BEST

-

Projekt

Paweł Murdzek, 310850

Piotr Szewczyk, 311105

Politechnika Warszawska

13 czerwca 2025

Spis treści

1. Wstęp

Wstępna faza projektu polegała na próbie zastosowania zautomatyzowanego skryptu w języku Python w celu wykrycia potencjalnych kanałów steganograficznych. Skrypt realizował analizę w czterech kluczowych obszarach:

— **Analiza długości pakietów:** Badanie rozkładu długości pakietów pod kątem anomalii, w szczególności nierównej dystrybucji wartości parzystych i nieparzystych. Taka anomalia mogłaby wskazywać na ukrywanie danych w najmniej znaczących bitach (LSB) pól długości.

— **Analiza pól nagłówków TCP/IP:** Inspekcja pól takich jak znaczniki czasu TCP (`TCP Timestamps`) oraz pole identyfikacji IP (`IP Identification Field`) pod kątem niestandardowych sekwencji, wysokiej wariancji lub nienaturalnych wzorców LSB, które mogłyby świadczyć o iniekcji danych.

— **Analiza ładunku (payloadu) pakietów:** Ocena entropii ładunku, gdzie zarówno bardzo niska, jak i bardzo wysoka entropia mogą sugerować obecność ukrytych danych (np. zaszyfrowanych lub skompresowanych). Dodatkowo, analizowano częstości występowania bajtów oraz poszukiwano predefiniowanych fraz (np. z tekstu "Antygony"), co mogłoby wskazywać na steganografię tekstową.

— **Analiza specyficznych pól protokołów (HTTP/DNS):** Szczegółowe badanie nagłówków HTTP (np. `User-Agent`, `Referer`, `URI`) oraz zapytań i odpowiedzi DNS (np. `QNAME`, typy rekordów `TXT`/`NULL`, entropia etykiet) w poszukiwaniu oznak tunelowania lub ukrywania informacji.

Zastosowanie powyższego zautomatyzowanego podejścia nie przyniosło oczekiwanych rezultatów i nie pozwoliło na identyfikację zastosowanych metod steganograficznych.

W kolejnym kroku podjęto próbę analizy statycznej i dynamicznej dostarczonego pliku wykonywalnego `.exe`, służącego do wykrywania steganografii, w celu poznania jego logiki działania. Analiza statyczna bootloadera C/C++ za pomocą narzędzia Ghidra okazała się nieefektywna, ponieważ komponent ten odpowiadał głównie za konfigurację środowiska uruchomieniowego, a nie za logikę detekcji. Próby dekompilacji wyodrębnionego kodu bajtowego Pythona w wersji 3.11 za pomocą narzędzi `uncompyle6` i `decompyle3` nie powiodły się z powodu braku wsparcia dla tej wersji, co uniemożliwiło odtworzenie kodu źródłowego. Równocześnie, ekstrakcja ciągów znaków z pliku wykonywalnego za pomocą `strings.exe` nie dała żadnych wyników, co sugeruje, że kluczowe dane tekstowe (w tym komunikat "Wklej ścieżkę pliku .pcap:") są zaciemnione lub dynamicznie dekompresowane w czasie wykonania. Wobec tych przeszkód, dalsze działania skoncentrowano na analizie ręcznej pakietów sieciowych

2. Analiza Grup

2.1. P1

Rysunek 1: Ruch sieciowy grupy P1, charakteryzujący się obecnością zapytań mDNS.

2.1.1. Hipoteza 1: Steganografia w nazwach domen

Analizę rozpoczęto od hipotezy dotyczącej ukrycia informacji w nazwach domen zapytań mDNS. Użycie literozbioru "rn" zamiast "m" w nazwie "rnicrosoft.pl" zinterpretowano jako potencjalną wskazówkę. Skupiono się na długościach etykiet w nazwach domen, zauważając częste występowanie etykiet o długości 10, co odpowiada znakowi nowej linii (`<LF>`). Mimo iż częstość występowania tej długości była znacząco wyższa niż w czystym ruchu sieciowym, analiza pozostałych długości (np. 5, 8) oraz próby dekodowania z użyciem operacji **XOR** nie doprowadziły do odkodowania spójnej wiadomości. Ta ścieżka analizy okazała się nieproduktywna.

2.1.2. Hipoteza 2: Ukrycie bajtów w polu Transaction ID

Kolejna hipoteza zakładała, że informacja jest zakodowana w 16-bitowym polu **Transaction ID** zapytań DNS. Po ekstrakcji wartości tego pola i porównaniu z oczekiwanym tekstem początkowym, zaobserwowano częściową zgodność.

