PSI - Dokumentacja projektu

- Temat: System konwersji CBOR/JSON.
- Autorzy: Dominik Sidorczuk, Tomasz Sroka, **Krzysztof Stawarski (lider)**, Fedir Tsupin
- Data sporządzenia: 31 maja 2023r. (poprawki stylistyczne 6 czerwca 2023r.)

Treść zadania

Zaprojektuj i zaimplementuj system bramy komunikacyjnej dla podsieci do 1024 urządzeń sensorycznych. Brama odbiera od urządzeń dokumenty w kodowaniu CBOR i rozsyła je do danego zbioru rejestratorów (do 32) w kodowaniu JSON. Brama opatruje dokumenty stemplem czasowym i podpisem cyfrowym. Komunikacja odbywa się po UDP. Zbiór aktywnych urządzań i rejestratorów może się zmieniać. Zaproponuj sposób ich autoryzacji.

Definicja bramy

Bramę definiujemy jako host w tej samej sieci lokalnej co sensory, który wysyła dane do rejestratorów po uprzedniej konwersji z CBOR do JSON.

Założenia funkcjonalne

(sensor <-> brama)

- Urządzenie sensoryczne uwierzytelnia się loginem i hasłem
- Brama autoryzuje sensor, lub informuje o błędzie
- Brama przydziela identyfikator sesji do komunikacji z sensorem, lub informuje sensor o braku wolnych miejsc
- Urządzenie negocjuje z bramą interwał wysyłania danych
- Brama czuwa nad przestrzeganiem tych interwałów (timeout po zbyt długim braku odpowiedzi)
- Sensor wysyła dane w formacie CBOR
- Brama dekoduje dane do formatu JSON (do dalszej komunikacji)
- Urządzenie może poinformować bramę o zakończeniu pracy

(brama <-> rejestrator)

- rejestrator autoryzuje się loginem i hasłem (unikalne dane logowania dla każdego)
- Brama autoryzuje rejestrator, lub informuje o błędzie
- Brama przydziela identyfikator sesji do komunikacji z rejestratorem, lub informuje sensor o braku wolnych miejsc
- Rejestrator może wskazać interesujące go grupy sensorów
- Brama po otrzymaniu danych od sensorów, wysyła je do zainteresowanych rejestratorów

- Rejestrator może poinformować bramę o zakończeniu pracy

Założenia niefunkcjonalne

- System powinien być szybki w konfiguracji, tzn. wystarczy ustalić dane logowania dla sensorów oraz rejestratorów
- Konfiguracja kont po stronie bramy jest dostępna przez SSH
- komunikacja między bramą a rejestratorami jest bezpieczna (szyfrowanie, podpis cyfrowy)
- Nie ma konieczności rejestrowania wszystkich danych (pewne braki są dopuszczalne)
- Zapisywane są niektóre dane statystyczne (kto się kiedy zalogował, jakie wystąpiły błędy)

Szyfrowanie komunikacji

Szyfrowanie:

Asymetryczne

występuje:

- na początku między sensorem a bramą kiedy to sensor szyfruje kluczem publicznym bramy swój login, hasło oraz od razu klucz AES do dalszej komunikacji.
- cały czas między bramą a rejestrem kiedy dane są przesyłane od bramy do rejestru. Są szyfrowane asymetrycznie za pomocą klucza prywatnego bramy, aby rejestratory wiedziały, że dane przychodzące są od bramy, bo ciężko będzie podrobić dane jak nie ma się klucza prywatnego.

Implementacja algorytmu szyfrowania asymetrycznego RSA została napisana własnoręcznie wraz z własną notacją zapisu kluczy publicznych i prywatnych. W tym algorytmie korzystamy z biblioteki pycryptodomex, z funkcji generującej liczby pierwsze o jakiejś długości w bitach. Przyjęliśmy założenie że liczby p i q są losowane o długości 512 bitów co daje 64 bajty. n = p * q jest długości 64 * 2 = 128 bajtów stąd też wiadomość nie może być dłuższa niż 128 bajtów, bo sama wiadomość jest zamieniana na liczbę o takiej samej długości zatem gdyby wiadomość była dłuższa niż 128 to wtedy wykonujemy działanie c = m ^ e % n gdzie m > n i wtedy po modulo powstaje nam liczba c < n więc jakbyśmy próbowali odszyfrować m1 = c ^ d % n to dostaniemy m1 < n co dałoby jakieś krzaczki (m1 != m). Gdyby trzeba było można by było dodać podział wiadomości na bloki i szyfrować każdy z osobna, ale w naszym projekcie nie było takiej potrzeby. Sama notacja kluczy wygląda następująco (jest długości parzystej, gdyż każdy bajt został zapisany heksadecymalnie czyli 2 bajty tej notacji = 1 bajt rzeczywistej wartości):

```
base64.b64encode(
```

```
  4 bajty mówią o długości całego klucza,
  2 bajty długości liczby n,
  2x bajtów liczby n,
  2 bajty długości liczby e lub d (w zależności od klucza),
  2y bajtów liczby e lub d
)
```

