

**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI**  
POLITECHNIKI RZESZOWSKIEJ

**Jakub Skrzynecki**

Testy penetracyjne sieci, serwerów i aplikacji

Ffuf – fuzzing tool

## Spis treści

<b>1. Wprowadzenie.....</b>	<b>4</b>
1.1. Cel projektu .....	4
1.2. Znaczenie testów penetracyjnych w cyberbezpieczeństwie .....	4
1.3. Krótka charakterystyka narzędzia ffuf .....	4
<b>2. Podstawowe funkcjonalności ffuf.....</b>	<b>5</b>
2.1. Mechanizm fuzzingu .....	5
2.1.1. Działanie fuzzingu w ffuf.....	5
2.1.2. Rodzaje fuzzingu w ffuf .....	5
2.2. Praca ze słownikami (wordlists) .....	5
2.2.1. Źródła słowników .....	6
2.2.2. Wykorzystanie słowników w ffuf .....	6
2.2.3. Praca z wieloma słownikami .....	6
2.2.4. Generowanie dynamicznych słowników .....	6
2.3. Opcje i parametry uruchomieniowe.....	6
2.3.1. Definiowanie celu testów .....	6
2.3.2. Równoczesne zapytania (konkurencyjność).....	6
2.3.3. Ustalanie metod HTTP .....	7
2.3.4. Nagłówki i ciasteczka.....	7
2.3.5. Formatowanie wyników .....	7
2.3.6. Timeout i limity czasowe .....	7
2.4. Filtrowanie wyników.....	7
2.4.1. Filtrowanie według kodów HTTP .....	7
2.4.2. Filtrowanie według długości odpowiedzi.....	7
2.4.3. Filtrowanie według wzorca w treści.....	7
2.5. Proxowanie.....	8
<b>3. Symulacja użycia ffuf na środowisku ffuf.me.....</b>	<b>9</b>

<b>3.1. Przygotowanie środowiska testowego .....</b>	<b>9</b>
<b>3.2. Scenariusze testowe – przykłady zastosowań .....</b>	<b>9</b>
3.2.1. Basic Content Discovery .....	9
3.2.2. Recursion .....	11
3.2.3. File Extensions .....	13
3.2.4. No 404 Status .....	14
3.2.5. Param Mining .....	17
3.2.6. Rate Limited .....	19
3.2.7. Pipes .....	22
3.2.8. Virtual host enumeration .....	27
<b>3.3. Analiza wyników symulacji.....</b>	<b>29</b>
<b>3.4. Wnioski z przeprowadzonych testów .....</b>	<b>30</b>
<b>4. Porównanie z innymi narzędziami do fuzzingu.....</b>	<b>30</b>
4.1. DirBuster.....	30
4.2. wfuzz.....	31
4.3. Zalety i wady w porównaniu z ffuf.....	31
<b>5. Podsumowanie i wnioski .....</b>	<b>31</b>
5.1. Zalety i ograniczenia ffuf.....	31
5.2. Rekomendacje dotyczące wykorzystania w testach penetracyjnych	32
5.3. Perspektywy rozwoju narzędzia .....	32
<b>6. Bibliografia .....</b>	<b>33</b>

# 1. Wprowadzenie

## 1.1. Cel projektu

Celem tego projektu jest dogłębna analiza narzędzia ffuf (Fuzz Faster U Fool) oraz przedstawienie jego zastosowań w testach penetracyjnych, ze szczególnym uwzględnieniem symulacji przeprowadzonych w środowisku ffuf.me. Projekt ma na celu pokazanie, w jaki sposób to narzędzie wspomaga identyfikację potencjalnych luk bezpieczeństwa w aplikacjach internetowych oraz jak jego zastosowanie może przyczynić się do poprawy zabezpieczeń testowanych systemów.

## 1.2. Znaczenie testów penetracyjnych w cyberbezpieczeństwie

W dobie rosnących zagrożeń w cyberprzestrzeni testy penetracyjne stanowią kluczowy element strategii ochrony systemów informatycznych. Umożliwiają one wykrywanie podatności, zanim zostaną one wykorzystane przez cyberprzestępców. Narzędzia takie jak ffuf automatyzują proces testowania, skracając czas potrzebny na przeprowadzenie szczegółowych analiz i umożliwiając skuteczne wykrywanie anomalii w funkcjonowaniu aplikacji.

## 1.3. Krótka charakterystyka narzędzia ffuf

Ffuf to narzędzie zaprojektowane do efektywnego wykonywania testów fuzzingu, czyli techniki polegającej na wysyłaniu różnych danych wejściowych do aplikacji w celu wykrycia ewentualnych luk bezpieczeństwa. Kluczowe cechy to:

- 1) **Wysoka wydajność** – Dzięki zoptymalizowanemu algorytmowi ffuf może szybko przeszukiwać duże przestrzenie adresowe i katalogi.
- 2) **Elastyczność** – Umożliwia dostosowanie parametrów testów, takich jak konfiguracja słowników (wordlists), co pozwala na przeprowadzanie ataków dostosowanych do konkretnego środowiska.
- 3) **Łatwość obsługi** – Intuicyjny interfejs wiersza poleceń sprawia, że narzędzie jest przystępne zarówno dla doświadczonych pentesterów, jak i osób dopiero rozpoczynających swoją przygodę z cyberbezpieczeństwem.
- 4) **Wszechstronność** – Ffuf może być wykorzystywany do różnych rodzajów testów, w tym wyszukiwania ukrytych katalogów, plików, parametrów, wirtualnych hostów oraz do testowania endpointów API.
- 5) **Zaawansowane filtrowanie wyników** – Możliwość filtrowania odpowiedzi na podstawie różnych kryteriów, takich jak kod odpowiedzi HTTP, rozmiar odpowiedzi czy zawartość, co znacznie ułatwia analizę dużej ilości danych.

## 2. Podstawowe funkcjonalności ffuf

### 2.1. Mechanizm fuzzingu

Fuzzing to technika testowania bezpieczeństwa, polegająca na automatycznym wysyłaniu dużej liczby losowych lub celowo spreparowanych danych wejściowych do aplikacji w celu wykrycia błędów i podatności. Ffuf implementuje fuzzing poprzez dynamiczne zastępowanie określonych fragmentów żądań danymi ze słowników i analizę odpowiedzi serwera.

#### 2.1.1. Działanie fuzzingu w ffuf

Ffuf działa na zasadzie iteracyjnego wstawiania wartości z podanego słownika w miejsce określonego ciągu znaków (np. FUZZ) w adresie URL, nagłówkach czy ciele żądania HTTP. Każda wygenerowana wartość jest wysyłana jako nowe zapytanie, a narzędzie analizuje odpowiedzi pod kątem różnic w kodach statusu, rozmiarze czy zawartości odpowiedzi.

Przykładowe użycie fuzzingu w ffuf:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt
```

W powyższym przykładzie narzędzie iteracyjnie podstawia wartości z `wordlist.txt` w miejsce `FUZZ`, sprawdzając, które żądania zwrócą sensowne odpowiedzi.

#### 2.1.2. Rodzaje fuzzingu w ffuf

- 1) **Fuzzing katalogów i plików** – Poszukiwanie ukrytych zasobów w aplikacjach webowych.
- 2) **Fuzzing parametrów GET/POST** – Weryfikacja obsługiwanych parametrów wejściowych.
- 3) **Fuzzing nagłówków HTTP** – Testowanie zabezpieczeń poprzez manipulację nagłówkami (np. User-Agent, Referer).
- 4) **Fuzzing sesji i tokenów** – Poszukiwanie wrażliwych identyfikatorów i metod autoryzacji.

### 2.2. Praca ze słownikami (wordlists)

Jednym z kluczowych elementów pracy z ffuf jest wykorzystanie słowników (wordlists), które pozwalają na skuteczne przeprowadzanie ataków fuzzingowych. Słowniki składają się z zestawów słów kluczowych, które są kolejno testowane w ramach różnych zapytań HTTP, w celu odkrycia ukrytych zasobów, parametrów czy wrażliwych punktów aplikacji.

### 2.2.1. Źródła słowników

Słowniki mogą pochodzić z różnych źródeł:

- 1) **Domyślne słowniki dostępne w narzędziach** – Na przykład popularne zestawy z pakietu SecLists.
- 2) **Własnoręcznie tworzone słowniki** – Można dostosować je do konkretnej aplikacji, np. na podstawie struktury URL lub logów serwera.
- 3) **Generowanie dynamiczne** – Tworzenie słowników na podstawie analizy dostępnych treści w aplikacji.

### 2.2.2. Wykorzystanie słowników w ffuf

Aby uruchomić test fuzzingowy z użyciem słownika, należy użyć opcji `-w`:

```
ffuf -u http://example.com/FUZZ -w /path/to/wordlist.txt
```

W powyższym przykładzie `FUZZ` jest zmienną, którą ffuf będzie zastępował kolejnymi wartościami z pliku `wordlist.txt`.

### 2.2.3. Praca z wieloma słownikami

Ffuf umożliwia użycie wielu słowników jednocześnie:

```
ffuf -u http://example.com/FUZZ/FUZZ2 -w /path/to/wordlist1.txt -w /path/to/wordlist2.txt
```

W tym przypadku narzędzie testuje kombinacje wartości z obu słowników.

### 2.2.4. Generowanie dynamicznych słowników

Za pomocą narzędzi takich jak `cewl` można stworzyć słownik na podstawie analizy treści strony:

```
cewl http://example.com -w custom_wordlist.txt
```

Następnie można użyć go w ffuf do precyzyjniejszych testów fuzzingu.

## 2.3. Opcje i parametry uruchomieniowe

Ffuf oferuje szeroki zestaw opcji konfiguracyjnych, które pozwalają dostosować testy do konkretnych potrzeb. Oto najważniejsze z nich:

### 2.3.1. Definiowanie celu testów

Podstawowym parametrem jest `-u`, który określa URL testowanej aplikacji:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt
```

### 2.3.2. Równoczesne zapytania (konkurencyjność)

Aby zwiększyć wydajność testów, można dostosować liczbę równoczesnych zapytań za pomocą `-t`:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -t 50
```

Wartość 50 oznacza, że narzędzie będzie wykonywać 50 zapytań jednocześnie.

### 2.3.3. Ustalanie metod HTTP

Domyślnie ffuf używa metody GET, ale można ją zmienić na POST, PUT itp.:

```
ffuf -u http://example.com/api -X POST -d "param=FUZZ" -w payloads.txt
```

Opcja -X definiuje metodę, a -d zawiera dane przesyłane w treści żądania.

### 2.3.4. Nagłówki i ciasteczka

Aby dodać niestandardowe nagłówki (np. autoryzacyjne), można użyć -H:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -H "Authorization:
Bearer TOKEN"
```

Podobnie można przekazywać ciasteczka za pomocą -b:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -b "session=abcdef"
```

### 2.3.5. Formatowanie wyników

Aby zapisać wyniki testów w formacie JSON:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -o results.json -of
json
```

Inne dostępne formaty to csv, html i md (Markdown).

