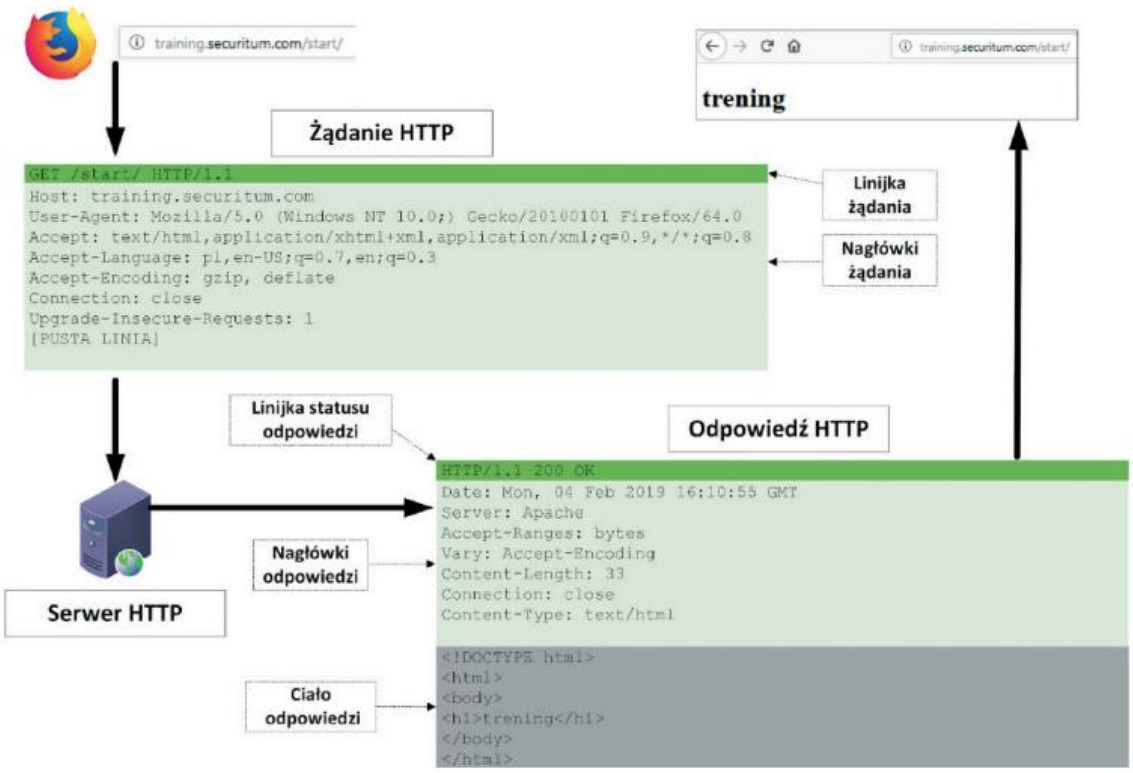
Bezpieczeństwo Aplikacji Internetowych

Podstawy protokołu http

1. Podstawowa komunikacja HTTP

Najczęściej spotkamy się z komunikacją HTTP z wykorzystaniem protokołu TCP (choć czasem można spotkać wykorzystanie protokołu UDP4). Komunikacja HTTP realizowana jest poprzez wysłanie żądania (ang. request) do serwera, który następnie generuje odpowiedź (ang. response). Przykład tego typu można zobaczyć na rysunku 1. Analizę komunikacji zacznijmy od żądania HTTP (dla uproszczenia przykładu usunąłem z niego niewymagane nagłówki): Listing 1. Prosta komunikacja HTTP GET /start/ HTTP/1.1 Host: training.securitum.com [pusta-linia]



Rysunek 1. Komunikacja HTTP

Pierwsza linijka żądania zawiera:
- metodę (ang. method lub verb; w naszym przypadku to: GET),
- adres URL (w naszym przypadku to: /start/),
- wersję protokołu (w naszym przypadku to: HTTP/1.1).