Symetryczne

Występuje do przesyłania danych pomiarowych między sensorem a bramą. Co ileś odebranych danych następuje prośba ze strony bramy na zmianę klucza symetrycznego. Wtedy sensor, gdy uda mu się odebrać odpowiednio zaszyfrowaną wiadomość w której jest kod wykonuje polecenie, generuje nowy klucz symetryczny oraz dokleja na początek kod wiadomości dla bramy i szyfruje całość symetrycznie starym kluczem. Wtedy brama wie, że to nowy klucz od sensora więc go odszyfrowuje starym i podmienia.

Do szyfrowania symetrycznego (AES) wykorzystaliśmy tę samę bibliotekę co wyżej, w której to poprostu generujemy zawsze klucz o długości 16 bajtów (zalecana długość oraz maksymalna) gdyż później wbudowane funkcje szyfrujące dzielą dane na 16 bajtowe bloki i każdą szyfrują z osobna. Sam algorytm szyfrowania jest dosyć skomplikowany nie mniej jednak jest jednym z silniejszych szyfrowań symetrycznych, ponieważ uzyskany szyfrogram został z jak największą nieliniowością przekształcony.

Logowanie

Do logowania został użyty moduł `logging` z biblioteki standardowej. Wszystkie rodzaje komunikatów (debug i wyżej) są wysyłane do plików .log, do konsoli są wysyłane tylko rodzaje info i wyżej. Jest to zrobione, żeby nie śmiecić konsoli użytkownika komunikatami, które mogą przydać się przy analizie działanie systemu, ale nie są istotne w czasie działania programu.

Formaty

Format linii w logach: `<timestamp>-<rodzaj>-<tekst>`

Np.: 2023-05-27 19:31:48,925 - DEBUG - Received sensor data from 127.0.0.1

Format linii w konsoli: `<rodzaj>-<tekst>`

Np.: INFO - UDP brama.py's listening on port 12345

Utrzymywanie połączenia

Sensory i rejestratory periodycznie wysyłają wiadomość "bicie serca". Jeżeli brama nie otrzyma kilku takich wiadomości pod rząd, uznaje, że urządzenie przestało odpowiadać, domyślnie jest to 5 utraconych wiadomości. Brama wtedy usuwa urządzenie z tablicy autoryzowanych adresów.

Zamiast używać kopca, tak jak proponowaliśmy początkowo, wykorzystujemy timery (jeden dla każdego urządzenia), którego czas jest przesuwany w przód z każdym biciem serca. Gdy timer dojdzie do zera, sensor jest rozłączany.

Przypadki użycia

- Czujniki domowe w różnych pokojach (np. temperatura, wilgoć, wykrywanie ruchu) przesyłane do różnych odbiorców (każdy domownik ma aplikacje na telefonie)
- Sensory na linach produkcyjnej, dane są wysyłane do rejestratorów zajmujących się różnymi typami sensorów
- Urządzenia GPS zamontowane w samochodach wjeżdżających do miasta, w celu badania zatłoczenia różnych dróg (?)

