

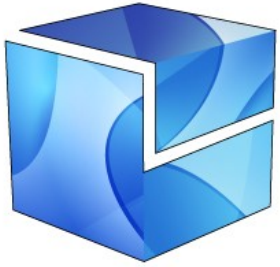


elite  
SOLUTIONS



# Fuzzing z wykorzystaniem języka Python

Piotr Łaskawiec  
plaskawiec@elitesolutions.pl



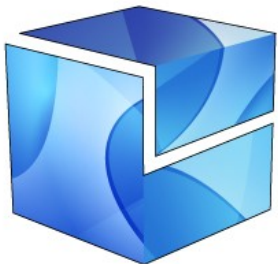
# Kim jestem...



HACK.pl



# ELITE SOLUTIONS

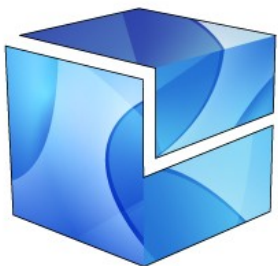


# Czym jest fuzzing?

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.





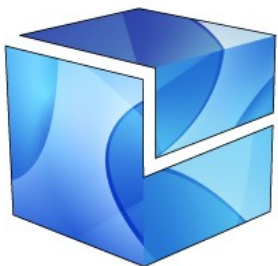
# Czym NIE jest fuzzing?

Należy rozgraniczyć standardowe metody testowania oprogramowania od testowania za pomocą danych pseudolosowych. Należy pamiętać, że fuzzing NIE ma nic wspólnego z testami:

- jednostkowymi
- funkcjonalnymi
- integracyjnymi
- wydajnościowymi

**FuzzTesting jest zupełnie odrębną techniką testowania i opiera się na zupełnie innych założeniach.**



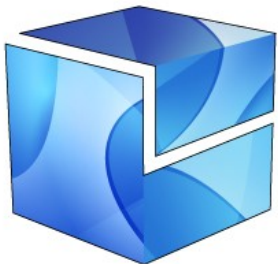


# Fuzzing - zastosowanie

Działaniu fuzzera możemy poddawać:

- Aplikacje działające lokalnie
- Aplikacje sieciowe
- Aplikacje internetowe
- Kontrolki ActiveX
- Biblioteki współdzielone
- Pliki
- Reszta celów ograniczona jest wyłącznie Waszą wyobraźnią

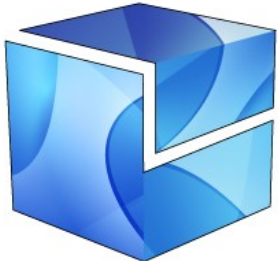




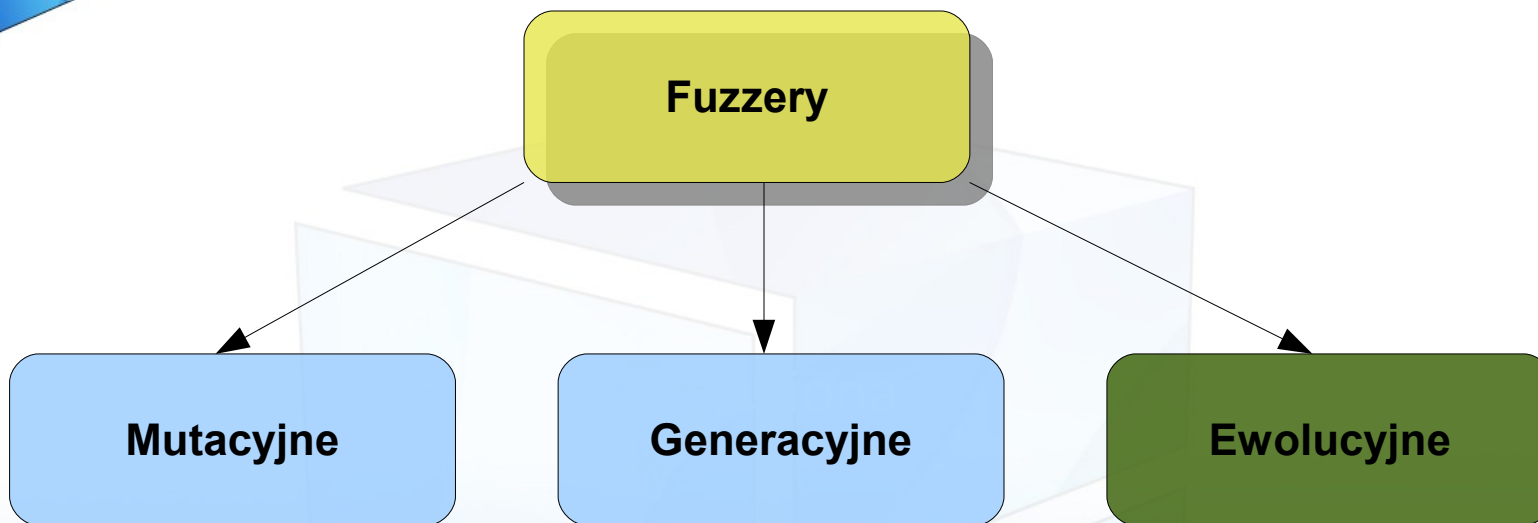
# Fuzzing – kilka faktów

- Fuzzing jest obecnie jedną z najczęściej stosowanych metod wykrywania luk w oprogramowaniu.
- Jest szeroko stosowany przez duże koncerny w procesie testowania aplikacji.
- Podstawą fuzzingu jest zasada, która mówi, że łamanie ogólnie przyjętych standardów może być przydatne. Aby przetestować program należy korzystać z niego w niestandardowy sposób.
- Python jest coraz częściej wykorzystywany do pisania fuzzerów i oprogramowania związanego z IT Security.





