

Linux'te Zararlı Yazılım Analizi için Faydalı Araçlar

written by Mert SARICA | 1 September 2014

Son zamanlarda Linux işletim sistemini (belki de *nix demeliyim) hedef alan zararlı yazılım salgınları (#1, #3) ile ilgili haberlere sıkça rastlıyoruz.

Sunucuların yanı sıra son kullanıcı sistemlerinin de hedef alınması (Hand of Thief bankacılık zararlı yazılımı), siber güvenlik uzmanlarının Windows gibi Linux işletim sistemi üzerinde de zararlı yazılım analizi yapabilecek bilgi ve beceriye sahip olması gerektiğini ortaya koymaktadır.

Fakat zararlı yazılım analizi ile ilgili kitaplara, eğitimlere, yazılara baktığınız zaman çoğunun sadece Windows işletim sistemi ile ilgili olduğunu görebilirsiniz. Her yıl yayınlanan siber güvenlik tehdit raporlarını incerseniz bunun en büyük nedeninin, geliştirilen zararlı yazılımların %90'ının Windows işletim sistemini hedef alması olduğunu anlayabilirsiniz. Windows işletim sistemi için geliştirilmiş olan bir zararlı yazılımı analiz etmek istediğiniz zaman, Linux'e kıyasla çok daha fazla araç bulmanız da bu sebepten ötürü şaşırtıcı değildir.

Linux'un açık kaynak kodlu ve özgür bir platform olması, barındırdığı araçlar ile zararlı yazılım analizine olanak tanısa da (strings, gdb, objdump, readelf, strace, file vb.), Windows'ta ücretsiz, kullanıcı dostu OllyDbg / Immunity Debugger ile kod analizi gerçekleştiren bir uzmanın Linux'te komut satırına mahkum kalması kimi zaman can sıkıcı olabilmektedir.

Tabii IDA Pro'nun disassembler ve hata ayıklayıcı olarak Linux dosya sistemini (ELF) ve işletim sistemini destekliyor olması (4 sene önceki IDA Pro ile Remote Linux Debugging yazıma buradan ulaşabilirsiniz) her ne kadar bu platform için büyük bir artı olsa da fiyatının ~1200\$ olması, kendini geliştirmek isteyen siber güvenlik uzmanları için büyük bir engel oluşturmaktadır.

İçinizi çok daraltmadan, OllyDbg / Immunity Debugger ve IDA Pro'ya alternatif olarak zararlı yazılım analizi ve tersine mühendislik için Linux üzerinde kullanabileceğiniz, benim de çok işime yarayan iki araçtan kısaca bahsetmek

istiyorum; EDB ve Hopper

EDB: Windows'ta OllyDbg ile sıkça kod analizi yapan kahramanımız Evan, günün birinde Linux'te OllyDbg gibi kullanışlı bir hata ayıklayıcıya ihtiyaç duyar, bulamaz ve EDB'yi geliştirmeye başlar. Windows'ta OllyDbg kullananlar için biçilmiş kaftan olan EDB ile ELF dosyalarını hem disassemble edebilir hem de hata ayıklayıcı olarak kullanarak rahatlıkla analiz edebilirsiniz. REMnux ile birlikte gelen EDB'yi çalıştırmak için komut satırında edb yazmanız yeterli.

Hopper: Hopper, IDA Pro'ya alternatif olarak kullanabileceğimiz, bütçenizi çok zorlamayacak (89\$), Mac OS X'te de çalışabilen, hem hata ayıklayıcı, hem disassembler hem de sözde (pseudocode kod çeviricisi (decompiler) olarak kullanabileceğiniz bir araçtır. Python desteğine ve Objective-C desteğine de sahip olduğunu hatırlatmak isterim.

Araçlara kısaca göz atmak için örnek bir zararlı yazılım üzerinden hızlıca ilerleyelim. Elimizde komuta kontrol merkezi ile haberleşen bir zararlı yazılım var ve bu zararlı yazılım komuta kontrol merkezine ait olan adresi strings, disassembler gibi araçlardan gizlemek için üzerinde gizli (encoded) olarak tutuyor.