2. Metody HTTP

Tym razem spróbujemy użyć innych metod niż GET. Specyfikacja HTTP definiuje ich kilka , a tak naprawdę można spotkać ich nawet kilkadziesiąt . Na początek spróbujmy wysłać żądanie metodą HEAD:

HEAD / HTTP/1.1
Host: training.sekurak.com
[pusta-linia]

Dostaniemy analogiczną odpowiedź jak wcześniej, ale już bez ciała odpowiedzi (jak widać, poprawna odpowiedź HTTP nie musi go zawierać), choć z jedną pustą linijką na końcu:

HTTP/1.1 200 OK
Date: Mon, 28 Jan 2019 18:37:40 GMT
Server: Apache
Last-Modified: Mon, 28 Jan 2019 18:11:05 GMT
ETag: "2d2e0-21-5808899aa4c5d"
Accept-Ranges: bytes
Content-Length: 33
Vary: Accept-Encoding
Content-Type: text/html
[pusta-linia]

Gdzie taka metoda może się przydać? Choćby podczas próby siłowego lokalizowania pewnych ukrytych plików i katalogów:

HEAD /test HTTP/1.1
Host: training.securitum.com

W odpowiedzi uzyskujemy informację bez ciała odpowiedzi – dzięki temu przyspieszamy pojedynczy test (serwer musi wysłać mniej bajtów w odpowiedzi):

HTTP/1.1 404 Not Found
Date: Mon, 28 Jan 2019 18:39:18 GMT
Server: Apache
Vary: Accept-Encoding
Content-Type: text/html; charset=iso-8859-1

a) GET: Metoda GET jest najbardziej powszechnie używaną i podstawową metodą HTTP. Służy do pobierania danych z określonego zasobu na serwerze. Parametry są przekazywane w URL jako część zapytania. Ta metoda jest zazwyczaj używana do odczytu danych, a nie do ich modyfikacji.

b) POST: Metoda POST jest używana do przesyłania danych do serwera w celu przetworzenia ich lub zapisania. Dane są przesyłane jako część ciała żądania, a nie w adresie URL. Jest używana, gdy klient chce przesłać dane, które mogą być poufne lub niebezpieczne do umieszczenia w URL.

c) PUT: Metoda PUT służy do aktualizacji istniejącego zasobu na serwerze lub tworzenia nowego, jeśli zasób nie istnieje. Podobnie jak w przypadku POST, dane są przesyłane jako ciało żądania. Jest to często używane do zastępowania całego zasobu.

d) DELETE: Metoda DELETE jest używana do usuwania określonego zasobu na serwerze. Wskazuje się zasób do usunięcia w adresie URL. Ta metoda jest używana do usuwania danych na serwerze.

e) PATCH: Metoda PATCH jest używana do częściowej aktualizacji zasobu na serwerze. Jest to przydatne, gdy chcemy zmienić tylko niektóre części zasobu, a nie cały zasób. Dane do aktualizacji są przesyłane jako ciało żądania.

f) HEAD: Metoda HEAD jest podobna do metody GET, ale serwer nie przesyła danych w odpowiedzi. Zamiast tego dostarcza tylko nagłówki HTTP. Jest używana, gdy klient chce uzyskać informacje o zasobie, ale nie potrzebuje pełnej zawartości.

g) OPTIONS: Metoda OPTIONS jest używana do pobierania informacji o dostępnych metodach i parametrach dla określonego zasobu. Jest to przydatne do sprawdzania, jakie operacje są dozwolone dla danego zasobu.

h) CONNECT: Metoda CONNECT jest używana do ustanawiania tunelu do serwera, który może działać jako proxy. Jest to często używane w celu umożliwienia klientowi połączenia z innym serwerem za pośrednictwem serwera proxy.

i) TRACE: Metoda TRACE służy do uzyskania informacji zwrotnych od serwera. Jest to używane w celu debugowania i analizy tras, jakie żądania HTTP przeszły przez różne serwery po drodze.

3) URL i URI

Uniform Resource Locator (URL) i Uniform Resource Identifier (URI) to dwa pojęcia związane z identyfikacją i odnośnikami do zasobów w sieci. Istnieją pewne subtelne różnice między nimi:

1. Uniform Resource Locator (URL):

- URL to specyficzny rodzaj URI: URL jest bardziej szczegółowym pojęciem, które jest podzbiorem URI. Oznacza to, że każdy URL jest również URI, ale nie każde URI jest URL.

- Zawiera informacje o lokalizacji: URL zawiera informacje, które pozwalają na zlokalizowanie i dostęp do konkretnego zasobu w sieci. Obejmuje on adres internetowy, protokół (np. HTTP, FTP), numer portu itp.

- Przykład: https://www.example.com:8080/index.html

2. Uniform Resource Identifier (URI):

- URI jest bardziej ogólnym pojęciem: URI jest identyfikatorem zasobu, który może być dowolnym identyfikatorem, nawet jeśli nie zawiera wszystkich informacji potrzebnych do bezpośredniego dostępu do zasobu.

- Może być relatywny lub absolutny: URI może być relatywny (odnosić się do zasobu względem bieżącego kontekstu) lub absolutny (zawierać pełną ścieżkę dostępu do zasobu).