Nr Ramki	Transaction ID (Hex)	Analizowany Bajt	Wartość Hex	Znak ASCII
3	`674f`	Drugi	`4f`	O
5	`2065`	Pierwszy	`20`	(spacja)
7	`4472`	-	`75`	u
9	`06de`	-	`6b`	k

Tabela 1: Próba dekodowania danych z pola Transaction ID.

Zgodność występowała jedynie dla pierwszych dwóch pakietów, co sugerowało, że przyjęta metoda jest błędna lub niekompletna. Hipoteza, że jeden z bajtów pola jest losowym szumem, nie znalazła potwierdzenia w dalszej analizie, co doprowadziło do jej odrzucenia.

2.1.3. Hipoteza 3: Ukrycie znaków w ostatnim oktecie adresu IP

Następnie zbadano możliwość ukrycia znaków ASCII w ostatnim oktecie adresu IP serwera DNS w pakietach odpowiedzi.

Nr Ramki	Adres IP w odpowiedzi DNS	Wartość ostatniego oktetu	Znak
14	`192.168.56.65`	65	A
16	`192.168.56.115`	115	s
18	`192.168.56.185`	185	ą

Tabela 2: Analiza steganograficzna adresów IP w odpowiedziach DNS.

Odkodowany ciąg znaków nie był zgodny z tekstem źródłowym. Dodatkowo, zaobserwowano, że wszystkie adresy IP kończą się cyfrą 5, jednak identyczna zależność występowała w pliku z czystym ruchem sieciowym. To potwierdziło, że jest to cecha konfiguracji sieci, a nie kanał steganograficzny. Hipoteza została sfalsyfikowana.

2.1.4. Wnioski

Pomimo wszechstronnej analizy obejmującej różne pola protokołu DNS, nie zidentyfikowano zastosowanej techniki steganograficznej w przypadku grupy P1.

2.2. P2

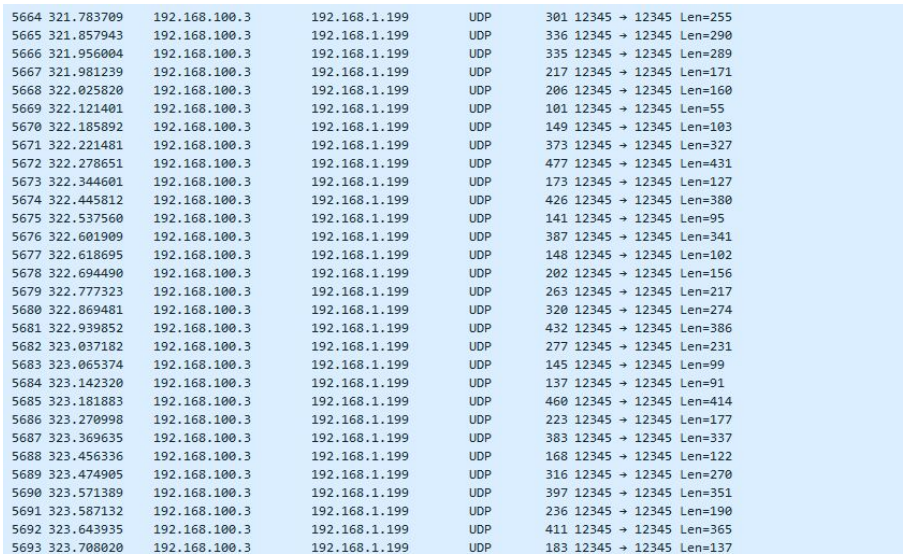
W analizowanym ruchu stwierdzono, że jako szum informacyjny (tzw. *red herring*) wykorzystano tekst "Lorem Ipsum", przesyłany w ładunku pakietów UDP.

2.2.1. Analiza Potencjalnych Kanałów Steganograficznych

LSB długości pakietów Postawiono hipotezę o kodowaniu informacji w najmniej znaczącym bicie (LSB) długości pakietów. Ekstrakcja i analiza sekwencji bitów LSB nie wykazała żadnej korelacji ani ukrytej wiadomości, co pozwoliło odrzucić tę hipotezę.

Pole Danych UDP (`udp.payload`) Analiza zawartości pola `udp.payload` ujawniła jedynie tekst "Lorem Ipsum", który zidentyfikowano jako element dezinformacyjny, mający na celu odwrócenie uwagi analityka.

