Sprawozdanie z projektu `tracks_hunter.py`

Jakub Aszyk
Indeks: 151841

9 czerwca 2026

1 Cel projektu

Projekt polega na przygotowaniu skryptu do odczytu danych EMV z karty testowej przez czytnik PC/SC. Końcowym programem jest `tracks_hunter.py`. Skrypt szuka danych Track 1 i Track 2, zapisuje wynik do czytelnego JSON-a i pozwala szybko testować kolejne karty.

Program nie czyta fizycznego paska magnetycznego. Korzysta z danych udostępnionych przez chip EMV: aplikacji płatniczych, rekordów AFL, tagów TLV oraz odpowiedzi APDU.

2 Zakres działania

Skrypt wykonuje pełny przebieg od wyboru aplikacji do zapisu raportu:

1. łączy się z kartą przez PC/SC,
2. wykrywa aplikacje płatnicze i sprawdza znane AID-y,
3. wysyła SELECT, GET PROCESSING OPTIONS, READ RECORD i GET DATA,
4. analizuje odpowiedzi TLV,
5. szuka Track 1 w tagu `56`, w surowych danych oraz przez rekonstrukcję z tagów EMV,
6. odczytuje Track 2 z tagów `57` i `9F6B`,
7. zapisuje wynik do `tracks_output.json`.

Najważniejsze grupy tagów w kodzie:

- `TRACK1_TAGS`: surowy Track 1, czyli tag `56`,
- `TRACK1_RELATED_TAGS`: dane pomocnicze Track 1, głównie `9F1F`,
- `TRACK2_TAGS`: Track 2 Equivalent Data, czyli `57` i `9F6B`,
- `SUPPORTING_TAGS`: dane pomocnicze, np. PAN, nazwa posiadacza i daty.

3 Najważniejsze elementy skryptu

Główna logika zbierania wyników znajduje się w klasie `HitStore`. Klasa zapisuje trafienia, usuwa duplikaty i zbiera fragmenty potrzebne do odtworzenia Track 1. Jeżeli ten sam tag zostanie znaleziony kilka razy, skrypt nie powiela rekordu, tylko dopisuje kolejne lokalizacje w polu `also_seen`.

Track 1 może pojawić się na trzy sposoby:

- jako surowy tag `56`,
- jako tekst pasujący do wzorca Track 1 w surowej odpowiedzi,
- jako wynik rekonstrukcji z tagów EMV.

Rekonstrukcja działa wtedy, gdy karta udostępnia potrzebne elementy:

- PAN z tagu `5A` albo `57`,
- datę ważności i service code z tagu `57` lub `9F6B`,
- nazwę posiadacza z tagu `5F20`,
- dane dyskrecjonalne Track 1 z tagu `9F1F`.

Z tych pól skrypt buduje Track 1 w formacie:

```
%B<PAN>^<NAME>^<YYMM><SERVICE_CODE><TRACK1_DISCRETIONARY>?
```

Track 2 jest prostszy. Skrypt zapisuje go jako `track2_equivalent`, gdy znajdzie tag `57` albo `9F6B`.

4 Opis działania programu

Program zaczyna pracę w funkcji `run_once()` albo `run_loop()`. Pierwsza wykonuje jeden odczyt, druga pozwala testować kolejne karty. W obu przypadkach właściwe skanowanie odbywa się w `scan_card()`. Ta funkcja otwiera połączenie z czytnikiem przez `main.PcscCard`, zapamiętuje nazwę czytnika, wykrywa aplikacje i tworzy obiekt `HitStore` na wyniki.

Wykrywanie aplikacji wykonuje `discover_apps()`. Najpierw korzysta ono z mechanizmu dostępnego w `main.py`, a potem sprawdza dodatkową listę AID-ów zapisaną w `tracks_hunter.py`. Dzięki temu skrypt nie kończy pracy na pierwszej oczywistej aplikacji, tylko próbuje także popularne warianty Visa, Mastercard i kilku innych systemów.

Dla każdej znalezionej aplikacji uruchamiana jest funkcja `scan_app()`. Skrypt wykonuje SELECT aplikacji, odczytuje FCI i sprawdza, czy już tam nie ma interesujących tagów. Następnie uruchamia `direct_get_data_probe()`, czyli bezpośrednie próby GET DATA dla tagów `56`, `9F1F`, `57`, `9F6B` i `5A`. To jest dodatkowa ścieżka, bo niektóre karty mogą zwrócić dane poza rekordami AFL.

Kolejnym etapem jest GET PROCESSING OPTIONS. Funkcja `get_processing_options_with_profile()` buduje dane wejściowe na podstawie PDOL i jednego z profili terminala. Skrypt próbuje kilka profili, np. `default`, `zero`, `contactless-default`, `contactless-online` i `magstripe-mode`. Po poprawnej odpowiedzi GPO program wyciąga AFL i czyta rekordy funkcją `read_afl_records()`.

Jeżeli włączony jest sweep, funkcja `sweep_records()` sprawdza również rekordy poza AFL. Przechodzi po kolejnych SFI i numerach rekordów, a odpowiedzi ze statusem `9000` przekazuje do analizy. To zwiększa szansę znalezienia danych zapisanych w niestandardowym miejscu.