- Przykład (absolutny URI): https://www.example.com/resource

- Przykład (relatywny URI): ../resource

Podsumowując, URI jest bardziej ogólnym pojęciem obejmującym wszelkiego rodzaju identyfikatory zasobów, podczas gdy URL jest konkretnym rodzajem URI, który zawiera pełne informacje o lokalizacji zasobu w sieci. W praktyce, terminy URL i URI są często używane wymiennie, ale warto zrozumieć różnicę między nimi z punktu widzenia teorii.

4) Nagłówki HTTP

Nagłówki HTTP (Hypertext Transfer Protocol) są elementami metadanych przesyłanymi w żądaniach i odpowiedziach HTTP między klientem a serwerem. Pełnią one ważną rolę w komunikacji między klientem a serwerem, pozwalając na przekazywanie informacji o żądaniu lub odpowiedzi oraz kontrolowanie zachowania tego procesu. Oto kilka najważniejszych nagłówków HTTP:

- User-Agent: Ten nagłówek zawiera informacje o przeglądarce lub aplikacji, która wysyła żądanie. Jest to przydatne do identyfikowania rodzaju i wersji oprogramowania używanego przez klienta.
- Host: Nagłówek Host jest niezbędny w żądaniach HTTP 1.1 i określa serwer docelowy, do którego skierowane jest żądanie. W wielu przypadkach jest to nazwa domeny, która pozwala na rozróżnienie wielu witryn na tym samym serwerze.
- Accept: Ten nagłówek wskazuje, jakie typy mediów (np. HTML, JSON, XML) akceptuje klient w odpowiedzi. Pomaga serwerowi dostosować zawartość odpowiedzi do preferencji klienta.
- Content-Type: W przypadku żądań POST i PUT oraz odpowiedzi serwera, ten nagłówek określa format danych zawartych w ciele żądania lub odpowiedzi, na przykład "application/json" lub "text/html".
- Content-Length: Nagłówek ten określa długość ciała żądania lub odpowiedzi w bajtach. Jest to przydatne, aby serwer wiedział, ile danych musi przeczytać.
- Location: Nagłówek Location jest często używany w odpowiedziach przekierowujących (status HTTP 3xx) i zawiera adres URL, do którego klient powinien zostać przekierowany.
- Cache-Control: Ten nagłówek pozwala na kontrolowanie sposobu, w jaki przeglądarka lub proxy przechowuje i zarządza zasobami w pamięci podręcznej. Można określić, czy zasób może być przechowywany w pamięci podręcznej, przez ile czasu i w jakich warunkach.
- Authorization: Nagłówek ten jest używany do przesyłania informacji uwierzytelniających, takich jak tokeny dostępu lub hasła, które są wymagane do autoryzacji dostępu do zasobów chronionych.
- Cookie: Ten nagłówek zawiera ciasteczka (cookies), które wcześniej zostały przesłane do klienta przez serwer. Umożliwia to serwerowi śledzenie sesji klienta i dostarczanie spersonalizowanych treści.
- Set-Cookie: W odpowiedzi serwera ten nagłówek jest używany do ustawienia ciasteczka w przeglądarce klienta. Dzięki temu serwer może zapamiętać pewne informacje o stanie sesji klienta.

- ETag: Nagłówek ETag służy do identyfikacji wersji zasobu. Jest używany w celu obsługi warunkowych żądań, co pozwala klientowi pobrać zasób tylko wtedy, gdy ten zasób uległ zmianie.
- If-Modified-Since: Ten nagłówek jest używany w warunkowych żądaniach GET, aby sprawdzić, czy zasób został zmodyfikowany od określonej daty. Jeśli nie, serwer może zwrócić status 304 Not Modified.
- Referer (sic!): Ten nagłówek wskazuje źródło, z którego klient przyszedł na bieżącą stronę. Jest przydatny w analizie danych statystycznych i do celów bezpieczeństwa.

Przykłady:
Oto przykłady niektórych nagłówków HTTP:

1. User-Agent:

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36

2. Host:

Host: www.example.com

3. Accept:

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,/*;q=0.8*

4. Content-Type:

Content-Type: application/json

5. Content-Length:

Content-Length: 1024

6. Location (przekierowanie):

Location: https://www.example.com/new-page

7. Cache-Control:

Cache-Control: max-age=3600, must-revalidate

8. Authorization (przy użyciu tokena Bearer):

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

9. Cookie:

Cookie: session_id=abc123; user_pref=dark_mode

10. Set-Cookie:

Set-Cookie: user_pref=dark_mode; expires=Sat, 22 Oct 2023 20:00:00 GMT; path=/

11. ETag:

ETag: "1a2b3c4d5e6f7g8h9i0j"

12. If-Modified-Since:

If-Modified-Since: Wed, 19 Oct 2022 14:30:00 GMT

13. Referer:

Referer: https://www.example.com/previous-page

Powyższe przykłady pokazują, jak wyglądają różne nagłówki HTTP w praktyce. Każdy z tych nagłówków ma określone zadanie i pomaga w prawidłowym przetwarzaniu żądań i odpowiedzi w protokole HTTP.