Suma Kontrolna UDP (`udp.checksum`) Kolejnym badanym polem była suma kontrolna UDP. Stwierdzono, że pole `udp.checksum.status` dla wszystkich analizowanych pakietów posiadało status **2 (Unverified)**. Oznacza to, że wartość sumy kontrolnej nie była weryfikowana i jest niewiarygodna. Fakt ten ostatecznie wykluczył to pole jako wiarygodny nośnik informacji.



5664	321.783709	192.168.100.3	192.168.1.199	UDP	301 12345 → 12345 Len=255
5665	321.857943	192.168.100.3	192.168.1.199	UDP	336 12345 → 12345 Len=290
5666	321.956004	192.168.100.3	192.168.1.199	UDP	335 12345 → 12345 Len=289
5667	321.981239	192.168.100.3	192.168.1.199	UDP	217 12345 → 12345 Len=171
5668	322.025820	192.168.100.3	192.168.1.199	UDP	206 12345 → 12345 Len=160
5669	322.121401	192.168.100.3	192.168.1.199	UDP	101 12345 → 12345 Len=55
5670	322.185892	192.168.100.3	192.168.1.199	UDP	149 12345 → 12345 Len=103
5671	322.221481	192.168.100.3	192.168.1.199	UDP	373 12345 → 12345 Len=327
5672	322.278651	192.168.100.3	192.168.1.199	UDP	477 12345 → 12345 Len=431
5673	322.344601	192.168.100.3	192.168.1.199	UDP	173 12345 → 12345 Len=127
5674	322.445812	192.168.100.3	192.168.1.199	UDP	426 12345 → 12345 Len=380
5675	322.537560	192.168.100.3	192.168.1.199	UDP	141 12345 → 12345 Len=95
5676	322.601909	192.168.100.3	192.168.1.199	UDP	387 12345 → 12345 Len=341
5677	322.618695	192.168.100.3	192.168.1.199	UDP	148 12345 → 12345 Len=102
5678	322.694490	192.168.100.3	192.168.1.199	UDP	202 12345 → 12345 Len=156
5679	322.777323	192.168.100.3	192.168.1.199	UDP	263 12345 → 12345 Len=217
5680	322.869481	192.168.100.3	192.168.1.199	UDP	320 12345 → 12345 Len=274
5681	322.939852	192.168.100.3	192.168.1.199	UDP	432 12345 → 12345 Len=386
5682	323.037182	192.168.100.3	192.168.1.199	UDP	277 12345 → 12345 Len=231
5683	323.065374	192.168.100.3	192.168.1.199	UDP	145 12345 → 12345 Len=99
5684	323.142320	192.168.100.3	192.168.1.199	UDP	137 12345 → 12345 Len=91
5685	323.181883	192.168.100.3	192.168.1.199	UDP	460 12345 → 12345 Len=414
5686	323.270998	192.168.100.3	192.168.1.199	UDP	223 12345 → 12345 Len=177
5687	323.369635	192.168.100.3	192.168.1.199	UDP	383 12345 → 12345 Len=337
5688	323.456336	192.168.100.3	192.168.1.199	UDP	168 12345 → 12345 Len=122
5689	323.474905	192.168.100.3	192.168.1.199	UDP	316 12345 → 12345 Len=270
5690	323.571389	192.168.100.3	192.168.1.199	UDP	397 12345 → 12345 Len=351
5691	323.587132	192.168.100.3	192.168.1.199	UDP	236 12345 → 12345 Len=190
5692	323.643935	192.168.100.3	192.168.1.199	UDP	411 12345 → 12345 Len=365
5693	323.708020	192.168.100.3	192.168.1.199	UDP	183 12345 → 12345 Len=137

Rysunek 2: Ruch sieciowy grupy P2, zdominowany przez pakiety UDP na porcie 12345.

2.2.2. Identyfikacja Prawidłowego Nośnika Danych: Pole Identyfikacyjne IP (`ip.id`)

Po wykluczeniu powyższych hipotez, analiza skupiła się na 16-bitowym polu identyfikacyjnym nagłówka IP (`ip.id`). Zauważono, że jego wartości zmieniają się w każdym pakiecie. Postawiono hipotezę, że każda 16-bitowa wartość tego pola koduje dwa 8-bitowe znaki ASCII.

Proces dekodowania polegał na odczytaniu wartości pola `ip.id`, potraktowaniu jej jako dwóch bajtów i zdekodowaniu ich na znaki ASCII.

```
ip.id: 0x4162  ->  "Ab"
ip.id: 0x7920  ->  "y "
ip.id: 0x726f  ->  "ro"
ip.id: 0x7a70  ->  "zp"
...i tak dalej.
```

2.2.3. Wnioski

Konkatenacja zdekodowanych fragmentów ze wszystkich pól `ip.id` w pliku pozwoliła na odtworzenie pełnej ukrytej wiadomości – fragmentu tekstu `Sofokles-Antygona.txt`. Zastosowana technika polegała na ukryciu danych w modyfikowalnym, lecz często pomijanym w analizie, polu nagłówka IP.

