Bezp. syst. i usług inform. 2
Keygen

Adam Zimny 209787

20 listopada 2016

1 Cel projektu

Celem laboratorium było stworzenie generatora kluczy poprzez odtworzenie odtworzenie algorytmu ich generowania na podstawie pliku binarnego.

2 Realizacja

W celu przystąpienia do analizy pliku należy poddać go deasemblacji przy pomocy komendy
`objdump -d reverse_lak > assembly`
Wywołanie tej komendy tworzy plik tekstowy o nazwie assembly, którego zawartością jest kod pliku `reverse_lak` w języku assembler.

Czytając kod programu odnaleźć można wywołania kolejnych funkcji. Prześledzenie działania programu po uruchomieniu pozwala wstępnie odtworzyć szkielet kodu programu:

Listing 1: Szkielet programu

```
success(){
        print("‘success"’);
}

failure(){
        print("‘failure"’);
}

generate_key(){
}

main(){
        print(login message);
        login = read_line();
        password = read_line();

        key = generate_key();
        if(password == key)
                success();
        else
                failure();
}
```

2.1 Odczyt poprawnego hasła

W celu dokładniejszego przeanalizowania operacji wykonywanych w programie wymagane jest użycie debuggera gdb. Debugger uruchamia się komendą:
`gdb reverse_lak`
Po uruchomieniu możliwe jest podejrzenie wartości zmiennych podczas działania programu. Na początek można wykorzystać to do odnalezienia poprawnego hasła dla wybranego numeru indeksu. W tym celu należy ustawić punkt przerwania po wywołaniu funkcji `generate_key();` w następujący sposób:

`b *0x80487c6`

Następnie należy uruchomić program, wprowadzić wymagane dane i wyświetlić zawartość rejestru przechowującego wygenerowany klucz.

`run`

USER: 209787

KEY: xxx

`x/s $esi`

co daje na wyjściu konsoli nastepujący wynik:

`0x804c068 "tahovcjqxelszgnubipwdkryfmtcludmvenwfoxgpyhqzirajsbk"`

2.2 Algorytm generowania hasła

Kolejnym krokiem jest analiza algorytmu funkcji `generate_key()` . Funkcja rozpoczyna się od pobrania ze stosu przekazanych parametrów i wywołania funkcji atoi na jednym z nich. Funkcja ta służy do konwersji ciągu znaków na wartość liczbową. Ustawiając punkt przerwania wewnątrz funkcji możliwe jest odczytanie parametrów przekazywanych do funkcji przez stos.

Listing 2: Parametry funkcji `generate_key()` odczytane w gdb

```
Breakpoint 1, 0x0804866e in generate_key ()
(gdb) x/ $ebp
0xffffd558:    -10872

(gdb) x/10x $ebp
0xffffd558:    0xffffd588    0x080487c6    0x0804c008    0x0804c068
0xffffd568:    0x00000040    0x0804c023    0xf7fbb000    0xf7fbb000
0xffffd578:    0xffffd5a0    0x00000000

(gdb) x/s 0x0804c008
0x804c008:    "209787\n"

(gdb) x/s 0x0804c068
0x804c068:    ""
```

Łatwo zauważyć, że przekazywaymi argumentami są wprowadzony numer indeksu, pusta tablica, oraz liczba całkowita, najprawdopodoniej oznaczająca rozmiar tablic. Wyświetlając wartości parametrów w trakcie przebiegu funkcji widać, że pusta tablica wykorzystywana jest do zwrócenia wygenerowanego klucza.

Przebieg funkcji można prześladzić korzystając z komendy `stepi` wykonującej kolejną linię programu. W celu łatwiejszej analizy działania programu warto skonfiguować w gdb wyświetlanie zawartości rejestrów po każdym kroku:

Listing 3: Konfiguracja gdb do wyświetlania rejestrów

```
display /i $pc            % wykonywana instrukcja
display $ecx              % licznik pętli
display /s 0x0804c068     % tablica wyjściowa
```

