

Web Application Fuzzing

Piotr Łaskawiec
piotr.laskawiec@elitesolutions.pl



Agenda

- Metody testowania oprogramowania
- Czym jest fuzzing?
- Proces fuzzingu
- Problemy
- Ograniczenia fuzzingu
- Dostępne rozwiązania
- Tworzenie własnego fuzzera
- **Prezentacja**
- Wady i zalety fuzzingu
- Przyszłość fuzzingu



Testowanie aplikacji

- Istnieje wiele zróżnicowanych metod testowania aplikacji – whitebox, blackbox, graybox.
- Każda z tych metod ma swoje wady jak i zalety – nie ma jednej, idealnej metody testowania.
- Dobór odpowiedniej metody testowania zależy od konkretnego problemu.



Testowanie aplikacji

Whitebox

Manualna analiza
kodu źródłowego

Automatyczna analiza
kodu źródłowego

Graybox

Manualny Reversing

Automatyczny Reversing

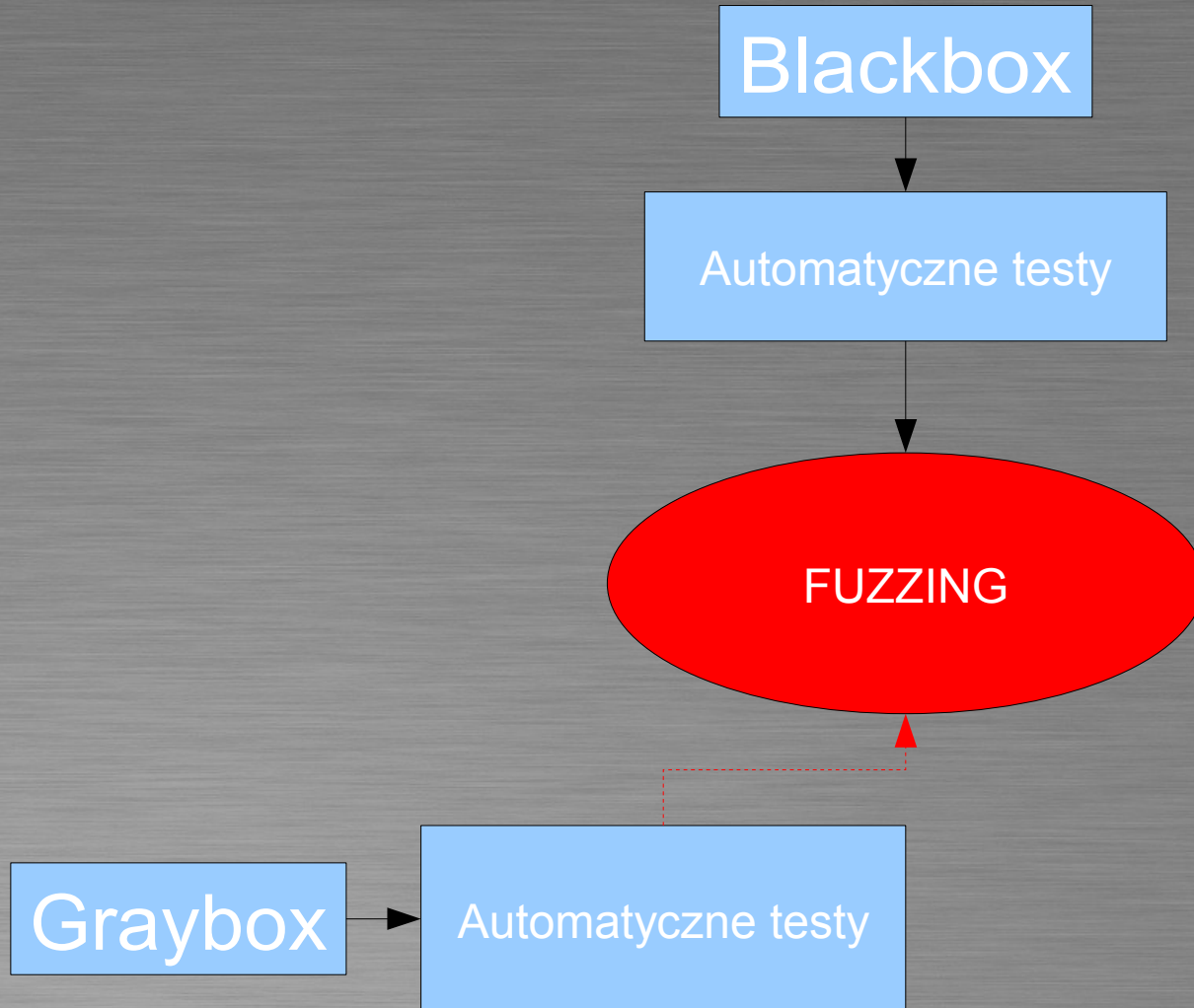
Blackbox

Manualne testy

Automatyczne testy



Testowanie aplikacji



Czym jest fuzzing?

Najogólniej mówiąc, fuzzing jest metodą testowania oprogramowania pod kątem występowania luk w bezpieczeństwie oraz nieprzewidzianych reakcji programu, za pomocą częściowo losowych danych.

W większości wypadków proces ten jest w pełni automatyczny, co pozwala na znaczne zredukowanie nakładów czasowych.



Fuzzing – co to znaczy w praktyce?