Hopper ile ELF dosyasına göz attığımızda yapacağımız ilk iş programın başlangıç fonksiyonu olan main fonksiyonuna göz atmak olacaktır. Bu fonksiyona göz attığımızda şüpheli DecryptData fonksiyonunun çağrılmadan önce yığına (stack) gizlenmiş veriyi (kyipvm-k`bl41177/bnn) kopyalandığını görebiliyoruz. DecryptData fonksiyonun içine girdiğimizde, sıradan disassembler araçları ile yapacağımız tek şey, komutların (opcode) üzerinden teker teker geçerek fonksiyonun gizlenmiş veriyi nasıl çözdüğünü anlamaya çalışmak ve ardından gizlenmiş veriyi çözen (decoder) bir araç hazırlamak olacaktır. Fakat Hopper ile gelen sözde kod (pseudocode) çevirici (decompiler) sayesinde bu fonksiyonu sözde koda (pseudocode) çevirmek ve ardından bu kodu Python'a çevirerek bir decoder yazmak gerçekten oldukça basit hale geliyor. Bunun için DecryptData fonksiyonunun üzerine iki defa bastıktan sonra ALT – Return tuşlarına basarak sözde kodu görüntüleyebiliyoruz. Bundan sonrası ise programlama bilginize kalıyor.

Hopper Disassembler v3 — Hopper Disassembler v3

File Edit Find Modify Navigate Scripts Window Help

rewgt3er4t X

Search

Labels Strings

Tags

- resolve
- V8Rand
- V8ULRand
- checksum
- UDPFlood
- SynFlood
- sub_80496b9
- DNSFlood
- sub_804a06d
- strstri
- DIYTp
- sub_804a8eb
- sub_804ac51
- TCPFlood
- sub_804b11a
- GetOnlineMessage
- countercalc
- CPUStatus
- sub_804b6ce
- sub_804b78c
- DOSreturn
- dosManageThread
- sub_804ba66
- dqwz
- StartAddress
- sub_804c584
- sub_804c64a
- install
- sub_804c8a0
- a
- sub_804cc55
- nodieth
- sub_804cd13
- nodie
- sub_804ce85
- main
- dthread.ioin

```

0804cfba    mov     edi, dword [ss:ebp-0x1318+var_4884]
0804cfbd    leave  ; endp
0804cfbe    ret

----- BEGINNING OF PROCEDURE -----

main:
0804cfbf    push  ebp
0804cfc0    mov   ebp, esp
0804cfc2    sub   esp, 0x8
0804cfc5    and   esp, 0xfffffff0
0804cfc8    mov   eax, 0x0
0804cfd0    sub   esp, eax
0804cfd2    push  0x1
0804cfd4    push  0xd
0804cfd6    call  signal
0804cfd8    add   esp, 0x10
0804cfe0    sub   esp, 0x8
0804cfe1    sub   esp, 0x4
0804cfe4    push  0x80bc800
0804cfe9    call  strlen
0804cfee    add   esp, 0x8
0804cff1    push  eax
0804cff2    push  0x80bc800
0804cff7    call  DecryptData
0804cfr2    push  0x80bc800
0804cfr7    call  DecryptData
0804cfff    call  a
0804d004    call  install
0804d009    call  nodie
0804d00e    push  0x0
0804d010    push  0x80481f0
0804d015    push  0x0
0804d017    lea  eax, dword [ss:ebp-0x8+var_4]
Background analysis ended
  