### 2.3.6. Timeout i limity czasowe

Aby uniknąć zawieszania się testów, warto ustawić limit czasu odpowiedzi:

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -timeout 5
```

W powyższym przykładzie każda odpowiedź musi nadejść w ciągu 5 sekund.

## 2.4. Filtrowanie wyników

Aby skupić się na istotnych wynikach, ffuf oferuje kilka metod filtrowania odpowiedzi:

### 2.4.1. Filtrowanie według kodów HTTP

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -fc 403
```

Odfiltrowuje wszystkie odpowiedzi z kodem 403 (Forbidden).

### 2.4.2. Filtrowanie według długości odpowiedzi

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -fl 2000
```

Usuwa odpowiedzi o długości 2000 bajtów.

### 2.4.3. Filtrowanie według wzorca w treści

```
ffuf -u http://example.com/FUZZ -w wordlist.txt -fr "Not Found"
```

Pomija odpowiedzi zawierające frazę "Not Found".

## 2.5. Proxowanie

Możemy przepuścić skan używając serwera proxy np. przepuszczanie przez burp suit.

Cały ruch może być skierowany na serwer proxy:

```
ffuf -u http://MACHINE_IP/FUZZ -c -w /usr/share/seclists/Discovery/Web-Content/common.txt -x http://127.0.0.1:8080
```

Jeśli chcemy proxować jedynie zapytania, które chcemy otrzymać możemy skorzystać

Z:

```
ffuf -u http://MACHINE_IP/FUZZ -c -w /usr/share/seclists/Discovery/Web-Content/common.txt -replay-proxy http://127.0.0.1:8080
```



### 3. Symulacja użycia ffuf na środowisku ffuf.me

Niniejszy etap projektu zakłada implementację i ewaluację funkcjonalności narzędzia ffuf w kontrolowanym środowisku testowym udostępnionym na platformie ffuf.me. Głównym celem przeprowadzanej symulacji jest kompleksowa weryfikacja skuteczności, wydajności oraz poprawności konfiguracji narzędzia w kontekście rzeczywistych scenariuszy testowych. Proces ten umożliwi precyzyjne określenie efektywności narzędzia w realizacji zadań związanych z technikami fuzzingu oraz identyfikacją potencjalnych wektorów ataku i punktów wejścia w aplikacjach webowych.

#### 3.1. Przygotowanie środowiska testowego

W tej części projektu skoncentrowano się na utworzeniu i konfiguracji środowiska umożliwiającego przeprowadzenie kompleksowej symulacji. Poniżej przedstawiono sekwencję kluczowych etapów procesu przygotowawczego:

- 1) **Weryfikacja instalacji narzędzia ffuf:** Przeprowadzono kontrolę poprawności instalacji narzędzia na dedykowanym systemie wirtualnym przeznaczonym do realizacji symulacji.
- 2) **Konfiguracja parametrów środowiska:** Zweryfikowano stabilność i niezawodność połączenia z platformą ffuf.me w celu zapewnienia integralności danych podczas testów.
- 3) **Akwizycja specjalistycznych wordlist:** Pozyskano i zaimplementowano zestaw wordlist opracowanych specjalnie na potrzeby skanowania platformy testowej. Zestaw obejmował pliki common.txt, parameters.txt oraz subdomains.txt, zawierające starannie wyselekcjonowane ciągi znaków do przeprowadzenia efektywnych testów.
- 4) **Walidacja gotowości infrastruktury:** Przeprowadzono serię testów weryfikacyjnych mających na celu potwierdzenie prawidłowego funkcjonowania środowiska oraz jego pełnej gotowości do realizacji zaawansowanych symulacji.

#### 3.2. Scenariusze testowe – przykłady zastosowań

##### 3.2.1. Basic Content Discovery

Pierwszym wdrożonym scenariuszem testowym było wyszukiwanie podstawowych plików przy wykorzystaniu następującej komendy:

```
ffuf -w ~/wordlists/common.txt -u http://ffuf.me/cd/basic/FUZZ
```

## FFUF.me Content Discovery - Basic

Lets start with the most basic fuzz first

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/common.txt -u http://ffuf.me/cd/basic/FUZZ
```

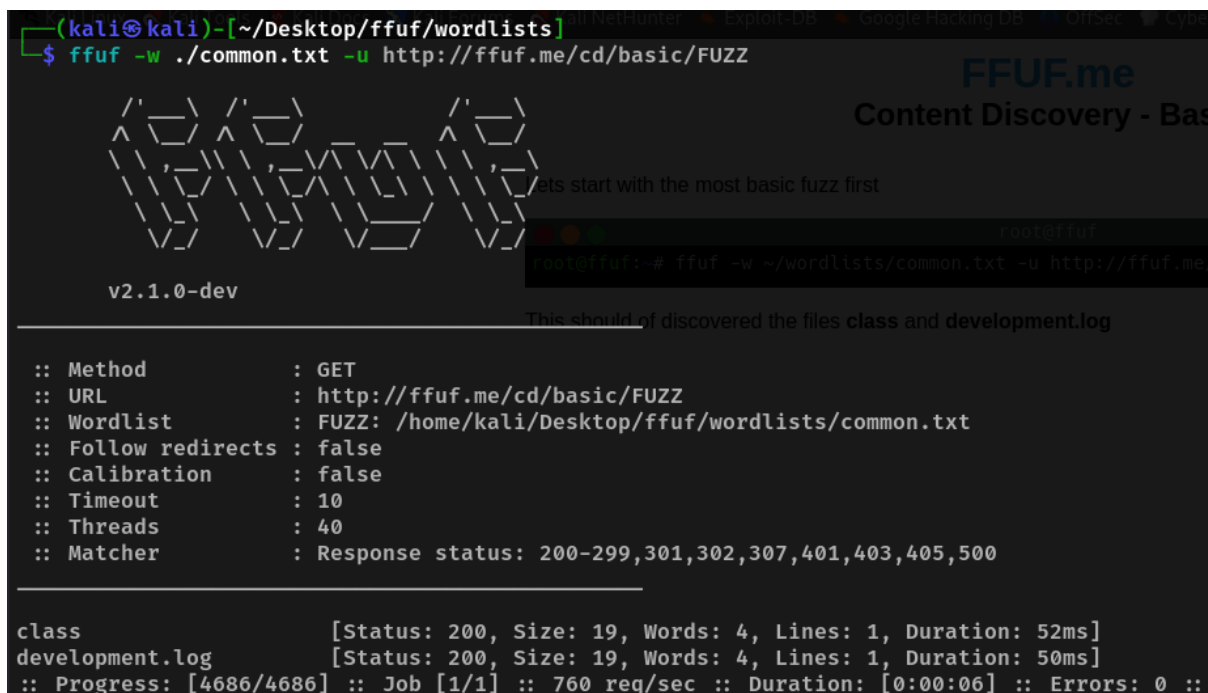
This should of discovered the files **class** and **development.log**