# Rodzaje fuzzerów

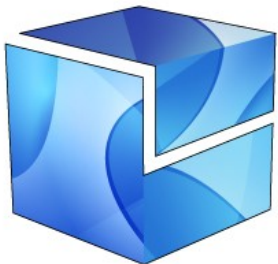


**Mutacyjne** – opierają się głównie na losowym modyfikowaniu dostarczonych próbek (np. plików danego formatu).

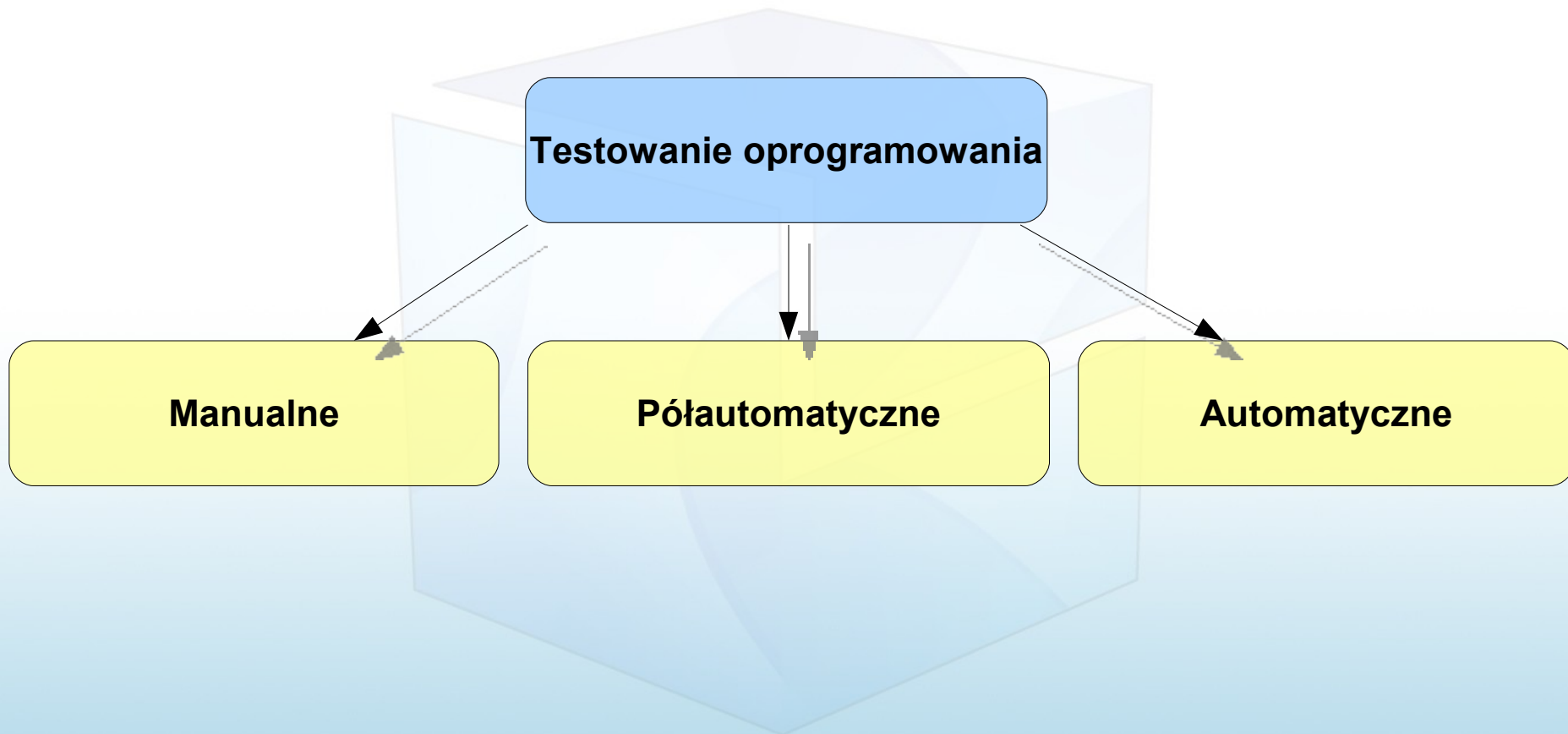
**Generacyjne** – bazują na dostosowywaniu się do konkretnych wymogów testowanych programów (do protokołu, formatu pliku itd.) i dzięki ich znajomości przynoszą lepsze rezultaty niż fuzzery mutacyjne. Wadą fuzzerów generacyjnych jest ich skomplikowanie (równie skomplikowaniu celu) oraz dłuższy czas potrzebny na ich stworzenie.

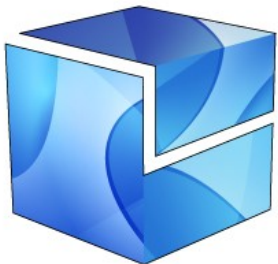
**Ewolucyjne** – aktualnie istnieją tylko założenia fuzzerów ewolucyjnych. Jedynym sensownym reprezentantem jest Evolving Fuzzer System (EFS). Fuzzery ewolucyjne są w stanie obserwować i uczyć się sposobu działania danego programu w celu lepszego dostosowania generowanych danych. Jest to przyszłość fuzzingu.





# Techniki testowania oprogramowania





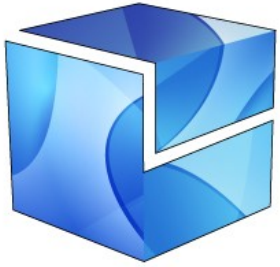
# Testowanie manualne

Testowanie manualne jest najprostszą formą fuzzingu, która jednak jest nadal szeroko stosowana.

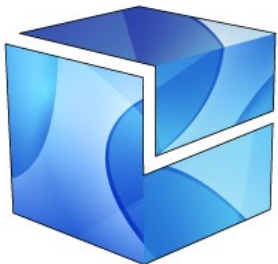
Payloady wprowadzane są ręcznie przez testera. Po wprowadzeniu spreparowanych danych obserwujemy reakcję programu. W przypadku *fuzzingu* manualnego oszczędność czasu w stosunku do analizy statycznej (analizy kodu źródłowego lub binarnego) nie jest wyraźnie widoczna.

**Fuzzing manualny nie jest dość ciekawy...**





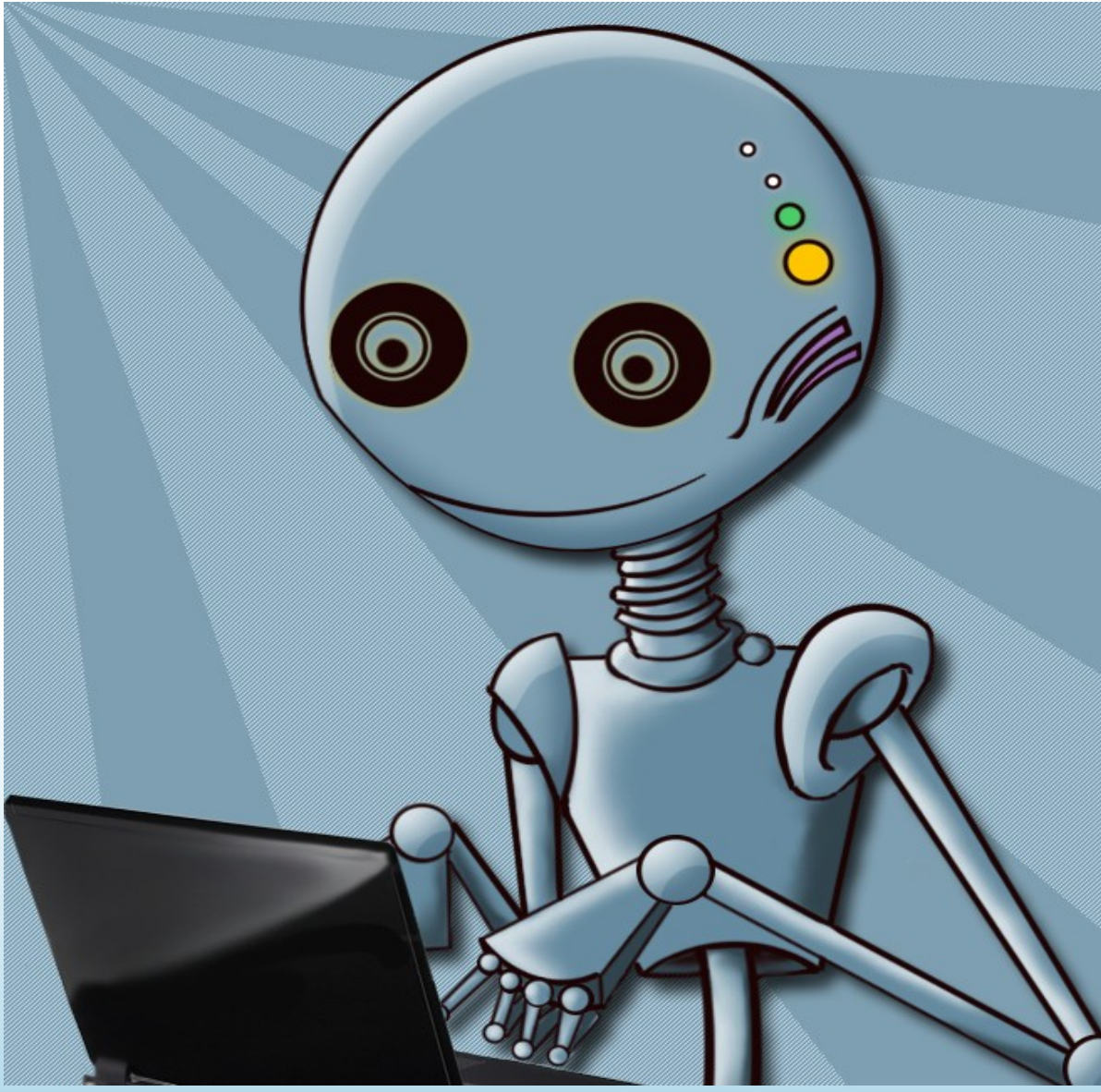
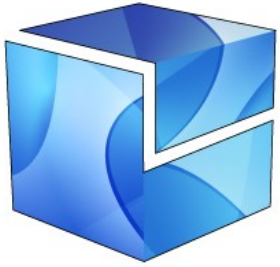
elite SOLUTIONS



