AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Informatyki, Elektroniki i Telekomunikacji
Katedra Informatyki

INŻYNIERIA OPROGRAMOWANIA - PROJEKT

WERYFIKACJA POPRAWNOŚCI WYNIKÓW ZWRACANYCH PRZEZ SERWERY DNS DLA SERWISU KLIENTA

DOKUMENTACJA DEWELOPERSKA

MATEUSZ BIELESZ, WOJCIECH KOSIOR, MAREK MORYL, KAMIL SZAREK

KIERUNEK:
Informatyka

OPIEKUN:
mgr inż. Witold Rakoczy

Kraków, 2020

Spis treści

1. Back-end

Przykłady komend konsolowych w niniejszym dokumencie idą za konwencją, wg. której komenda wykonywana jako użytkownik root poprzedzona jest znakiem '#', a komenda wykonywana jako inny użytkownik - znakiem '$'.

2. Wymagania części back-endowej systermu

2.1. System

Ze względu na użycie linuksowych przestrzeni nazw, back-end 0tdns działa obecnie wyłącznie pod systemami z rodziny GNU/Linux.

Dodatkowo, konieczne jest wsparcie jądra dla przekazywanie pakietów IP (forwarding), sieciowych przestrzeni nazw (namespace'ów), iptables i urządzeń tun.

2.2. Środowisko

system operacyjny	Ubuntu 18.04
język	Python3 (system był tworzony z użyciem wersji 3.5.3)
zależności	cron (w dużej części dystrybucji automatycznie zainstalowany)
	libunbound z bindingiem do pythona3; w rodzinie Debiana możliwy do zainstalowania przez `# apt install python3-unbound`
	openvpn w rodzinie Debiana możliwy do zainstalowania przez `# apt install openvpn`
	iptables; w rodzinie Debiana możliwe do zainstalowania przez `# apt install iptables`
baza danych	PostgreSQL wesja 10.12

2.3. Uwagi

Skrypty systemu są uruchamiane cyklicznie przez demona crona. Aby system działał, musi być uruchomiony demon crona. Najskuteczniejszym rozwiązaniem jest uruchamianie crond

przy starcie systemu (w niektórych dystrybucjach, w szczególności tych z rodziny Debiana, domyślnie włączone).

3. Instalacja części back-endowej

3.1. Pobranie systemu

W celu zainstalowania systemu klonujemy repozytorium: `https://repo.or.cz/0tDNS.git`

```
$ git clone https://repo.or.cz/0tDNS.git && cd 0tdns
```

3.2. Instalacja plików wchodzących w skład systemu

Wykonujemy jako użytkownik root:

```
# ./install.sh
```

Skrypt `install.sh` przekopiowuje do `/var/lib/0tdns/` oraz do `/usr/sbin/` i `/usr/bin/` skrypty wchodzące w skład systemu.

Oprócz tego `install.sh` umieszcza importowany przez inne skrypty `ztdnslib.py` w `/usr/lib/python3/dist-packages/` oraz kopiuje `db_connection_config.yml` do katalogu `/etc/0tdns/`.

Przy wywołaniu `install.sh` można ustawić odpowiednią zmienną środowiskową

```
$ INSTALL_ROOT=/some/path/ ./install.sh
```

lub podać argument do skryptu

```
$ ./install.sh /some/path/
```

Efekt jest w obu przypadkach taki sam - wszystkie pliki zostaną zainstalowane w `/some/path/`, co może zostać wykorzystane do instalacji wewnątrz chroot'a lub do stworzenia pakietu dystrybucji.

3.3. Automatyczna część konfiguracji systemu

Po instalacji plików w systemie konieczne jest utworzenie użytkownika 0tdns i dodanie wpisów do crontaba, co wykonać można wywołaniem skryptu

```
# ./setup.sh
```

Funkcjonalność `setup.sh` i `install.sh` została rozdzielona z myślą o dystrybucji systemu.