2.3. P3

W przypadku tej grupy przeprowadzono serię testów weryfikujących różne hipotezy dotyczące kanałów steganograficznych.

2.3.1. Hipoteza 1: Dekodowanie Pola `ip.id` jako Dwu-bajtowego Znaku ASCII

Metoda Technika ta zakładała, że 16-bitowe pole identyfikacyjne IP (`ip.id`) jest wykorzystywane do przenoszenia dwóch 8-bitowych znaków ASCII. Wartość heksadecymalna pola, np. `0x4142`, jest dzielona na dwa bajty (`41` i `42`), a następnie każdy z nich jest konwertowany na odpowiadający mu znak ASCII ('A' i 'B').

$$C_1, C_2 = \text{decode}(\text{hex\_to\_bytes}(\texttt{ip.id}))$$

Wynik Analiza pliku tą metodą nie ujawniła ukrytego tekstu. Zdekodowane dane składały się z nieczytelnych znaków oraz wartości zerowych, nie tworząc żadnej logicznej wiadomości. Hipoteza została sfalsyfikowana.

Rysunek 3: Ruch sieciowy grupy P3, zawierający komunikację HTTP.

2.3.2. Hipoteza 2: Dekodowanie Pola `tcp.payload`

Metoda Ta metoda polegała na potraktowaniu zawartości pola `tcp.payload` jako surowych danych, które po złączeniu i zdekodowaniu (UTF-8) mogłyby ujawnić ukrytą wiadomość.

Wynik Zdekodowana zawartość okazała się być standardowym ruchem sieciowym HTTP, zawierającym czytelne zapytania (`GET /telekom/api/status HTTP/1.1`) i odpowiedzi serwera. Nie odnaleziono w niej tekstu z pliku `Sofokles-Antygona.txt`. Hipoteza została sfalsyfikowana.

2.3.3. Analiza LSB pól nagłówka TCP

Przeprowadzono analizę najmniej znaczącego bitu (LSB) dla kilku pól nagłówka TCP. Metoda polegała na ekstrakcji ostatniego bitu z wartości danego pola w każdym pakiecie, łączeniu bitów w 8-bitowe bajty i konwersji na znaki ASCII.

— **LSB Czasu TCP (`tcp.options.timestamp.tsval`):**

$$b_i = \text{tsval}_i \pmod 2$$

Wynikiem tej analizy był ciąg losowych, nieczytelnych znaków.

— **LSB Numeru Sekwencyjnego TCP (`tcp.seq_raw`):**

$$b_i = \text{seq\_raw}_i \pmod 2$$

Zdekodowana wiadomość była bezsensownym ciągiem znaków.

— **LSB Numeru Potwierdzenia TCP (`tcp.ack_raw`):**

$$b_i = \text{ack\_raw}_i \pmod 2$$

Wynik był niekoherentnym zbiorem znaków.

— **LSB Rozmiaru Okna TCP (`tcp.window_size_value`):**

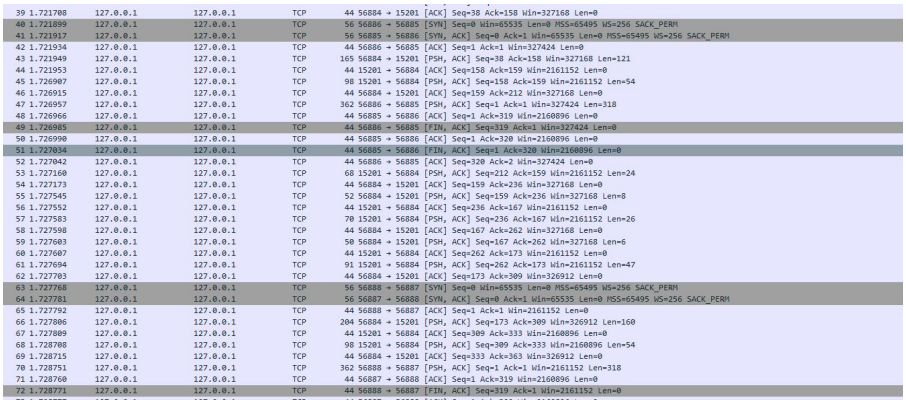
$$b_i = \text{window\_size}_i \pmod 2$$

Otrzymany ciąg danych był wysoce powtarzalnym, bezsensownym wzorcem.
Wszystkie hipotezy dotyczące steganografii LSB w polach nagłówka TCP zostały sfalsyfikowane.