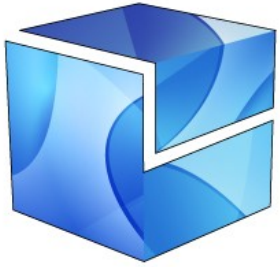
# Testowanie automatyczne

- Testowanie automatyczne jest najbardziej zaawansowana formą fuzzingu, w której przekazanie danych do aplikacji, monitorowanie testowanego programu, jak i sporządzenie raportu jest w pełni zautomatyzowane.
- Zadaniem osoby przeprowadzającej test jest JEDNORAZOWE przygotowanie środowiska, danych testowych oraz narzędzi monitorujących. Po wykonaniu tych czynności możemy spokojnie oczekiwać na efekty...

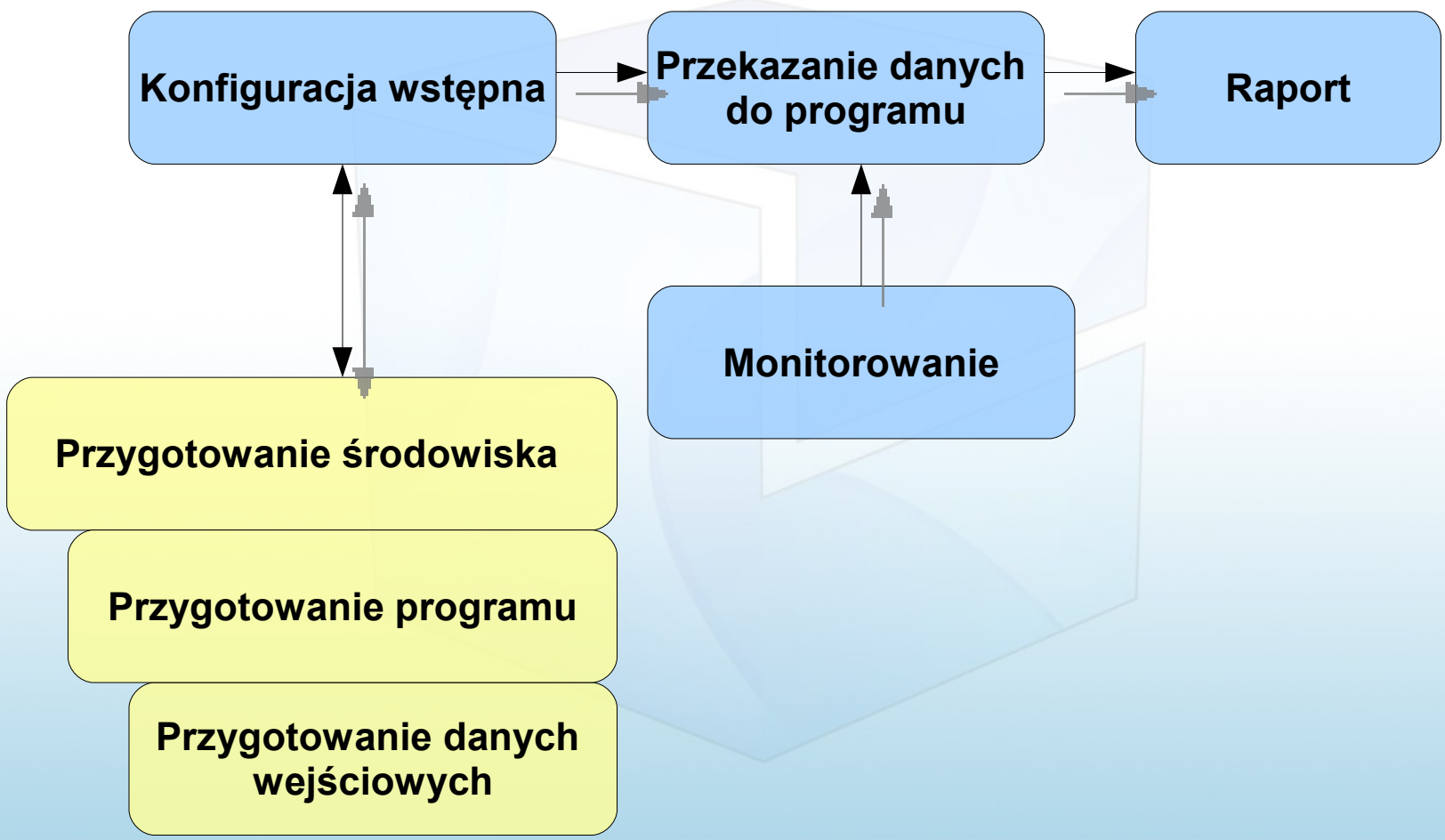


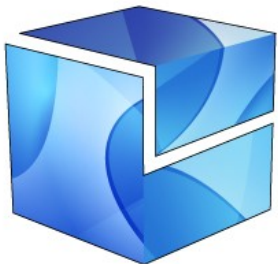


elite SOLUTIONS



# Schemat procesu testowania automatycznego

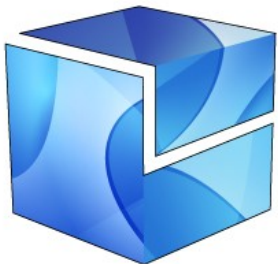




# Popularne fuzzery napisane w Pythonie

- Taof
- Wapiti
- untidy
- ProxyFuzz
- Powerfuzzer
- Revvv fuzzer
- FileP

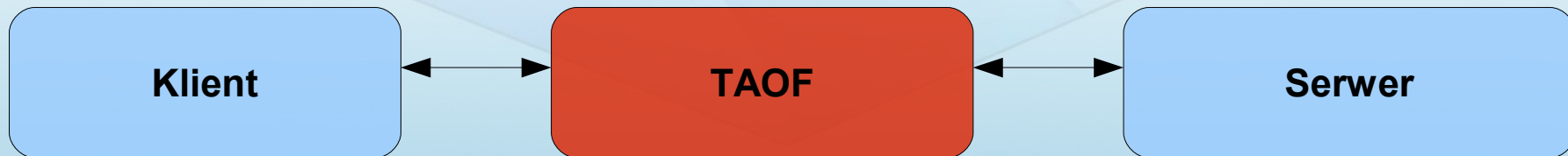




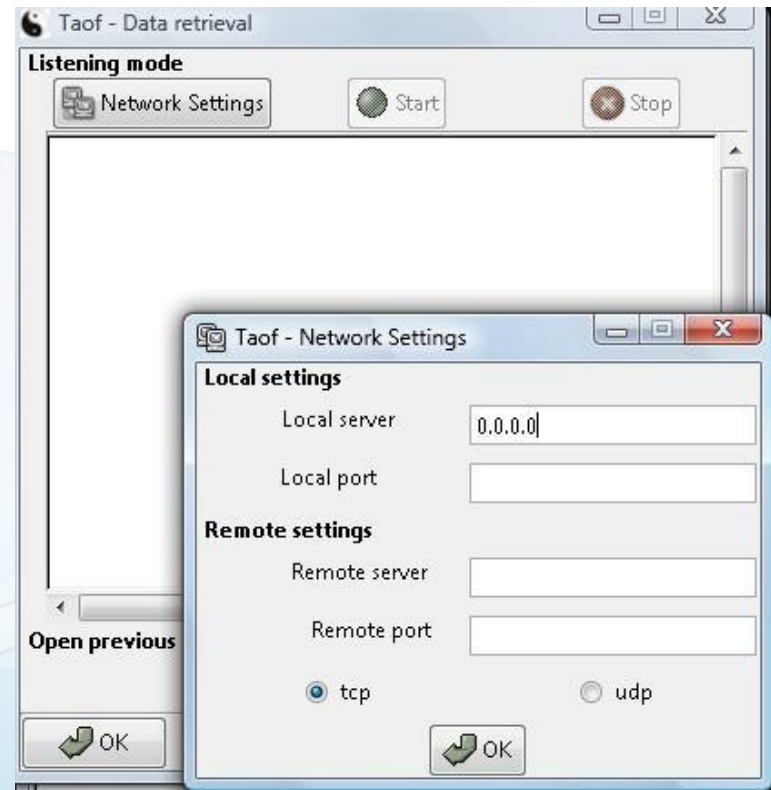
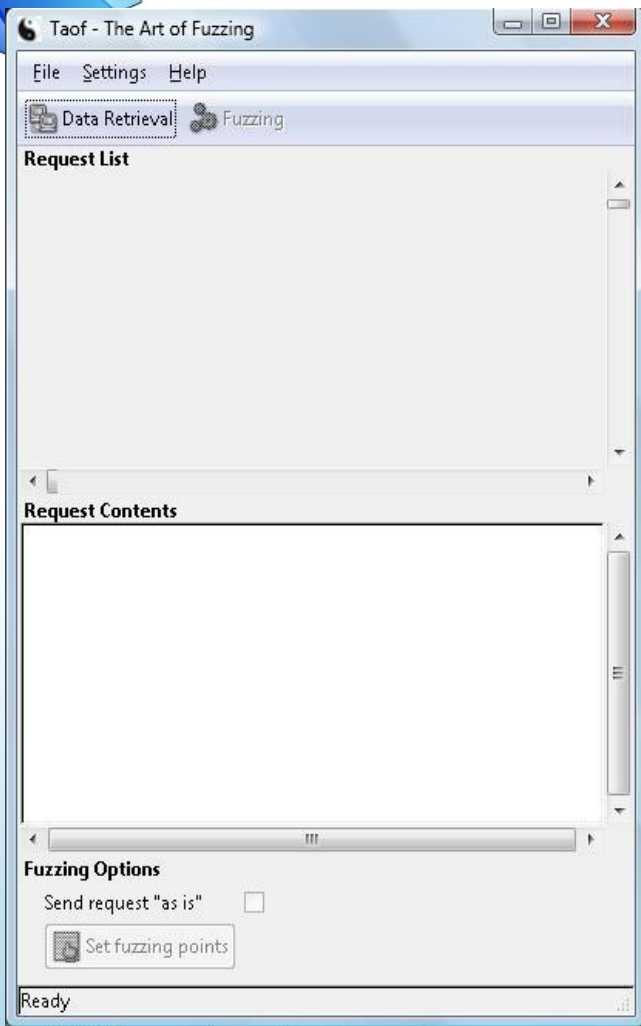
# Taof

- Fuzzer protokołów sieciowych
- Cechuje się dużą prostotą i przyjemnym GUI
- Szczególnie przydatny podczas testowania nieudokumentowanych protokołów.
- Idealny przykład potężnego fuzzera, który mimo swoich możliwości nie przytłacza użytkownika.

## Zasada działania



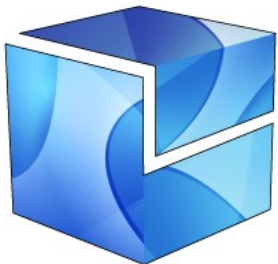
# Taof



## Znalezione błędy:

- WarFTPd
- Savant Web Server
- NaviCOPA
- ..... (miejsce na Twój błąd :))

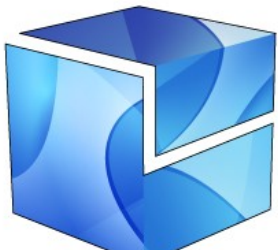




# Wapiti

- Skanuje aplikacje webowe w poszukiwaniu popularnych błędów.