5) WARTOŚCI PRZEKAZYWANE DO APLIKACJI PROTOKOŁEM HTTP

To jeden z najistotniejszych, podstawowych tematów w kontekście bezpieczeństwa. Przytłaczająca liczba podatności w aplikacjach webowych może być wykorzystana dzięki odpowiedniemu (złośliwemu) manipulowaniu wartościami przekazywanymi do aplikacji. W którym zatem miejscu żądania HTTP mogą znaleźć się parametry? Nieco wymijająca odpowiedź brzmi: wszędzie. Poczynając od linijki żądania, przez nagłówki, aż po ciało. Przyjrzyjmy się kilku popularnym miejscom, w których możemy znaleźć parametry.

Nagłówki żądania HTTP

To miejsce (może poza nagłówkiem Cookie, o którym będzie mowa nieco później) jest często niesłusznie ignorowane w kontekście potencjalnego zagrożenia związanego z niebezpiecznym użyciem wartości nagłówka. Ilustracją tego problemu będą dwie po75 Wartości przekazywane do aplikacji protokołem HTTP datności. Pierwsza to SQL Injection, zlokalizowana 22 marca 2019 roku w nagłówku User-Agent21. Nagłówek z wstrzyknięciem mógł w tym przypadku wyglądać np. tak:

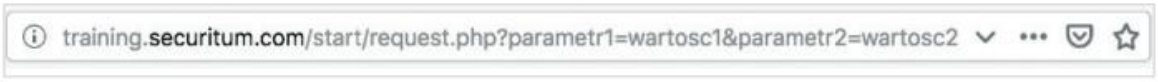
*Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) ↙ Chrome/55.0.2883.87'XOR(if(now()=sysdate(),sleep(5*5),0))OR'*

Druga podatność to Path Traversal w Ruby on Rails (CVE-2019-5418). W tym przypadku zupełnie niewinny nagłówek żądania Accept (mówiący normalnie, jakie formaty odpowiedzi akceptuje klient) mógł w pewnych sytuacjach doprowadzić do możliwości nieautoryzowanego odczytywania plików na serwerze:

Accept: ../../../../../../../../../../etc/passwd{{

URL

Parametry przekazywane w URL najczęściej zobaczymy w zapytaniach typu GET. Wielu z nas widziało na pewno tego typu konstrukcję, umieszczoną w pasku adresu przeglądarki internetowej:



Przeglądarka na podstawie takiego adresu realizuje następujące żądanie HTTP:

```
GET /start/request.php?parametr1=wartosc1&parametr2=wartosc2 HTTP/1.1
Host: training.securitum.com
```

```
HTTP/1.1 200 OK
Date: Mon, 04 Mar 2019 09:15:30 GMT
Server: Apache
Content-Length: 39
Content-Type: text/html; charset=UTF-8

parametr1: wartosc1
parametr2: wartosc2
```

Pamiętajmy, że to tylko pewna konwencja. Aplikacja jako wartości może równie dobrze użyć nazwy parametru (w naszym przypadku jest to parametr1). Co się stanie, gdy dodamy jakiś dodatkowy parametr, nieobsługiwany przez aplikację? Na przykład parametr3? Najczęściej – nic. Czasem jednak aplikacja analizuje wszystkie przesłane (np. metodą GET) parametry, więc warto sprawdzić, co się stanie, jeśli przekażemy tego typu dodatkową wartość.

Zaleca się, aby metodą GET nie przesyłać poufnych czy wrażliwych informacji, np. haseł czy identyfikatorów sesyjnych. Dlaczego? Parametry te widać bezpośrednio w pasku URL przeglądarki. Są one domyślnie logowane przez serwery HTTP, jak również widoczne w wynikach wyszukiwarek internetowych.

POST: application/x-www-form-urlencoded

Drugim często spotykanym sposobem wysyłania parametrów jest przesyłanie ich w ciele żądania, z wykorzystaniem metody POST. Często w ten sposób wysyłane są dane z formularzy HTML:

```
<form action="/request.php" method="POST">
Podaj parametr 1: <input type="text" name="parametr1">
Podaj parametr 2: <input type="text" name="parametr2">
<input type="submit">
</form>
```

W tagu <form> możemy podać również parametr enctype, wskazujący przeglądarce, w jaki sposób powinna przesłać dane z formularza w żądaniu:

```
<form action="/request.php" method="POST"
enctype="application/x-www-form-urlencoded">
```

Wartość application/x-www-form-urlencoded jest domyślna (jeśli więc chcemy z niej skorzystać, możemy całkowicie pominąć parametr enctype).