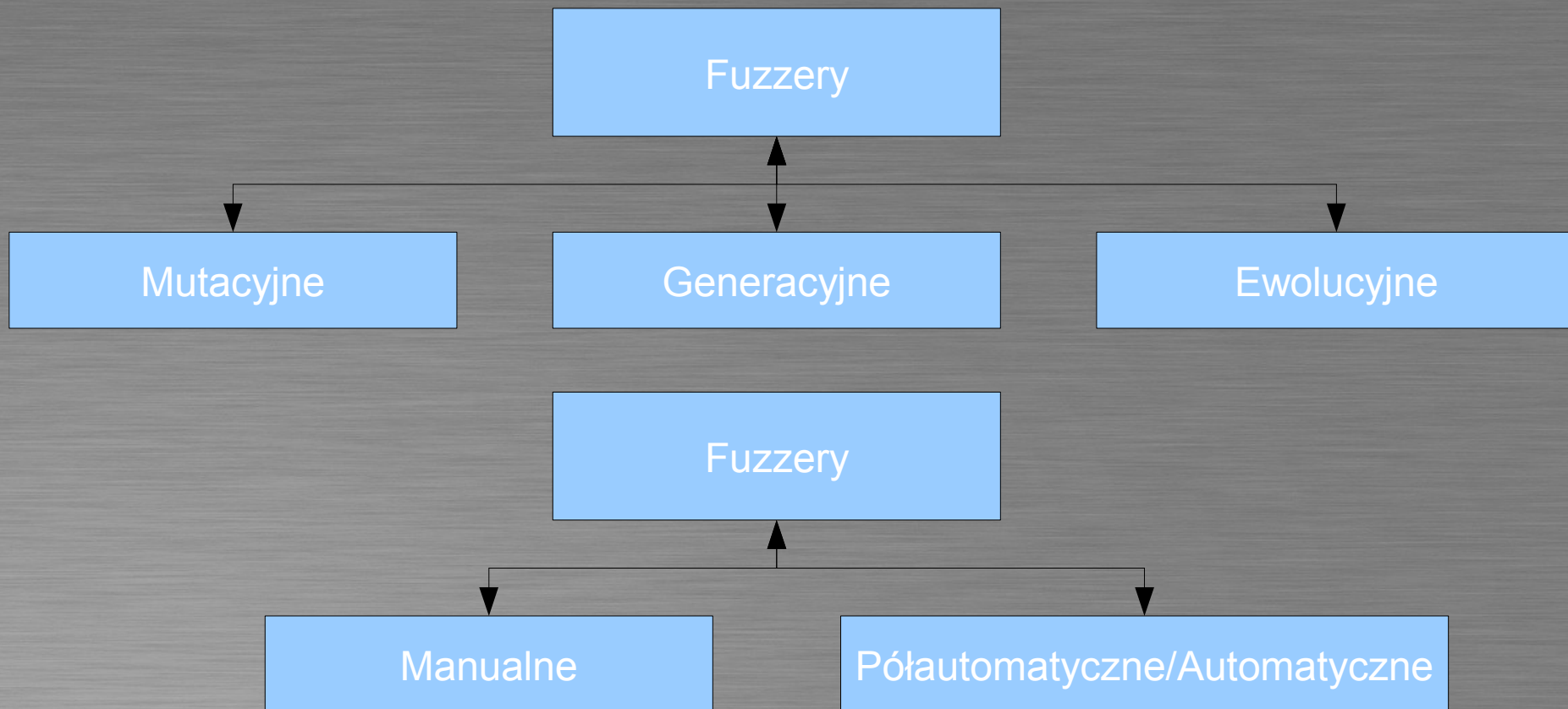
Fuzzing opiera się na przekazaniu do testowanego programu wadliwych danych (złe formatowanie, niepoprawna długość itp.), które zostaną zaakceptowane przez aplikację i spowodują nieprzewidzianą reakcję programu (zawieszenie się aplikacji, wyświetlenie błędu itd.).



Proces fuzzingu



Typy fuzzerów



Fuzzing – zastosowanie

Działaniu fuzzera możemy poddawać:

- Aplikacje działające lokalnie
- Aplikacje sieciowe
- **Aplikacje webowe**
- Kontrolki ActiveX
- Biblioteki współdzielone
- Pliki
- ...



Rozwój aplikacji webowych

- Obecnie aplikacje webowe przeżywają okres dynamicznego rozwoju
- Aplikacje webowe coraz częściej zastępują oprogramowanie „desktopowe” - przykładem może być np. Google Docs czy Photoshop Express
- Aplikacje webowe związane są z wieloma dziedzinami życia – od ochrony zdrowia po rozrywkę.



Popularne błędy

- Cross Site Scripting
- Cross Site Request Forgery
- Injections Bugs (SQL Injections, XPath Injections, SMTP Injections, Code Injections...)
- Bad Session Management
- Remote File Execution
- Logical Bugs
- ...



Stopień trudności

- Testowanie aplikacji pod kątem występowania nieskomplikowanych podatności (np. XSS czy SQL Injection) jest relatywnie proste.
- Poszukiwanie błędów, które nie posiadają wyraźnej zależności pomiędzy wysłanymi danymi testowymi a odpowiedzią programu jest o wiele trudniejsze.
- Wykrycie błędu logicznego jest praktycznie niemożliwe.



Proces fuzzingu aplikacji webowej



Problemy

- Identyfikacja punktów wejściowych
 - Skąd wiemy gdzie przekazywać dane?
- Generacja danych testowych
- Identyfikacja błędu
 - Skąd wiemy, że wystąpił błąd?
 - Jak wykryć błędy, które nie dają wyraźnych oznak istnienia.
- Zakończenie procesu testowania
 - Kiedy zakończyć fuzzing?



Identyfikacja punktów wejściowych



HTTP GET Request

Wartości przekazywane metodą GET:
XSS, Injection Bugs, RFI/LFI ...

Katalog:
Path Traversal Bug

```
GET /local_directory/test.php?param=1 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: test_cookie=1000
```

Wersja przeglądarki:
Injections Bugs, XSS

Ciasteczka:
Injection Bugs, XSS...



HTTP POST Request

Wersja protokołu:
Niepoprawne działanie aplikacji

POST /test.php **HTTP/1.1**
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Referer: **example2.com/index.php**
Keep-Alive: 300
Connection: keep-alive
Content-Length: 12

Strona odsyłająca:
Injection Bugs

test=**test123**

Wartości przekazywane metodą POST:
XSS, Injection Bugs, RFI/LFI ...