Next : Recursion

Rys 3.1 Interfejs scenariusza Basic Content Discovery na platformie ffuf.me

W wyniku przeprowadzonego fuzzingu na podstronie uwidocznionej na Rysunku 3.1 zidentyfikowano dwa pliki: class oraz development.log.

```
(kali@kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./common.txt -u http://ffuf.me/cd/basic/FUZZ
```



```
FFUF.me
Content Discovery - Bas

Lets start with the most basic fuzz first

root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/common.txt -u http://ffuf.me/

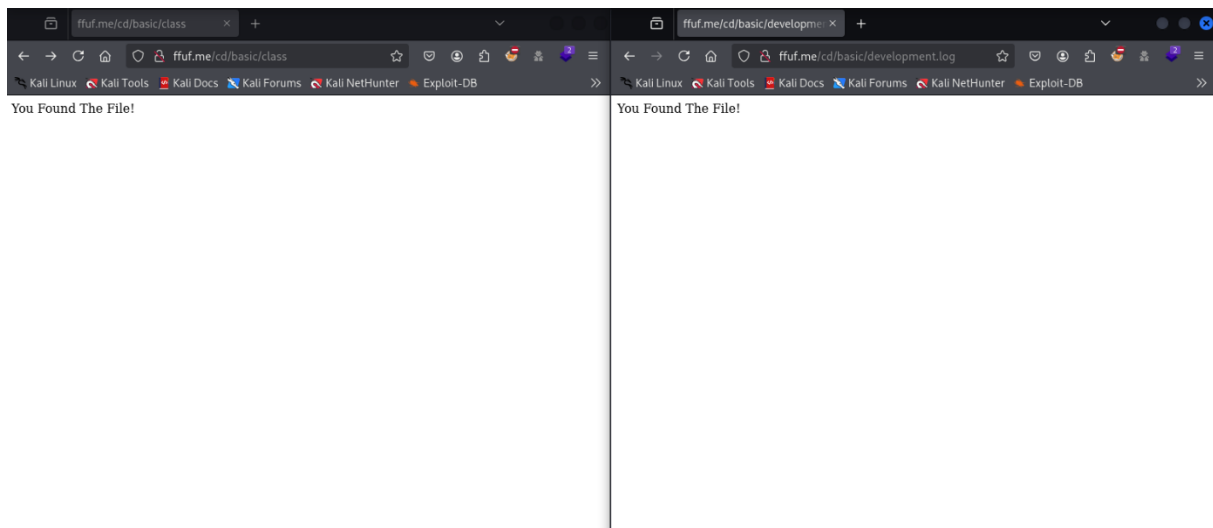
This should of discovered the files class and development.log

:: Method      : GET
:: URL         : http://ffuf.me/cd/basic/FUZZ
:: Wordlist    : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500

class          [Status: 200, Size: 19, Words: 4, Lines: 1, Duration: 52ms]
development.log [Status: 200, Size: 19, Words: 4, Lines: 1, Duration: 50ms]
:: Progress: [4686/4686] :: Job [1/1] :: 760 req/sec :: Duration: [0:00:06] :: Errors: 0 ::
```

Rys 3.2 Rezultaty wyszukiwania z wykorzystaniem wordlisty common.txt

Na Rysunku 3.1 zaprezentowano stronę poddaną procesowi skanowania, natomiast Rysunek 3.2 ilustruje zastosowaną komendę, która skutkowałą identyfikacją dwóch plików: class oraz development.log.



Rys 3.3 Zawartość zidentyfikowanych plików

### 3.2.2. Recursion

Kolejnym zaimplementowanym scenariuszem jest zastosowanie przełącznika `-recursion`, który funkcjonuje analogicznie do poprzedniego mechanizmu, jednakże wprowadza istotną modyfikację: gdy narzędzie ffuf podczas procesu skanowania identyfikuje katalog, inicjuje automatycznie kolejny proces skanowania dla tego katalogu, kontynuując tę procedurę rekurencyjnie aż do momentu wyczerpania możliwości lokalizacji kolejnych plików.

## FFUF.me Content Discovery - Recursion

This is similar to the first scan but this time we're using the `-recursion` switch. This switch tells ffuf that if it encounters a directory it should start another scan within that directory and so on until no more results are found

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/common.txt -recursion -u http://ffuf.me/cd/recursion/FUZZ
```

This scan should uncover the directory `/admin` and then `/admin/users` and finally the file `/admin/users/96`

[Back : Basic](#)

[Next : File Extensions](#)

Rys 3.4 Interfejs scenariusza Recursion na platformie ffuf.me

Implementacja skanowania na podstronie przedstawionej na Rysunku 3.4 z wykorzystaniem przełącznika `-recursion` skutkowałą identyfikacją katalogów `admin` oraz `users`, a także pliku o identyfikatorze `96`.

```

v2.1.0-dev
FFUF.me
Content Discovery - Recursion

:: Method      : GET
:: URL         : http://ffuf.me/cd/recursion/FUZZ
:: Wordlist     : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Follow redirects : false
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500

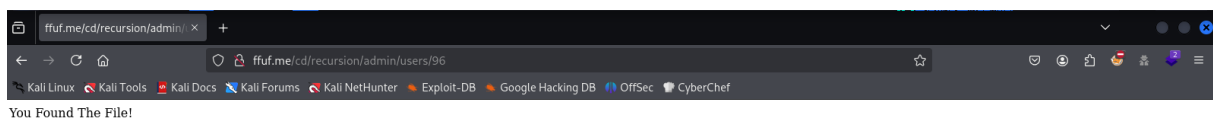
admin [Status: 301, Size: 0, Words: 1, Lines: 1, Duration: 54ms]
[INFO] Adding a new job to the queue: http://ffuf.me/cd/recursion/admin/FUZZ
[INFO] Starting queued job on target: http://ffuf.me/cd/recursion/admin/FUZZ

users [Status: 301, Size: 0, Words: 1, Lines: 1, Duration: 59ms]
[INFO] Adding a new job to the queue: http://ffuf.me/cd/recursion/admin/users/FUZZ
[INFO] Starting queued job on target: http://ffuf.me/cd/recursion/admin/users/FUZZ

96 [Status: 200, Size: 19, Words: 4, Lines: 1, Duration: 51ms]
:: Progress: [4686/4686] :: Job [3/3] :: 749 req/sec :: Duration: [0:00:06] :: Errors: 0 ::
```

Rys 3.5 Zidentyfikowane podkatalogi i pliki po przeprowadzeniu rekurencyjnego skanowania

Proces skanowania przedstawiony na Rysunku 3.5 skutecznie zidentyfikował uprzednio zaimplementowane katalogi /admin, /users oraz plik 96.



Rys 3.6 Zawartość zlokalizowanego pliku

Funkcjonalność `-recursive` wykazuje szczególną użyteczność w kontekście przeszukiwania złożonych struktur katalogów lub zagnieżdżonych podstron podczas realizacji procedur rekonesansu aktywnego.

### 3.2.3. File Extensions

Wykorzystując narzędzie ffuf, istnieje możliwość implementacji skanowania ukierunkowanego na konkretne rozszerzenia plików. W prezentowanym scenariuszu zastosowano przełącznik -e, który odpowiada za precyzyjne określenie typów rozszerzeń plików podlegających procesowi skanowania.



Rys 3.7 Interfejs scenariusza File Extensions na platformie ffuf.me

W procesie przeszukiwania katalogu widocznego na Rysunku 3.7 przewidziano identyfikację pliku users.log.

```
(kali@kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./common.txt -e .log -u http://ffuf.me/cd/ext/logs/FUZZ

FFUF.me
Content Discovery - File Extension Fuzzing

You've come across a directory called /logs but we cannot view its content
that the files stored in here are using the .log extension

Use the below scan with the -e switch to specify the extension type to add
wordlist to find the correct log

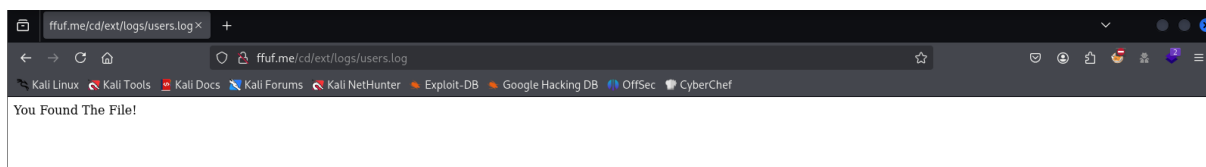
v2.1.0-dev

:: Method      : GET
:: URL         : http://ffuf.me/cd/ext/logs/FUZZ
:: Wordlist    : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Extensions : .log
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500

users.log [Status: 200, Size: 19, Words: 4, Lines: 1, Duration: 51ms]
:: Progress: [9372/9372] :: Job [1/1] :: 784 req/sec :: Duration: [0:00:14] :: Errors: 0 ::
```

Rys 3.8 Rezultaty skanowania z implementacją przełącznika filtrującego rozszerzenie .log

W wyniku implementacji fuzzingu uwidocznionego na Rysunku 3.8 poprawnie zidentyfikowano plik users.log.



Rys 3.9 Zawartość zlokalizowanego pliku users.log

Implementacja przełącznika -e umożliwia efektywną i precyzyjną identyfikację plików charakteryzujących się określonym rozszerzeniem, znacząco optymalizując proces skanowania.

### 3.2.4. No 404 Status

Narzędzie ffuf implementuje zaawansowaną funkcjonalność filtrowania odpowiedzi serwera na podstawie ich długości mierzonej w bajtach. Zastosowanie parametru -fs <Długość odpowiedzi> umożliwia efektywne filtrowanie odpowiedzi, które z wysokim prawdopodobieństwem nie zawierają poszukiwanych danych.

## FFUF.me

Lets try running the below ffuf example and see what happens

```
root@ffuf
```

```
root@ffuf:~# ffuf -w ~/wordlists/common.txt -u http://ffuf.me/cd/no404/FUZZ
```

From the ffuf response you'll notice that every file you've requested has come back as been found! It's not that you've got lucky and come across a load of content it's that the webpage displaying the "Page Cannot Be Found" message is not returning a 404 header

You'll notice that the "Page Cannot Be Found" page consistently has a file size of 669 bytes. Let's re-run the ffuf command but with the -fs switch which filters out any results that are 669 bytes in length.

