.zip
- https://jwt.io/introduction
- https://jwt.io/
- https://owasp.org/www-community/attacks/Session_fixation
- https://owasp.org/www-community/HttpOnly
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies
- https://devszczepaniak.pl/komunikacja-http-w-javascript/

Zadania

0.1 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch0-ex01:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `1101` działa serwer obsługujący protokół HTTP w wersji 1.1. Przy użyciu narzędzia `cURL` wyślij do serwera zapytanie, za pomocą którego zalogujesz się na serwer. Rozważ 2 przypadki:

- zaloguj się używając prawidłowego loginu i hasła (`kali` / `orange`)
- zaloguj się używając nieprawidłowego loginu i hasła

Jaki rodzaj uwierzytelniania oferuje serwer?

0.2 Używając pythonowego modułu `Scapy`, napisz program, przy użyciu którego przeanalizujesz plik `ex02.pcap`. Odpowiedź na poniższe pytania:

- Na jaką stronę wchodził użytkownik? Podaj nazwę i adres IP.
- Z jakiego adresu IP użytkownik łączył się do serwera?
- Jaki rodzaj uwierzytelniania obsługuje serwer?
- Czy użytkownikowi udało się zalogować? Jakie hasło / hasła podawał?

0.3 Poniżej znajduje się request protokołu HTTP/1.1 w zapisie szesnastkowym. Wiedząc, że znak `\r` w zapisie szesnastkowym to `0x0d`, natomiast `\n` to `0x0a` oraz wiedząc, że w zapisie szesnastkowym jedna cyfra reprezentuje 4 bity, napisz program, w którym sprawdzisz, jaki rodzaj autoryzacji został wykorzystany w pakiecie, oraz jakie hasło i nazwę użytkownika podawał użytkownik podczas uwierzytelnienia. Aby rozwiązać zadanie, napisz program w języku Python.

```
474554202f7769726573686172 6b2d6c6162732f70726f7465637465645f70616765732f485454502d7769726573686172 6b2d
66696c65352e68746d6c20485454502f312e310d0a486f73743a20676169612e63732e756d6173732e6564750d0a436f6e6e65
6374696f6e3a206b6565702d616c6976650d0a43616368652d436f6e74726f6c3a206d61782d6167653d300d0a417574686f72
697a6174696f6e3a204261736963206347467a637a707361573531654 84a765932747a0d0a557067726164652d496e73656375
72652d52657175657374733a20310d0a557365722d4167656e743a204d6f7a696c6c612f352e3020285831313b204c696e7578
207838365f363429204170706c655765624b69742f3533372e333620284b48544d4c2c206c696b65204765636b6f2920436872
6f6d652f3131342e302e302e3020536166617269 2f3533372e33360d0a4163636570743a20746578742f68746d6c2c6170706c
69636174696f6e2f7868746d6c2b786d6c2c6170706c69636174696f6e2f786d6c3b713d302e392c696d6167652f617669662c
696d6167652f776562702c696d6167652f61706e672c2a2f2a3b713d302e382c6170706c69636174696f6e2f7369676e65642d
65786368616e67653b763d62333b713d302e370d0a4163636570742d456e636f64696e673a20677a69702c206465666c617465
0d0a4163636570742d4c616e67756167653a20706c2d504c2c706c3b713d302e392c656e2d55533b713d302e382c656e3b713d
302e370d0a0d0a
```

0.4 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch0-ex04:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `1104` działa serwer obsługujący protokół HTTP w wersji 1.1. Wiedząc, że dostęp do serwera jest zabezpieczony hasłem, oraz, że zarówno login jak i hasło pochodzą z bazy polskich haseł opublikowanej przez CERT Polska, zaloguj się do serwera.

0.5 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch0-ex05:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `1105` działa serwer obsługujący protokół HTTP w wersji 1.1. Przy użyciu narzędzia `cURL` wyślij do serwera zapytania, za pomocą których:

- zalogujesz się na serwer
- pobierzesz zawartość strony `http://127.0.0.1:1105/dashboard.php`
- wylogujesz się z serwera

Jaki rodzaj uwierzytelniania oferuje serwer?