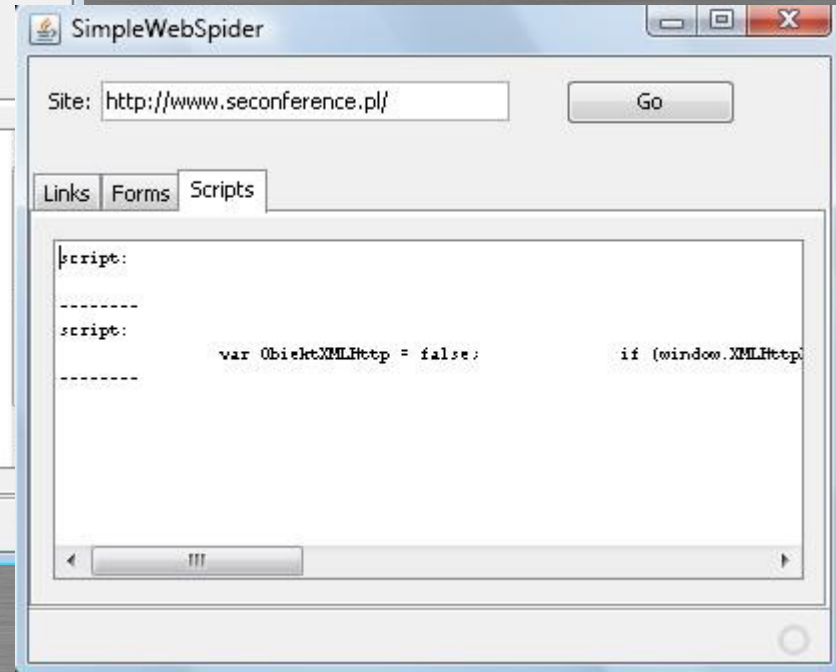
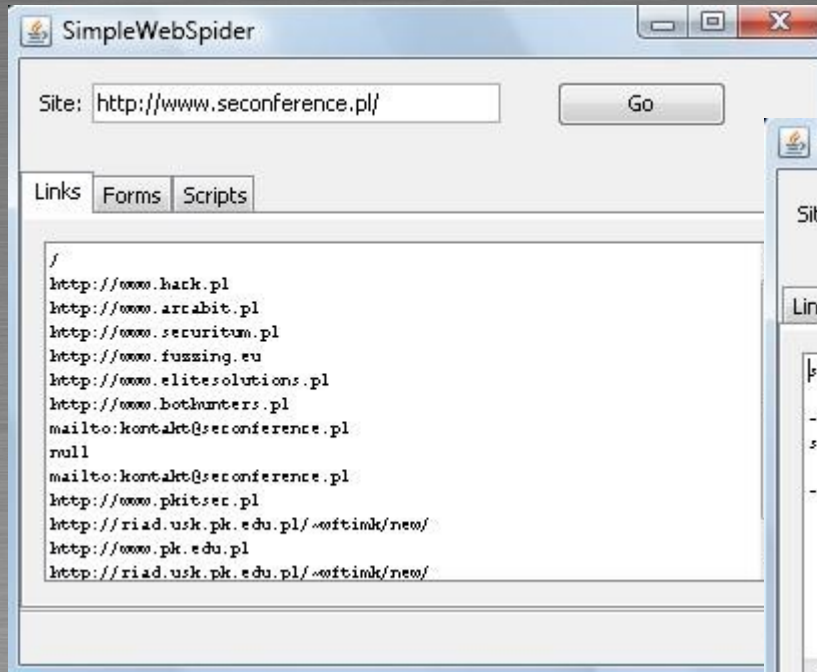


WebSpidering

- Odnośniki wraz z parametrami
- Formularze (ukryte pola)
- Skrypty (ActiveXObject, XMLHttpRequest...)
- Komentarze (odnośniki, skrypty)



WebSpidering



- Ogólnodostępne narzędzia
- Autorskie rozwiązania



Google

- Wyszukiwarki internetowe (w szczególności Google) mogą dostarczyć wiele cennych informacji. Dają nam dostęp do konkretnych podstron aplikacji wraz z listą przyjmowanych parametrów.
- Jesteśmy ograniczeni tylko do parametrów przekazywanych metodą GET.



Generowanie i przekazywanie danych



Rodzaje danych testowych



Dane zapisane w programie

- Opierają się głównie na popularnych payload'ach
- Program zostanie przetestowany tylko i wyłącznie pod kątem wcześniej zdefiniowanych danych
- Zupełny brak losowości
- Różnicowanie wykrytych podatności zależy tylko i wyłącznie od różnicowania danych testowych
- Stworzenie zbioru danych wejściowych jest czasochłonne



Dane zapisane w programie – WebFuzz i XSS

- `<script>alert('XSS')</script>`
- `<script>alert("XSS")</script>`
- `<script>alert("XSS");</script>`
- `<script language="javascript" type="text/javascript">alert("XSS");</script>`
- `test@<script>alert("XSS")</script>.com`
- `1"style="background:url(javascript:alert('XSS'))"%20"`
- `>"><script>alert('XSS')</script>`
- `>"><script>alert("XSS")</script>`
- `</textarea><script>alert('XSS')</script>`
- `>"><img%20src="javascript:alert('XSS')">`
- `--><script>alert('XSS')</script>`
- `'+alert('XSS')+'`
- `"+alert('XSS')+"`
- `<IMG SRC="javasc
ript:alert('XSSx');">`
- `%3Cimg+src%3D%27http%3A%2F%2Flocalhost%2Fblah%27%3E`



Dane zapisane w programie – WebFuzz i SQL Injection

- '
- ' or 1=1
- sutton' or '1' = '1
- 'sutton' or 'x'='x'
- ' or like '%
- '%20OR



Atak siłowy – bruteforce

- Testowane są wszystkie kombinacje wartości z podanego przedziału.
- Jest to metoda bardzo nieefektywna, ale przydatna w niektórych sytuacjach:
 - Wyszukiwanie ukrytej funkcjonalności
 - Wyszukiwania plików i katalogów
 - Ominięcie procesu autoryzacji (przykład)
- Nie nadaje się do testowania pod kątem konkretnych podatności takich jak np. XSS



Atak siłowy - bruteforce

www.example.com/test.php?id=[BF]

id=a
id=b
id=c
...
id=admin

www.example.com/test.php?[BF]=[BF]

a=a
a=b
a=c
...
admin=1

www.example.com/[BF].php

a
b
...
add

www.example.com/[BF]/

a
...
db
admin



Automatyczne generowanie danych

- Generowanie danych testowych na podstawie ustalonych wzorców.
- Generowane dane dopasowywane są do konkretnej podatności – zwiększa to efektywność fuzzingu.
- Możliwość znajdowania nietypowych błędów (zaleta metody bruteforce) połączona z szybkością działania (zaleta predefiniowanych wzorców).
- Problemem może być opracowanie odpowiedniej metody generacji danych.