3.4. Ładowanie niezbędnych modułów jądra

Dla działania systemu potrzebne są sterowniki tun i ip_tables. W zależności od dystrybucji mogą one być wbudowane w jądro lub dostarczone w postaci modułów. W tym drugim przypadku konieczne jest załadowanie modułów:

```
# modprobe tun
# modprobe ip_tables
```

Tak załadowany moduł działa do momentu ponownego uruchomienia systemu operacyjnego. Aby moduły były automatycznie ładowane przy starcie, należy wpisać je do odpowiedniego pliku konfiguracyjnego:

```
# echo tun >> /etc/modules
# echo ip_tables >> /etc/modules
```

3.5. Włączenie przekierowywania pakietów

Aby 0tdns działał, jądro musi przekazywać pakiety IP z domyślnego urządzenia sieciowego na urządzenia veth, które będą tworzone (za włączenie przekazywania pakietów w drugą stronę - od urządzenia veth - odpowiadają skrypty systemu). Ta funkcjonalność jest domyślnie wyłączona w przypadku większości dustrubucji. Jeśli dane jądro ją wspiera, można ją włączyć poprzez zapis do odpowiedniego pliku w systemie plików `/proc`.

```
# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
```

Powyższy przykład pokazuje, jak włączyć przekazywanie ruchu ze wszystkich urządzeń. Znając nasze główne urządzenie (tj. to łączące nas z siecią internet), możemy włączyć przekazywanie jedynie pakietów przychodzących z niego. Przykładowo, jeśli tym urządzeniem jest eth0:

```
# echo 1 > /proc/sys/net/ipv4/conf/eth0/forwarding
```

Tak wprowadzona zmiana nie jest jednak persystentna ze względu na ponowne uruchomienia systemu operacyjnego. Aby tak się stało, należy zmodyfikować odpowiedni plik konfiguracyjny:

```
# echo 'net.ipv4.conf.eth0.forwarding=1' >> /etc/sysctl.conf
```

3.6. Uzupełnienie pliku konfiguracyjnego

Programy działające w ramach systemu automatycznie sukają konfiguracji systemu pod `/etc/0tdns/db_connection_config.yml`. Dostarczony plik jest domyślną konfiguracją, którą administrator systemu powinien zmodyfikować precyzując w niej adekwatny adres bazy danych i inne parametry zgodnie ze specyfikacją w rozdziale poniżej.

4. Konfiguracja części back-endowej

Konfiguracja systemu odbywa się poprzez plik `/etc/0tdns/db_connection_config.yml`. Domyślna, wymagająca uzupełnienia konfiguracja, jest dostarczona z systemem.

Wykorzystany jest tu format yaml. Możliwe jest dodawanie komentarzy zaczynających się od znaku '#'.

Poniżej wyjaśnione jest znaczenie poszczególnych zmiennych w pliku konfiguracyjnym.

4.1. Zmienna user

Zawiera nazwę używanego użytkownika (roli) w bazie danych. Nazwa może - choć jeśli nie zawiera znaków specjalnych, to nie musi - być zamknięta w cudzysłów.

Przykład:

```
user: postgres
```

4.2. Zmienna password

Zawiera hasło do bazy podanego w zmiennej user użytkownika. Nazwa może - choć jeśli nie zawiera znaków specjalnych, to nie musi - być zamknięta w cudzysłów.

Przykład:

```
password: postgres
```

4.3. Zmienna host

Zawiera adres bazy danch. Może to być zarówno nazwa domenowa, jak i adres IP.

Przykład:

```
host: "127.0.0.1"
```

4.4. Zmienna port

Zawiera port, na którym należy łączyć się z bazą danych. Port może, choć nie musi, być wzięty w cudzysłów.

Przykład:

```
port: "5432"
```

4.5. Zmienna database

Zawiera nazwę używanej bazy w systemie bazodanowym. Nazwa może - choć jeśli nie zawiera znaków specjalnych, to nie musi - być zamknięta w cudzysłów.