Obsługa sytuacji błędnych i złośliwych

Brama stara się obsługiwać niektóre sytuacje wyjątkowe:

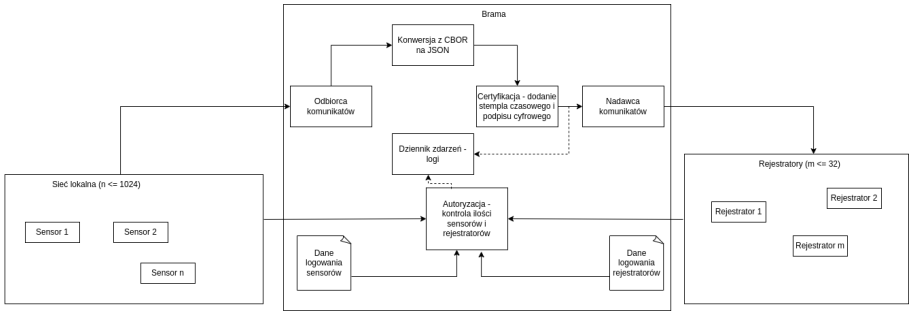
- Wszelkie nieznane wiadomości (zły kod) są ignorowane, a pojawienie się takiej wiadomości jest notowane w logach
- Próby ignorowania odświeżania klucza asymetrycznego przez sensor są tolerowane tylko do pewnego czasu, potem brama rozłącza sensor.
- Brama nie pozwala połączyć się więcej niż 1024 sensorom oraz 32 rejestratorom
- Próby logowania za pomocą złego loginu lub hasła są zapisywane w logach

Wybrane środowisko sprzętowo-programowe (systemy operacyjne, biblioteki programistyczne) i narzędziowe (debugowanie, testowanie)

- Środowisko: Linux, a konkretniej - kontenery z dystrybucją Linuxa
- Język przewodni: Python
- Wykorzystywane biblioteki Pythona:
  - biblioteka standardowa - m.in.: json, logging
  - cbor2 - do parsowania formatu CBOR
  - pycryptodomex - do szyfrowania komunikacji pomiędzy bramą a klientami
- Inne narzędzia pomocnicze indywidualne: debugger wybrany indywidualnie przez każdego członka zespołu, zgodny z używanym przez niego środowiskiem
- Narzędzia wspierające:
  - Docker/Kubernetes - do łatwego powielania klientów i reprodukowalności środowiska testowego

– Git - system konroli wersji, używany w połączeniu z wydziałowym serwerem gitlab

Architektura rozwiązania



- Przed wysyłaniem/otrzymaniem danych sensory/rejestratory muszą zautoryzować się przez blok autoryzacji. Dane do logowania będziemy przechowywać w plikach. Podczas tego procesu też będą wysyłane logi do bloku Dziennika zdarzeń.
- Po zalogowaniu sensory mogą wysyłać dane w formacie CBOR do bloku odbiorcy komunikatów, który będzie analizował ich poprawność.
- Następnie po stronie bramy następuje konwersja z CBORa na JSON oraz dołączenie stempla czasowego oraz podpisu cyfrowego.
- Dane ostatnieczne są rozsyłane do rejestratorów.

Szczegółowy opis wiadomości

Brama rozpoznaje siedem typów wiadomości:

```
class CODES(Enum):
    sensor_auth = 0
    rejestrator_auth = 1
    auth_success = 2
    auth_error = 3
    sensor_data = 4
    heartbeat = 5
    sensor_required_tochange_aeskey = 6
```

Wszystkie z nich są przesyłane w tym samym formacie - pierwszy bajt to numer wiadomości, następnie jest jej zawartość.

Opis poszczególnych wiadomości

0. `sensor_auth`:

format: `0<login>&<hasło>#<klucz_AES>`

Wysyłana tylko przez sensor w celu początkowej autoryzacji. Brama odpowiada poprzez `auth_success` kiedy login i hasło pasuje do któregoś w bazie, lub `auth_error`. Login i hasło jest rozdzielone separatorem '&', można to zmienić w pliku `common.py`. Na końcu jest klucz wygenerowany przez sensor do komunikacji asymetrycznej.

Ta wiadomość jest zawsze szyfrowana asymetrycznie kluczem publicznym bramy, każda nowa wiadomość od nieznanego adresu jest poddawana próbie deszyfrowania.

1. `rejestrator_auth` format: `1<login>&<hasło>`

analogicznie do poprzedniej wiadomości, ale rejestrator nie używa szyfrowania symetrycznego, więc nie wysyła takiego klucza.