```

Address 0x804cfb. Segment Segment 0. main + 0. Section .text. file offset 0x4fbf

Hopper Disassembler v3 — Hopper Disassembler v3

File Edit Find Modify Navigate Scripts Window Help

rewgt3er4t X

Search

Labels Strings

Tags

- _init
- _start
- call_gmon_start
- _do_global_dtors_aux
- frame_dummy
- closefirewall
- EnBuffer
- DecryptData
- ltrim
- resolve
- V8Rand
- V8ULRand
- checksum
- UDPFlood
- SynFlood
- sub_80496b9
- DNSFlood
- sub_804a06d
- strstri
- DIYTp
- sub_804a8eb
- sub_804ac51
- TCPFlood
- sub_804b11a
- GetOnlineMessage
- countercalc
- CPUStatus
- sub_804b6ce
- sub_804b78c
- DOSreturn
- dosManageThread
- sub_804ba66
- dqwz
- StartAddress
- sub_804c584
- sub_804c64a
- install

```

080486c7    jmp   0x804868a
080486c9    leave ; endp
080486ca    ret

----- BEGINNING OF PROCEDURE -----

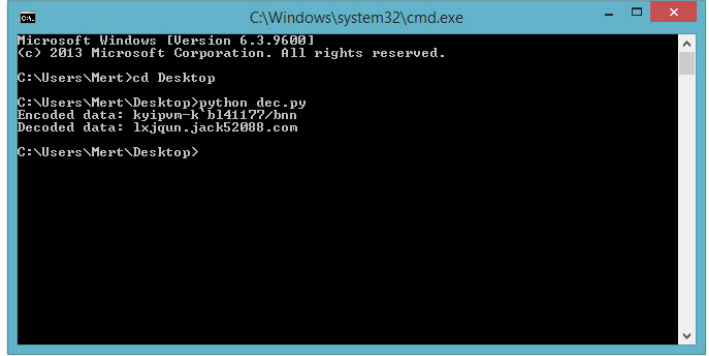
DecryptData:
080486cb    push  ebp
080486cc    mov   ebp, esp
080486ce    sub   esp, 0x8
080486d1    mov   dword [ss:ebp-0x8+var_4], 0x0
  
```

Pseudo Code — Hopper Disassembler v3

```

function DecryptData {
    var_4 = 0x0;
    do {
        eax = var_4;
        if (eax >= arg_offset_x4) {
            break;
        }
        temp_0 = var_4 / 0x3;
        temp_1 = var_4 % 0x3;
        if (temp_1 == 0x1) {
            if ((*int8_t*)(var_4 + arg_offset_x0) > 0x20) && ((*int8_t*)(var_4 + arg_offset_x0) != 0x7f) {
                (*int8_t*)(var_4 + arg_offset_x0) = ((*int8_t*)(var_4 + arg_offset_x0) - 0x1);
            }
            else {
                if ((*int8_t*)(var_4 + arg_offset_x0) > 0x20) && ((*int8_t*)(var_4 + arg_offset_x0) != 0x7f) {
                    (*int8_t*)(var_4 + arg_offset_x0) = ((*int8_t*)(var_4 + arg_offset_x0) + 0x1);
                }
            }
            var_4 = var_4 + 0x1;
            continue;
        } while (true);
        return eax;
    }
}
  
```

```
1 #function DecryptData (
2 #   var_4 = 0x0;
3 #   do {
4 #       eax = var_4;
5 #       if (eax >= arg_offset_x4) {
6 #           break;
7 #       }
8 #       temp_0 = var_4 / 0x3;
9 #       temp_1 = var_4 % 0x3;
10 #       if (temp_1 == 0x1) {
11 #           if ((*int8_t *) (var_4 + arg_offset_x0) > 0x20) && ((*int8_t *) (var_4 + arg_offset_x0) != 0x7f) {
12 #               *(int8_t *) (var_4 + arg_offset_x0) = *(int8_t *) (var_4 + arg_offset_x0) - 0x1;
13 #           }
14 #       } else {
15 #           if ((*int8_t *) (var_4 + arg_offset_x0) > 0x20) && ((*int8_t *) (var_4 + arg_offset_x0) != 0x7f) {
16 #               *(int8_t *) (var_4 + arg_offset_x0) = *(int8_t *) (var_4 + arg_offset_x0) + 0x1;
17 #           }
18 #       }
19 #       var_4 = var_4 + 0x1;
20 #       continue;
21 #   } while (true);
22 #   return eax;
23 # }
24
25 encdata = "kypvm-k`bl41177/bnn"
26 decdata = ""
27 m = 0
28
29 print "Encoded data:", encdata
30
31 for i in encdata:
32     temp_1 = m % 3
33     if (temp_1 == 0x1):
34         if ((ord(i) > 0x20) and (ord(i) != 0x7f)):
35             decdata = decdata + chr(ord(i) - 0x1)
36     else:
37         if ((ord(i) > 0x20) and (ord(i) != 0x7f)):
38             decdata = decdata + chr(ord(i) + 0x1)
39     m = m + 1
40
```