Automatyczne generowanie danych - przykład

Przykładem może być generowanie ciągów znaków, które pozwalają na wykrycie błędu XSS występującego w testowanej aplikacji. Strukturę ciągu testowego możemy podzielić na logiczne bloki i przyporządkować im odpowiednie zbiory wartości.

<START><MAIN><STOP>

START: "> ; ">< ; <; , </script>>, ">...



Automatyczne generowanie danych - przykład

```
Administrator: C:\Windows\system32\cmd.exe
<<script>alert('test')<</script>
<<script>alert('test')</script>>
<<script>alert('test')">
<<script>alert("test") <</script>
<<script>alert("test") </script>>
<<script>alert("test") ">
>javascript:alert('XSS')<</script>
>javascript:alert('XSS')</script>>
>javascript:alert('XSS')">
"><script>alert('test')<</script>
"><script>alert('test')</script>>
"><script>alert('test')">
"><script>alert("test") <</script>
"><script>alert("test") </script>>
"><script>alert("test") ">
">javascript:alert('XSS')<</script>
">javascript:alert('XSS')</script>>
">javascript:alert('XSS')">
<<script>alert('test')<</script>
<<script>alert('test')</script>>
<<script>alert('test')">
<<script>alert("test") <</script>
<<script>alert("test") </script>>
<<script>alert("test") ">
<javascript:alert('XSS')<</script>
<javascript:alert('XSS')</script>>
<javascript:alert('XSS')">
<img src=""><script>alert('test')<</script>
<img src=""><script>alert('test')</script>>
```



Automatyczne generowanie danych - strategie

- Określenie odpowiedniego wzorca, do którego będą dopasowywane dane.
- Określenie zbioru/zbiorów wartości elementarnych.
- Zastosowanie odpowiedniej metody generowania zbioru wynikowego – kombinacje, permutacje...
- Wykorzystanie pierwiastka pseudolosowości.



Identyfikacja błędu



Identyfikacja błędu

- W przypadku aplikacji webowych jest to najważniejsza część procesu fuzzingu.
- Często aplikacje webowe nie dają wyraźnych znaków świadczących o obecności błędu.
- Jeżeli nie mamy dostępu do serwera, jedynym źródłem informacji o zaistniałym błędzie jest skromna odpowiedź aplikacji na nasze zapytanie.
- Powszechnie uważa się, że fuzzing aplikacji webowych pozwala na znalezienie podatności, których występowanie jest wyraźnie dostrzegalne – **czy aby na pewno jest to prawda?**



Jak identyfikować błędy – metody tradycyjne

- Kody odpowiedzi HTTP:
 - 200 – OK
 - 401 – Nieautoryzowany dostęp
 - 403 – Dostęp zabroniony
 - 404 – Nie znaleziono
 - 500 – Wewnętrzny błąd serwera
- Dane zawarte w kodzie zwracanej strony:
 - Błędy SQL
 - Kod JS
 - Specyficzne dane (błędy LFI/RFI)
- Jeżeli mamy dostęp do serwera: logi systemowe, monitorowanie działania aplikacji



Przykład - Powerfuzzer

```
if data.find("You have an error in your SQL syntax")>=0:  
    err="MySQL Injection"  
if data.find("supplied argument is not a valid MySQL")>0:  
    err="MySQL Injection"  
if data.find("[Microsoft][ODBC Microsoft Access Driver]")>=0:  
    err="MSSQL Injection"  
if data.find("java.sql.SQLException: Syntax error or access violation")>=0:  
    err="SQL Injection"  
if data.find("XPathException")>=0:  
    err="XPath Injection"  
if data.find("supplied argument is not a valid ldap")>=0:  
    err="LDAP Injection"  
if data.find("javax.naming.NameNotFoundException")>=0:  
    err="LDAP Injection"
```



