Rozwiązania testu - Projektowanie i wdrażanie systemów w chmurze

Mateusz Zając (298654)

29 września 2021

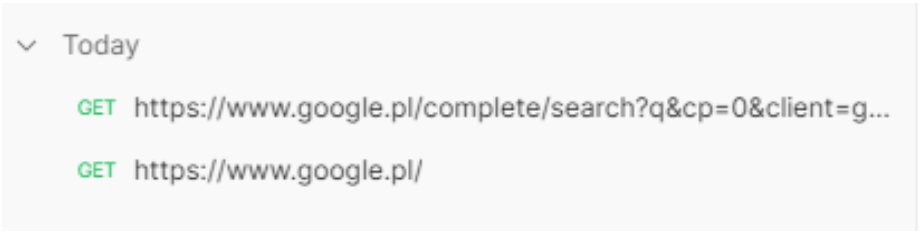
Zadanie 1

Badając problemy z aplikacją webową nabierasz podejrzenia, że przeglądarka internetowa wysyłając zapytanie do serwera umieszcza w pewnym nagłówku HTTP inną wartość, niż spodziewa się serwer. Podaj *kilka* sposobów, które można użyć by potwierdzić tę hipotezę i sprawdzić faktyczną wartość wybranego nagłówka.

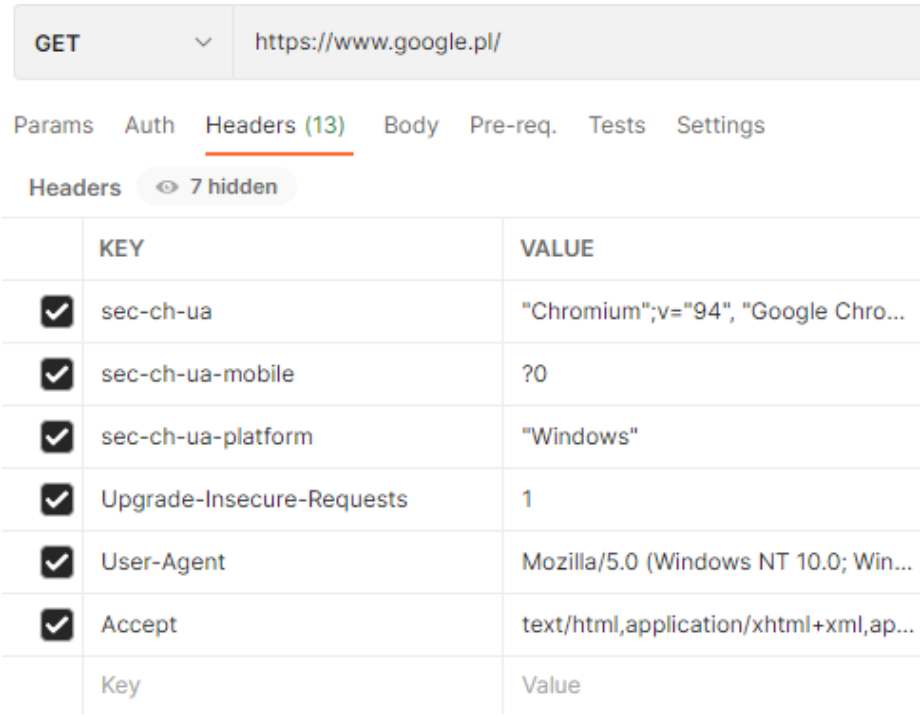
Żeby potwierdzić tę hipotezę, muszę przyjrzeć się odpowiedniemu nagłówkowi HTTP i zweryfikować jego zawartość. W tym celu muszę mieć możliwość obejrzenia poszczególnych zapytań HTTP, na przykład tak:

1. **Użycie Postmana i przejrzenie zapytań wysyłanych przez przeglądarkę.**

   Aplikacja Postman posiada rozszerzenie do przeglądarki Google Chrome, która po aktywacji zapisuje zapytania HTTP w historii zapytań aplikacji. Po wejściu na stronę Google historia zapytań wygląda tak:



Możemy podejrzeć nagłówek zapytania:



2. **Użycie curl**

   Możemy również użyć programu "curl" w konsoli linuxowej. Jeśli chcę podejrzeć wysyłane nagłówki zapytania np. do http://www.google.pl, mogę wywołać curl w ten sposób: `curl -v http://www.google.pl`. Dzięki temu zobaczę wszystkie nazwy nagłówków oraz ich zawartość:

```
mzet@mzet-VirtualBox:~/Desktop$ curl -v http://www.google.pl
*   Trying 216.58.209.3:80...
* TCP_NODELAY set
* Connected to www.google.pl (216.58.209.3) port 80 (#0)
> GET / HTTP/1.1
> Host: www.google.pl
> User-Agent: curl/7.68.0
> Accept: */*
```

3. **Sprawdzenie pakietów przy pomocy narzędzi deweloperskich Google Chrome**

   Alternatywnie, mogę użyć wbudowanych narzędzi deweloperskich przeglądarki Google Chrome, aby podejrzeć wysyłane zapytania. Mogę tutaj podejrzeć zarówno zapytanie, jak i odpowiedź na nią.

Zadanie 2

Masz pełny dostęp administracyjny do zdalnego serwera z systemem linuksowym[2]. Potrzebujesz na nim umieścić kod programu, który przygotowałeś na lokalnym komputerze[3], składający się z dużej liczby plików. Następnie będziesz edytował te pliki - czasami lokalną kopię, a czasami zdalną. Jakich narzędzi, komend i sposobów użyjesz, by wygodnie i sprawnie przesyłać cały komplet plików tam i z powrotem?