- Potrafi wykrywać różne wariacje wstrzykiwania kodu, XSS, CRLF i LDAP Injection oraz tzw. „include bugs”.
- Obsługuje pliki cookie.
- Zawiera kilka różnych parserów kodu HTML.





# Wapiti

```
Administrator: CMD Shell
Wapiti-1.1.7-alpha - A web application vulnerability scanner
Usage: python wapiti.py http://server.com/base/url/ [options]

Supported options are:
-s <url>
--start <url>
    To specify an url to start with

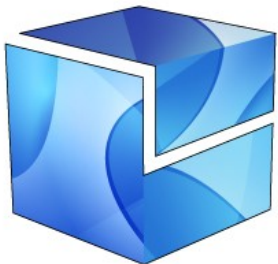
-x <url>
--exclude <url>
    To exclude an url from the scan (for example logout scripts)
    You can also use a wildcard (*)
    Exemple : -x "http://server/base/?page=*&module=test"
    or -x http://server/base/admin/* to exclude a directory

-p <url_proxy>
--proxy <url_proxy>
    To specify a proxy
    Exemple: -p http://proxy:port/

-c <cookie_file>
--cookie <cookie_file>
    To use a cookie
```

```
Administrator: CMD Shell
Wapiti-1.1.7-alpha <wapiti.sourceforge.net>
THIS IS AN ALPHA VERSION - PLEASE REPORT BUGS
*
Attacking urls <GET>...
-----
Attacking forms <POST>...
-----
Looking for permanent XSS
-----
C:\hack\wapiti-1.1.7-alpha>_
```



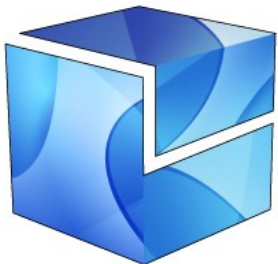


# untidy

- Fuzzer XML
- Untidy przekształca wejściowy ciąg znaków reprezentujących XML do potencjalnie niebezpiecznych modyfikacji.
- Charakteryzuje się bardzo prostą budową – idealnie nadaje się do analizy kodu.
- Do jego budowy wykorzystano jedynie podstawowe komponenty języka Python – cały kod programu zamyka się w ~400 liniach kodu.