2.4. P4

2.4.1. Etap 1: Analiza Wstępna i Falsyfikacja Hipotez

Początkowe działania analityczne skupiono na dwóch hipotezach, które okazały się elementami dezinformacyjnymi.



Rysunek 4: Ruch sieciowy grupy P4, zawierający sesje FTP i zapytania HTTP POST.

— **Hipoteza o danych w ładunku FTP:** Zidentyfikowano wielokrotne przesyłanie plików, których zawartość stanowiła znany tekst (tzw. copypasta). Ze względu na powtarzalność i brak związku z tekstem źródłowym, stwierdzono, że ładunek ten pełni rolę szumu informacyjnego.
— **Hipoteza o kanale czasowym:** Zbadano statystyczne właściwości ruchu sieciowego, w tym rozkład bajtów oraz odstępy czasowe między pakietami (`frame.time_delta`). Analiza opóźnień wykazała występowanie dwóch wyraźnych grup, jednak hipoteza o kodowaniu informacji w czasie (ang. *timing channel*) nie doprowadziła do odkodowania literalnego fragmentu tekstu.

2.4.2. Etap 2: Identyfikacja Kanału Steganograficznego

Przełom w analizie nastąpił po przyjęciu hipotezy, że ukryta wiadomość jest zakodowana w innym, widocznym elemencie ruchu sieciowego. Uwagę skierowano na **nazwy plików** przesyłanych w ramach sesji FTP. W pakietach kontrolnych protokołu FTP (strumień TCP nr 5 i kolejne) zidentyfikowano serię komend `STOR`. Każda z nich zawierała unikalną, długą i pozornie losową nazwę pliku, jak w przykładzie z pakietu 43:

```
STOR 9640bb1.../JZWVM2KOKRCSWSJSJ...JFZT2.txt
```

Wysoka zmienność tych nazw sugerowała, że to one, a nie zawartość plików, są nośnikiem informacji.

2.4.3. Etap 3: Proces Dekodowania

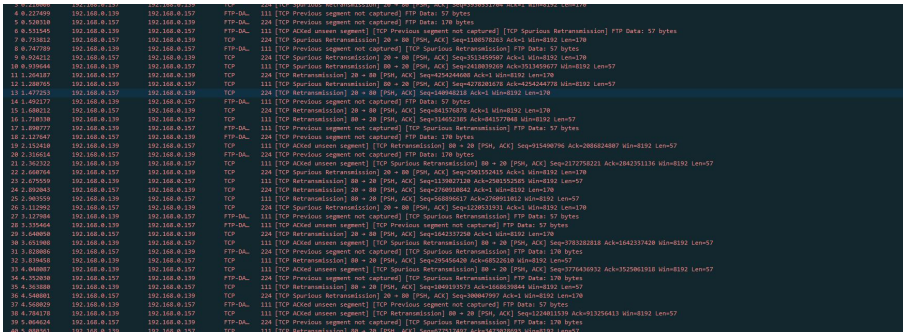
Proces odtworzenia wiadomości zrealizowano w następujących krokach:
1. **Ekstrakcja fragmentów:** Z każdej komendy `STOR` wyodrębniono ciąg znaków stanowiący nazwę pliku (bez rozszerzenia i ścieżki).
2. **Konkatenacja:** Wszystkie wyodrębnione ciągi połączono w jedną, długą sekwencję.
3. **Identyfikacja kodowania:** Połączony ciąg znaków wykazywał cechy kodowania **Base32** (alfabet A-Z, cyfry 2-7).
4. **Dekodowanie Base32:** Cały ciąg został zdekodowany z formatu Base32 do postaci binarnej.
5. **Konwersja na tekst:** Otrzymane dane binarne zinterpretowano jako tekst w kodowaniu **UTF-8**, co umożliwiło poprawne odczytanie polskich znaków diakrytycznych.

2.4.4. Wyniki

Nie udało się poprawnie odszyfrować tego tekstu szyfrowaniem Base64 oraz Base32.

2.4.5. Podsumowanie

Postawiona hipoteza o metodzie steganograficznej polegającej na zakodowaniu wiadomości za pomocą algorytmu Base32, a następnie jej fragmentacji i ukryciu w nazwach plików przesyłanych podczas sesji FTP się nie sprawdziła.