Trudne nie znaczy niemożliwe



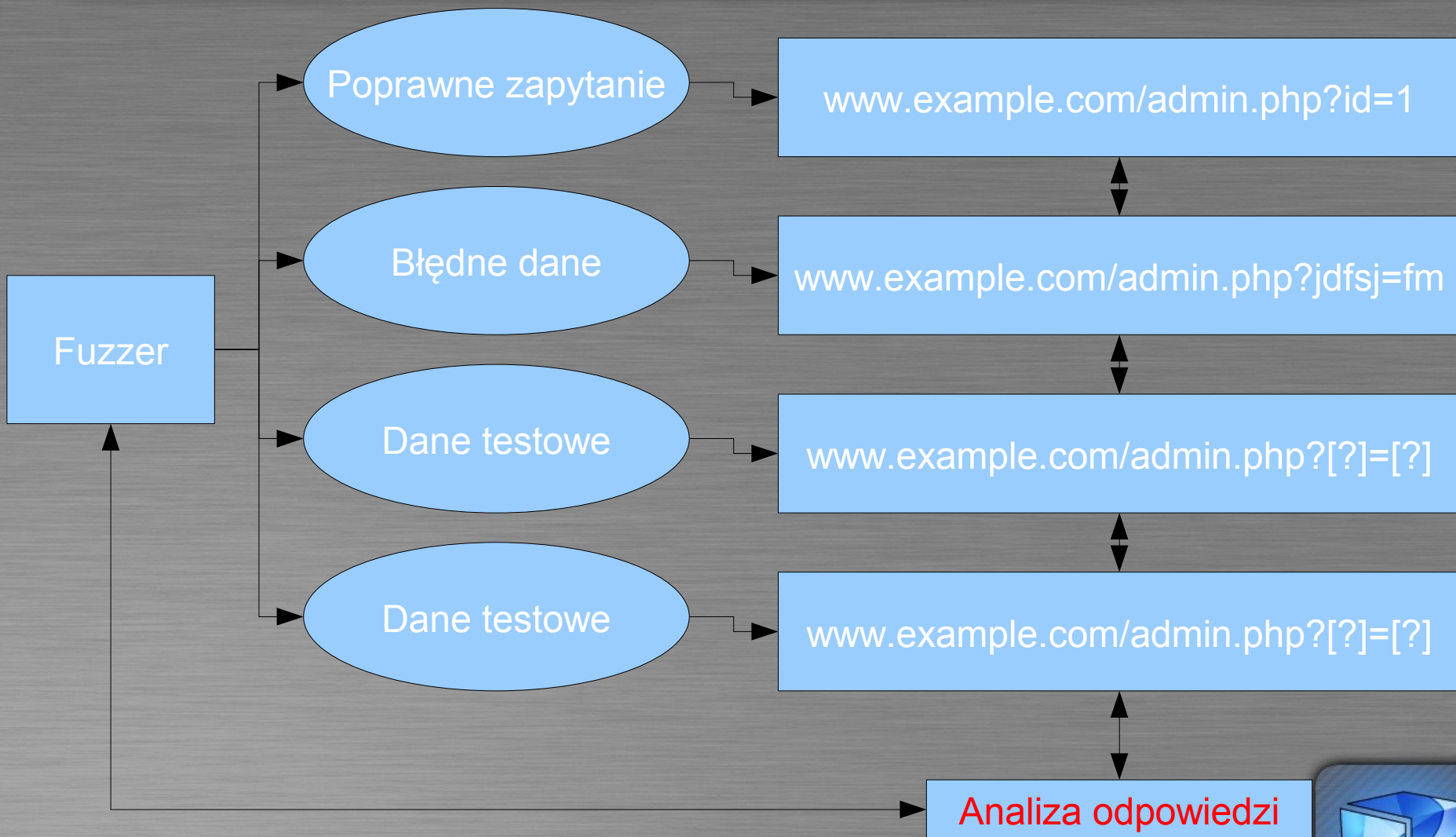
Identyfikacja błędu – metody zaawansowane

Aplikacje webowe cechują się dużą zmiennością jeżeli chodzi o zwracaną strukturę strony oraz treść. Nie możemy więc skorzystać ze statycznych wzorców, które umożliwiałyby dopasowanie do każdej testowanej aplikacji. Możemy za to wykorzystać:

- Wielokrotne zapytania i porównywanie zwróconych wyników
- **Porównywanie struktury strony a nie treści na niej zamieszczonej**
- Porównywanie danych jednoznacznie identyfikujących daną stronę – np. tag <title>
- Ataki czasowe