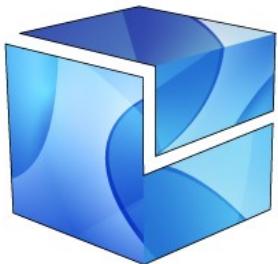




# ProxyFuzz

- Bardzo prosta budowa.
- Działanie na zasadzie man-in-the-middle.
- Niezależny od protokołu.
- Korzysta z Twisted.
- Jego praca polega na losowym zmienianiu danych przechodzących przez sieć, co prowadzi do wystąpienia nieoczekiwanych reakcji w pracy programów odbierających/wysyłających dane.



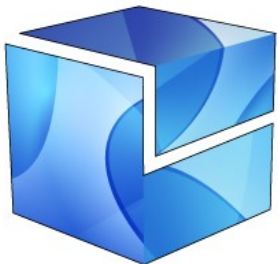


# Devvv

- Autorski projekt cechujący się dość unikalnym podejściem do tematu fuzzingu aplikacji webowych.
- Testowanie przy użyciu generowanych danych jak i payload'ów pobieranych ze specjalnego repozytorium.
- Zaawansowany system pobierania i analizowania odnośników umieszczonych na danej stronie oraz mechanizm web-crawlera.
- Obsługa plików cookie, autoryzacji, proxy.
- Devvv zostanie opublikowany w ciągu najbliższych 2 miesięcy.

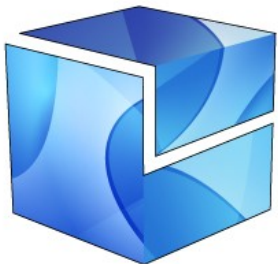
**Na chwilę obecną, za pomocą wczesnej wersji Devvv udało się znaleźć błędy na kilkunastu znaczących polskich stronach internetowych.**





# Własny fuzzer – planowanie

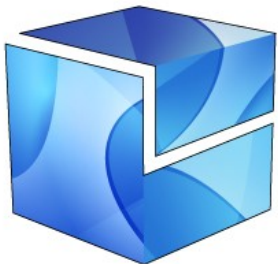




# Własny fuzzer - narzędzia

- Python
- Python :)
- Python :) :)
- pyCURL
- python-ptrace
- pcap
- Impacket
- BeautifulSoup
- Twisted
- PyDBG
- scrapy
- lswwww
- Urllib2
- HTMLParser/Tidy



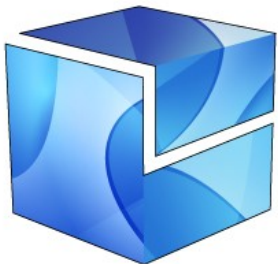


# Fuzzing Frameworks

Tworzenie unikalnego rozwiązania dla każdego problemu jest wyjściem najbardziej dokładnym, ale pochłania zbyt dużo zasobów czasowych a możliwość ponownego wykorzystania kodu jest znikoma.

Rozwiązaniem mogą być zaawansowane Frameworki, za pomocą których możemy zbudować fuzzer dostosowany do konkretnych potrzeb bez konieczności powielania setek linii kodu.



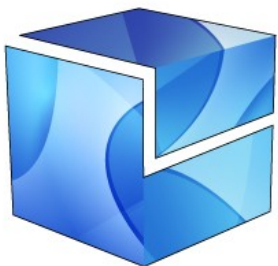


# Potrzebny nam bohater...

Dobry framework powinien posiadać kilka głównych cech:

- Powinien uprościć techniki reprezentacji danych oraz zapewnić możliwość modelowania dowolnych grup wartości.
- Powinien zapewnić metody transmisji danych oraz monitorowania stanu testowanej aplikacji.
- Powinien mieć możliwość wykrywania zaistniałych błędów oraz być w stanie odtworzyć stan aplikacji przed wystąpieniem błędu.
- Powinien być efektywny.
- Powinien być w stanie wskazać porcję danych powodującą błąd.



