Bezpieczeństwo komputerowe

Laboratoria lista nr 2

Piotr Jaroszewski
229805

Opis zadania

Zadanie polega na złamaniu szyfrowania AES 256 w trybie CBC. W każdym podpunkcie zadania mamy podany sufiks klucza który jest coraz krótszy wraz z kolejnymi podpunktami zadania. Znaki klucza to wartości hexadecymalne. Pełny klucz ma 64 znaki co daje 2^{64} możliwości.

Proces rozwiązywania

Implementacja – ogólny zarys

Do znalezienia klucza wykorzystałem własną implementacje algorytmu brute force w C++ z wykorzystaniem biblioteki openSSL w której znajduje się funkcja deszyfrująca AES.
Program korzysta również z implementacji dekodera base64 znalezionego na stronie stackoverflow[1].
Implementacja rozprasza się na tyle wątków ile podamy w stałej w kodzie programu.

Optymalizacje kompilatora

W zadaniu korzystam tylko i wyłącznie z flagi -O3 podczas kompilacji.

Operacje bitowe

Program wykorzystuje operacje bitowe w każdym możliwym miejscu które może nie być oczywiste dla kompilatora do zoptymalizowania.

Poprawność odszyfrowanego tekstu

Do sprawdzania czy deszyfracja przebiegła pomyślnie sprawdzane są pierwsze znaki zdeszyfrowanej wiadomości. Jeżeli znak należy do zbioru [a-zA-Z .,] to jest uznawany za dobry. Na końcu sprawdzania brany jest procent dobrych w stosunku do całego textu, jeżeli przekracza on pewien próg to tekst uznawany jest za dobrze odszyfrowany. Dobranie odpowiedniego progu poprawności jest oparte o serię eksperymentów które doprowadziły do dobrania współczynnika na poziomie **70%**.

Generator liczb prefiksów

Do tego fragmentu kodu została przyłożona największa uwaga, co przekłada się na wysoką optymalizacje.
Klucz dostajemy w postaci stringa (64 znaki) jednak należy to przerobić na tablice bajtów, gdzie na każdy bajt przypadają dwa znaki hexadecymalne z klucza.
Prototyp całego programu powstawał w Pythonie3 jednak został poniechany na etapie generowania prefiksu klucza który generował osmioznakowe prefiksy około 17 minut. Rozwiązanie w C++ dla 8 znaków generowanie trwa 19sekund. Czyli przejście na C++ daje przyśpieszenie 53-krotne.
Początkowa faza w języku C++ zakładała rozwiązanie konwersji stringu na tablice bajtów w pętli co generowało bardzo duży czas szukania klucza w porównaniu do aktualnego stanu.
Rozwiązanie generowania klucza rekurencyjnie zostało całkowicie poniechane z oczywistych względów.

Podejście jakie zostało zaimplementowane zaczerpnięte zostało ze poraz kolejny ze stackoverflow[2]. Rozwiązanie opiera się o jedną pętlę z licznikiem w której odbywa się cały proces łamania oraz drugą wewnętrzną w której przez pewne operacje bitowe ustawiane są kolejne nibble.
Rozwiązaniem branym pod uwagę było też ustawienie adresu początku tablicy z kluczzem jako adres licznika pętli głównej. Niestety nie dawało to dużych profitów, a powodowało błędy podczas szukania prefiksów o nieparzystej długości.

Implementacja funkcji deszyfrującej
Podczas eksperymentów branych pod uwagę było parę implementacji z których najbardziej odpowiednia okazała się ta z biblioteki openSSL.
Pętla główna funkcji deszyfrującej została tak napisana by jak najmniej wywoływać kolejne funkcje prowadzące do deszyfracji, w związku z czym są tam dwa warunki powstrzymujące kolejne etapy deszyfracji przed wykonaniem wrazie nie powodzenia.