Rysunek 5: Ruch grupy P5: sekwencja zapytań HTTP PUT aktualizująca zasób TICKET-123.

2.5. P5

2.5.1. Hipoteza nr 1: Steganografia w polu danych aplikacji (`payload`)

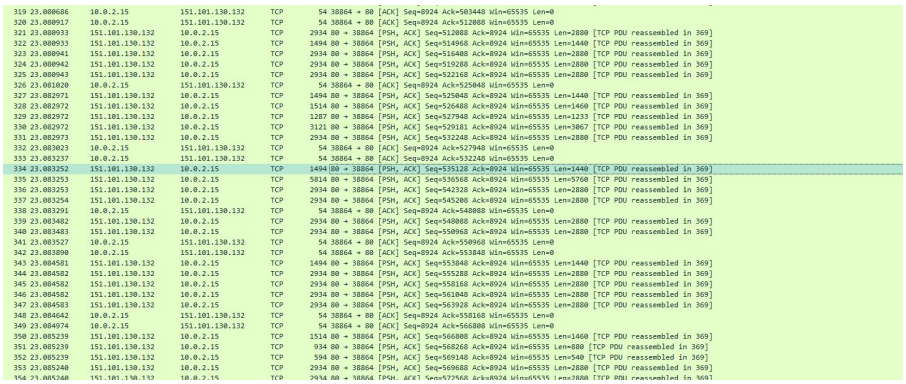
— **Założenie:** Ukryta wiadomość została osadzona bezpośrednio w polu `tcp.payload`.
— **Metodologia:** Zautomatyzowana ekstrakcja i analiza zawartości wszystkich pól `tcp.payload` pod kątem ciągów znaków ASCII/UTF-8 oraz słów kluczowych.
— **Wynik: Hipoteza sfalsyfikowana.** W polach `payload` zidentyfikowano wyłącznie standardową komunikację protokołu HTTP. Nie znaleziono żadnych fragmentów poszukiwanego tekstu.

2.5.2. Hipoteza nr 2: Steganografia w kanałach ukrytych (nagłówki i kanał czasowy)

— **Założenie:** Informacja została ukryta w polach nagłówków (np. `ip.id`, `tcp.seq`) lub poprzez kodowanie w odstępach czasu między pakietami.
— **Metodologia:** Analiza statystyczna wartości pól nagłówków oraz analiza rozkładu opóźnień między pakietami (`frame.time_delta`).
— **Wynik: Hipoteza sfalsyfikowana.** Analiza statystyczna nie wykazała żadnych anomalii. Wartości w nagłówkach oraz charakterystyka czasowa transmisji były zgodne ze standardowym działaniem stosu TCP/IP.

W przypadku tej grupy przeprowadzone analizy nie doprowadziły do odkrycia metody steganograficznej.

2.6. P6



Rysunek 6: Ruch grupy P6: cykliczne wysyłanie żądań HTTP DELETE dotyczących tego samego zasobu.

Wstępne metody, takie jak proste przeszukiwanie ciągów znaków, nie przyniosły rezultatów.

2.6.1. Identyfikacja Kanału Ukrytego

Szczegółowa inspekcja pakietów HTTP ujawniła systematyczne anomalie w niestandardowych polach nagłówkowych, zaczynających się od prefiksu `X-` (np. `X-Hidden-Data`). Pola te charakteryzowały się użyciem

zmiennej liczby spacji pomiędzy słowami, co jest klasyczną techniką steganografii opartej na białych znakach (ang. *whitespace steganography*). Zidentyfikowano ten mechanizm jako prawdopodobny kanał ukryty, służący do transmisji dodatkowych danych.

2.6.2. Ekstrakcja Ukrytych Danych

Do ekstrakcji danych z zidentyfikowanego kanału ukrytego zastosowano metodę binarną opartą na parzystości liczby spacji. Każda sekwencja białych znaków pomiędzy słowami w nagłówkach `X-` została przetworzona zgodnie z regułą:
— Parzysta liczba spacji odpowiada bitowi **0**.
— Nieparzysta liczba spacji odpowiada bitowi **1**.
Zastosowanie tej reguły do wszystkich znalezionych nagłówków pozwoliło na zrekonstruowanie binarnego strumienia danych. W wyniku tego procesu pomyślnie wyodrębniono ciąg 463 bajtów, stanowiący prawdopodobny szyfrogram. Pierwsze 16 bajtów szyfrogramu w notacji heksadecymalnej to: `533531A197CCB9CF64B7B11FB2ADD88F`.