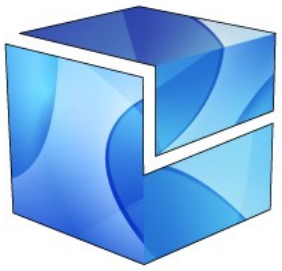


# Poznajcie Sulley'a

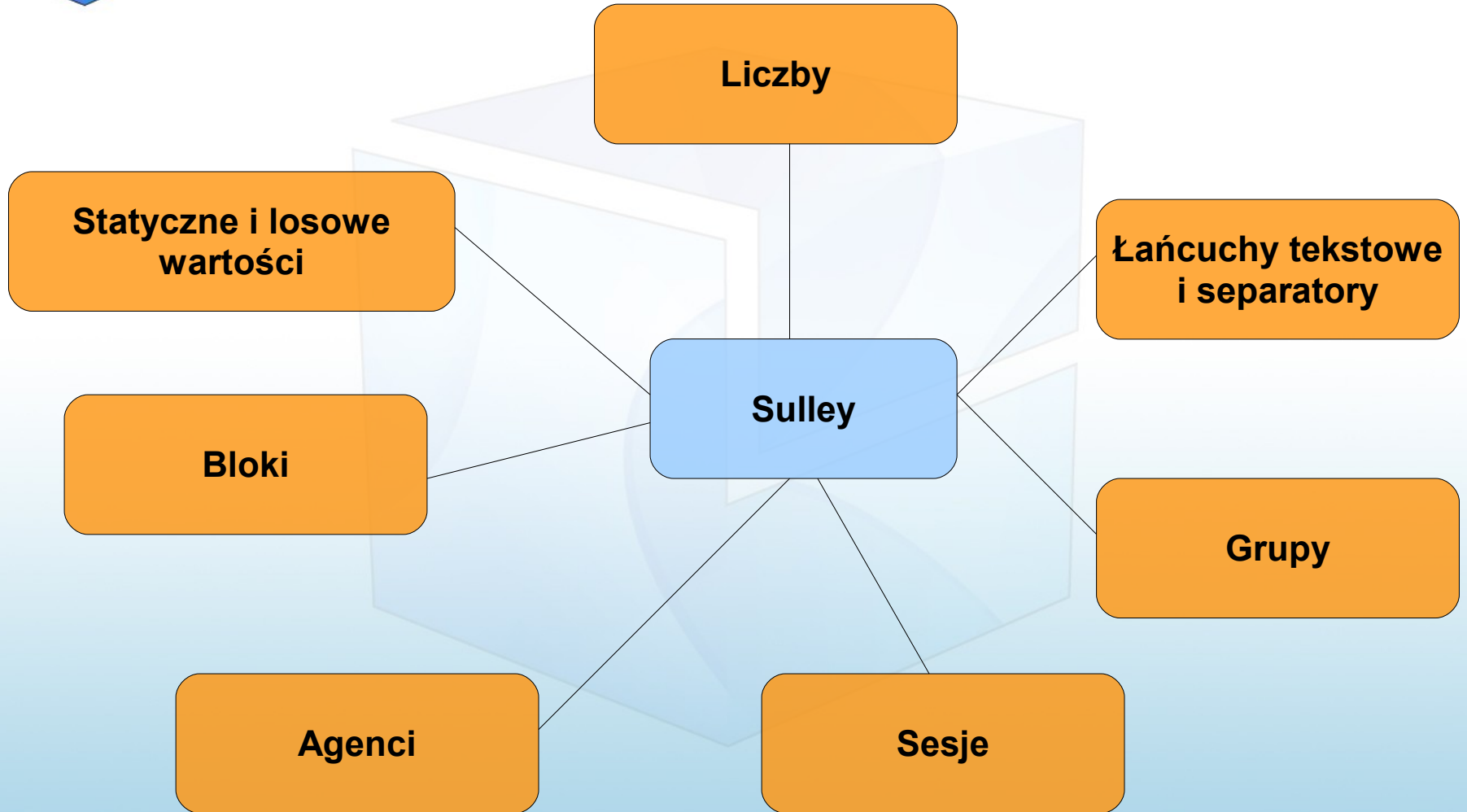


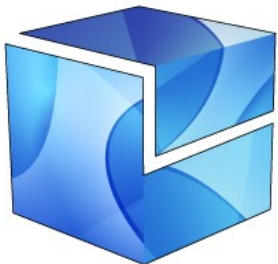
- Sulley spełnia wszystkie wymienione wcześniej wymogi.
- Jest kompletnym narzędziem do pisania nawet najbardziej zaawansowanych fuzzerów.
- Jest relatywnie prosty i co najważniejsze, jego budowa jest przejrzysta i przystępna.
- Mimo rzadkich uaktualnień jest dojrzałym projektem.
- Ograniczeniem może być współpraca z Pythonem wersji 2.4 i dość małe wsparcie dla fuzzingu plików o różnych formatach.



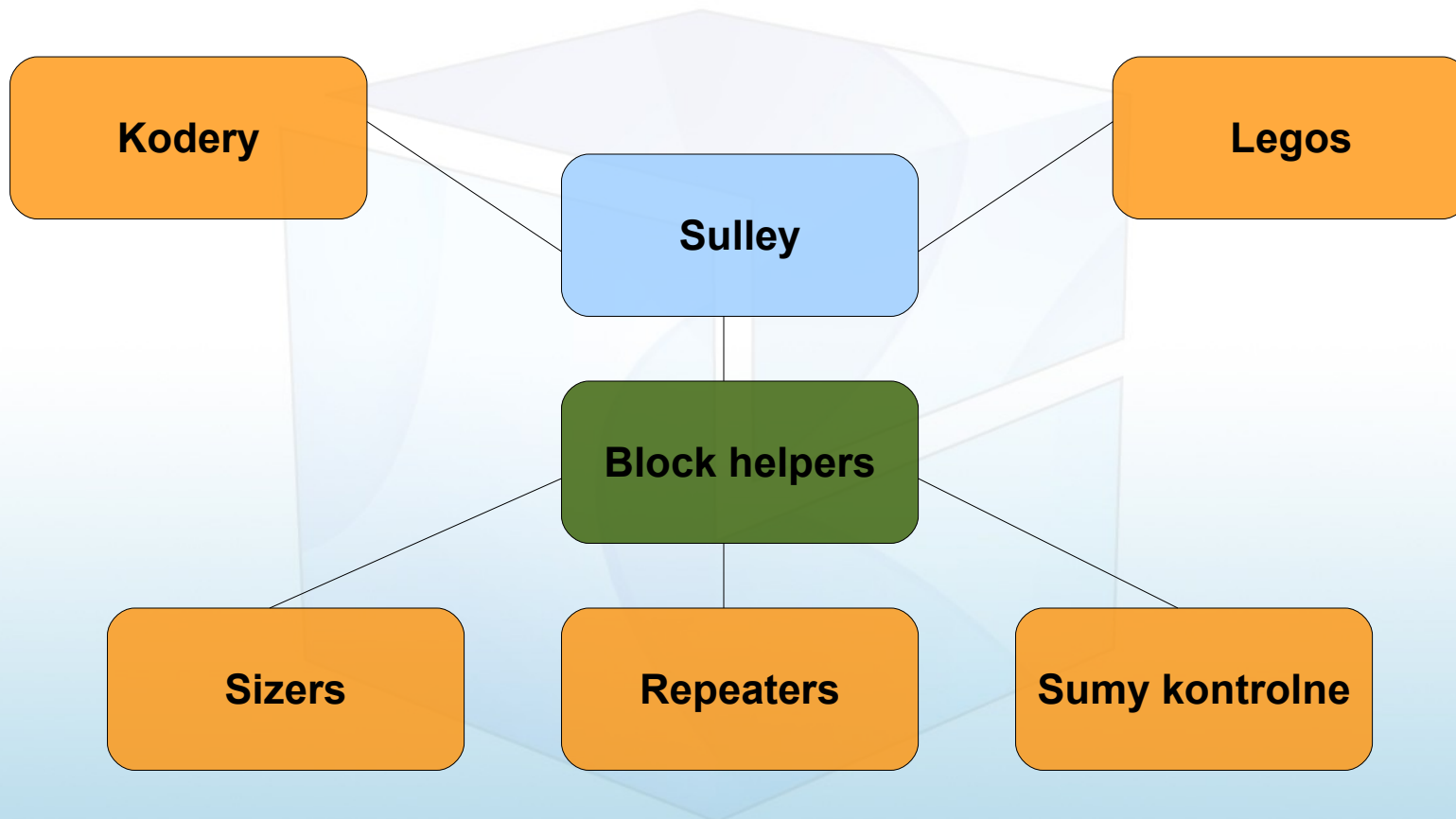


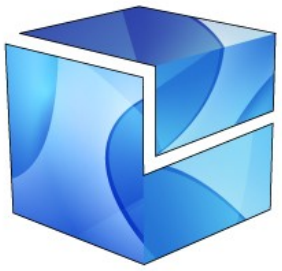
# Sulley – komponenty podstawowe





# Sulley – komponenty dodatkowe

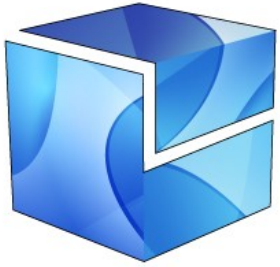




# Sulley – reprezentacja danych

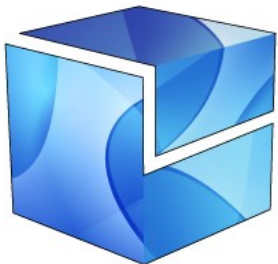
- W Sulley, poszczególne grupy danych wysyłane do testowanego programu budowane są z bloków danych oraz wartości podstawowych.
- Bloki danych mogą być wielokrotnie zagnieżdżone.
- Dane mogą być także modelowane przy wykorzystaniu zależności, które pozwalają na lepszą personalizację.
- Mamy dostęp do wielu predefiniowanych wartości podstawowych oraz generatorów odpowiedzialnych za tworzenie sum kontrolnych oraz określanie wielkości danych na podstawie wybranych elementów.