Wielokrotne zapytania

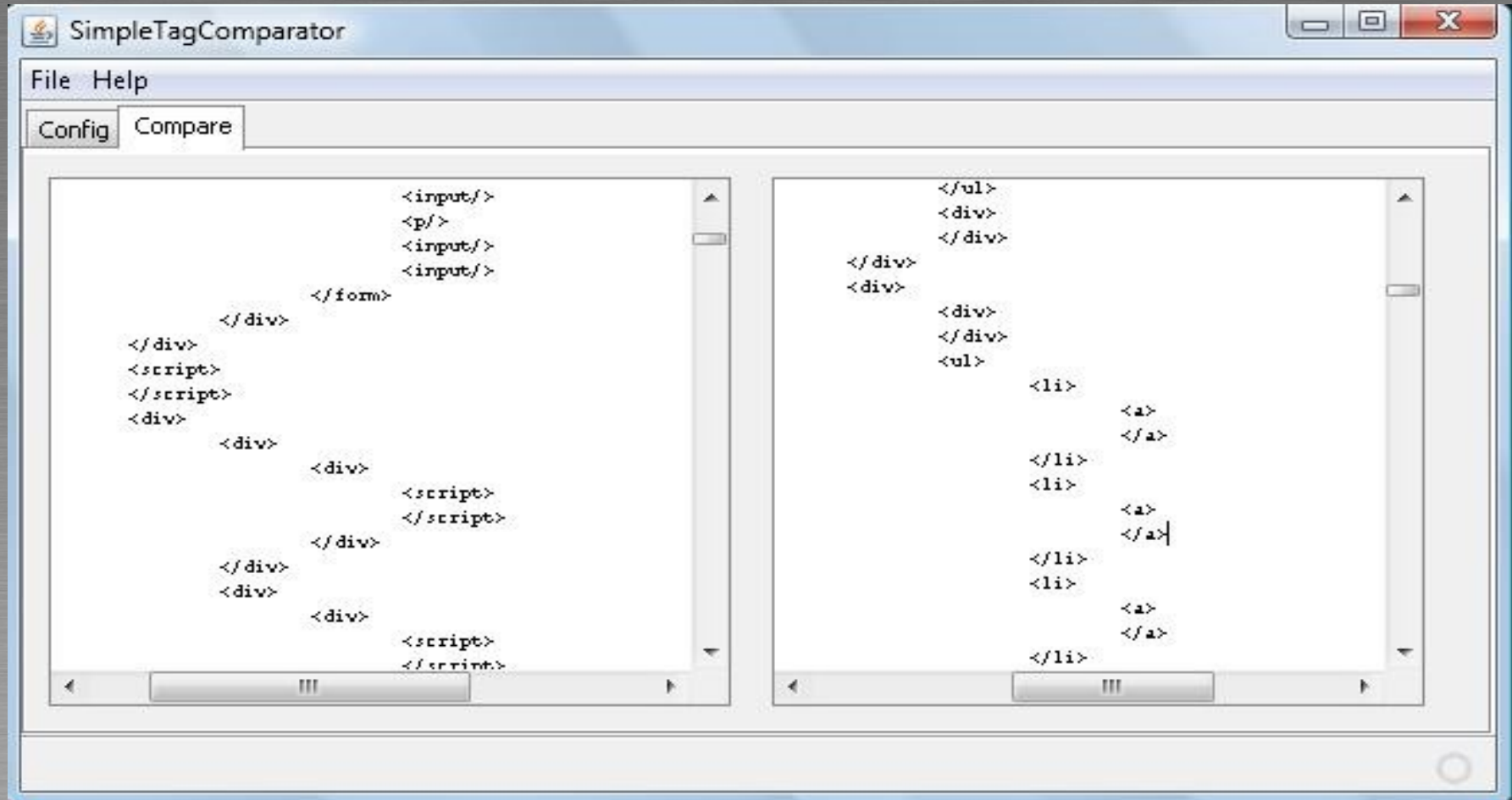


Porównywanie struktury strony

- Porównujemy odpowiednie rozmieszczenie kluczowych tagów (table, div...)
- Jeżeli różnica w budowie strony jest znaczna (czyli struktura strony wyraźnie różni się od próbki) to fuzzer sygnalizuje potencjalny błąd.
- Porównywanie struktury strony może być przydatne np. podczas poszukiwania błędu polegającego na ominięciu autoryzacji i uzyskaniu dostępu do panelu administracyjnego.

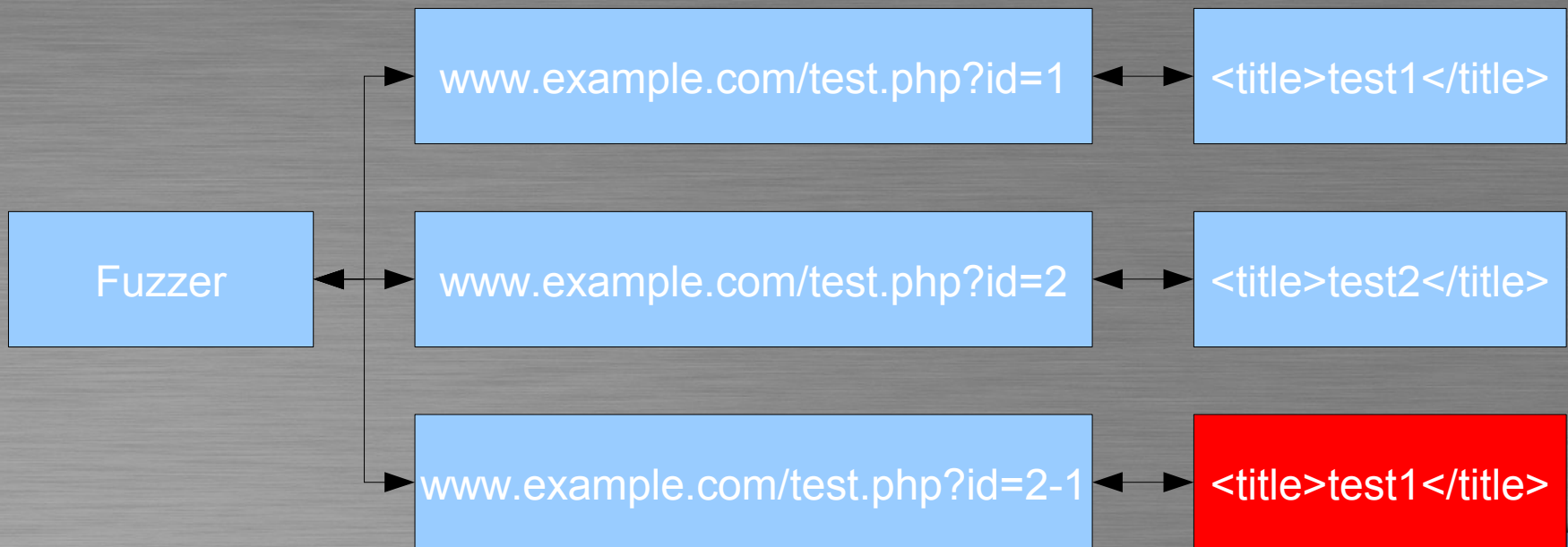


Porównywanie struktury strony



Analiza zwracanej treści

- Analizowanie zwracanych przez aplikację webową danych pod kątem występowania zmian pomiędzy kolejnymi zapytaniami.



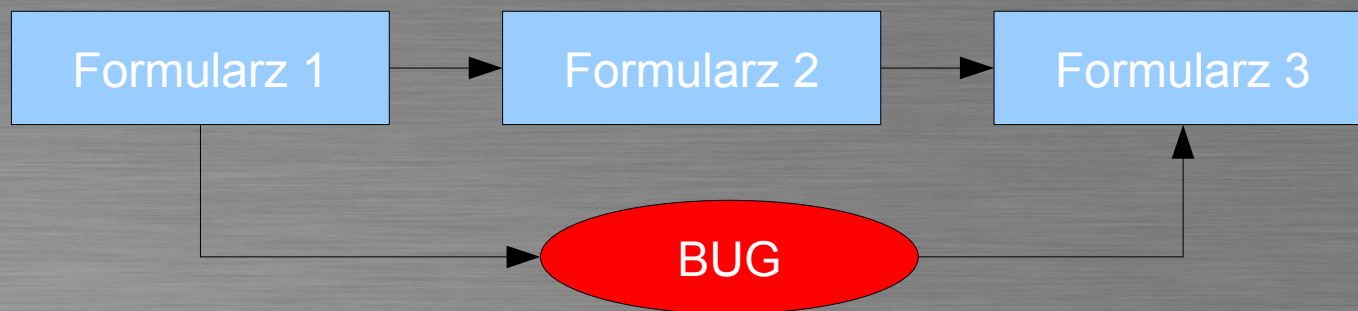
Ograniczenia

- Analizowanie struktury i treści strony jest czynnością czasochłonną.
- Wymaga dużej mocy obliczeniowej.
- Nie daje tak jednoznacznych i pewnych wyników jak testowanie oparte na poszukiwaniu konkretnych danych (błędy SQL, kod JS) zwracanych czy analizie kodów odpowiedzi HTTP.
- Jest trudne do zaimplementowania.
- **Na chwilę obecną jest to zbyt mało efektywna technika.**



