Efektywny atak na karty Mifare Classic

Marek Marczykowski

21 czerwca 2010

1 Wstęp

Opisany poniżej atak opiera się głównie na wiedzy zawartej w [1]. Nie będę tutaj tej wiedzy dublował, w szczególności opis karty i protokołu Mifare ograniczę do niezbędnego minimum. Do przeprowadzania ataku wykorzystuję sprzęt: Proxmark 3 (`http://www.proxmark.org/`. Oprogramowanie w nim zastosowane bazuje w większości na oryginalnym jego programie (rewizja 438 z 2010-04-20), [2], oraz implementacji algorytmu Crypto-1 `http://code.google.com/p/crapto1/`.

2 Niezbędne minimum wiedzy o Mifare

Karta Mifare logicznie składa się z 16 sektorów. Do każdego sektora są ustalone dwa klucze (A i B), które mogą mieć skonfigurowane różne prawa dostępu (zapis/odczyt fragmentów sektorów i samych kluczy i praw). Zwykle (poza kartami 4K) sektor składa się z 4 bloków po 16 bajtów. W dalszej części wszędzie (również w parametrach poleceń) gdzie będzie mowa o id sektora - będzie to numer dowolnego bloku wewnątrz niego (np. pierwszy blok w sektorze).

Protokół Mifare opiera się na ISO14443-A (13.56MHz). Sama autoryzacja jest już specyficzna. Opiera się ona na dwóch rejestrach przesuwnych ze sprzężeniem zwrotnym. Jeden (32bit) jest używany jako PRNG (generator liczb pseudolosowych) i taktowany falą nośną, drugi (48bit) - generator bitów szyfrujących.

Ten drugi rejestr jest na początku autoryzacji inicjowany kluczem. Następnie na wejście, wraz ze sprzężeniem zwrotnym (xor) jest podawana 32-bitowa liczba losowa (z PRNG) połączona (też xor) z ID karty. Ta sama liczba jest również wysyłana jako challenge karty. Następnie czytnik (w założeniu znający klucz) wykonuje te same operacje na swojej kopii rejestru.

Od tego miejsca cała komunikacja jest szyfrowana przy pomocy xor z generowanym na podstawie rejestru ciągiem bitów (jeden bit zależący od wybranych 20 bitów rejestru).

Drugi krok autoryzacji, to odpowiedź czytnika. Składa się ona z challenge czytnika (podobnie dodawanego na wejście rejestru), oraz odpowiedzi na challenge karty (już nie wpływającej bezpośrednio na stan rejestru). Całość (64bity) jest zaszyfrowana.

Jeśli karta poprawnie zweryfikuje odpowiedź karty - odpowiada odpowiedzią na challenge czytnika, tak na prawdę zależącego od challenge karty.

Szczegółowy opis algorytmu znajduje się w [1].

Wykorzystam tu szereg słabości tego algorytmu, w szczególności:

- przewidywalność PRNG
- możliwość odtworzenia poprzedniego stanu rejestru na podstawie bieżącego (cofanie przesuwania)
- możliwość odtworzenia możliwych stanów rejestru na podstawie ciągu bitów szyfrujących

Dodatkowo wykorzystam odmianę autoryzacji, która następuje po pierwszej autoryzacji. Różni się ona tym, że pierwszy challenge karty jest przekazywany już zaszyfrowany.

3 Opis ataku

Na początku musimy wybrać jakiś stały czas pomiędzy kolejnymi autoryzacjami. Będziemy tu korzystać z faktu, że generator liczb pseudolosowych jest przewidywalny i jego wartości zależą od czasu.

Pierwszym krokiem będzie przeprowadzenie normalnej autoryzacji do dowolnego sektora przy pomocy znanego klucza. Następnie po ustalonym (dowolnie, ale precyzyjnie odmierzonym) czasie wykonujemy reautoryzację do tego samego sektora. W odpowiedzi dostajemy challenge karty (wygenerowany z PRNG) zaszyfrowany przy pomocy klucza tego sektora. Na tej podstawie możemy (po odszyfrowaniu) obliczyć ilość cykli PRNG w tym ustalonym czasie.

Następnym krokiem będzie już zbieranie informacji potrzebnych bezpośrednio do złamania kluczy. Dla każdego klucza (sektora i klucza A/B) powtarzamy procedurę:

1. Autoryzujemy się znanym kluczem (zapamiętując challenge karty)
2. Czekamy ustalony czas
3. Próbujemy zautoryzować się nowego sektora
4. Zapisujemy odpowiedź i przerywamy operację

W jej wyniku dostajemy parę challenge, między którymi znamy odległość (wyrażoną w cyklach PRNG), przy czym druga z nich jest zaszyfrowana kluczem, który próbujemy złamać. Ponieważ znamy (przybliżoną) wartość PRNG nie zaszyfrowaną, możemy odtworzyć 32 bity strumienia szyfrującego wygenerowanego przez algorytm Mifare. Ponieważ cały klucz ma 48 bitów, na tej podstawie, przy pomocy Crapto1, jesteśmy w stanie tylko ograniczyć zbiór możliwych kluczy (do 2^16).

Dla każdego klucza powtarzamy całą procedurę i otrzymane dwa zbiory przecinamy. Statystycznie dostajemy przecięcie jednoelementowe, które jest poszukiwanym kluczem.