# Schemat procesu testowania automatycznego



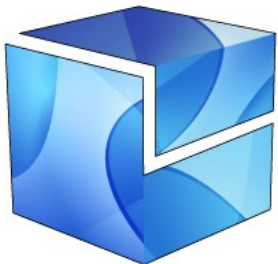


# Sulley – HTTP fuzzing (http-test.py)

```
from sulley import *
```

```
s_initialize("HTTP")  
s_group("methods", values=["GET", "HEAD", "POST", "TRACE", "PUT"])  
if s_block_start("main", group="methods"):  
    s_delim(" ")  
    s_delim("/")  
    s_string("index.htm")  
    s_delim(" ")  
    s_string("HTTP")  
    s_delim("/")  
    s_string("1")  
    s_delim(".")  
    s_string("1")  
    s_static("\r\n\r\n")  
s_block_end(,main)
```





# Sulley – custom protocol

Łańcuch tekstowy 1	Łańcuch tekstowy 2	Suma kontrolna łańcucha 1	Rozmiar łańcucha 2	Typ – 4 bajty
N bajtów	N bajtów	MD5	2 bajty	4 bajty

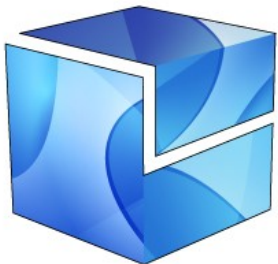
Sulley zapewnia łatwość modelowania dowolnych protokołów.

Założmy, że nasz testowy protokół złożony jest z dwóch łańcuchów tekstowych, sumy kontrolnej obliczanej na podstawie pierwszego łańcucha, rozmiaru drugiego łańcucha oraz typu, który nie został jednoznacznie określony.

Za pomocą Sulley'a postaramy się zbudować model reprezentujący powyższy protokół. Jest to czynność niezwykle prosta i intuicyjna.

**Popatrzmy na kod...**





# Sulley – Custom fuzzing (custom.py)

```
from sulley import *
```

```
s_initialize("CUSTOM")
```

```
if s_block_start("custom"):  
    if s_block_start("string1"):  
        s_string("A" *20)  
    s_block_end()
```

```
    if s_block_start("string2"):  
        s_string("A" *20)  
    s_block_end()
```

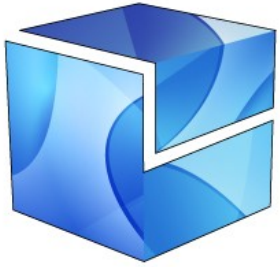
```
s_checksum("string1", algorithm="md5")
```

```
s_size("string2", length=2)
```

```
s_random("\x41\x41\x41\x41", 4, 4)
```

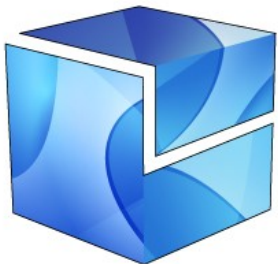
```
s_block_end(„custom”)
```





# Schemat procesu testowania automatycznego





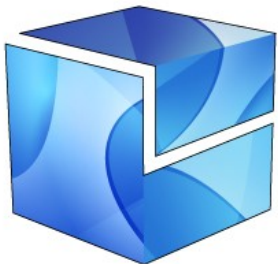
```
from sulley import *  
import custom  
#import http
```

```
def start():
```

```
    sesja = sessions.session(session_filename="audits/custom.session")  
  
    cel = sessions.target("192.168.0.1", 9999)                #host, port  
    cel.netmon = pedrpc.client("192.168.0.1", 26001)        #host,port=26001  
    cel.procmon = pedrpc.client("192.168.0.1", 26002)       #host,port=26002  
    cel.procmon_options = \  
    {  
        "proc_name" : "program.exe",  
        "stop_commands" : ['F:\\custom\\program.exe -stop'],  
        "start_commands" : ['F:\\custom\\program.exe -run'],  
    }  
    sesja.add_target(target)  
    sesja.connect(s_get("CUSTOM"))        #wykorzystujemy dane z custom.py  
    sesja.fuzz()
```

```
start()
```

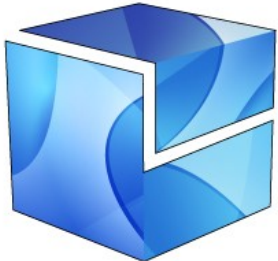




# Sulley - monitorowanie

- **network\_monitor.py** – monitoruje i zapisuje ruch sieciowy do plików w formacie PCAP. Network monitor pozwala na analizę danych, które spowodowały nieprzewidziane zachowanie programu.
- **process\_monitor.py** – monitoruje i archiwizuje działanie testowanej aplikacji i po każdym przesłaniu do niej losowych danych sprawdza czy nie wystąpiły błędy w jej działaniu.
- **vmcontrol.py** – jest to jeden z ciekawszych elementów Sulley, pozwalający na kontrolowanie wirtualnej maszyny (m.in. uruchomienie, zatrzymanie, tworzenie oraz przywracanie snapshotów).



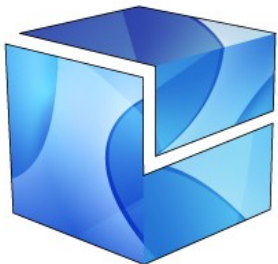


```
Administrator: CMD Shell
ERR> USAGE: process_monitor.py
<-c|--crash_bin FILENAME> filename to serialize crash bin class to
[-p|--proc_name NAME] process name to search for and attach to
[-i|--ignore_pid PID] ignore this PID when searching for the target process
[-l|--log_level LEVEL] log level <default 1>, increase for more verbosity
[--port PORT] TCP port to bind this agent to
```

```
Administrator: CMD Shell
ERR> USAGE: network_monitor.py
<-d|--device DEVICE #> device to sniff on (see list below)
[-f|--filter PCAP FILTER] BPF filter string
[-P|--log_path PATH] log directory to store pcaps to
[-l|--log_level LEVEL] log level <default 1>, increase for more verbosity
[--port PORT] TCP port to bind this agent to

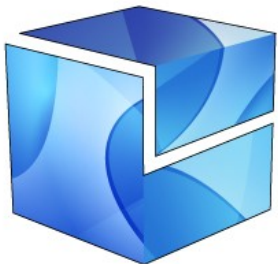
Network Device List:
[0] <7BF0FFED-E211-4F5E-845E-053243D8B51F> 192.168.1.12
[1] <96D16C82-1E3B-44B4-B5D5-99FE59F88571> 192.168.152.1
[2] <D350C2BB-87AA-466E-99A7-BFEC5EAD477A> 192.168.0.6
[3] <711D6536-A9C4-4CD6-BEB2-C106238F2489> 192.168.175.1
[4] <F599839C-39A4-4F1F-9C23-B85F3339F89C>
```





# Schemat procesu testowania automatycznego





# Sulley – raport

- `crashbin_explorer.py` – pozwala na analizę miejsc, w których praca testowanego programu została zaburzona. Crashbin dostarcza nam wielu przydatnych danych, takich jak wartości rejestrów w chwili wystąpienia błędu oraz umożliwia wygenerowanie grafu ze ścieżką błędu.
- Web Monitoring Interface



## Sulley Fuzz Control

RUNNING

Total: 809 of 27,138 [= ] 2.981%

CUSTOM: 809 of 27,138 [= ] 2.981%

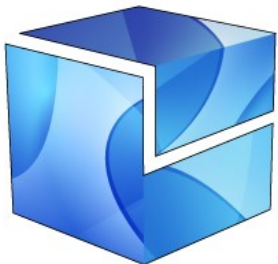
Pause

Resume

Test Case #

Crash Synopsis

Captured Bytes



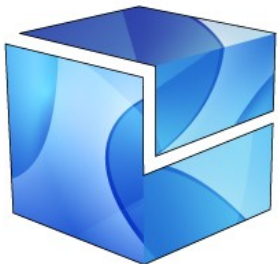
# Przykład – Simple HTTP Server

*http.py*

```
from sulley import *

s_initialize("HTTP")
s_group("methods", values=["GET", "HEAD", "POST", "TRACE", "PUT"])
if s_block_start("main", group="methods"):
    s_delim(" ")
    s_delim("/")
    s_string("index.htm")
    s_delim(" ")
    s_string("HTTP")
    s_delim("/")
    s_string("1")
    s_delim(".")
    s_string("1")
    s_static("\r\n\r\n")
s_block_end(„main”)
```