Disassembler ve programlama ile uğraşmak istemiyorum, kısa yoldan OllyDbg'da olduğu gibi fonksiyonun (DecryptData) üzerinden hızlıca geçerek fonksiyonun gizlenmiş veriyi çözmesini sağlayayım diyenler için ise EDB hemen imdadımıza yetişiyor. EDB'ye zararlı yazılımı yükledikten sonra main fonksiyonuna gitmek için, Plugins menüsü altında SymbolViewer eklentisini çalıştırdıktan sonra ilgili yere main yazıp üzerine iki defa basarsanız ana fonksiyonuna geçiş yapabilirsiniz. Ardından DecryptData (call 0x08048cb) üzerine breakpoint (sağ tuş -> Add Breakpoint) koyarak F9 (Run) butonuna basarak DecryptData fonksiyonuna kadar programın devam etmesini sağlayabilirsiniz. Sağ alt kısımda yer alan yığın (stack) bölümünde gördüğünüz gizlenmiş verinin (kypvm-k`bl41177/bnn) çözülmüş halini görmek için F8 (Step Over) tuşuna bastığınızda verinin lxjqun.jack52088.com adresi olarak çözüldüğünü görebilirsiniz.

edb - /home/remnux/Desktop/upx/rewgtf3er4t [5892]

File View Debug Plugins Options Help

No Analysis Found For This Region

0804:cfbf	55	push ebp
0804:cfc0	89 e5	mov ebp, esp
0804:cfc2	83 ec 08	sub esp, 8
0804:cfc5	83 e4 f0	and esp, 0xf0
0804:cfc8	b8 00 00 00 00	mov eax, 0
0804:cfd	29 c4	sub esp, eax
0804:cfcf	83 ec 08	sub esp, 8
0804:cfd4	6a 01	push 1
0804:cfd4	6a 0d	push 13
0804:cfd6	e8 fd 6f 00 00	call 0x08053fd8 <rewgtf3er4t::ssignal>
0804:cfdb	83 c4 10	add esp, 16
0804:cfd	83 ec 08	sub esp, 8
0804:cfe1	83 ec 04	sub esp, 4
0804:cfe4	68 00 c8 0b 08	push 0x080bc800
0804:cfe9	e8 b2 40 01 00	call 0x080610a0 <rewgtf3er4t::strlen>
0804:cfee	83 c4 08	add esp, 8
0804:cff1	50	push eax
0804:cff2	68 00 c8 0b 08	push 0x080bc800
0804:cff7	e8 cf b6 ff ff	call 0x080486cb <rewgtf3er4t::DecryptData>
0804:cfff	83 c4 10	add esp, 16
0804:cfff	e8 b7 fb ff ff	call 0x0804cbbb <rewgtf3er4t::a>
0804:d004	e8 5c f6 ff ff	call 0x0804c665 <rewgtf3er4t::install>
0804:d009	e8 c0 fd ff ff	call 0x0804cdce <rewgtf3er4t::nodie>
0804:d00e	6a 00	push 0
0804:d010	68 f0 81 04 08	push 0x080481f0
0804:d015	6a 00	push 0
0804:d017	8d 45 fc	lea eax, [ebp-4]

0x080486cb = 080486cb <rewgtf3er4t::DecryptData+0>

Registers

General Purpose

- EAX: 00000014
- EBX: 00000000
- ECX: ffffffff
- EDX: 00000000
- EBP: bfabd6b8
- ESP: bfabd6a0
- EIP: 0804cf7c <rewgtf3er4t::...
- EFLAGS: 00200282