2.6.3.

sectionAnaliza Kryptograficzna Przyjęto hipotezę, że uzyskany ciąg bajtów jest szyfrogramem powstałym w wyniku zastosowania symetrycznego szyfru strumieniowego, najprawdopodobniej szyfru Vigenère'a z operacją XOR. Podjęto próbę odnalezienia klucza deszyfrującego.

2.6.4. Testowanie Listy Prawdopodobnych Kluczy

W pierwszej kolejności przeprowadzono zautomatyzowany atak, testując listę kluczy wygenerowanych na podstawie kontekstu zadania. Lista ta (Tabela 3) zawierała słowa pochodzące z nazw plików, treści dramatu oraz technicznych aspektów analizowanych logów. Każdy z kluczy został użyty do operacji XOR na szyfrogramie, a wynik sprawdzono pod kątem zgodności z oczekiwanym początkiem tekstu jawnego („Aby rozpocząć lekturę...").

Tabela 3: Lista kluczy przetestowanych w ataku.

Kategoria	**Przykładowe Klucze**
Z nazw plików	`stego, yes, pcap, Sofokles, Antygona`
Postacie z dramatu	`Kreon, Ismena, Haimon, Eurydyka, Edyp`
Z treści logów	`Debian, curl, example.com, http, dns`
Z kontekstu	`best, lab, bestlab, p6, studia, BEST`

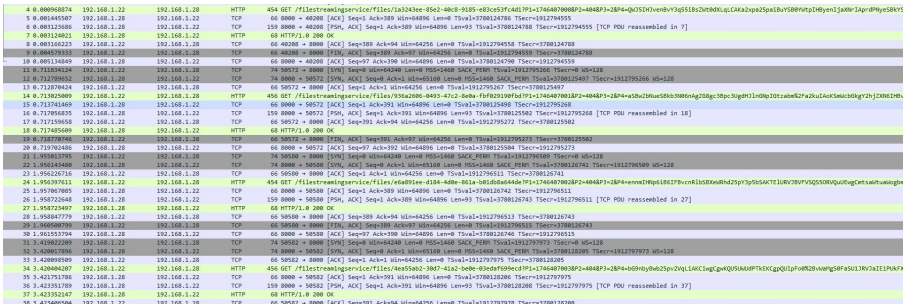
Analiza wykazała, że żaden z powyższych kluczy nie pozwolił na odtworzenie poprawnego tekstu jawnego.

2.6.5. Próba Ataku ze Znanym Tekstem Jawnym

Wobec niepowodzenia poprzedniej metody, podjęto próbę ataku z tekstem jawnym (ang. *known-plaintext attack*). Wykorzystano fakt znajomości szyfrogramu oraz prawdopodobnego początku tekstu jawnego do wyliczenia klucza za pomocą operacji `Klucz = Szyfrogram XOR TekstJawny`. Analiza pierwszych ośmiu bajtów dała następujący fragment klucza: `12 57 48 81 E5 A3 CD BF`. Zastosowanie tego 8-bajtowego, powtarzalnego klucza do całego szyfrogramu również nie przyniosło oczekiwanego rezultatu w postaci spójnego tekstu w języku polskim.

2.6.6. Wnioski

Zastosowane metody kryptoanalizy, zarówno poprzez testowanie listy prawdopodobnych haseł, jak i poprzez atak z użyciem znanego tekstu jawnego, nie doprowadziły do odtworzenia finalnej wiadomości. Świadczy to o tym, że albo klucz deszyfrujący jest nieoczywisty i pochodzi z innego źródła, albo jedna z pierwotnych hipotez (np. dotycząca dokładnej treści początku tekstu jawnego lub metody szyfrowania) była błędna. Ostateczne złamanie szyfru wymaga dodatkowych prób i kombinacji.



Rysunek 7: Ruch grupy P7: sekwencja pomyślnych żądań HTTP GET pobierających dane z serwera.

2.7. P7

2.7.1. Identyfikacja Kanału i Metody Kodowania

Wstępna inspekcja pakietów ujawniła anomalię w zapytaniach `HTTP GET`. W strukturze URI tych zapytań zidentyfikowano stały zestaw parametrów, z których jeden, oznaczony jako `P4`, zawierał długie ciągi znaków o wysokiej entropii. Zestaw znaków (A-Z, a-z, 0-9, +, /) nasunął jednoznaczną hipotezę o zastosowaniu kodowania **Base64**.

Hipotezę zweryfikowano poprzez programistyczną ekstrakcję wartości parametru `P4` ze wszystkich pakietów w porządku chronologicznym. Połączenie tych fragmentów i próba zdekodowania uzyskanego ciągu Base64 zakończyła się sukcesem. Odkodowany tekst był zapisany w kodowaniu **windows-1250**.