# Przykład – Simple HTTP Server

*http-test.py*

```
from sulley import *  
import http
```

```
def start():
```

```
    sesja = sessions.session(session_filename="audits/simple-server.session")
```

```
    cel = sessions.target("192.168.0.6", 80)
```

```
    cel.netmon = pedrpc.client("192.168.0.7", 26001)
```

```
    cel.procmon = pedrpc.client("192.168.0.6", 26002)
```

```
    cel.procmon_options = \
```

```
{  
    "proc_name" : "http.exe",  
    "stop_commands" : ['taskkill /IM http.exe'],  
    "start_commands" : ['c:\\http\\http.exe'],  
}
```

```
    sesja.add_target(target)
```

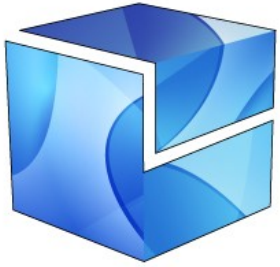
```
    sesja.connect(s_get("HTTP"))
```

```
    sesja.fuzz()
```

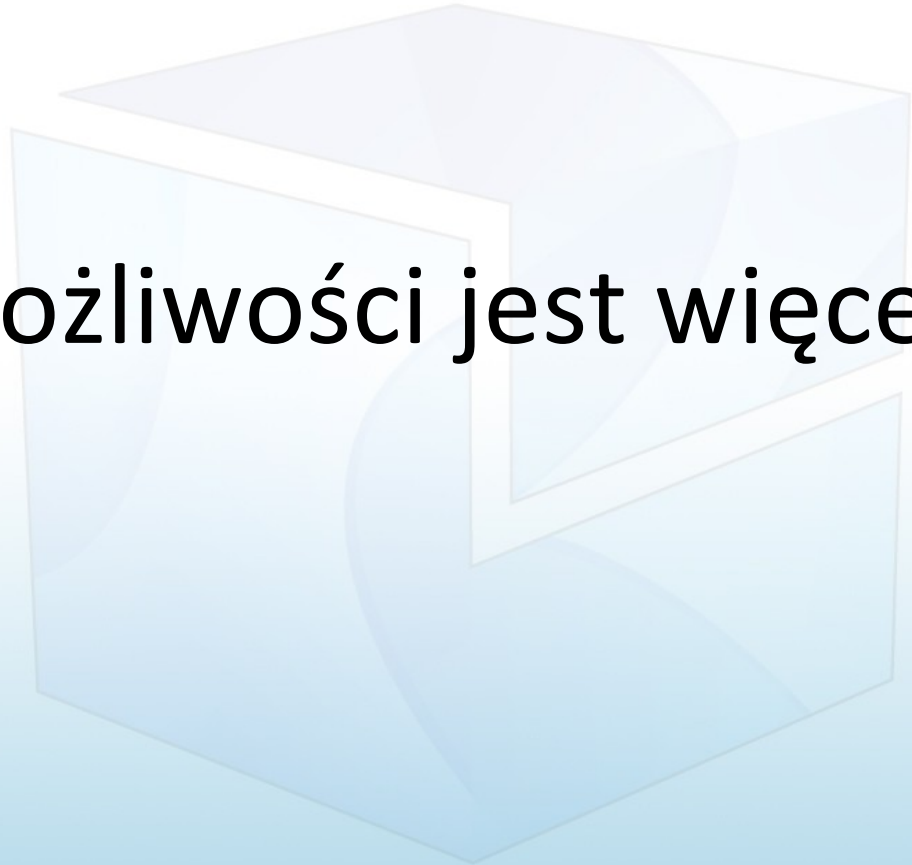
```
start()
```



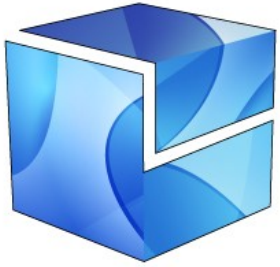




Możliwości jest więcej...

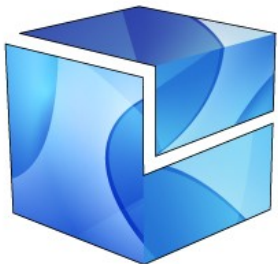


elite SOLUTIONS



# Peach

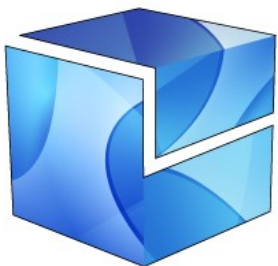




# Fusil

- Fusil jest nowym, ciągle rozwijanym frameworkiem służącym do pisania fuzzerów.
- Zapewnia wiele zróżnicowanych możliwości wykrywania błędów – od analizy kodu powrotu, poprzez sprawdzanie logów systemowych, a kończąc na monitoringu użycia CPU.
- Fusil zapewnia także wsparcie w tworzeniu środowiska testowego i zarządzania zasobami. Kontroluje on priorytety procesów, tworzy katalogi tymczasowe itd.

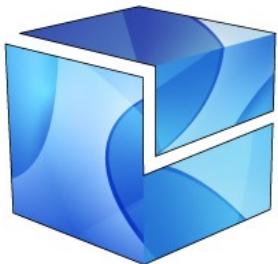




# Wady fuzzingu

- Praktyczny brak możliwości wykrycia błędów logicznych występujących w oprogramowaniu.
- Niemożność przewidzenia czasu (choćby przybliżonego), który potrzebny jest na wykrycie podatności.
- Wraz ze stopniem skomplikowania badanego programu rośnie złożoność fuzzera.
- Techniki zapobiegające fuzzingowi:
  - Spełnianie określonych wymagań przez plik/protokół (źle zbudowany zbiór danych jest pomijany)
  - Wczesna walidacja i odrzucanie niepoprawnych danych

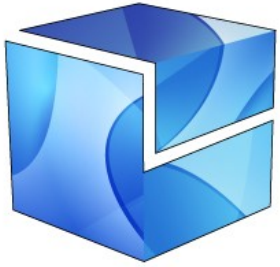




## W sieci...

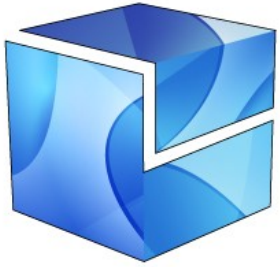
- [fuzzing.org](http://fuzzing.org)
- [fuzzing.eu](http://fuzzing.eu)
- [darknet.org.uk](http://darknet.org.uk)
- [owasp.org/index.php/Fuzzing](http://owasp.org/index.php/Fuzzing)





# PYTANIA...



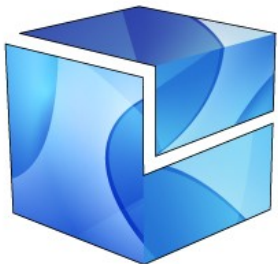


# SEConference



[www.seconference.pl](http://www.seconference.pl)





# Bardzo dziękuję za uwagę

Bardzo dziękuję Działce i Mateuszowi za grafiki :)

Piotr Łaskawiec  
plaskawiec@elitesolutions.pl



ELITE SOLUTIONS