I. Treść zadania.

Jak wiemy, w profesjonalnych sieciach przemysłowych stosowane są różnorodne metody weryfikacji poprawności przesyłanych danych. Dość powszechnie stosowana jest metoda nadmiarowej redundancyjnej sumy kontrolnej (CRC). Metoda ta jest wykorzystywana między innymi w sieciach: MODBUS, PROFIBUS, HART, CAN, Foundation Fieldbus, itd. Istnieje jednak pewne skończone prawdopodobieństwo, że mimo tego że przesyłana ramka komunikacyjna zostanie zniekształcona, to odebrana suma kontrolna będzie poprawna. Taka ramka zostanie poprawnie przyjęta przez urządzenie odbiorcze, co może mieć poważne konsekwencje. Możemy w tym przypadku stwierdzić, że zostało narażone bezpieczeństwo przesyłu danych. Zwykle, obok CRC, systemy komunikacyjne wykorzystują uzupełniające metody weryfikacji poprawności przesyłania danych np. bity parzystości, kontrolę zakresu wartości określonych pól ramki komunikacyjnej itp.

Opis sytuacji:

Pewien haker należący do organizacji Against War zdobył informację, że dynamiczny kod dostępu do tajnego składu broni masowej zagłady pewnej organizacji militarnej przesyłany jest siecią Modbus RTU.

Informacje szczegółowe, które dodatkowo zdobył są następujące:

- jednostka master wysyła kod rozkazem o kodzie 0x10;
- system zamków magazynu jest obsługiwany przez jednostkę slave o stałym adresie 0x0A;
- kod dostępu jest kodem 128 bitowym;
- jednokrotne podanie niewłaściwego kodu do zamka wiąże się z jego automatycznym zniszczeniem;
- poprawność samego kodu jest weryfikowana dodatkowo przez CRC, która dla zmylenia wyznaczana jest tak jak standardowa CRC w specyfikacji sieci Modbus.

Skaner, który został zainstalowany przez przyjaciół hakera tuż przy jednostce master zarejestrował ramkę z nieczytelnym kodem dostępu do składu broni o postaci:

Pole adresu		0x0A
Pole funkcji		0x10
Adres pierwszego rejestru kodu		0x0000
Adres ostatniego rejestru kodu		0x0008
Liczba bajtów pola danych		0x12
Pole danych:	kod	0x????
	kod	0x????
	kod	0x????
	kod	0x????
	kod	0x????
	kod	0x????
	kod	0x????
	kod	0x????
CRC kodu		0xA77C
CRC ramki		0x????

Zadanie 3

a) Odtworzyć kod dostępu do zamka składu broni.

b) Czy taki kod jest tylko jeden? Odpowiedź udowodnić.

II. Rozwiązanie.

W celu uzyskania kodu dostępu do zamka składu broni można skorzystać (na pierwszy rzut oka) z dwóch sposobów. W pierwszym należałoby podłączyć się własnym urządzeniem do sieci Modbus RTU i podszywając się za jednostkę master, zażądać od jednostki slave o adresie 0x0A wydania wartości rejestrów o adresach od 0x0000 do 0x0008. Jednak prawdopodobnie w sieci o takim znaczeniu nie byłoby to możliwe, dlatego ten sposób zostanie pominięty.

Drugim sposobem jest odtworzenie kodu dostępu na podstawie zarejestrowanego CRC kodu (nie ramki) o wartości 0xA77C. Jest ono wyznaczane tak samo jak standardowe CRC. Przyjęto, że zarejestrowana wartość CRC jest w postaci po wykonaniu algorytmu, a nie w postaci w jakiej jest dołączana do wiadomości. Dlatego w celu odtworzenia kodu należy wykonać algorytm wyznaczenia CRC od tyłu. Kolejność działań będzie następująca:

1. Załadowanie do 16-bitowego rejestru o wartości 0xA77C (wartość CRC).
2. Jeśli MSB=1 wtedy przeprowadzić XOR rejestru z wartością 0xA001, jeśli nie przejść do punktu 2.
   (wynika to z tego że w czasie obliczania CRC MSB musi równać się zero po przesunięciu, więc jeśli MSB jest 1, to musiała być przeprowadzona operacja XOR z wartością 0xA001).
3. Przesunięcie rejestru o jeden bit w lewo i wstawienie wartości 1 w miejsce LSB, jeśli w poprzednim punkcie MSB było równe 1. Jeśli MSB było 0, to należy wstawić 0.
4. Powtórzyć punkty 2 i 3 do ośmiu przesunięć.
5. Na podstawie młodszego bajtu rejestru można obliczyć CRC, ale tylko dla wiadomości o rozmiarze do jednego bajta (co jest sprzeczne z zadaniem). Należy to zrobić poprzez XOR z wartością 0xFF.

Nie jest możliwe odwrócenie operacji z algorytmu wyznaczania CRC „XOR młodszego bajtu rejestru z kolejnym bajtem wiadomości", ponieważ mamy tylko dany wynik tej operacji, a nie znamy kolejnego bajtu wiadomości i wartości młodszego bajtu rejestru (przed operacją XOR). Tylko, jeśli wiadomość miała jeden bajt, jest znana wartość młodszego bajtu rejestru (0xFFFF).

Dlatego w ten sposób nie można odtworzyć szukanego kodu dostępu. Dodatkowo można udowodnić, że istnieje wiele wiadomości o długości 128 bitów, dające CRC takie samo jak zarejestrowana wartość CRC kodu.

Skoro XOR jest przeprowadzany na 8 bitach, a każdy bit wyniku (0 lub 1) można uzyskać na 2 sposoby (wynik „0" dają dwa zera lub dwie jedynki; wynik „1" dają „0 i 1" lub „1 i 0" – kolejność ma znaczenie), to ilość sposobów w jaki da się uzyskać wynik operacji XOR z bajtem wiadomości jest równy: $2^8 = 256$. Co teoretycznie dla kodu o długości 128 bitów, czyli 16 bajtów, da liczbę możliwych kodów dających takie samo CRC równą: 256^15 = $1{,}3*10^{36}$. Do obliczenia trzeba było użyć liczby bajtów pomniejszonej o 1, ponieważ przy operacji z pierwszym bajtem znamy początkową wartość rejestru.

Dlatego też odpowiedź na pytania z punktu b brzmi**: istnieje więcej niż jeden kod który można odtworzyć na podstawie CRC**.

W celu udowodnienia istnienia wielu kodów dostępu, dających takie samo CRC jak te które zostało zarejestrowane, został napisany program w języku C++, w którym zostaje generowana liczba (zwiększana o 1 w kolejnych iteracjach) i na jej podstawie zostaje obliczane CRC dla kodu o długości 128 bitów, zawierające wartość tej liczby. Jeśli CRC jest równe 0xA77C, to zostaje wyświetlony kod dostępu w postaci wartości jego kolejnych bajtów (bajt o wartości 0x0A wyświetli się jako A).

Skoro kod dostępu jest w postaci kodu 128 bitowego, to istnieje $2^{128} \approx 3{,}4*10^{38}$ wszystkich możliwych kodów. Odpowiada to takiej ilości liczb całkowitych, jakie trzeba wygenerować. Ma to znaczenie, ponieważ trzeba pamiętać jaka zmienna zostanie użyta do generowania liczb, np. zmienna ***unsigned int*** pozwoli wygenerować liczby tylko do 4 294 967 295.

Do znalezienia przykładowych kodów przyjęto zakres liczb od 0 do 7 999 999. Poniżej przedstawiono uzyskane wyniki (co może też być uznane za odpowiedź na podpunkt a):

```
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5a 79 31
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5b b8 f1
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5c f9 33
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5d 38 f3
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5e 78 f2
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5f b9 32
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 f9 22
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 61 38 e2
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 62 78 e3
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 63 b9 23
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 64 f8 e1
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 65 39 21
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 66 79 20
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 67 b8 e0
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 68 f8 e4
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 69 39 24
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6a 79 25
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6b b8 e5
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6c f9 27
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6d 38 e7
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6e 78 e6
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6f b9 26
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 70 f8 ee
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 71 39 2e
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 72 79 2f
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 73 b8 ef
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 74 f9 2d
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 75 38 ed
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 76 78 ec
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 77 b9 2c
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 78 f9 28
KOD: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 79 38 e8
Dla 8000000 iteracji, otrzymano: 122 kodow.

Process returned 0 (0x0)   execution time : 2.970 s
Press any key to continue.
```

Rysunek 1 – Wynik z wykonanego programu.