Każda odpowiedź przechodzi przez `inspect_response()`. Jeżeli odpowiedź jest TLV, skrypt analizuje wszystkie tagi. Jeżeli nie jest TLV, sprawdza jeszcze surowe bajty wyrażeniem regularnym dla Track 1. Na końcu `group_hits()` dzieli trafienia na sekcje `track1`, `track2`, `track1_related` i `supporting`, a `write_json()` zapisuje raport.

5 Przebieg skanowania

Program zaczyna od sparsowania argumentów. Najczęściej używane opcje to:

- `-reader`: wybór czytnika po fragmencie nazwy,
- `-json`: ścieżka do pliku wynikowego,
- `-redact-json`: maskowanie danych wrażliwych,
- `-profile-count`: liczba profili GPO,
- `-sweep`: szersze sprawdzanie rekordów,
- `-verbose`: diagnostyka APDU.

Po połączeniu z kartą skrypt wykrywa aplikacje. Najpierw używa standardowego wykrywania z modułu `main`, a później próbuje dodatkowych AID-ów wpisanych w `tracks_hunter.py`. Dla każdej aplikacji wykonuje SELECT i analizuje FCI.

Następnie skrypt próbuje kilka profili GPO: `default`, `zero`, `contactless-default`, `contactless-online` i `magstripe-mode`. Różnią się one wartościami podawanymi w PDOL. Dzięki temu karta jest sprawdzana w kilku wariantach zachowania terminala.

Po GPO skrypt odczytuje rekordy wskazane przez AFL. Jeżeli włączony jest sweep, sprawdza także rekordy SFI poza AFL. To pozwala znaleźć dane zapisane w mniej typowych miejscach.

Każda odpowiedź jest parsowana jako TLV. Skrypt sprawdza tagi Track 1, Track 2, tagi pomocnicze oraz wzorzec tekstowy Track 1 w surowych bajtach.

6 Raport JSON

Wynik zapisywany jest domyślnie do pliku `tracks_output.json`. Raport ma kilka głównych sekcji:

- `apps`: znalezione aplikacje płatnicze,
- `card_hint`: skrócony opis karty, np. zamaskowany PAN i data ważności,
- `track1`: wpisy Track 1, w tym surowe i rekonstruowane,
- `track2`: wpisy Track 2,
- `track1_related`: np. dane z tagu `9F1F`,
- `supporting`: opcjonalne tagi pomocnicze,
- `diagnostics`: statusy APDU, jeśli użyto `-verbose`.

Przykładowe podsumowanie z konsoli:

```
Track1 FOUND; Track2 FOUND; apps=1 track1=1 raw=0
reconstructed=1 track2=1 9F1F=1
```

Oznacza to, że skrypt znalazł Track 1 i Track 2. W tym przypadku Track 1 nie pochodził z surowego tagu `56`. Został odtworzony z tagów EMV, ponieważ karta wystawiła `9F1F`.

Po użyciu `-redact-json` skrypt maskuje PAN, usuwa surowe wartości heksadecymalne i ukrywa dane dyskrecjonalne. Jest to przydatne, gdy raport ma być przekazany dalej albo dołączony do dokumentacji.

Wynik dla testowanej karty

Dla testowanej karty skrypt wykrył aplikację Visa Electron z AID:

```
A0000000032010
```

Wynik skanowania:

- Track 1 został znaleziony jako `reconstructed_track1`,
- surowy Track 1 w tagu `56` nie był dostępny,
- Track 2 został znaleziony w tagu `57`,
- tag `9F1F` dostarczył dane dyskrecjonalne Track 1.

Dodatkowo z karty można było odczytać m.in. datę ważności, service code, nazwę posiadacza, kraj emitenta, walutę aplikacji, CVM list, Application Usage Control oraz dane związane z offline authentication.

Historia rozmowy zapisana w JSON-ie

Na końcu pracy powstał plik `conversation_history.json`. Zapisano w nim widoczną historię rozmowy, skróty decyzji, wywołania komend, edycje plików i artefakty wygenerowane w projekcie. Nie zapisano prywatnego ukrytego chain-of-thought. Zamiast tego w pliku znajdują się bezpieczne streszczenia działań.

Najważniejsze etapy rozmowy:

1. Najpierw sprawdzono istniejący `main.py`. Początkowy odczyt pokazał Track 2, ale nie pokazał surowego Track 1.
2. Następnie utworzono `track1_hunter.py`, czyli skrypt do intensywniejszego szukania Track 1.
3. Po głębszym skanie ustalono, że karta nie zwraca tagu `56`, ale udostępnia elementy pozwalające zrekonstruować Track 1.
4. Skrypt rozbudowano do wersji końcowej `tracks_hunter.py`, dodając Track 2 i ładny JSON.
5. Utworzono sprawozdanie w Markdownu, przepisano je na LaTeX i zbudowano PDF.
6. Dodano skill `humanize-writing`, a następnie użyto go do poprawienia stylu dokumentu.

W pliku `conversation_history.json` są też sekcje `command_invocations`, `file_edits`, `generated_or_updated_artifacts` oraz `important_results`. Dzięki temu można prześledzić, jakie komendy zostały wykonane i jakie pliki powstały podczas pracy.

Podsumowanie

Efektem projektu jest skrypt `tracks_hunter.py`. Program odczytuje dane EMV przez PC/SC, szuka Track 1 i Track 2, potrafi odtworzyć Track 1 z tagów EMV i zapisuje wynik w czytelnym JSON-ie. Dla testowanej karty najważniejszy wynik był taki, że Track 1 nie wystąpił jako tag `56`, ale dało się go poprawnie złożyć z danych EMV.