```
root@ffuf
```

```
root@ffuf:~# ffuf -w ~/wordlists/common.txt -u http://ffuf.me/cd/no404/FUZZ -fs 669
```

This should cut the results down to just one file **secret**

[Back : File Extensions](#)

Next : Param Mining

Rys 3.10 Interfejs scenariusza No 404 Status na platformie ffuf.me

```
ffuf -w /usr/share/wordlists/ffuf/wordlists/common.txt -u http://ffuf.me/cd/no404/FUZZ -s 200-299,301,302,307,401,403,405,500 -t 10 -p 40 -c 40 -m GET
```

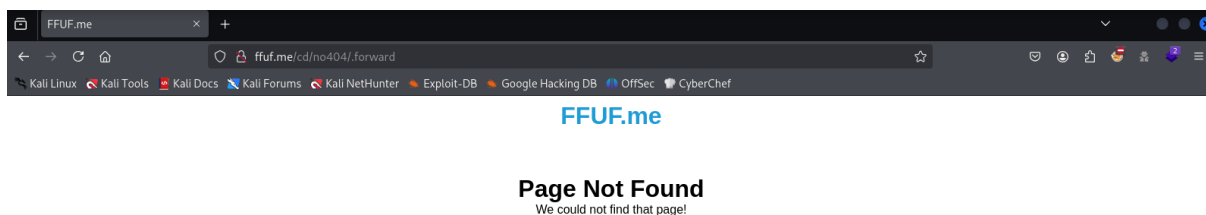
**FFUF.me**  
Content Discovery - No 404

In a perfect world all websites would respond correctly with the correct HTTP status codes. Lets try running the below ffuf example and see what happens

```
.gitkeep [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 50ms]
.git/index [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 51ms]
.mysql_history [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 51ms]
.git/config [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 54ms]
.git/logs/ [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 55ms]
.svn/entries [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 56ms]
.bash_history [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 57ms]
.sh_history [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 59ms]
.gitconfig [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 59ms]
.rhosts [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 62ms]
.swf [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 63ms]
.git-rewrite [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 61ms]
.forward [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 61ms]
.bashrc [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 80ms]
.cvsignore [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 78ms]
.svn [Status: 200, Size: 669, Words: 126, Lines: 23, Duration: 80ms]
```

Rys 3.11 Rezultaty procesu skanowania bez implementacji przełącznika filtrującego

Realizacja procesu skanowania bez implementacji mechanizmu filtrowania skutkuje otrzymaniem odpowiedzi oznaczonych kodem statusu 200 (Rysunek 3.11), jednak istotną obserwacją jest fakt, że długość odpowiedzi wynosi konsekwentnie 669 bajtów. Uzasadnione jest przeprowadzenie weryfikacji zawartości stron zwracających odpowiedzi o takiej charakterystyce.



Rys 3.12 Wizualizacja niestandardowej obsługi błędu 404

Pomimo otrzymania kodu odpowiedzi 200, strona przedstawiona na Rysunku 3.12 zawiera jednoznaczną informację o niedostępności zasobu. W konsekwencji niezbędne jest zastosowanie filtra długości odpowiedzi w celu efektywnego odfiltrowania stron zawierających komunikat „Page Not Found”.



```
(kali@kali)~[~/Desktop/ffuf/wordlists]
$ ffuf -w ./common.txt -u http://ffuf.me/cd/no404/FUZZ -fs 669

FFUF.me
Content Discovery - No 404

In a perfect world all websites would respond correctly with the correct HTTP status code.
Let's try running the below ffuf example and see what happens

v2.1.0-dev

:: Method      : GET
:: URL         : http://ffuf.me/cd/no404/FUZZ
:: Wordlist    : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
:: Filter      : Response size: 669

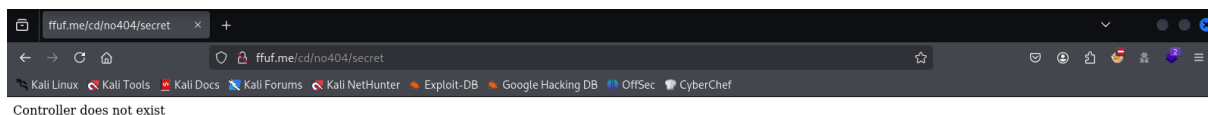
From the ffuf response you'll notice that every file you've requested has come back with a 404.
nd come across a load of content it's that the webpage doesn't exist.
You'll notice that the "Page Cannot Be Found" page consistently has a file size of 669 bytes.
command but with the -fs switch which filters out any results that are 669 bytes in size.

This should cut the results down to just one file secret

secret [Status: 200, Size: 25, Words: 4, Lines: 1, Duration: 72ms]
:: Progress: [4686/4686] :: Job [1/1] :: 743 req/sec :: Duration: [0:00:06] :: Errors: 0 ::
```

Rys 3.13 Rezultaty procesu skanowania z implementacją przełącznika filtrującego

W wyniku implementacji mechanizmu filtrowania uwidocznionego na Rysunku 3.13 skutecznie zidentyfikowano stronę o nazwie "secret".



Rys 3.14 Zawartość zlokalizowanej strony "secret"

Implementacja mechanizmu filtrowania długości odpowiedzi wykazuje szczególną użyteczność w kontekście eliminacji niepożądanych odpowiedzi, np. podczas realizacji ataków typu brute force na mechanizmy uwierzytelniania lub w sytuacjach analogicznych do prezentowanej, gdy zaimplementowano zaawansowane mechanizmy ochronne w postaci modyfikacji standardowych kodów statusu odpowiedzi serwera.

### 3.2.5. Param Mining

Kolejnym zaimplementowanym scenariuszem testowym jest identyfikacja parametrów zapytań, które potencjalnie umożliwiają dostęp do zastrzeżonych zasobów serwera.

## FFUF.me

### Content Discovery - Param Mining

Viewing the page </cd/param/data> you are greeted with the message "Required Parameter Missing" and the page is set to a HTTP Status Code of 400 which means Bad Request.

Using the below request we can try and find the missing parameter.

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/parameters.txt -u http://ffuf.me/cd/param/data?FUZZ=1
```

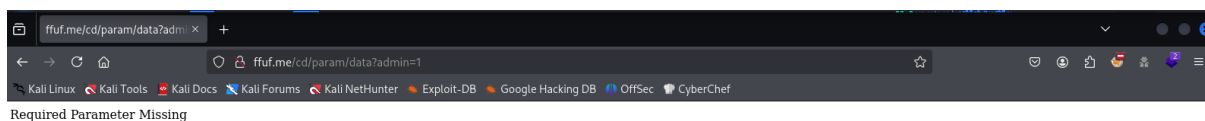
The above command should of returned the missing parameter of **debug**.

Back : No 404 Status

Next : Rate Limited

Rys 3.15 Interfejs scenariusza Param Mining na platformie ffuf.me

W ramach procedury testowej przeprowadzono proces identyfikacji parametrów umożliwiających autoryzowany dostęp do zasobów zlokalizowanych pod ścieżką `/cd/param/data`.



Rys 3.16 Weryfikacja dostępu do zasobów bez implementacji parametrów

W sytuacji braku implementacji odpowiedniego parametru, dostęp do zasobów zlokalizowanych pod ścieżką `/data` jest efektywnie zablokowany.

```
(kali@kali)~[~/Desktop/ffuf/wordlists]
$ ffuf -w ./parameters.txt -u http://ffuf.me/cd/param/data?FUZZ=1

FFUF.me
Content Discovery - Param I

Viewing the page /cd/param/data you are greeted with the message "Required parameter is missing. Please set a HTTP Status Code of 400 which means Bad Request."
Using the below request we can try and find the missing parameter.

v2.1.0-dev

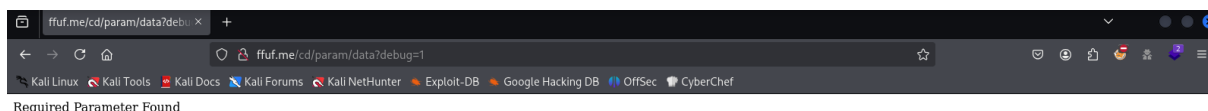
root@ffuf: # ffuf -w ~/wordlists/parameters.txt -u http://ffuf.me/cd/param/data?FUZZ=1

:: Method      : GET
:: URL         : http://ffuf.me/cd/param/data?FUZZ=1
:: Wordlist    : FUZZ: /home/kali/Desktop/ffuf/wordlists/parameters.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500

debug [Status: 200, Size: 24, Words: 3, Lines: 1, Duration: 62ms]
:: Progress: [2588/2588] :: Job [1/1] :: 819 req/sec :: Duration: [0:00:05] :: Errors: 0 ::
```

Rys 3.17 Rezultaty procesu identyfikacji parametrów

W wyniku przeprowadzenia kompleksowego przeszukiwania listy potencjalnych parametrów, jak uwidoczniono na Rysunku 3.17, zidentyfikowano parametr "debug", którego implementacja umożliwia efektywny dostęp do zasobów strony, co manifestuje się zwróceniem kodu statusu 200, sygnalizującego pomyślne wyświetlenie zawartości.



Rys 3.18 Weryfikacja poprawności zidentyfikowanego parametru

Po przeprowadzeniu weryfikacji parametru "debug" uwidoczniono na Rysunku 3.18, że stanowi on prawidłowy parametr umożliwiający autoryzowany dostęp do zastrzeżonych zasobów strony.

### 3.2.6. Rate Limited

Kolejnym zaimplementowanym scenariuszem testowym jest interakcja z serwerem implementującym limity częstotliwości zapytań, których przekroczenie skutkuje tymczasowym zablokowaniem możliwości komunikacji. W celu efektywnego obejścia mechanizmów zabezpieczających przed atakami typu DoS, niezbędna jest implementacja przełącznika -p, który wprowadza kontrolowane opóźnienie po realizacji każdego zapytania, oraz parametru -t,

definiującego liczbę równoległych wątków wykorzystywanych do transmisji zapytań do serwera.

## FFUF.me Content Discovery - Rate Limited

Sometimes services are rate limited, this means you're only allowed to send a certain amount of requests per second. In this instance the directory we are going to fuzz is limited to 50 requests per second. Try running the below command and you'll soon notice you're receiving a lot of 429 HTTP Statuses which means you're temporarily banned from making requests for a few seconds.

We're using the -mc switch to only display http statuses 200 and 429.