Osadzenie obliczeń na GPU
Była to jedna z pierwszych myśli które przyszły do głowy kiedy miej więcej w połowie procesu optymalizacji program znajdował prefik ośmioznakowy w czasie kilku minut na czterech wątkach CPU.
W celu postawienia procesu na wątkach GPU rozpatrzone zostały dwie biblioteki służące do pracy z szyfrowaniem AES 256 w trybie CBC.
Jedna z nich została znaleziona na githubie[3]. Ta biblioteka mimo usilnych prób edycji nie chciała pracować poprawnie ze względu na użycie statycznych zmiennych i logiki która nie pozwalała rozproszyć biblioteki na wątki.
Drugiej biblioteki niestety nie została odnaleziona w czasie pisania tego tekstu.
Kolejnym rozwiązaniem które też nie zadziałało poprawnie była biblioteka libgpucrypto również znaleziona na githubie[4].
Biblioteka straciła wsparcie wiele lat temu dlatego nie chciała nawet się skompilować wraz z technologią CUDA 8 aktualnie zainstalowaną na platformie do której uzyskałem dostęp. Libgpucrypto zatrzymała się na CUDA 4.
Kolejnym krokiem po wielu nieudanych próbach adaptacji było zapytanie społeczności stackoverflow o rozwiązanie które mógłbym zaadaptować do swojego projektu. Wtedy zamieścił ktoś informację o specjalnych częściach procesorów Intel służących do obliczeń kryptograficznych. Sugerowało to iż na GPU deszyfracja będzie trwać dużo dłużej. Jednak przy możliwości rozproszenia tego na więcej niż 300 wątków nie miało to znaczenia, w ciągu dalszym było by to nieporównywalnie szybciej niż 4 wątki CPU nawet ze specjalnymi poleceniami na poziomie assemblera.
Wszystkie próby jakie podjąłem osadzenia obliczeń na kartach graficznych w celu rozproszenia tego na kilkaset wątków nie powiodły się z racji nie znalezienia dostatecznie dobrej implementacji.
Osobiście nie podjąłem się pisania biblioteki samemu.

Rozwiązanie

Poniżej zamieszczam tabelę z danymi które udało mi się zebrać podczas wykonywania zadania na kodzie takiej postaci jak w momencie oddawania.

Długość prefiksu	Czas przeszukania całej przestrzeni kluczy	Statystyczny czas szukania klucza
4	0.03s	<1s
5	0.092s	<1s
6	1.045s	<1
7	17.607s	~8s
8	4m 37s	~2m
9	68m	~35m
10	ok. 16h	~8h
11	(szacowany) 10d 16h	(szacowany) ~5d 8h
...		
64	(szacowany) $\sim 2*10^{62}$	

Z zarejestrowanych czasów widać (od prefiksu długości 6) liniowy wzrost czasu potrzebnego na znalezienie klucza. Z takiej statystyki widać też nieopłacalność czasową (co przekłada się na pieniężną) łamania kluczy długości 256b (32B).
Długość życia wszechświata jest nieprównywalnie mniejsza niż czas potrzebny do złamania całego klucza na konfiguracji która została użyta to rozwiązania zadania.
Koszt łamania klucza z nieznanym prefiksem długości 10 to ok. 1.14zł przy założeniu iż 1kWh kosztuje 55 groszy. Pod uwagę został wzięty sam procesor bez uwzględniania płyty głównej i zasilacza, gdyby to uwzględnić koszt wzrósłby do 6.60zł.

Obsługa programu

kompilacja:
g++ main.cpp utility.cpp -lpthread -lssl -lcrypto -O3

uruchomienie
./a.out $crypto $IV $sufiks_klucza $liczba_znaków_do_zgadnięcia

[1] https://stackoverflow.com/a/34571089/2008148
[2] https://stackoverflow.com/a/22983144/2008148
[3] https://github.com/kokke/tiny-AES-c
[4] https://github.com/kay21s/hammer/tree/master/libgpucrypto