W odczytanym kodzie programu odnaleźć można liczne instrukcje skoków warunkowych. Przechodząc komendą `stepi` przez funkcję można zauważyć, że występują w niej dwie pętle, a rejestr `ecx` wykorzystywany jest jako licznik. Przekształcając instrukcję assemblera do bardziej czytelnej postaci uzyskać można pewnien rodzaj pseudokodu, który później może zostać wykorzystany do napisania odpowiedniego fragmentu w wybranym języku programowania. W celu uproszczenia kodu zastosowano pewne podstawienia:

index - liczba, dla której generowany jest klucz, przechowywana w rejestrze `%ebx`

tab - komórka pamięci o adresie 0x8049e18, z której program pobiera wartości przesunięte o `edx`

key - tablica wyjściowa, jej adres przechowywany jest w rejestrze `%edi`

i - rejestr ecx wykorzystywany jako licznik pętli

Listing 4: Pętle wewnątrz funkcji `generate_key()`

```
804868f:        imul    $0x7,%ecx,%eax                  % eax = i*7
                xor     %edx,%edx                       % edx = 0;
                add     %ebx,%eax                       % eax = eax + index;
                div     %esi                            % edx = eax mod esi;
                mov     0x8049e18(%edx),%al             % eax = tab[edx];
                mov     %al,(%edi,%ecx,1)               % key[i] = eax;
                inc     %ecx                            % i++;
                cmp     0x10(%ebp),%ecx                 % i == 64  ?
                jae     80486ac <generate_key+0x52>     % break;
                cmp     $0x19,%ecx                      % i == 25 ?
                jbe     804868f <generate_key+0x35>     % loop;


80486b8:        lea     (%ecx,%ecx,8),%eax              % eax = i + i*8;
                xor     %edx,%edx                       % edx = 0;
                add     %ebx,%eax                       % eax = eax + index;
                div     %esi                            % edx = eax mod esi;
                mov     0x8049e18(%edx),%al             % eax = tab[edx];
                mov     %al,(%edi,%ecx,1)               % key[i] = eax;
                inc     %ecx                            % i++;
                cmp     0x10(%ebp),%ecx                 % i == 64 ?
                jae     80486d5 <generate_key+0x7b>     % break;
                cmp     $0x33,%ecx                      % i == 51 ?
                jbe     80486b8 <generate_key+0x5e>     % loop;
```

Zawartość komórki pamięci `0x8049e18` wykorzystywanej w pętli wyświetlona przy pomocy komendy `x/s 0x8049e18` ukazuje zbiór znaków używanych do budowania klucza:

```
0x8049e18 <alpha>: "abcdefghijklmnopqrstuvwxyzO"
```

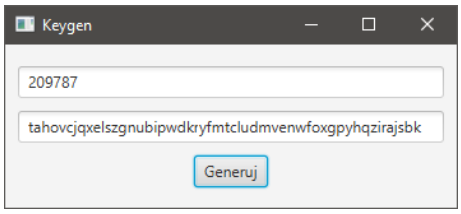
Na podstawie tych informacji możliwe jest odtworzenie algorytmu generującego klucze.

Listing 5: Odtworzony algorytm generowania klucza

```
char[] alphabet = "abcdefghijklmnopqrstuvwxyzO".toCharArray();

private void generateKey() {
      int index = Integer.parseInt(indexField.getText());// indexField - pole
          tekstowe
      int modulo = index % 26;
      int a, c;
      int esi = 26;
      String key = "";
      for (c = 0; c < 26; c++) {
          a = c * 7;
          a += modulo;
          key += alphabet[a % esi];
      }
      for (c = 26; c < 52; c++) {
          a = c + c * 8;
          a+= modulo;
          key += alphabet[a % esi];
      }
      keyField.setText(key); // keyField - pole tekstowe
  }
```

Sprawdzenie w stworzonym generatorze klucza dla uzytego wcześniej numeru indeksu 209787 potwierdza poprawność algorytmu. Wygenerowany klucz zgadza się w odczytanym wcześniej przy uzyciu debuggera gdb.



Rysunek 1: Zrzut ekranu stworzonego generatora

3 Podsumowanie

Zadanie to pokazało jak poprzez analizę kodu programu możliwe jest obejście pewnych zabezpieczeń. W tym celu wymagana jest umiejętność obsługi debuggera gdb, znajomość podstawowych komend systemu Linux oraz języka assembler. Analiza przetłumaczonego kodu maszynowego może być czasochłonna, jednak przy odpowiednim wykorzystaniu dostepnych narzędzi da się odtworzyć funkcje programu, które powinny być ukryte przed użytkownikiem.