Dokumentacja projektu TIN

Zespół:
Paweł Andruszkiewicz
Marek Moraczyński
Alexis Nowikowski
Paweł Wiszenko

Temat projektu: Implementacja serwera umożliwiającego zdalną rekonfigurację zapory ogniowej działającą w oparciu o program zarządzający iptables. Zlecenie wykonania rekonfiguracji wykorzystuje połączenie z wykorzystaniem protokołu HTTP i notacji/składni JSON. Uwaga !!! W protokole komunikacji należy zaproponować schemat uwierzytelniania bez przesyłania hasła w postaci jawnej.

Wstęp

Tematem projektu jest implementacja serwera umożliwiająca zdalną rekonfigurację zapory ogniowej w oparciu o program zarządzający iptables.

Środowisko

Program serwera działa w systemie operacyjnym Linux. Jako, że tematem projektu jest implementacja samego serwera, klientem będziemy nazywać dowolne narzędzie, które pozwala wykonywać zapytania z użyciem protokołu HTTP i notacją JSON. Aby jednak ułatwić testowanie zdania napisany został prosty klient HTTP.

Klient HTTP

Aby zaobserwować działanie serwera można wykorzystać przykładowego klienta HTTP napisanego w Pythonie - ***client.py***. Za pomocą tego klienta można wykonywać wszystkie obsługiwane polecenia i dobrze przetestować działanie serwera.

Uwaga! Klienta należy uruchamiać po uruchomieniu serwera. Należy też mieć na uwadze, że jest to tylko program do prostego testowania i nie oferuje możliwości logowania (loguje się na domyślnie ustawione użytkownika i hasło) oraz nie pozwala na ponowne zalogowanie po wykonaniu błędnego polecenia.

Serwer

Serwer uruchamia się jako program konsolowy. Serwer jest napisany w języku C/C++. Jeden wątek przeznaczony jest na nasłuchiwanie nowych połączeń, a gdy takie się pojawią

tworzony będzie nowy wątek, w którym będą obsługiwane zapytania użytkowników. Synchronizacji wymagają pliki log.

Sposób uruchomienia serwera: ***sudo ./serwer*** oraz opcjonalnie argumenty:

- ścieżka do pliku konfiguracyjnego: ***-config=[path_to_config_file]***
- włączenie poziomów logowania: **-log=[e][i][a]**, gdzie:
    - e - włączenie logowania błędów (error)
    - a - włączenie logowania informacji o dostępie (access)
    - i - włączenie logowania informacyjnego (info)

Przykładowe wywołanie: **sudo ./serwer -config=/bin/usr/iptables.conf -log=eai**

Instalacja

Kompilacja, konfiguracja oraz instalacja programu polega na uruchomieniu skryptu w Bash. Skrypt skompiluje program oraz utworzy plik konfiguracyjny.

Wykorzystywane biblioteki:
Do parsowania JSON:
https://github.com/open-source-parsers/jsoncpp
Do obsługi hashowania MD5:
http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5

Konfiguracja serwera

Konfiguracja serwera będzie przechowywana w pliku konfiguracyjnym *iptables.conf*. Przechowywane tam są informacje takie jak:

- adres IP nasłuchu (brak pola oznacza dowolny adres),
- port TCP nasłuchu,
- timeout sesji,
- timeout transmisji,
- ścieżka do katalogu w którym będą generowane pliki log,
- ścieżka do pliku z listą użytkowników, którzy mogą dokonać uwierzytelnienia z serwerem,
- ścieżka do pliku z listą zabronionych zakresów IP klientów.

Ścieżka do pliku konfiguracyjnego może zostać podana jako argument wywołania (patrz rozdział “Serwer”), gdy nie zostanie podana - program spodziewa się znaleźć plik konfiguracyjny (o nazwie *iptables.conf*) w folderze z którego uruchamiany jest serwer.

Przykładowy plik konfiguracyjny:

```
# iptables.conf
[core]
host_name HOST_NAME
```

```
;server_ip 127.0.0.1
server_port 12345

;timeout is in seconds
session_timeout 600
transmission_timeout 30

[paths]
log_path ./
users_path ../users.txt
blacklist_path ../blacklist.txt
```

Przykładowy plik blacklist.txt:

```
# blacklist - blocked ip addresses
88.23.1.115
192.168.1.120-192.168.1.144
192.168.1.1
192.168.1.101-192.168.1.102
```

Przykładowy plik users.txt:

```
# users.txt - name:password
mareczek:123456
andrusza2:trolololo
aasd:asda sad asdas das
```