W tego rodzaju formularzach powinno nam przyjść do głowy pytanie: co się stanie, gdy zmienimy metodę POST na GET w naszym żądaniu? W większości przypadków nie powinno to spowodować żadnych problemów. Nie mniej jednak tego typu błąd został odkryty w API wordpressa, gdzie zmiana metody z POST na GET umożliwiała anonimową zmianę dowolnego wpisu.

POST: multipart/form-data

Czasem parametry wysyłane metodą POST korzystać będą właśnie z kodowania multipart/form-data. Przykład tego typu kodowania najczęściej znajdziemy w formularzach uploadujących pliki. Wygląda ono np. tak:

```
POST /request.php HTTP/1.1
Host: training.securitum.com
Content-Type: multipart/form-data; boundary=awnwWejh23kl
Content-Length: 323

--awnwWejh23kl
content-disposition: form-data; name="parametr1"

wartosc
jeden

--awnwWejh23kl
content-disposition: form-data; name="parametr2"

wartosc
dwa
--awnwWejh23kl
content-disposition: form-data; name="nasz_plik"; filename="file1.txt"
Content-Type: text/plain

Zawartosc pliku tekstowego
--awnwWejh23kl--
```

Warto zwrócić tutaj uwagę na tzw. ogranicznik (ang. boundary29). Najczęściej jest to losowa zawartość, choć, nomen omen, mająca pewne ograniczenia (np. posiada limit długości). Pełny ogranicznik to znaki CRLF (znacznik końca linii), dwa znaki minus (--) oraz wartość wskazana parametrem boundary w nagłówku. Czyli w naszym przykładzie jest to --awnwWejh23kl (oraz wcześniejszy znacznik końca linii). Ostatni ogranicznik dodatkowo posiada na końcu dwa kolejne znaki --. W naszym przykładzie to --awnwWejh23kl-- (oraz wcześniejszy znacznik końca linii). W skrócie, ogranicznik rozdziela ciało żądania na wiele części (ang. multipart). W rozważanym przypadku mamy ich trzy. Każda z nich posiada nagłówek definiujący nazwę zmiennej: content-disposition: form-data; name="nazwa". Następnie po dwóch znakach końca linii (CRLF) następuje przekazanie wartości zmiennej. Koniec wartości sygnalizowany jest przez ogranicznik.

Ciasteczka

Czyli popularne cookie, nazywane niekiedy niepoprawnie „plikami cookie”32. W kontekście naszych rozważań ciastka (mogące zawierać parametry) możemy znaleźć w nagłówku Cookie żądania HTTP. W jednym nagłówku może znaleźć się kilka różnych ciasteczek (mimo to nagłówek cały czas będzie nazywał się Cookie, nie Cookies). W którym miejscu ciastka są wysyłane z serwera? W odpowiedzi, w nagłówku Set-Cookie.

Zobaczmy przykład obu nagłówków. Po wpisaniu w przeglądarkę adresu: http:// training.securitum.com/cookie.php serwer ustawia dwa ciastka (nagłówek odpowiedzi Set-Cookie):

```
GET /cookie.php HTTP/1.1
Host: training.securitum.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:65.0) ⤶
Gecko/20100101 Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/ ⤶
webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
Date: Sun, 03 Feb 2019 16:58:11 GMT
Server: Apache
Set-Cookie: testowe-ciastko1=testowa-wartosc2
Set-Cookie: testowe-ciastko2=testowa-wartosc2
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html
```

Kolejne zapytanie z przeglądarki jest już automatycznie uzupełnione o nagłówek Cookie:

```
GET /cookie.php HTTP/1.1
Host: training.securitum.com
Accept-Language: en-US,en;q=0.5
DNT: 1
Connection: close
Cookie: testowe-ciastko1=testowa-wartosc2; testowe-ciastko2=testowa-wartosc2
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

LITERATURA

Bezpieczeństwo Aplikacji Webowych - Michał Bentkowski / Artur Czyż / Rafał 'bl4de' Janicki / Jarosław Kamiński Adrian 'vizzdoom' Michalczyk / Mateusz Niezabitowski / Marcin Piosek Michał Sajdak / Grzegorz Trawiński / Bohdan Widła - ISBN: 978-83-954853-2-9 - Kraków 2020