0.6 Rozwiąż zadanie **0.5** za pomocą skryptu w języku Python:

- Użyj modułu `requests`
- Użyj gniazd (moduł `socket`)

0.7 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch0-ex07:latest`), uruchom serwer HTTP. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `1107` działa serwer obsługujący protokół HTTP w wersji 1.1. Serwer wspiera uwierzytelnianie za pomocą tokenów JWT oraz udostępnia poniższe endpointy:

- `http://127.0.0.1:1107/signup`
- `http://127.0.0.1:1107/login`
- `http://127.0.0.1:1107/user`

Używając narzędzia `cURL` zarejestruj nowego użytkownika, zaloguj się na nowo utworzone konto, a następnie wyświetl informacje o wszystkich użytkownikach zarejestrowanych na serwerze (opcja dostępna tylko dla zalogowanych użytkowników). Ze względu na prostotę serwera, można na nim zarejestrować jedynie użytkownika o podanych niżej danych:

```
user_data = {
    'name': 'John',
    'email': 'john@example.com',
    'password': 'password123'
}
```

0.8 Rozwiąż zadanie **0.7** za pomocą skryptu w języku Python:

- Użyj modułu `requests`
- Użyj gniazd (moduł `socket`)

0.9 *`memos` is a privacy-first, lightweight note-taking service. Easily capture and share your great thoughts. In version `0.12.2`, `memos` is vulnerable to Improper Access Control*

(a) Zapoznaj się z opisem podatności Improper Access Control (CWE-284: Improper Access Control).

(b) Korzystając z przygotowanego obrazu Dockerowego (mazurkatarzyna/bsk-book-p1-ch0-ex09:latest), uruchom serwer `memos`.

(c) Zarejestruj w aplikacji 3 użytkowników: `root`, `student`, `bsk`.

(d) Wiedząc, że aplikacja do uwierzytelniania użytkowników wykorzystuje JWT, będąc zalogowanym jako użytkownik `student`, zmień nazwę użytkownika `root`, oraz nazwę użytkownika `bsk`.

0.10 Korzystając z przygotowanego obrazu Dockerowego (`mazurkatarzyna/bsk-book-p1-ch0-ex010:latest`), uruchom aplikację `rdiffweb`. Po uruchomieniu obrazu, pod adresem IPv4 `127.0.0.1` na porcie TCP o numerze `1110` działa serwer obsługujący aplikację `rdiffweb` służącą do zarządzania backupami Linuksa przez interfejs webowy.

(a) Jako administrator, dodaj do aplikacji swój klucz SSH (wygeneruj przykładowy klucz SSH).

(b) Wiedząc, że aplikacja do uwierzytelniania wykorzystuje `Cookies` oraz jest podatna na Session Fixation, wyloguj się z konta administratora, a następnie wykorzystaj narzędzie `cURL` do pobrania zawartości stron: `http://127.0.0.1:1110/admin/sysinfo`, `http://127.0.0.1:1110/prefs/general` oraz usunięcia klucza SSH administratora.

(c) Zaloguj się na konto administratora, aby sprawdzić, czy klucz SSH został usunięty.

0.11 *`memos` is a privacy-first, lightweight note-taking service. Easily capture and share your great thoughts. In version `0.12.2`, `memos` is vulnerable to Cross-Site Request Forgery (CSRF)* (CWE-352).

(a) Zapoznaj się z opisem podatności Cross-Site Request Forgery (CSRF) (CWE-352).

(b) Korzystając z przygotowanego obrazu Dockerowego (mazurkatarzyna/bsk-book-p1-ch0-ex011:latest), uruchom serwer `memos`.

(c) Zarejestruj w aplikacji 3 użytkowników: `root`, `student`, `bsk`.

(d) Wiedząc, że aplikacja posiada endpoint `/o/get/image?url=`, który odpowiada za pobieranie obrazków z zewnętrznych źródeł, oraz, że jest on podatny na CSRF, usuń wybranego użytkownika.