Przykład:

```
database: "ztdns"
```

4.6. Zmienna enabled

Precyzuje, czy instancja back-endu ma pracować. Jeśli zmienna ustawiona jest na “yes” - system działa normalnie i cogodzinnie wykonywane są zapytania. Jeśli ustawiona jest na “no” - system jest wyłączony.

Dzięki tej zmiennej możliwe jest czasowe wyłączenie danej instancji systemu, np. w wypadku awarii. W dostarczonym szablonie pliku konfiguracyjnego system nie jest włączony.

Wartość zmiennej może - choć nie musi - buć wzięta w cudzysłów.

Przykłady:

```
enabled: no
```

```
enabled: "yes"
```

4.7. Zmienna handled_vpns

Ma ona zawierać listę bazodanowych id serwerów VPN, które ta instancja back-endu ma obsługiwać. Tym samym możliwe jest rozdzielenia pracy na kilka maszym.

Ta zmienna może pozostać nieustawiona - wtedy dana instancja systemu będzie obsługiwała wszystkie połączenia VPN w bazie danych.

W liście mogą się znaleźć także id, które nie występują w bazie danych.

Przykład:

```
handled_vpns: [1, 2, 17]
```

4.8. Zmienna parallel_vpns

Zmienna, która decyduje o tym, ile połączeń VPN może być zestawionych jednocześnie. System w żadnym momencie nie nawiąże więcej połączeń VPN, niż wynosi wartość tej zmiennej. Umożliwia to administratorowi kontrolowanie obciążenia maszyny i łącza sieciowego.

Zmienna powinna być liczbą, nieotoczoną cudzysłowami.

Przykład:

```
parallel_vpns: 20
```

4.9. Zmienna private_addresses

Zmienna zawiera listę zakresów adresów IP, które system może nadawać parom veth. Format pojedynczego zakresu to “<adres_ipv4> - <adres_ipv4>”. Znaki białe naokoło myślnika są opcjonalne. Spośród podanych w tej zmiennej zakresów powinno dać się wybrać przynajmniej tyle podsieci z maską /30, ile wynosi wartość zmiennej parallel_vpns (niespełnienie tego wymogu objawi się odpowiednimi wpsami w logach).

Przykład:

```
private_addresses: ["10.25.25.0 - 10.25.25.59", "10.25.26.0 - 10.25.26.
```

5. Deinstalacja części back-endowej

W celu odwrócenia zmian dokonanych przez `setup.sh` należy wywołać (jako root)

```
# ./uninstall.sh
```

Jeśli z systemu mają być usunięte także pliki zainstalowane przez skrypt `install.sh`, można użyć następującej flagi

```
# uninstall.sh --delete-files
```

6. Działanie części back-endowej systemu

6.1. Cogodzinne odytywanie serwerów DNS

Demon crona co godzinę uruchamia skrypt `hourly.py`, który czyta konfigurację `/etc/0tdns/db_connection_config.yml`, następnie łączy się z bazą danych i pobiera z niej id wszystkich serverów VPN, z którymi ma nawiązać połączenie w celu odpytywania serwerów DNS.

Dla każdego serwera VPN `hourly.py` sprawdza, czy jego plik konfiguracyjny znajduje się w systemie. Jeżli nie, jest pobierany z bazy i zapisywany pod `/var/lib/0tdns/`.

Przez zestawione połączenia VPN odpytywane są odpowiednie serwery DNS, a wyniki są umieszczane w bazie danych.

6.2. Sprawdzanie zakończenia wykonania

Skrypt `hourly.py` na początku działania tworzy plik `/var/lib/0tdns/lockfile`, który na końcu działania usuwa. Jeśli plik istnieje w systemie, oznacza to, że jakaś instancja `hourly.py` pracuje.