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/common.txt -u http://ffuf.test/cd/rate/FUZZ -mc 200,429
```

Now try running the below command, the -p switch causes the application to pause 0.1 seconds per request and the -t switch creates 5 versions of ffuf which means a maximum of 50 requests per second.

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/common.txt -t 5 -p 0.1 -u http://ffuf.test/cd/rate/FUZZ -mc 200,429
```

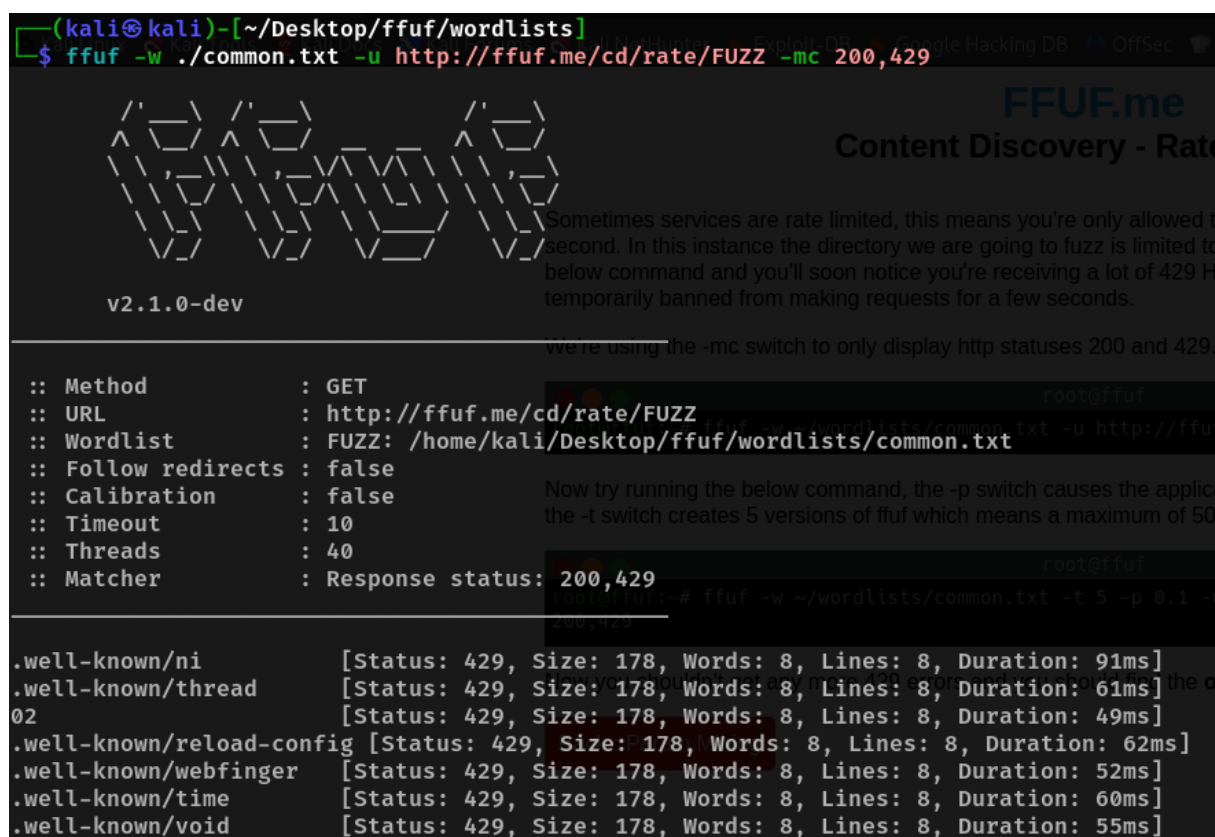
Now you shouldn't get any more 429 errors and you should find the **oracle** file.

[Back : Param Mining](#)[Next : Pipes](#)

Rys 3.19 Interfejs scenariusza Rate Limited na platformie ffuf.me

Strona przedstawiona na Rysunku 3.19 została skonfigurowana z implementacją limitów częstotliwości zapytań i poddana zostanie procesowi skanowania z wykorzystaniem narzędzia ffuf.

```
(kali㉿kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./common.txt -u http://ffuf.me/cd/rate/FUZZ -mc 200,429
```



```
FFUF.me
Content Discovery - Rate

Sometimes services are rate limited, this means you're only allowed to
second. In this instance the directory we are going to fuzz is limited to
below command and you'll soon notice you're receiving a lot of 429 H
temporarily banned from making requests for a few seconds.

We're using the -mc switch to only display http statuses 200 and 429.

:: Method      : GET
:: URL         : http://ffuf.me/cd/rate/FUZZ
:: Wordlist     : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Follow redirects : false
:: Calibration  : false
:: Timeout      : 10
:: Threads     : 40
:: Matcher      : Response status: 200,429

.well-known/ni      [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 91ms]
.well-known/thread  [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 61ms]
02                  [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 49ms]
.well-known/reload-config [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 62ms]
.well-known/webfinger [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 52ms]
.well-known/time     [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 60ms]
.well-known/void     [Status: 429, Size: 178, Words: 8, Lines: 8, Duration: 55ms]
```

Rys 3.20 Rezultaty procesu skanowania bez implementacji limitów częstotliwości

Proces skanowania uwidoczniiony na Rysunku 3.20 jednoznacznie wskazuje, że serwer implementuje automatyczną blokadę już na poziomie początkowych odpowiedzi, co wynika z transmisji nadmiernej liczby zapytań w jednostce czasu. W procesie testowym zaimplementowano również przełącznik `-mc`, którego celem było filtrowanie odpowiedzi w celu wyświetlenia wyłącznie tych oznaczonych kodami 200 lub 429.

```
(kali@kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./common.txt -t 5 -p 0.1 -u http://ffuf.me/cd/rate/FUZZ -mc 200,429

v2.1.0-dev

Sometimes services are rate limited, this means you're only allowed to send
second. In this instance the directory we are going to fuzz is limited to 50 req
below command and you'll soon notice you're receiving a lot of 429 HTTP St
temporarily banned from making requests for a few seconds.

We're using the -mc switch to only display http statuses 200 and 429.

:: Method      : GET
:: URL         : http://ffuf.me/cd/rate/FUZZ
:: Wordlist     : FUZZ: /home/kali/Desktop/ffuf/wordlists/common.txt
:: Follow redirects : false
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 5
:: Delay       : 0.10 seconds
:: Matcher     : Response status: 200,429

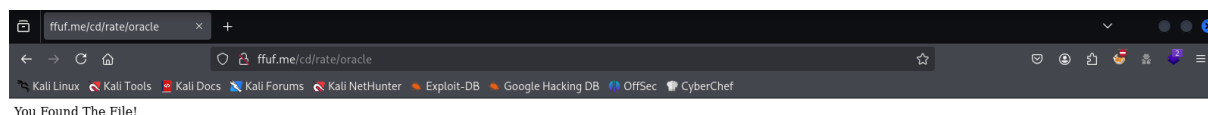
Now try running the below command, the -p switch causes the application to
ates 5 versions of ffuf which means a maximum of 50 requests

Now you shouldn't get any more 429 errors and you should find the oracle fi
Back - Param Mining

oracle [Status: 200, Size: 19, Words: 4, Lines: 1, Duration: 49ms]
:: Progress: [4172/4686] :: Job [1/1] :: 33 req/sec :: Duration: [0:02:07] :: Errors: 0 ::
```

Rys 3.21 Rezultaty procesu skanowania z implementacją limitów częstotliwości zapytań

Proces skanowania zrealizowany i zaprezentowany na Rysunku 3.21 umożliwia sformułowanie konkluzji, że implementacja mechanizmów limitujących częstotliwość zapytań przyniosła oczekiwane rezultaty. W wyniku przeprowadzonego skanowania skutecznie zidentyfikowano podstronę o nazwie "Oracle".



Rys 3.22 Weryfikacja zawartości zlokalizowanej strony

### 3.2.7. Pipes

Mechanizm pipowania (pipelines) w narzędziu ffuf umożliwia bezpośrednią transmisję danych z zewnętrznych narzędzi lub poleceń, stanowiąc alternatywę dla konwencjonalnych, predefiniowanych list słów. Funkcjonalność ta jest realizowana poprzez implementację parametru -w -, który instruuje narzędzie ffuf o konieczności akwizycji danych wejściowych bezpośrednio z potoku (stdin).

## FFUF.me Content Discovery - Pipes

Sometimes there's situations where the wordlists that we have aren't useful for the job required.

For instance you might want to test something for an IDOR vulnerability and try some integers against it.

Instead of creating a wordlist with the required integers for the job we can use the linux seq command instead and pipe the result straight into ffuf.

```
root@ffuf
root@ffuf:~# seq 1 1000 | ffuf -w - -u http://ffuf.me/cd/pipes/user?id=FUZZ
```

The above command should discover a valid result of **657**

What if the ID's are hashed in base64 or md5, this is a common way for developers to try and obfuscate the input to make it appear more random.