Błędy logiczne i kombinowane

- Na chwilę obecną nie istnieją sposoby detekcji błędów logicznych.
- Nie ma także możliwości wykrycia skomplikowanych błędów kombinowanych:



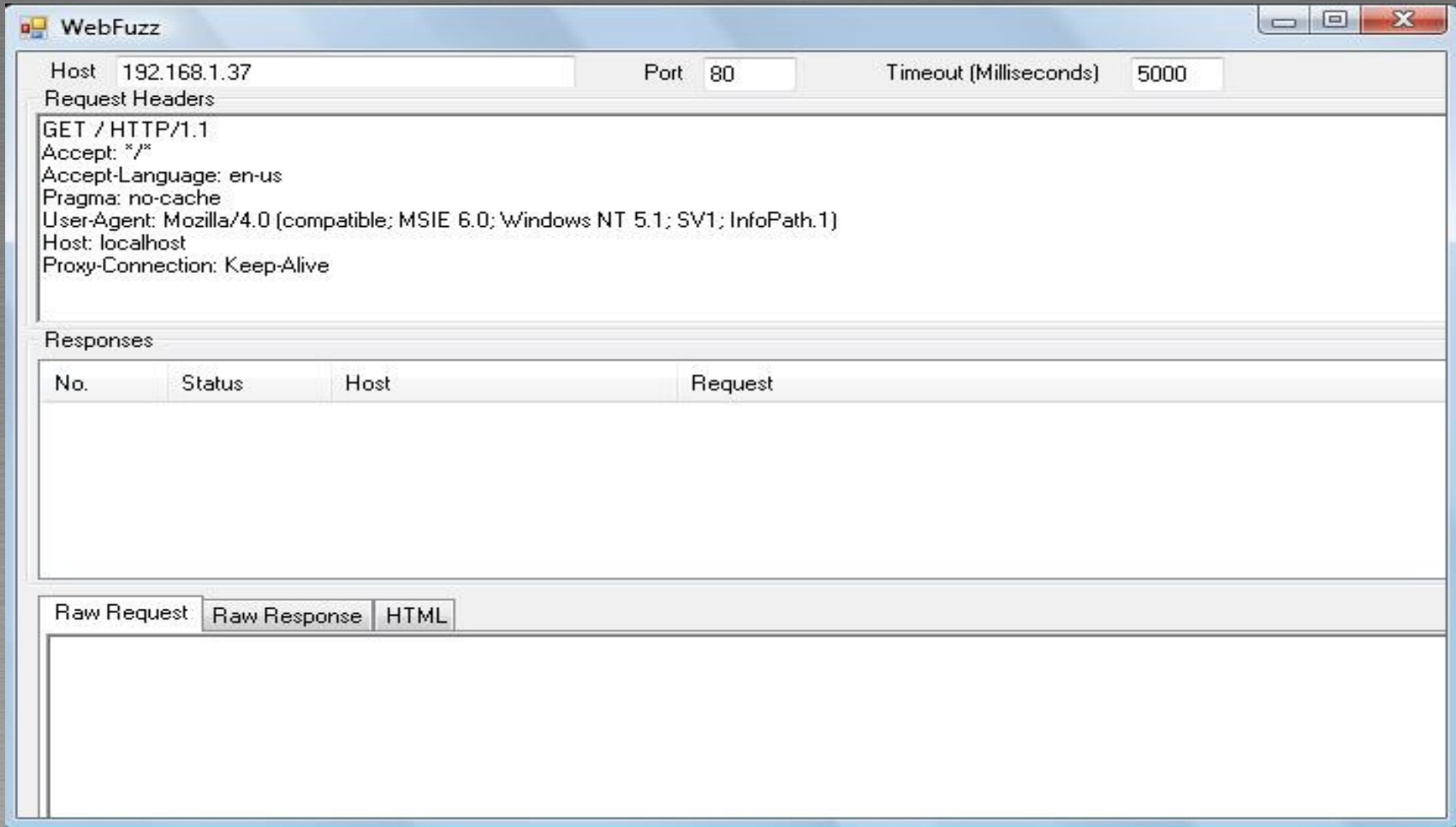
Wraz z rozwojem sztucznej inteligencji i wzrostem jej znaczenia w procesie fuzzingu wykrycie takich błędów prawdopodobnie stanie się możliwe...



Dostępne oprogramowanie



WebFuzz – prekursor Web Fuzzingu



The screenshot shows the WebFuzz application window. At the top, there are input fields for Host (192.168.1.37), Port (80), and Timeout (5000 milliseconds). Below these is the 'Request Headers' section, which contains the following text:

```
GET /HTTP/1.1
Accept: */*
Accept-Language: en-us
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; InfoPath.1)
Host: localhost
Proxy-Connection: Keep-Alive
```

Below the request headers is a 'Responses' section containing a table with the following columns: No., Status, Host, and Request. The table is currently empty.

At the bottom of the window, there are three tabs: 'Raw Request', 'Raw Response', and 'HTML'. The 'Raw Request' tab is currently selected, and the area below it is empty.



Powerfuzzer

Powerfuzzer v1 BETA

File

```
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd%00
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=..%2F..%2F..%2F..%2F..%2F..%2F..%2Fboot.ini
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=..%2F..%2F..%2F..%2F..%2F..%2F..%2Fboot.ini%00
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=a%3Benv
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=a%29%3Benv
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=%2Fe%00
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=%BF%27%22%28
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=<script>var+pf_687474703a2f2f6c6f63616c686f737473a383038342f54657374536974652f7365636f6e6666572656e63652f446174612e616374696f6e3b6a73657373696f6e69643d4235353231464131443230343242394239463830314635443634304134383143_6964=new+Boolean();</script>
+ http://localhost:8084/TestSite/seconference/Data.action;jsessionid=B5521FA1D2042B9B9F801F5D640A481C?id=http%3A%2F%2Fwww.google.com%0D%0APowerfuzzer%3A+v1+BETA
Attacking urls (GET)...
```

Credentials

User

Password

Proxy

Timeout

Verbosity Low Medium High

Target URL

Exclude URL(s) or dir

Done.