Z powodu niedokładności pomiaru czasu i ew. zakłóceń PRNG można dodać tolerancję i w poprzednim kroku wyliczyć kilka zestawów dla przypuszczalnych wartości PRNG. W użytej implementacji przyjąłem margines 50 cykli.

4 Analiza

Opisany atak ma niewątpliwą przewagę nad innymi do tej pory opisanymi: jest przeprowadzony bezpośrednio na ka karcie (bez potrzeby posiadania czytnika znającego klucze), oraz wymaga wykonania w sumie około 40 prób autoryzacji. Kosztem tej efektywności jest konieczność posiadania co najmniej jednego znanego klucza, ale zazwyczaj sektor 0 ma ustawiony domyślny klucz (A0A1A2A3A4A5). Ten jeden klucz można też zdobyć przy pomocy zastosowania innego ataku.

5 Implementacja

Przygotowana implementacja składa się z dwóch części: oprogramowania na Proxmark 3 służącego do wykonania części aktywnej ataku (komunikacji z kartą), oraz kilku programów służących do złamania klucza na postawie zebranych danych.

5.1 Użycie części na Proxmark

Załączona implementacja dodaje kilka poleceń. W kontekście tego ataku ich użycie powinno wyglądać następująco:

- `hf 14a mfcalibrate WYBRANY_CZAS` - Będzie to czas (w tyknięciach zegara 25MHz) oczekiwany pomiędzy kolejnymi próbami autoryzacji. Ja używałem wartości rzędu pół sekundy. Jeśli nie podamy czasu, urządzenie spróbuje dobrać czas równy wielokrotności okresu PRNG (przydatne przy innych atakach).

- `hf 14a mfstartauth SEKTOR` - Wybór sektora, do którego będzie druga autoryzacja (czyli atakowany sektor). Dodatkowo, jeśli chcemy żądać użycia klucza B (zamiast domyślnego A), należy dodać 256.
- `hf 14a mfauth SEKTOR KTORY_KLUCZ CHALLENGE KLUCZ` - Właściwa autoryzacja, wraz z następującą próbą drugiej autoryzacji (jeśli wydano wcześniej poprzednie dwa polecenia). Opis parametrów:
  - SEKTOR Sektor do którego znamy klucz
  - KTORY_KLUCZ Klucz do autoryzacji (A - 0, B - 1). Dodatkowo dodaje się tu liczbę powtórzenia całej procedury, pomnożoną przez 2
  - CHALLENGE Challenge, którego ma się spodziewać od karty. Ponieważ PRNG jest przewidywalny, da się określić czas (z pewną dokładnością), w którym można się spodziewać konkretnej wartości. W tym ataku nie jest to potrzebne i należy podać 0
  - KLUCZ Znany klucz do autoryzacji, np a0a1a2a3a4a5

5.2 Użycie części na PC

Spis narzędzi:

- `auth` - generuje odpowiedź czytnika na podstawie danych od karty i klucza
- `dist` - podaje odległość pomiędzy dwoma challenge, przy czym drugi z nich odszyfrowuje
- `mifarecrack2` - generuje listę możliwych kluczy na podstawie dwóch challenge: jawnego i zaszyfrowanego
- `nonce-shift` - wylicza challenge przesunięty o zadaną ilość cykli PRNG
- `process-out.awk` - z wyjścia generowanego przez Proxmark wybiera istotne do ataku wartości
- `crack-full.sh` - na podstawie danych z Proxmark (wybranych przez skrypt powyżej) łamie klucz.
- `dist-test.sh` - skrypt pomagający dobrać odpowiednie przesunięcia challenge

Użycie:

1. Najpierw trzeba wszystkie narzędzia skompilować (make)
2. W skrypcie dist-test.sh wypełniamy ID karty (pierwszy parametr dist) i znany klucz (drugi argument)
3. Po zebraniu danych kalibracyjnych - ponownej autoryzacji do sektora o znanym kluczu - przez Proxmark (hf 14a mfauth ...) przetwarzamy przez `process-out.awk`.
4. Wygenerowane dane podajemy na wejście do `dist-test.sh` i wybieramy "sensowny" zakres różnic. Można też np. pomijać zawsze pierwszą autoryzację, jeśli występują tam większe zaburzenia.
5. Wybrane wartości ustawiamy w skrypcie `crack-full.sh` (razem z ID karty)
6. Ponownie zbieramy dane przy pomocy Proxmark - tym razem już do docelowych sektorów - i przetwarzamy przez `process-out.awk`.
7. Te dane przekazujemy do skryptu `crack-full.sh`, który przeprowadzi właściwy atak.

Uwaga: `crack-full.sh` generuje pliki tymczasowe (o konfigurowalnej nazwie), które mogą wymagać znacznej ilości wolnego miejsca (nawet rzędu kilku gigabajtów).

5.3 Załączniki

Implementacja jest do pobrania z `http://students.mimuw.edu.pl/~marmarek/mifare/`.

Literatura

[1] Flavio D. Garcia, Gerhard de Koning Gans, Peter van Rossum Ruben Muijrers, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling mifare classic. 2008.

[2] Kyle E. Penri-Williams. Implementing an rfid 'mifare classic' attack, 2009.