Unfortunately Linux doesn't have a good built in way to do this when piping in multiple lines so I developed the below bash script which can be altered to your own requirements. You'll need to save it in the `/usr/local/bin` with execute permissions so it's available from any directory.

```
root@ffuf
#!/bin/bash
while read i
do
  if [ "$i" == "md5" ]; then
    echo -n $i | md5sum | awk '{ print $1 }'
  elif [ "$i" == "b64" ]; then
    echo -n $i | base64
  else
    echo $i
  fi
done
```

Try the below to find the base64 encoded ID

```
root@ffuf
root@ffuf:~# seq 1 1000 | hashit b64 | ffuf -w - -u http://ffuf.me/cd/pipes/user2?id=FUZZ
```

The above command should discover a valid result of **ODg4Cg==** which is Integer **888**

And lastly the below to find the md5 hashed ID

```
root@ffuf
root@ffuf:~# seq 1 1000 | hashit md5 | ffuf -w - -u http://ffuf.me/cd/pipes/user3?id=FUZZ
```

The above command should discover a valid result of **4daa3db355ef2b0e64b472908cb70f0d** which is integer **934**

[Back : Rate Limited](#)

[Content Discovery Module Complete](#)

Rys 3.23 Interfejs scenariusza Pipes na platformie ffuf.me

Na Rysunku 3.23 zaprezentowano interfejs webowy platformy ffuf.me dla scenariusza testowego "Pipes". Wyjaśnia on koncepcję wykorzystania potoków do dynamicznego generowania danych wejściowych dla ffuf, na przykład sekwencji liczb za pomocą polecenia seq.

```
(kali@kali)~[~/Desktop/ffuf/wordlists]
$ seq 1 1000 | ffuf -w - -u http://ffuf.me/cd/pipes/user?id=FUZZ
```

FFUF.me  
Content Discovery - Pipes

Sometimes there's situations where the wordlists that we have aren't useful. For instance you might want to test something for an IDOR vulnerability and instead of creating a wordlist with the required integers for the job we can use a sequence of integers and pipe the result straight into ffuf.

v2.1.0-dev

```
root@ffuf:~# seq 1 1000 | ffuf -w - -u http://ffuf.me/cd/pipes/user?id=FUZZ
The above command should discover a valid result of 657

What if the ID's are hashed in base64 or md5, this is a common way for developers to make it appear more random.

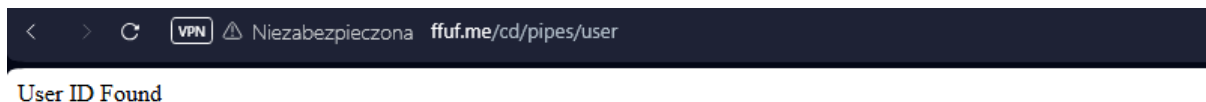
Unfortunately Linux doesn't have a good built in way to do this when piping a sequence of integers. You'll need to use a tool like perl to generate a sequence of integers that meets the requirements. You'll need to execute permissions so it's available from any directory.
```

:: Method	: GET
:: URL	: http://ffuf.me/cd/pipes/user?id=FUZZ
:: Wordlist	: FUZZ: -
:: Follow redirects	: false
:: Calibration	: false
:: Timeout	: 10
:: Threads	: 40
:: Matcher	: Response status: 200-299,301,302,307,401,403,405,500

```
657 [Status: 200, Size: 13, Words: 3, Lines: 1, Duration: 56ms]
:: Progress: [1000/1000] :: Job [1/1] :: 749 req/sec :: Duration: [0:00:03] :: Errors: 0 ::
```

Rys 3.24 Rezultaty procesu skanowania bez implementacji mechanizmów kodowania

Rysunek 3.24 przedstawia wyniki wykonania pierwszego polecenia ffuf wykorzystującego pipowanie. Polecenie `seq 1 1000` generuje liczby od 1 do 1000, które są przekazywane jako wejście (-w -) do ffuf w celu testowania parametru id w URL. Skanowanie pomyślnie zidentyfikowało wartość 657 jako prawidłowy identyfikator.



Rys 3.25 Weryfikacja poprawności zidentyfikowanego identyfikatora

Ten rysunek (Rysunek 3.25) pokazuje weryfikację wyniku uzyskanego w poprzednim kroku. Otwarcie w przeglądarce adresu URL `ffuf.me/cd/pipes/user` z parametrem `id=657` potwierdza znalezienie prawidłowego identyfikatora komunikatem "User ID Found".



```
(kali㉿kali)-[~/Desktop/ffuf/wordlists]
$ seq 1 1000 | while read i; do echo -n "$i" | base64; done | ffuf -w - -u http://ffuf.me/cd/pipes/user2?id=FUZZ

name-that-hash | Kali Linux Tools
https://www.kali.org/tools/name-that-hash/
name-that-hash 3.x which supports the identification of over 175 unique hash types
https://github.com/j0hnn1e/nthash
name-that-hash 3.x is able to identify a ...

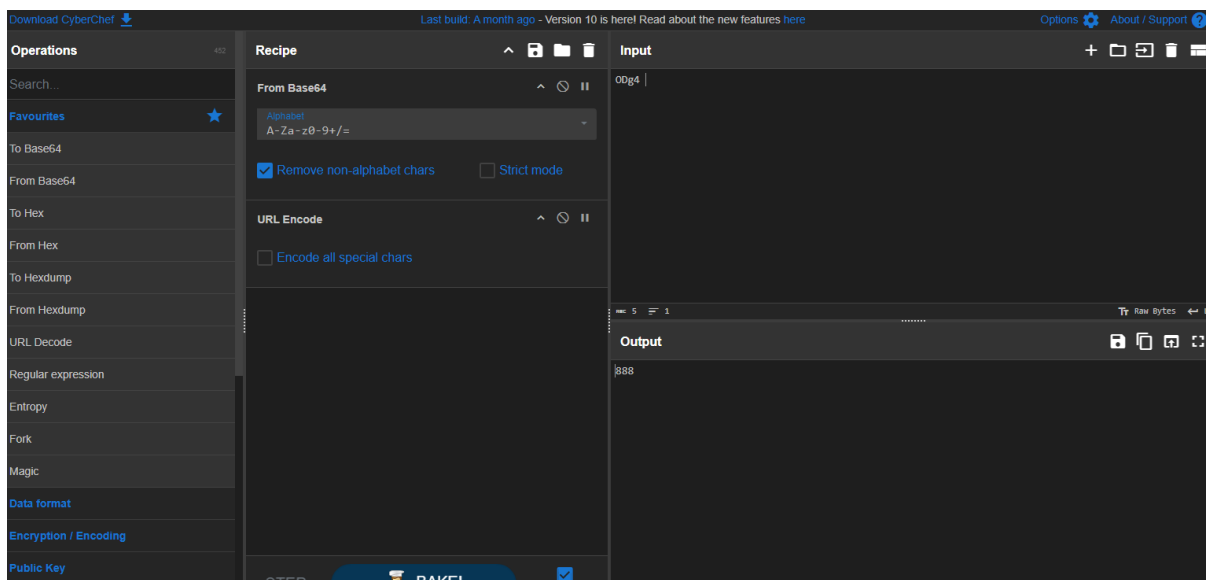
v2.1.0-dev

:: Method      : GET
:: URL         : http://ffuf.me/cd/pipes/user2?id=FUZZ
:: Wordlist     : FUZZ: -
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500

ODg4 [Status: 200, Size: 13, Words: 3, Lines: 1, Duration: 50ms]
:: Progress: [1000/1000] :: Job [1/1] :: 775 req/sec :: Duration: [0:00:01] :: Errors: 0 ::
```

Rys 3.26 Rezultaty procesu skanowania z implementacją kodowania Base64

W przypadku drugiego użytkownika (user2) niezbędna była implementacja mechanizmu kodowania wartości w standardzie Base64 przed transmisją zapytania. W wyniku procesu skanowania zidentyfikowano identyfikator w postaci zakodowanej: ODg4. W celu efektywnego wykorzystania uzyskanej informacji, konieczne jest przeprowadzenie procesu dekodowania w celu ustalenia pierwotnej wartości liczbowej.



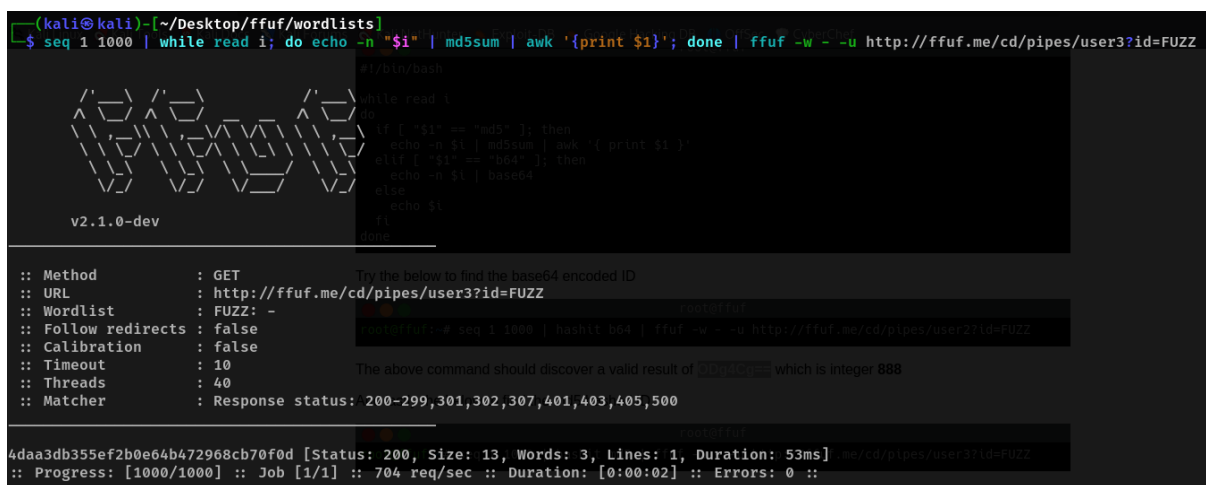
Rys 3.27 Proces dekodowania zidentyfikowanego identyfikatora

Rysunek 3.27 ilustruje proces dekodowania wartości ODg4 uzyskanej w poprzednim skanowaniu. Użyto do tego narzędzia CyberChef, które potwierdziło, że zakodowana wartość Base64 odpowiada liczbie 888.