Najszybszym i najprostszym sposobem na transfer dużej ilości plików w bezpieczny sposób przez sieć jest użycie protokołu SFTP. Jeśli na serwerze jest włączona ta usługa, mogę się do niego zdalnie podłączyć, używając menedżera FTP (np Filezilla lub Total Commander). Po prostu podaję adres serwera, loguję się i mogę swobodnie transferować pliki z obie strony. Mogę też edytować pliki bezpośrednio na serwerze.

Aby uruchomić program na serwerze zdalnym oraz wykonywać komendy, mogę użyć SSH. Tam także wystarczy podanie adresu serwera oraz zalogowanie. Od tej chwili mogę korzystać z konsoli serwerowej na komputerze lokalnym.

Innym rozwiązaniem, bez użycia SFTP może być komenda `scp`. Służy do bezpiecznego kopiowania plików z lokalnego komputera na serwer. Lepszym rozwiązaniem jest natomiast użycie `rsync`, który robi to szybciej i sprawniej.

Jeśli nad kodem aplikacji pracuje więcej osób, należy rozważyć użycie repozytorium kodu (np. Git) wraz z synchronizacją kodu serwera z kodem repozytorium (takie rozwiązaniem stosuje np. Heroku).

Zadanie 3

Uruchamiasz na zdalnym serwerze własny program, który nasłuchuje połączeń na porcie 1234. Bezproblemowo łączysz się do tego programu z lokalnego komputera. Następnie przestawiasz serwer, by tym razem słuchał na porcie 5678, i wtedy już nie udaje się nawiązać połączenia z lokalnego komputera. Podaj kilka hipotez jakie zjawiska mogą prowadzić do takiego problemu i wymień krótko w jaki sposób te hipotezy można zweryfikować.

Należy sprawdzić, jaki stan ma ten port na serwerze. Możemy to zrobić przy pomocy `netstat` (w konsoli serwera). Potem od razu filtrujemy wynik narzędziem `grep`: `netstat | grep "5678"`. Jeśli nastąpi jakiś problem (port, który wybraliśmy nie ma statusu "LISTEN"), oznacza to, że mamy problem z nasłuchiwaniem na tym porcie. Być może inna aplikacja z niego korzysta lub nasz program nie ma uprawnień do zajęcia portu.

Bezpośrednim sprawdzeniem czy mamy połączenie z adresem serwera na danym porcie jest wywołanie: `telnet (adresserwera) (port)`. Jeżeli uda nam się poprawnie połączyć, po stronie sieci nie ma problemów z połączeniem. Problem może leżeć w aplikacji.

Jeśli natomiast nie możemy się połączyć, sprawdzić musimy także, czy nasza brama sieciowa (np. router, który łączy serwer z internetem) umożliwia łączenie na tym porcie. Możemy to sprawdzić w ustawieniach bezpieczeństwa na urządzeniu sieciowym. Potrzebnym może być dodanie wyjątku w zaporze sieciowej urządzenia.

Zadanie 4

Ile czasu, orientacyjnie, zajęłoby Ci napisanie prostego serwera HTTP[4], który przyjmuje zapytania na kilku różnych ścieżkach (np. `/index`, `/test/test1`, `/test/test2` oraz `/przyklad`) odpowiadając dynamiczną zawartością (np. aktualna godzina, zużycie pamięci na serwerze, wiadomość ustawiona przez poprzednie zapytanie)? Jakich narzędzi i/lub technik użyjesz, dzięki którym uda Ci się to aż tak szybko?

Najprostszym dla mnie sposobem na stworzenie tego serwera jest napisanie prostej aplikacji w `Node.js` przy pomocy frameworka `Express`. Mogę szybko zdefiniować ścieżki, na które ma reagować serwer, a dzięki bibliotece standardowej javaScript mogę wyciągnąć potrzebne informacje o czasie i pamięci maszyny:

```
1   // Mateusz Zajac
2   // 298654
3   // Projektowanie i wdrazanie systemow w chmurze
4
5   const express = require('express')
6   const app = express()
7   const port = 1234
8   const os = require('os')
9
10  var lastVal = ""
```

```
11
12      // This path prints the current server time
13      app.get('/index', (req, res) => {
14        var mytime = new Date(Date.now())
15        res.send(`My current time is: ${mytime}`)
16      })
17
18      // This path prints free memory in Megabytes
19      app.get('/test/test1', (req, res) => {
20        // It returns number in bytes,
21        // i want to cast it to MB for readability,
22        //  therefore division by 1e6
23        var freemem = os.freemem/(1e6)
24        res.send(`My free memory: ${parseInt(freemem)}MB`)
25      })
26
27      // This path sets lastVal variable to "George has a cat"
28      app.get('/test/test2', (req, res) => {
29        lastVal = "George has a cat"
30        res.redirect("/przyklad")
31      })
32
33      // This path sets lastVal variable to "Cat has George"
34      app.get('/test/test3', (req, res) => {
35        lastVal = "Cat has George"
36        res.redirect("/przyklad")
37      })
38
39      // This path prints lastVal variable on the screen
40      app.get('/przyklad', (req, res) => {
41        res.send(`My last value is: ${lastVal}`)
42      })
43
44      // This one runs the whole app on port 1234
45      // And prints the message to confirm it
46      app.listen(port, () => {
47        console.log("I'm ready!")
48      })
```

Napisanie całości zajmuje około godzinę (uzwględniając instalację `Node.js`, frameworku `Express` oraz inicjalizację projektu za pomocą `npm init`).