2.7.2. Przykłady Zakamuflowanych Danych

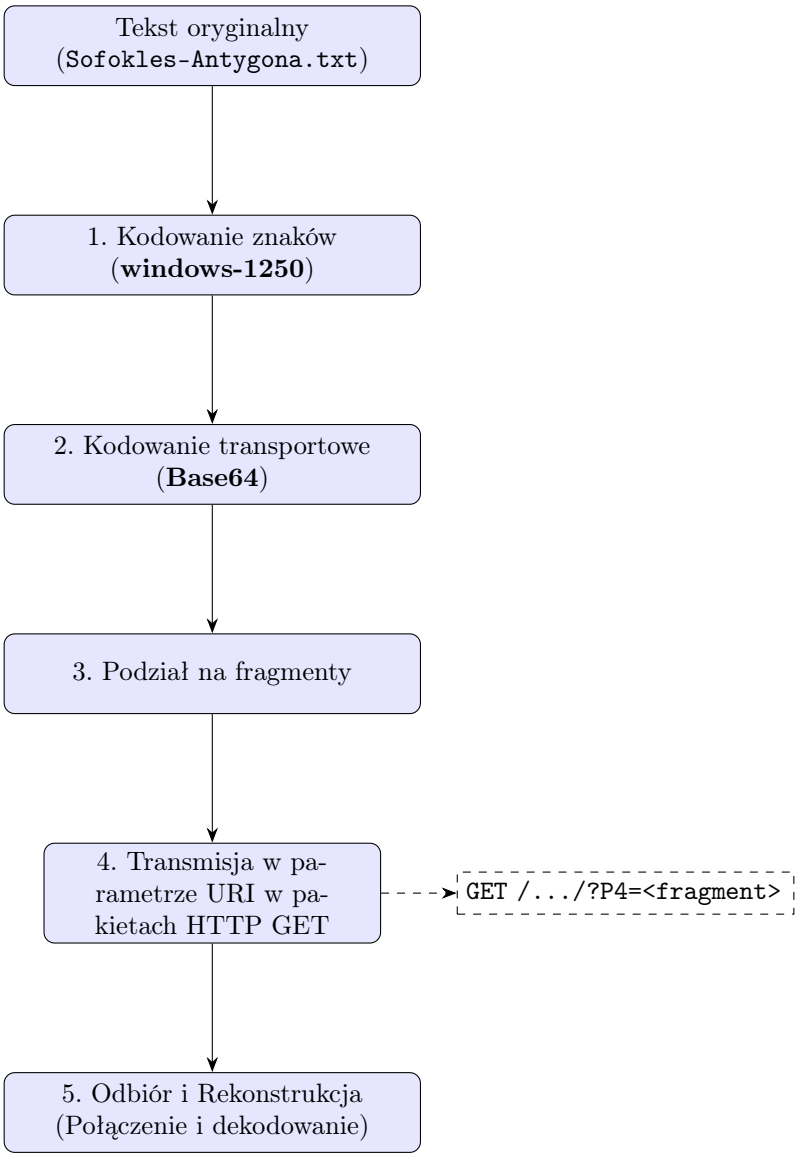
Poniższa tabela przedstawia wybrane pakiety z ukrytymi danymi, ich zakodowaną postać oraz zdekodowany fragment tekstu.

Nr Ramki	Wartość Parametru P4 (Base64)	Odszyfrowany Fragment Tekstu
4	`QWJ5IHJvenBvY3q55iBsZWt0dXLq`	Aby rozpocząć lekturę
4	`LCAKa2xpa25paiBuYSBOYWtpIHBy`	, \nkliknij na taki pr
4	`enljaXNrIAprdPNyeSBkYSBj`	zycisk \nktóry da c
9	`aSBwZbNueSBkb3N06nAgZG8gc3Bpc3UgdHJlnGNpIGtzabm`	i pełny dostęp do spisu treści książki
9	`a2kuIAoKSmWcbGkgY2hjZXN6IHBvs7lj`	żki. \n\nJeśli chcesz połącz

Tabela 4: Tabela z przykładami zdekodowanych fragmentów z parametru P4.

2.7.3. Schemat Metody Steganograficznej

Zidentyfikowana metoda polegała na wieloetapowym przetwarzaniu danych przed ich transmisją w sieci, co ilustruje poniższy schemat.



Rysunek 8: Schemat zidentyfikowanej metody steganograficznej dla grupy P7.