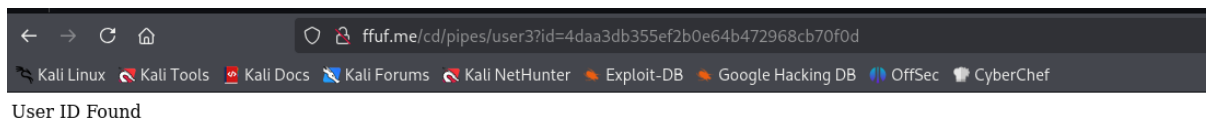
Rys 3.28 Weryfikacja poprawności zdekodowanego identyfikatora

Ten rysunek (Rysunek 3.28) przedstawia weryfikację zdekodowanego identyfikatora. Dostęp do adresu URL `ffuf.me/cd/pipes/user2` z parametrem `id=ODg4` (czyli zdekodowanym 888) skutkuje wyświetleniem komunikatu "User ID Found", co potwierdza poprawność identyfikatora.



Rys 3.29 Rezultaty procesu skanowania z implementacją hashowania MD5

Rysunek 3.29 ukazuje wyniki skanowania dla ścieżki `user3`. W tym przypadku liczby generowane przez `seq 1 1000` są hashowane algorytmem MD5 przy użyciu `md5sum` i `awk`, a następnie przekazywane do `ffuf`. Narzędzie identyfikuje hash `4daa3db355ef2b0e64b472968cb70f0d` jako prawidłowy identyfikator.



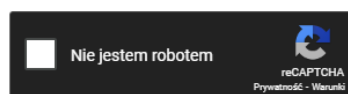
Rys 3.30 Weryfikacja poprawności zidentyfikowanego identyfikatora z wykorzystaniem hashowania MD5

Na Rysunku 3.30 widzimy proces łamania (crackowania) hasha MD5 uzyskanego w poprzednim kroku. Użyto do tego celu narzędzia online CrackStation, które pomyślnie zidentyfikowało, że hash `4daa...` odpowiada wartości 934.

## Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

4daa3db355ef2b0e64b472968cb70f0d



Crack Hashes

**Supports:** LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, rpeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults

Hash	Type	Result
4daa3db355ef2b0e64b472968cb70f0d	md5	934

**Color Codes:** **Green** Exact match, **Yellow** Partial match, **Red** Not found.

[Download CrackStation's Wordlist](#)

### Rys 3.31 Weryfikacja poprawności zidentyfikowanego identyfikatora użytkownika

Ostatni rysunek w tej sekcji (Rysunek 3.31) pokazuje finalną weryfikację. Próba dostępu do URL `ffuf.me/cd/pipes/user3` z parametrem `id` ustawionym na złamany hash (odpowiadający 934) skutkuje wyświetleniem strony z komunikatem "User ID Found", co kończy scenariusz Pipes.

### 3.2.8. Virtual host enumeration

Virtual host enumeration stanowi zaawansowaną technikę identyfikacji wirtualnych hostów zaimplementowanych na serwerze internetowym.

## FFUF.me Subdomains - Virtual Host Enumeration

FFUF can be used to discovery subdomains by the use of virtual hosts and changing the Host header

Try running the below ffuf:

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me
```

You'll see from the results that every result comes back with a size of 1495 Bytes.

Now try running the below ffuf scan but this time using the `-fs` switch to filter out any results that are 1495 bytes.

```
root@ffuf
root@ffuf:~# ffuf -w ~/wordlists/subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me -fs 1495
```

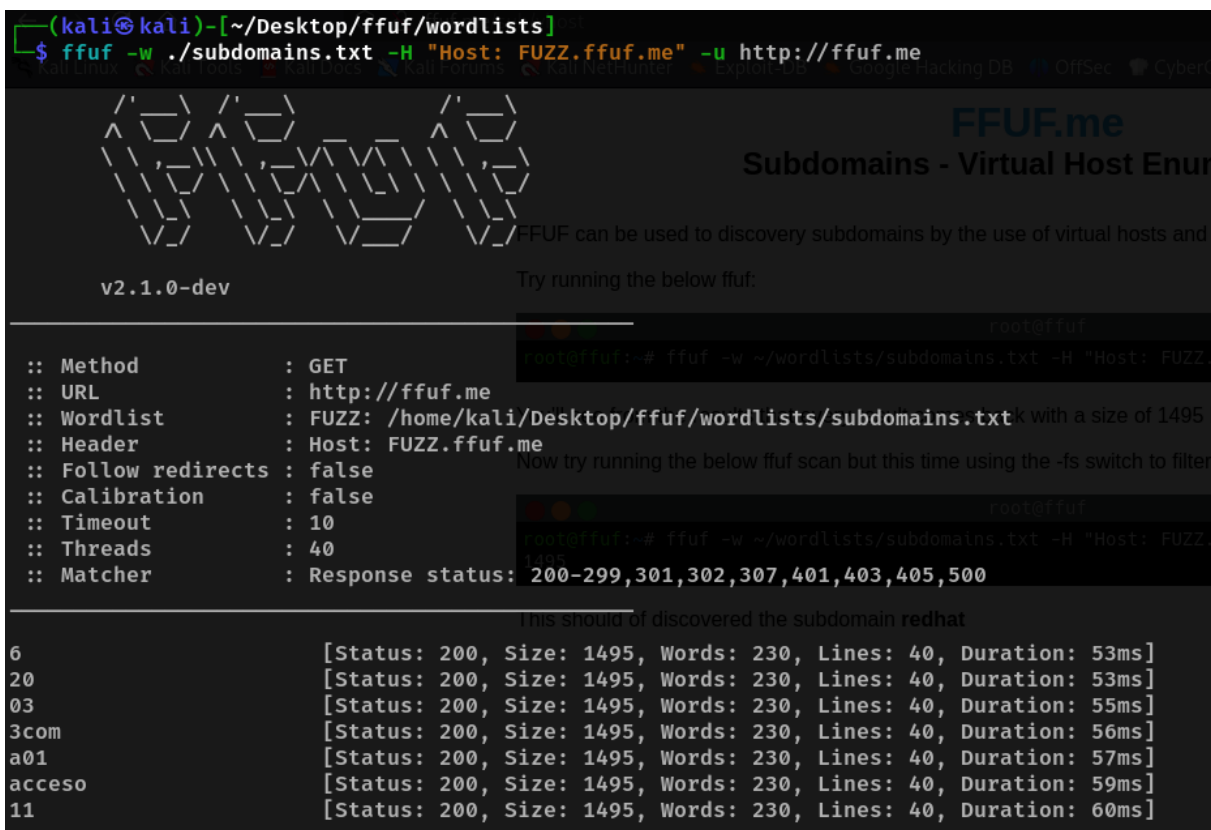
This should of discovered the subdomain **redhat**

Module Complete

### Rys 3.32 Interfejs scenariusza Virtual host enumeration na platformie ffuf.me

Na Rysunku 3.32 przedstawiono interfejs platformy ffuf.me dla scenariusza enumeracji hostów wirtualnych. Wyjaśniono w nim, że ffuf może być użyty do odkrywania subdomen poprzez modyfikację nagłówka Host i podano przykładowe polecenie inicjujące skanowanie bez filtrowania.

```
(kali@kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me
```



**FFUF.me**  
Subdomains - Virtual Host Enumeration

v2.1.0-dev

Try running the below ffuf:

```
root@ffuf:~# ffuf -w ~/wordlists/subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me
```

Now try running the below ffuf scan but this time using the -fs switch to filter

```
root@ffuf:~# ffuf -w ~/wordlists/subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me -fs
```

This should have discovered the subdomain redhat

Subdomain	Status	Size	Words	Lines	Duration
6	200	1495	230	40	53ms
20	200	1495	230	40	53ms
03	200	1495	230	40	55ms
3com	200	1495	230	40	56ms
a01	200	1495	230	40	57ms
accesso	200	1495	230	40	59ms
11	200	1495	230	40	60ms

Rys 3.33 Wstępna analiza odpowiedzi serwera podczas procesu skanowania

Pomimo otrzymywania odpowiedzi oznaczonych kodem statusu 200 widoczne na rysunku 3.33, ewidentna jest implementacja zaawansowanego mechanizmu bezpieczeństwa, który celowo nie transmituje standardowego kodu 404 dla nieistniejących hostów. W konsekwencji niezbędna jest implementacja zaawansowanego mechanizmu filtrowania odpowiedzi na podstawie ich długości.