Powerfuzzer online

Home

Powerfuzzer Online v1

"Your automated low hanging fruit vulnerability scanner that will not cost you a fortune to use™"

Home

What is it ?

Terms of Use

Privacy Policy

Become a partner

Become a reseller

Contact Us

Sample scan



Scanner



Check what bad guys already know about your website security. Minimize risk of your website being compromised.

Scanner submission

Name

E-Mail

URL

Comments

Referrer code (optional)

Authorization

I am authorized to

Top 10 Vulnerabilities

- A1 - Cross Site Scripting (XSS)
- A2 - Injection Flaws
- A3 - Malicious File Execution
- A4 - Insecure Direct Object Reference
- A5 - Cross Site Request Forgery (CSRF)
- A6 - Information Leakage and Improper Error Handling
- A7 - Broken Authentication and Session Management
- A8 - Insecure Cryptographic Storage
- A9 - Insecure Communications
- A10 - Failure to Restrict URL Access



JBroFuzz

The screenshot shows the JBroFuzz application window titled "JBroFuzz - Untitled". The interface includes a menu bar (File, Edit, Format, View, Panel, Options, Help) and a toolbar with icons for play, stop, add, help, and refresh. The main area is divided into several sections:

- URL:** A text box containing `https://www.owasp.org`.
- Request:** A text area displaying the following HTTP request:

```
GET /index.php/Main_Page HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.8.1.1) Gecko/200
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
Accept-Language: en-gb,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```
- Payloads:** A section with a tab labeled "On The Wire (0)" and a sub-section "Added Payloads Table" containing an empty table with columns "Category", "Start", and "End". To the right of the table are a red plus button and a minus button.
- Output:** A table with the following headers: "No", "Target", "Timestamp", "Status", "Response Time", and "Response Size". The table body is currently empty.
- Bottom Panel:** A row of tabs labeled "Fuzzing", "Graphing", "Payloads", "Headers", and "System (0)".



Pozostałe rozwiązania

- MielieTools - <http://packetstormsecurity.org/UNIX/utilities/mielietools-v1.0.tgz>
- Spike Proxy - <http://www.immunitysec.com/downloads/SP148.tgz>
- Wfuzz - <http://www.edge-security.com/wfuzz.php>

• Venom Fuzzer – inteligentny fuzzer aplikacji webowych. Oparty na wirtualnej maszynie. Zdolny do monitorowania wielu części składowych aplikacji webowej w celu detekcji błędów.

Strona projektu: <http://code.google.com/p/venom-fuzzer/>

- Proxy: Burp, WebScarab, Spike



Tworzenie własnego fuzzera

Autorskie fuzzery przydają się jeżeli naszym celem jest nietypowa aplikacja o znanej strukturze. Ich tworzenie jest także uzasadnione w przypadku gdy nie podobają nam się rozwiązania już istniejące...

- Język – dowolny. Polecane: Python, Ruby, Java
- Pomocne narzędzia:
 - RFuzz – rozbudowany framework napisany w Ruby służący do budowy fuzzerów aplikacji webowych. Prosty a zarazem oferujący duży wachlarz możliwości.
 - Parsery – istnieje wiele ciekawych rozwiązań w zależności od wybranego języka programowania: Jericho (Java), Beautiful Soup (Python)...



Tworzenie własnego fuzzera

Blok odpowiadający za nawiązywanie połączeń, odbieranie oraz kodowanie danych

Blok odpowiedzialny za generowanie danych.

Fuzzer

Blok odpowiedzialny za identyfikowanie punktów wejściowych.

Blok odpowiedzialny za wykrywanie błędów.

Blok odpowiedzialny za sporządzanie raportu.



Web Service fuzzing

- Blackhat 2007 – Michael Sutton
- WSDL (Web Service Description Language)
 - ogromny zbiór informacji
- WSFuzzer -

http://www.owasp.org/index.php/Category:OWASP_WSFuzzer_Project



Podsumowanie



Zalety fuzzingu

- Pełna automatyzacja procesu testowania.
- Umożliwia szybkie znalezienie popularnych błędów.
- Umożliwia znalezienie błędów powiązanych z nietypowymi danymi wejściowymi.
- Duży wybór gotowych narzędzi.
- Prostota (w większości przypadków)



Wady fuzzingu

- Brak możliwości wykrycia błędu logicznego.
- Brak możliwości wykrycia złożonych podatności.
- Brak możliwości określenia dokładnego czasu potrzebnego na wykonanie testów w przypadku automatycznej generacji danych.



Przyszłość fuzzingu

- Rozwój zunifikowanych środowisk testowych.
- Wzrost znaczenia fuzzingu ewolucyjnego.
- Analiza składniowa (Gfuzz)
- Fuzzery online



Prezentacja



Na koniec...

- Należy pamiętać, że fuzzing nie jest tylko i wyłącznie metodą stosowaną w celu wykrycia luk w bezpieczeństwie. Fuzzing może być integralną częścią procesu tworzenia oprogramowania. Testowanie software'u za pomocą fuzzerów może wyeliminować wiele błędów trudnych do wychwycenia przy użyciu innych form testowania.
- Fuzzing nie jest rodzajem błędu ani metodą ataku. Jest to metoda testowania, której celem jest wykrycie potencjalnych błędów.



Informacje dodatkowe

- W sieci:
 - fuzzing.eu
 - fuzzing.org
 - krakowlabs.com/lof.html
- Literatura
 - **Fuzzing: Brute Force Vulnerability Discovery**
 - Fuzzing for Software Security Testing and Quality Assurance
 - Open Source Fuzzing Tools
- Prezentacje:
 - Smashing Web Apps (M.Sutton): blackhat.com



Dziękuję za uwagę



Pytania