Jeśli w momencie rozpoczęcia wykonania `hourly.py` plik lockfile istnieje, skrypt, aby uniknąć kolizji ze swoją wcześniejszą instancją, zakończy działanie bez wykonania zapytań DNS. Ten mechanizm stanowi zabezpieczenie na wypadek sytuacji szczególnej, jak np. błąd systemu. W normalnych warunkach `hourly.py` powinien zakończyć działanie w czasie dużo krótszym, niż godzina.

Dodatkowo, 15, 30 i 45 minut po każdej godzine uruchamiany jest przez demon crona skrypt `check_if_done.py`, który sprawdza, czy plik lockfile istnieje i jeśli tak - wykonuje zapis do logów oraz wysyła maila do administratora (niezaimplementowane).

Na wypadek nagłego wyłączenia systemu operacyjnego, do crontaba dodawane jest polecenie usunięcia pliku lockfile w momencie ponownego uruchomienia.

6.3. Powiadamienie mailowe

O każdej równej godzinie uruchamiany jest skrypt send_emails, który sprawdza w bazie danych, jakie błędne adresy były zwrócone w poprzedniej godzinie i wysyła alert do odpowiednich użytkowników.

6.4. Logi

Logi zapisywane są do pliku `/var/log/0tdns.log`. Znajdują się tam oznaczone godziną informacje m.in. o nawiązywanych i nieudanych połączeniach VPN, działających zbyt długo instancjach skryptu `hourly.py` i błędach w podanych w konfiguracji zakresach adresów IP.

6.5. Zmiany w bazie danych

Wraz z wykonaniem zapytań DNS dodawane są do tabeli user_side_responses oraz ew. user_side_response wpisy z wynikami. Możliwe wartości pola 'results' w user_side_responses to:

- 'successful' - od serwera DNS przyszła odpowiedź, że domena istnieje (jedyny przypadek, kiedy mogą (choć nie muszą) być do tego rekordu dowiązane rekordy w user_side_response)
- 'not exists' - serwer DNS twierdzi, że domena nie istnieje
- 'no reponse' - nie przyszła żadna odpowiedź od serwera DNS
- 'DNS error: <jaki błąd>' - jeśli zwrócony został inny kod błędu, niż powyższe
- 'internal failure: out of memory' - wykorzystywany libunbound zwrócił wartość informującą, że zapytanie DNS się nie powiodło z powodu braku pamięci
- 'internal failure: vpn_connection_failure' - nie udało się nawiązać danego połączenia VPN
- 'internal failure: process_crash' - błąd skryptu wchodzącego w skład systemu

Nie rozróżniamy na tym etapie, czy ip zwrócone zgadzają się z oczekiwanymi.

Razem z dodaniem wpisu innego typu, niż 'internal failure:', zmniejszany jest licznik ważności odpowiedniego rekordu w tabeli user_side_queries lub rekord jest usuwany (jeśli licznik osiągnął 0).

7. Front-end

7.1. Środowisko

Przy realizacji projektu korzystano z następującej konfiguracji:

system operacyjny	Windows 10
język	Python3 (system był tworzony z użyciem wersji 3.8.1)
framework	Django wersja 3.0.6
	sqlparse wersja 0.3.1
	psycopg2 wersja 2.8.5
	psycopg2-binary wersja 2.8.5
	DateTime wersja 4.3
baza danych	PostgreSQL wersja 10.12 dostępna zdalnie, SQLite dostępna lokalnie

7.2. Uruchomienie servera django

Aby uruchomić serwer django należy wykonać następujące kroki:

1. W folderze o nazwie django stworzyć za pomocą konsoli wirtualne środowisko pythona

   ```
   python -m venv venv
   ```

2. Aktywować środowisko

   ```
   venv\Scripts\activate
   ```

3. Zainstalować potrzebne zależności

   ```
   pip install -r requirements.txt
   ```

4. Uruchomić serwer

   ```
   python manage.py runserver
   ```

7.3. Uruchomienie oprogramowania w usłudze hostingowej

Najprostszym sposobem uzyskania zdalnego dostępu do front-endu jest wykupienie usługi hostingowej. Przykładem takiej usługi może być hosting *https://hostovita.pl* czy *pythonanywhere.com*. Przy wyborze tej opcji należy zastosować następujące zmiany w pliku settings.py:

- zmienić wartość DEBUG z True na False,
- zmienić wartość SECRET_KEY na bezpieczną, niedostępną z zewnątrz wartość klucza, który będzie używany przy zapewnieniu bezpieczeństwa danych, w niektórych przypadkach klucz ten można znaleźć w plikach hostingu,
- należy wykonać następującą komendę: *python manage.py check –deploy* w celu sprawdzenia konfiguracji,

7.4. Uruchomienie własnego serwera obsługującego część front-endową

Uruchomienie projektu na własnym serwerze jest alternatywą dla powyższego rozwiązania, dostępną przy posiadaniu dostępu do takiego serwera. Przy pisaniu konfiguracji założono, że na serwerze działa Linux Ubuntu, dla innej dystrybucji systemu należy wykonać analogiczne komendy. Aby uruchomić projekt w tej konfiguracji należy wykonać następujące kroki:

1. zaloguj się na serwer przy użyciu SSH,
2. zainstaluj pythona z poziomu superusera:
   apt-get install python3
   apt-get install python3-pip
   pip3 install virtualenv
3. z poziomu użytkownika utwórz wirtualne środowisko, wykonując następujące komendy:
   cd
   mkdir django-apps
   cd django-apps
   virtualenv env
   source env/bin/activate
   pip install django
4. w tym momencie framework django został zainstalowany, możesz sprawdzić jego wersję przy użyciu komendy:
   django-admin –version
5. przystąpimy teraz do konfiguracji django, jako użytkownik wykonaj następujące komendy: cd /django-apps
   django-admin startproject mysite
6. za pomocą dowolnego edytora tekstowego otwórz */home/<username>/django-apps/mysite/mysite/settings.py*, gdzie *<username>* jest twoją nazwą użytkownika,
7. w pliku settings.py ustaw: DEBUG = True
8. znajdź linię zaczynającą się od ALLOWED_HOSTS i zmień wartość ALLOWED_HOSTS = ['xxx.xxx.xxx.xxx'], zastępując xxx.xxx.xxx.xxx adresem IP twojego serwera,

9. zapisz zmiany w pliku settings.py,

10. uruchom serwer WWW, wykonując następujące polecenie w terminalu:
    python /django-apps/mysite/manage.py runserver xxx.xxx.xxx.xxx:8000
    gdzie xxx.xxx.xxx.xxx jest adresem IP twojego serwera,

11. używając przeglądarki internetowej wejdź na http://xxx.xxx.xxx.xxx:8000, gdzie xxx.xxx.xxx.xxx jest adresem IP twojego serwera, powinieneś zobaczyć następującą wiadomość: *The install worked successfully! Congratulations!*

12. pozostało jeszcze tylko skonfigurować interfejs administratora, w terminalu należy wykonać następujące komendy: cd
    python /django-apps/mysite/manage.py migrate
    python /django-apps/mysite/manage.py createsuperuser

13. dostaniemy tutaj dostęp do możliwości utworzenia konta administratora, w pole Username należy wpisać nazwę użytkownika dla konta administratora, w pole Email address należy wpisać adres email administratora, na końcu dwókrotnie mależy wpisać hasło w pole Password, należy tutaj zwrócić uwagę aby hasło administratora było bezpieczne,

14. jeżeli serwis nie jest uruchomiony należy wykonać nastęującą komendę:
    python /django-apps/mysite/manage.py runserver xxx.xxx.xxx.xxx:8000
    gdzie xxx.xxx.xxx.xxx jest adresem IP twojego serwera,

15. używając przeglądarki internetowej wejdź na http://xxx.xxx.xxx.xxx:8000/admin,

16. zaloguj się przy użyciu wcześniej ustawionych nazwy użytkownika i hasła,