```
(kali@kali)-[~/Desktop/ffuf/wordlists]
$ ffuf -w ./subdomains.txt -H "Host: FUZZ.ffuf.me" -u http://ffuf.me -fs 1495

v2.1.0-dev

:: Method      : GET
:: URL         : http://ffuf.me
:: Wordlist    : FUZZ: /home/kali/Desktop/ffuf/wordlists/subdomains.txt
:: Header     : Host: FUZZ.ffuf.me
:: Follow redirects : false
:: Calibration : false
:: Timeout     : 10
:: Threads    : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
:: Filter     : Response size: 1495

redhat [Status: 200, Size: 15, Words: 2, Lines: 1, Duration: 47ms]
:: Progress: [1907/1907] :: Job [1/1] :: 793 req/sec :: Duration: [0:00:02] :: Errors: 0 ::
```

Rys 3.34 Rezultaty procesu skanowania z implementacją filtrowania

Ten rysunek (Rysunek 3.34) ilustruje rezultaty skanowania po zastosowaniu mechanizmu filtrowania. Użycie opcji -fs 1495 pozwoliło odrzucić wszystkie odpowiedzi o rozmiarze 1495 bajtów. W efekcie ffuf pomyślnie zidentyfikował tylko jedną, prawidłową subdomenę redhat, która zwróciła odpowiedź o innym rozmiarze (15 bajtów).

### 3.3. Analiza wyników symulacji

Analiza wyników symulacji przeprowadzonych na platformie ffuf.me wykazała, że narzędzie ffuf umożliwia precyzyjną identyfikację zasobów oraz potencjalnych wektorów ataku w aplikacjach webowych. W trakcie testów zastosowano różnorodne scenariusze, takie jak podstawowe odkrywanie zawartości, rekurencyjne przeszukiwanie struktur katalogowych, filtrowanie wyników na podstawie rozszerzeń plików oraz analizę niestandardowych odpowiedzi serwera (np. mechanizm No 404 Status). Wyniki symulacji potwierdziły, że zastosowanie dedykowanych wordlist, precyzyjne filtrowanie odpowiedzi (m.in. według długości) oraz automatyzacja procesów skanowania znacząco zwiększa efektywność wykrywania ukrytych zasobów i nieprawidłowości konfiguracji. Testy wykazały także, że implementacja mechanizmów takich jak param mining, rate limiting oraz integracja metod

pipowania pozwala na dynamiczne dostosowanie strategii badawczej do specyfiki badanej infrastruktury, eliminując fałszywe alarmy i umożliwiając dokładną analizę bezpieczeństwa.

### **3.4. Wnioski z przeprowadzonych testów**

Wnioski płynące z przeprowadzonych symulacji wskazują, że narzędzie ffuf stanowi nieocenione wsparcie w procesach audytu bezpieczeństwa systemów webowych. Otrzymane wyniki jednoznacznie sugerują, iż wdrożenie zaawansowanych technik fuzzingu, łączących precyzyjną identyfikację zasobów z dynamiczną analizą parametrów oraz filtrowaniem odpowiedzi, jest kluczowe dla skutecznej oceny stanu zabezpieczeń aplikacji. Rekomenduje się dalszą integrację narzędzi fuzzingowych z systemami monitoringu i automatyzacji, co umożliwi bieżące aktualizowanie metod ochrony oraz szybką reakcję na pojawiające się zagrożenia. Kompleksowa analiza symulacji podkreśla konieczność stosowania zautomatyzowanych procedur testowych w celu minimalizacji ryzyka naruszenia bezpieczeństwa, a także wskazuje na potencjalne obszary, w których konieczne są dodatkowe mechanizmy ochronne, by zapewnić wysoki poziom integralności i dostępności systemów IT.

## **4. Porównanie z innymi narzędziami do fuzzingu**

W dzisiejszym świecie cyberbezpieczeństwa skuteczne narzędzia do fuzzingu odgrywają kluczową rolę w wykrywaniu potencjalnych luk i słabości aplikacji webowych. W tym podrozdziale przyjrzymy się trzem popularnym rozwiązaniom – DirBuster, wfuzz i ffuf – analizując ich specyfikę, możliwości konfiguracji, szybkość działania oraz wygodę użytkowania.

### **4.1. DirBuster**

DirBuster to narzędzie stworzone do wykrywania ukrytych katalogów i plików na serwerach WWW. Działa na zasadzie ataku brute-force, wykorzystując obszerną listę predefiniowanych słów kluczowych, co pozwala na dokładne zmapowanie struktury serwera. Jego największym atutem jest intuicyjny interfejs graficzny, który ułatwia analizę wyników, co czyni go dobrym wyborem dla osób mniej zaznajomionych z narzędziami działającymi wyłącznie w trybie wiersza poleceń. Jednak w porównaniu z nowszymi rozwiązaniami DirBuster może mieć pewne ograniczenia pod względem elastyczności i szybkości działania, co może stanowić wyzwanie w dynamicznie zmieniającym się środowisku cyberbezpieczeństwa.

## 4.2. wfuzz

Wfuzz to wszechstronne narzędzie, które oferuje szerokie możliwości, nie tylko w klasycznym fuzzingu, ale także w przeprowadzaniu bardziej skomplikowanych ataków na aplikacje webowe. Dzięki zaawansowanym opcjom konfiguracji użytkownik może precyzyjnie dostosować parametry testów, w tym obsługę autoryzacji czy dynamiczne modyfikowanie zapytań. Wfuzz jest ceniony przez specjalistów ds. bezpieczeństwa, którzy potrzebują elastycznego i skalowalnego narzędzia, choć jego efektywne wykorzystanie wymaga zaawansowanej znajomości wiersza poleceń.

## 4.3. Zalety i wady w porównaniu z ffuf

Ffuf wyróżnia się przede wszystkim wysoką wydajnością i szybkością działania, co jest możliwe dzięki optymalizacji kodu i wsparciu dla procesów równoległych. Narzędzie to pozwala na szybkie skanowanie zasobów i łatwą integrację z innymi rozwiązaniami, co czyni je atrakcyjnym wyborem dla doświadczonych analityków. Dużą zaletą ffuf jest elastyczność konfiguracji, umożliwiającą dostosowanie strategii ataku do konkretnych wymagań testowych. Minusem może być brak interfejsu graficznego, co dla osób preferujących bardziej wizualne narzędzia może stanowić barierę. Dodatkowo, choć ffuf oferuje wiele opcji analizy wyników, nie są one tak intuicyjne, jak w przypadku DirBuster.

Wybór odpowiedniego narzędzia do fuzzingu zależy od specyfiki testów, poziomu doświadczenia użytkownika oraz oczekiwań dotyczących szybkości i przejrzystości wyników.

- 1) **DirBuster** sprawdzi się, jeśli priorytetem jest graficzna wizualizacja i łatwość obsługi.
- 2) **wfuzz** będzie najlepszym wyborem w przypadku bardziej skomplikowanych scenariuszy testowych, gdzie liczy się precyzyjna konfiguracja.
- 3) **ffuf** to idealne rozwiązanie dla zaawansowanych użytkowników, którzy cenią wydajność i elastyczność.

Każde z tych narzędzi ma swoje unikalne cechy, które można optymalnie wykorzystać w zależności od specyfiki testów i wymagań w zakresie cyberbezpieczeństwa.

# 5. Podsumowanie i wnioski

## 5.1. Zalety i ograniczenia ffuf

Ffuf to jedno z najbardziej efektywnych narzędzi do fuzzingu, wyróżniające się przede wszystkim wysoką wydajnością oraz możliwością wykonywania operacji równoległych.

Dzięki temu proces testowania jest znacznie szybszy i bardziej efektywny. Jego największą zaletą jest elastyczność konfiguracji, która pozwala precyzyjnie dostosować parametry testów do specyfiki badanego systemu. To szczególnie istotne w świecie cyberbezpieczeństwa, gdzie zagrożenia nieustannie ewoluują.

Mimo licznych zalet, ffuf ma też swoje ograniczenia. Brak graficznego interfejsu użytkownika oraz ograniczone możliwości wizualizacji wyników mogą stanowić wyzwanie dla osób preferujących bardziej intuicyjne, graficzne narzędzia. Niemniej jednak, jego funkcjonalność oraz możliwość integracji z innymi systemami analitycznymi sprawiają, że jest to niezwykle wartościowe wsparcie w testach penetracyjnych.

## **5.2. Rekomendacje dotyczące wykorzystania w testach penetracyjnych**

W kontekście testów penetracyjnych ffuf doskonale sprawdza się jako element zautomatyzowanego systemu wykrywania luk bezpieczeństwa. Specjaliści ds. cyberbezpieczeństwa mogą integrować go z innymi narzędziami diagnostycznymi, co pozwala na kompleksową analizę infrastruktury IT. Aby uzyskać jak najdokładniejsze dane, warto regularnie walidować wyniki oraz dostosowywać parametry testowe do specyfiki badanego środowiska. W praktyce ffuf najlepiej sprawdza się jako uzupełnienie tradycyjnych metod testowania, dodając do nich dynamiczne i szybkie podejście do identyfikacji potencjalnych zagrożeń.

## **5.3. Perspektywy rozwoju narzędzia**

Patrząc w przyszłość, perspektywy rozwoju ffuf wydają się bardzo obiecujące. Dalsza optymalizacja kodu oraz wdrożenie bardziej zaawansowanych funkcji, takich jak lepsza wizualizacja wyników czy integracja z nowoczesnymi systemami raportowania, mogłyby znacząco zwiększyć atrakcyjność narzędzia. W miarę wzrostu zagrożeń cybernetycznych i rosnących wymagań wobec bezpieczeństwa aplikacji internetowych, ffuf będzie musiał ewoluować, by sprostać nowym wyzwaniom. Inwestycje w rozwój tego narzędzia mogą odegrać kluczową rolę w podnoszeniu poziomu ochrony systemów informatycznych na globalną skalę.



## 6. Bibliografia

- 1) Oficjalna dokumentacja i repozytorium ffuf: Głównie źródło informacji na temat narzędzia, jego opcji oraz przykładów użycia, dostępne na platformie GitHub.
  - URL: <https://github.com/ffuf/ffuf>
- 2) Materiał wideo (YouTube) - HackerSploit: "Fuzzing & Directory Brute-Force With ffuf": Wideo omawiające techniki fuzzingu i brute-force katalogów przy użyciu narzędzia ffuf, pokazujące praktyczne zastosowania i konfiguracje.
  - Kanał: HackerSploit
  - URL: <https://www.youtube.com/watch?v=9Hik0xy9qd0>
- 3) Materiał wideo (YouTube) - NahamSec: "What is Fuzzing (using ffuf)": Film wyjaśniający koncepcję fuzzingu w kontekście bezpieczeństwa aplikacji webowych, demonstrujący użycie ffuf do tego celu, wraz z przykładami i wskazówkami.
  - Kanał: NahamSec
  - URL: <https://www.youtube.com/watch?v=0v1CTSyRpMU>