2. `auth_success`

format: `2<informacja>`

3. `auth_error`

format: `3<informacja>`

Odpowiedzi na próby autoryzacji, zawiera krótkie podsumowanie tego co zaszło

4. `sensor_data`

format od sensora do bramy: `4<dane_cbor>`

format od bramy do rejestratora: `4<dane json>`

Dowolne dane zbierane przez sensor.

5. `heartbeat` format: `5<tekst>`

Periodyczna wiadomość, informująca bramę, że sensor/rejestrator nadal jest aktywny. Domyślnie po 5 opuszczonych interwałach brama cofa autoryzację dla danego urządzenia.

6. `sensor_required_tochange_aeskey`

format w stronę sensora: 6

format w stronę bramy: `6<nowy_klucz>`

Po przekroczeniu dopuszczonej długości sesji na tym samym kluczu, brama prosi sensor o wygenerowanie nowego klucza. Sensor powinien w ciągu kilku wiadomości odpowiedzieć (domyślnie limit 20), w przypadku ignorowania polecenia sensor jest deautoryzowany.

Sposób testowania

Poza doraźnym testowaniem manualnym, wykorzystujemy testy integracyjne, gdzie zestawiamy ze sobą poszczególne elementy systemu i obserwujemy wyniki

ich działania poprzez analizę logów post mortem. W szczególności, zależy nam na przetestowaniu dwóch rzeczy - minimalnego działającego systemu i systemu przy maksymalnym przewidywanym obciążeniu. Niestety, ze względu na ograniczenia sprzętowe, nie jesteśmy w stanie zasymulować ponad 1000 urządzeń jednocześnie przekazujących sobie komunikaty. Udało się jednak uzyskać dostatecznie dużą (300) liczbę urządzeń na potrzeby testów, które przeprowadzamy z większą częstotliwością wysyłanych komunikatów, co w pewien sposób wyrównuje trudność testu, skoro obciążenie zostaje zwiększone inną metodą.

Wnioski z testowania

Widzimy, że system spełnia postawione mu wymagania - przekazuje wiadomości, zmienia klucze po określonej ilości wysłanych wiadomości z danymi, dokonuje retransmisji utraconych komunikatów sterujących, wykrywa odłączenie się sensorów i rejestratorów. Pojawiają się też jednak problemy - przy testach obciążeniowych, przy większych ilościach sensorów (np. 800), sensory nie są w stanie uwierzytelnić się. Wynika to pewnie z ograniczeń sprzętowych i narzutu na moc obliczeniową, wynikającego z zastosowania kontenerów oraz bardziej kosztownego obliczeniowo języka (Python, a nie chociażby C, C++ czy Go).

Podział prac

Dominik Sidorczuk - szyfrowanie

Tomasz Sroka - konwersja CBOR -> JSON, obsługa komunikacji

Krzysztof Stawarski - infrastruktura (CI, konteneryzacja), testy

Fedir Tsupin - loggowanie, autoryzacja klientów

Podsumowanie

Wyniesione doświadczenia

- Należy bardzo uważać na to jak interpretowane są zwracane dane. Przesył danych po sieci nie daje nam żadnej gwarancji sprawdzania typów danych, co najczęściej występuje w lokalnym środowisku. W naszym wypadku kosztownym w debugowaniu okazała się sytuacja, gdy string z informacją o podaniu niepoprawnego hasła był odsyłany z powrotem do sensora, który traktował go jako klucz i zwracał błąd na poziomie szyfrowania danych. Oczywiście błędy programu były zwracane na kompletnie innym etapie, co skutecznie zbiło nas z tropu.
- Warto wziąć się za poszczególne etapy projektu wcześniej, niż na wieczór przed oddaniem danego etapu. W nocy milej się śpi, niż pisze kod, a już

na pewno milej, niż pisze się dokumentację.

- Tworzenie dokumentacji zajmuje niespodziewanie długo. Warto brać to pod uwagę przy planowaniu pracy i zakładać, że nawet połowa czasu spędzonego nad projektem zostanie poświęcona na dokumentację.

Statystyki dotyczące stworzonych plików

Ilość linijek w plikach danego typu:

- Python - 857
- Bash - 113
- Dockerfile - 62

Sumarycznie: 1032

Szacowany czas spędzony nad projektem

Dominik Sidorczuk - 18 godzin

Tomasz Sroka - 20 godzin

Krzysztof Stawarski - 18 godziny

Fedir Tsupin - 18 godzin