Jak widać uzyskano 122 kody dające CRC o wartości 0xA77C, przy niezmiennych 13 bajtach wiadomości równych 0.

Należy też zweryfikować uzyskane wyniki, dlatego dla trzech uzyskanych kodów zostanie obliczone CRC za pomocą kalkulatora CRC z następującej strony internetowej: https://www.lammertbies.nl/comm/info/crc-calculation .

Wybrane kody są następujące:

- 0000 0000 0000 0000 0000 0000 005f b932
- 0000 0000 0000 0000 0000 0000 0075 38ed
- 0000 0000 0000 0000 0000 0000 0079 38e8

Uzyskane wartości CRC (interesuje nas pozycja CRC-16 (Modbus)):

"00000000000000000000000005fb932" (hex)	
1 byte checksum	**74**
CRC-16	**0x57C2**
CRC-16 (Modbus)	**0xA77C**
CRC-16 (Sick)	**0x3C07**
CRC-CCITT (XModem)	**0xC07B**
CRC-CCITT (0xFFFF)	**0xAA71**
CRC-CCITT (0x1D0F)	**0x85D0**
CRC-CCITT (Kermit)	**0xC336**
CRC-DNP	**0x0D20**
CRC-32	**0x768B66A9**

Rysunek 2 – Wartość CRC dla pierwszego kodu.

"00000000000000000000000007538ed" (hex)	
1 byte checksum	**154**
CRC-16	**0x57C2**
CRC-16 (Modbus)	**0xA77C**
CRC-16 (Sick)	**0x49D3**
CRC-CCITT (XModem)	**0x9347**
CRC-CCITT (0xFFFF)	**0xF94D**
CRC-CCITT (0x1D0F)	**0xD6EC**
CRC-CCITT (Kermit)	**0xECFD**
CRC-DNP	**0xCDB5**
CRC-32	**0x77A5C3D0**

Rysunek 3 – Wartość CRC dla drugiego kodu.

"00000000000000000000000007938e8" (hex)	
1 byte checksum	**153**
CRC-16	**0x57C2**
CRC-16 (Modbus)	**0xA77C**
CRC-16 (Sick)	**0x7CCB**
CRC-CCITT (XModem)	**0xB683**
CRC-CCITT (0xFFFF)	**0xDC89**
CRC-CCITT (0x1D0F)	**0xF328**
CRC-CCITT (Kermit)	**0xE20F**
CRC-DNP	**0xCBF1**
CRC-32	**0x0ED5CE3B**

Rysunek 4 – Wartość CRC dla trzeciego kodu.

Jak widać, wybrane kody dają identyczne CRC, co świadczy o poprawnym wybraniu kodów.

III. Wnioski.

- Nie da się odtworzyć kodu o długości większej niż jeden bajt na podstawie jego wartości CRC.
- Dla danej wartości CRC, da się uzyskać wiele różnych kodów dających takie samo CRC. Stanowi to pewien poważny problem z punktu widzenia komunikacji master-slave, ponieważ slave może odebrać inną wiadomość niż wysłał master i uznać ją za poprawną na podstawie obliczonej przez siebie wartości CRC. Dlatego istnieją dodatkowe metody weryfikacji takie jak np. bity parzystości, kontrola zakresu wartości określonych pól ramki komunikacyjnej
- W przypadku opisanej sytuacji haker praktycznie nie ma szans, aby uzyskać kod dostępu, ponieważ musiałby strzelać z ilości $1{,}3*10^{36}$ kodów dających zarejestrowaną wartość CRC. Więc można z tego punktu widzenia uznać sieć Modbus RTU za bezpieczną.