Bookmarks

Address	Comment

Add Del Clear

Data Dump

08048000-080bc000

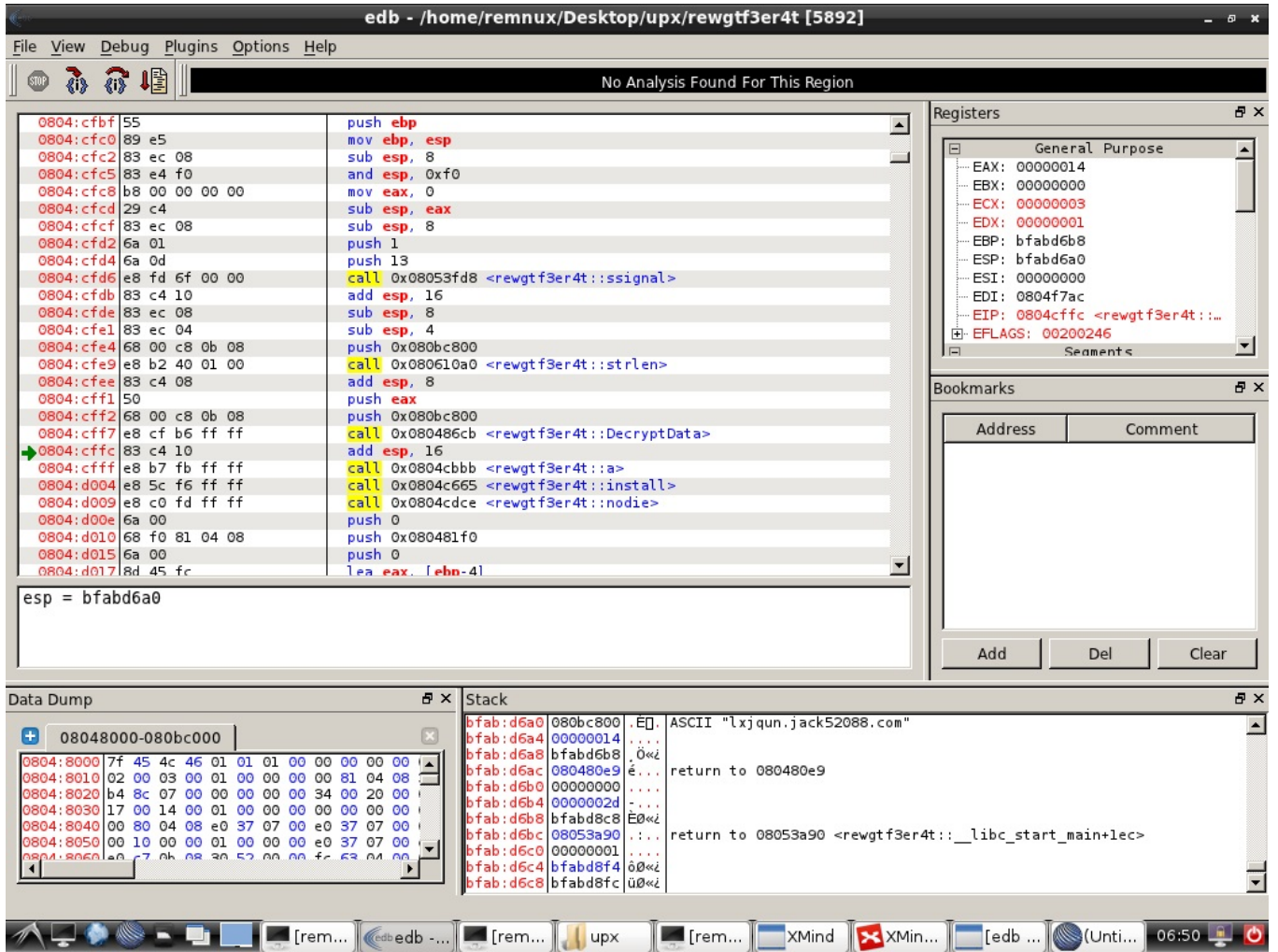
0804:8000	7f 45 4c 46 01 01 01 00 00 00 00 00
0804:8010	02 00 03 00 01 00 00 00 00 81 04 08
0804:8020	b4 8c 07 00 00 00 00 00 34 00 20 00
0804:8030	17 00 14 00 01 00 00 00 00 00 00 00
0804:8040	00 80 04 08 e0 37 07 00 e0 37 07 00
0804:8050	00 10 00 00 01 00 00 00 e0 37 07 00
0804:8060	e0 7 0b 08 30 52 00 00 f0 63 04 00

Stack

bfab:d6a0	080bc800	.E	ASCII "kyipvm-k`bl41177/bnn"
bfab:d6a4	00000014	...	
bfab:d6a8	bfabd6b8	0<<	
bfab:d6ac	080480e9	é...	return to 080480e9
bfab:d6b0	00000000	...	
bfab:d6b4	0000002d	...	
bfab:d6b8	bfabd8c8	E0<<	
bfab:d6bc	08053a90	...	return to 08053a90 <rewgtf3er4t::__libc_start_main+1ec>
bfab:d6c0	00000001	...	
bfab:d6c4	bfabd8f4	00<<	
bfab:d6c8	bfabd8fc	00<<	

paused

[rem... edb - ... [rem... upx [rem... XMind XMin... [edb ... (Unti... 06:49



Linux üzerinde tersine mühendislik ve zararlı yazılım analizi ile ilgilenmek isteyenler için faydalı bir yazı olduğunu ümit ederek bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.