Funkcjonalność

- Serwer umożliwia wyświetlanie listy reguł
- Serwer umożliwia usuwanie reguł
- Serwer umożliwia dodawanie reguł blokowania/przepuszczania połączeń pochodzących z konkretnych adresów IP
- Serwer umożliwia dodawanie reguł blokowania/przepuszczania ruchu na konkretnym porcie TCP lub UDP
- Serwer umożliwia dodawanie reguł blokowania/przepuszczania połączeń pochodzących z konkretnych adresów MAC
- Serwer umożliwia dodawanie reguł blokowania/przepuszczania połączeń wychodzących do konkretnych adresów IP

Lista dostępnych poleceń:

LOGIN_INIT = 0

LOGOUT = 1,
GET_ALL_RULES = 2,
DELETE_RULE = 3,
BLOCK_IP = 4,
BLOCK_TCP_PORT = 5,
BLOCK_UDP_PORT = 6,
BLOCK_INCOMING_MAC = 7,
RAW = 8

W każdym z przypadków należy podać parametry: challenge, hash, command, params. Tam gdzie one nie są potrzebne należy podać wartości puste.

Format poleceń

Zapytania i odpowiedzi serwera przesyłane są w formacie JSON. Url zapytania HTTP musi być określony jako: *http://HOST:PORT/iptables_mgmt*, gdzie HOST może być albo adresem IP serwera albo nazwą hosta, a PORT to numer portu, na którym nasłuchuje serwer.

Format wysłania polecenia do serwera jest następujący:

```
{
  “challenge”: “pkNhbA89D3b0iZGTwFVslXtc”,
  “hash”: “457ec35e8959d02dfceb2e219c5837d3”,
  "command": "",
  "params": {
        "param1" : "",
        "param2" : "",
        .
        .
        .
        “paramN” : “”
  }
}
```

Pole “challenge” identyfikuje użytkownika - dzięki temu serwer wie, który użytkownik wysłał polecenie. Pole “command” będzie reprezentować identyfikator polecenia do wykonania, natomiast tablica “params”, argumenty związane z tym poleceniem.

Przykłady JSONów z poleceniem:

- **żądanie uwierzytelnienia**

```
{
"challenge": "",
 "hash": "",
 "command": 0,
 "params": {
        "username": "mareczek"
```

```
        }
    }
```

- **blokowanie ruchu z danego adresu IP**

```
    {
        "challenge": "SjYRsV8KUruEvnikv38ZqjTG",
        "hash": "639623df7ab5a309b1b74b3c16336b06",
        "command": 4,
        "params": {
            "chainType": 0,
            "ip": "70.70.70.80"
        }
    }
```

Więcej przykładów w rozdziale “**Przykłady JSONów**” na końcu tego dokumentu.

Odpowiedzią natomiast będzie JSON:

```
{
  "error_code": ,
  "error_message": "",
  "challenge": "pkNhbA89D3b0iZGTwFVslXtc",
  "data”: ""
}
```

Przykładowe możliwe odpowiedzi JSON:

Odpowiedź w przypadku kiedy użytkownik nieuwierzytelniony próbuje wysłać komendę do serwera niezwiązaną z uwierzytelnianiem.

```
{
  "error_code": 20,
  "error_message": ".",
  "challenge": "pkNhbA89D3b0iZGTwFVslXtc"
}
```

Odpowiedź na LOGIN_INIT:

```
{
  "error_code": 0,
  "error_message": "OK",
  "challenge": "pkNhbA89D3b0iZGTwFVslXtc"
}
```

Pole error_code reprezentuje kod błędu (albo sukcesu). Dodatkowo możemy wysłać w polu “error_message” komunikat błędu dla danego “error_code” albo komunikat sukcesu (“OK”). Pole “data” będzie zawiera informacje zwrotne dot. wykonanego polecenia np: w przypadku wyświetlenia listy reguł iptables, jest to tablica reguł iptables. Pole “challenge” zawiera wyzwanie, które będzie używane przy wykonywaniu kolejnego zapytania.

Uwierzytelnianie

Aby skorzystać z usług serwera, należy się uwierzytelnić w systemie. Przebieg uwierzytelniania wygląda następująco:

1. Klient wysyła do serwera zapytanie z informacją o chęci próby uwierzytelnienia oraz nazwą użytkownika.
2. Serwer sprawdza czy adres IP nie jest na liście zabronionych adresów ip. Jeżeli nie jest - serwer odsyła odpowiedź w postaci losowego ciągu znaków (wyzwanie). Serwer zapamiętuje krotkę {aktualne_wyzwanie, nazwa_użytkownika}, gdzie aktualne_wyzwaniem jest identyfikatorem krotki.
3. Klient, aby wykonać polecenie, hashuje za pomocą MD5 hasło użytkownika wraz z wyzwaniem serwera i treścią wiadomości i wysyła hash, wyzwanie i wiadomość.
4. Serwer również oblicza hash MD5 hasła użytkownika wraz z wyzwaniem i treścią wiadomości. Jeżeli hashe są takie same, użytkownik zostaje uwierzytelniony i wykonane są przesłane polecenia i wysyła odpowiedź wraz z nowym wyzwaniem, które musi zostać użyte przy następnym zapytaniu.
5. Po uwierzytelnieniu, aby zrobić kolejne zapytanie będąc uwierzytelnionym należy powtórzyć kroki 3-4 z nowym wyzwaniem otrzymanym w odpowiedzi w punkcie 4.

Sytuacje wyjątkowe

W sytuacji wyjątkowej serwer próbuje przechwycić wyjątek oraz zapisać go do pliku error log. Następnie wysyła odpowiedź z informacją o błędzie do klienta (w formacie JSON).

Logi

Serwer będzie zapisywał logi do różnych plików log. Istnieją trzy poziomy logów: **ERROR, ACCESS, INFO**. Każdy z nich zapisywany jest do innego pliku. Error log będzie przechowywał wszystkie logi, które reprezentują błędy. Access (audit) log będzie zawierał informacje dotyczące udanych i nieudanych prób uwierzytelniania do serwera. Info log będzie przechowywał informacje o działaniu serwera i wykonanych poleceniach iptables.
Aktywne poziomy logowania można ustawić podczas uruchamiania serwera (jako argumenty wywołania - patrz rozdział “Serwer”), zaś ścieżką w której zapisywane będą logi można ustawić w pliku konfiguracyjnym (patrz rozdział “Konfiguracja serwera”).

Logi error będą w formacie:
%DATE% %ERROR_MESSAGE%
Logi access będą w formacie:
%DATE% %USER% %AUTHENTICATION_STATUS%
Logi audit będą w formacie:
%DATE% %USER% %INFORMATION_MESSAGE%

Przykładowe fragmenty plików log:

- error log

```
26/11/2015:11:01:13 Incorrect configuration file
26/11/2015:11:01:21 OutOfMemoryException
```

- access log

```
26/11/2015:11:01:13 kowalski Authentication request received
26/11/2015:11:01:21 kowalski Successful authentication!
26/11/2015:11:04:10 mareczek Authentication request received
26/11/2015:11:04:15 mareczek FAILED: Password incorrect
26/11/2015:11:05:00 mareczek Authentication request received
26/11/2015:11:05:46 mareczek Successful authentication!
26/11/2015:11:07:12 kowalski Logged out
```

- audit log

```
26/11/2015:11:01:13 kowalski Command executed: iptables -L -n
26/11/2015:11:01:21 kowalski Command executed: iptables -D INPUT -s 127.0.0.1 -p tcp
--dport 111 -j ACCEPT
```

Przykłady JSONów

LOGIN_INIT REQUEST

```
{
"challenge": "",
 "hash": "",
 "command": 0,
 "params": {
          "username": "mareczek"
          }
}
```

LOGOUT REQUEST

```
{
"challenge": "PtglhGXrcqHPMpsrjDbPP1Kh",
 "hash": "7d4ed2a063403e9bb2a3b338660eaaac",
 "command": 1,
 "params":””
}
```

GET_ALL_RULES REQUEST:

```
{
"challenge": "fa37JncCHryDsbzayy4cBWDx",
 "hash": "8b3898ffcb49939cb631141ced38b5bd",
 "command": 2,
 "params":""
}
```

DELETE_RULE REQUEST:

```
{
"challenge": "fa37JncCHryDsbzayy4cBWDx",
 "hash": "8b3898ffcb49939cb631141ced38b5bd",
 "command": 3,
 "params": {
          "chainType": 0,
          "line": 3
          }
}
```

BLOCK_IP REQUEST:

```
{
          "challenge": "SjYRsV8KUruEvnikv38ZqjTG",
          "hash": "639623df7ab5a309b1b74b3c16336b06",
          "command": 4,
          "params": {
                    "chainType": 0,
                    "ip": "70.70.70.80"
          }
}
```

GET_ALL_RULES RESPONSE

```
{
   "challenge" : "pkNhbA89D3b0iZGTwFVslXtc",
   "data" : "Chain INPUT (policy ACCEPT)\ntarget     prot opt source               destination
\n\nChain FORWARD (policy ACCEPT)\ntarget     prot opt source               destination
```

```
\n\nChain OUTPUT (policy ACCEPT)\ntarget     prot opt source               destination
\n",
   "error_code" : 0,
   "error_message" : "OK"
}
```


DELETE_RULE RESPONSE


```
{
   "challenge" : "7M087wp5h4gvOIUK5FW6C7nr",
   "error_code" : 0,
   "error_message" : "OK"
}
```


Kody błędów

10 - Request was not in JSON format
11 - Request was not in valid format
12 - Error: invalid command.
13 - Error: executing command.
20 - Authorization failed.
21 - Specified username not exist.