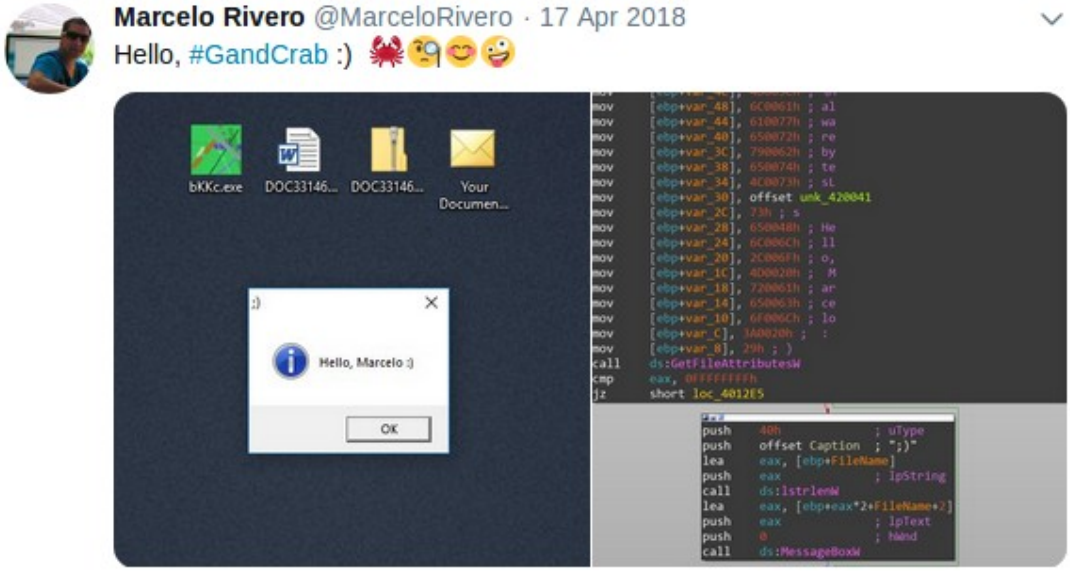
# Deobfuskácia textových reťazcov v ransomvéri GandCrab

## Úvod

Ransomvér GandCrab sa na scéne objavuje približne od začiatku roka 2018 a stále patrí medzi najrozšírenejší malvér. V priebehu času sa vyskytlo viacero verzií ransomvéru GandCrab, v súčasnosti je už rozšírená verzia 5.3. Nové verzie vychádzajú pomerne často, avšak s nie príliš výraznými zmenami medzi verziami. Napriek tomu však niektoré nástroje, ktoré boli špeciálne vytvorené, aby pomáhali pri analýze tohto ransomvéru, s novými verziami prestali fungovať. Vyzerá to tak, že autori GandCrabu sledujú prácu analytikov a občas dokonca nechávajú v ransomvéri špeciálne odkazy priamo pre nich.




*Obr. 1: Personalizovaná vzorka ransomvéru GandCrab, zdroj: https://twitter.com/MarceloRivero/status/986045072272691201*

V tomto článku sa pozrieme na spôsob, akým sú obfuskované textové reťazce, ktoré využíva ransomvér GandCrab verzie 5.x a predstavíme skript pre analytický nástroj IDA (interaktívny disasembler), ktorý dokáže tieto reťazce deobfuskovať bez spustenia analyzovanej vzorky ransomvéru. Takýto nástroj síce existoval pre prvé verzie ransomvéru, avšak nebol už dlho aktualizovaný a medzičasom sa použitá metóda obfuskácie výrazne zmenila.

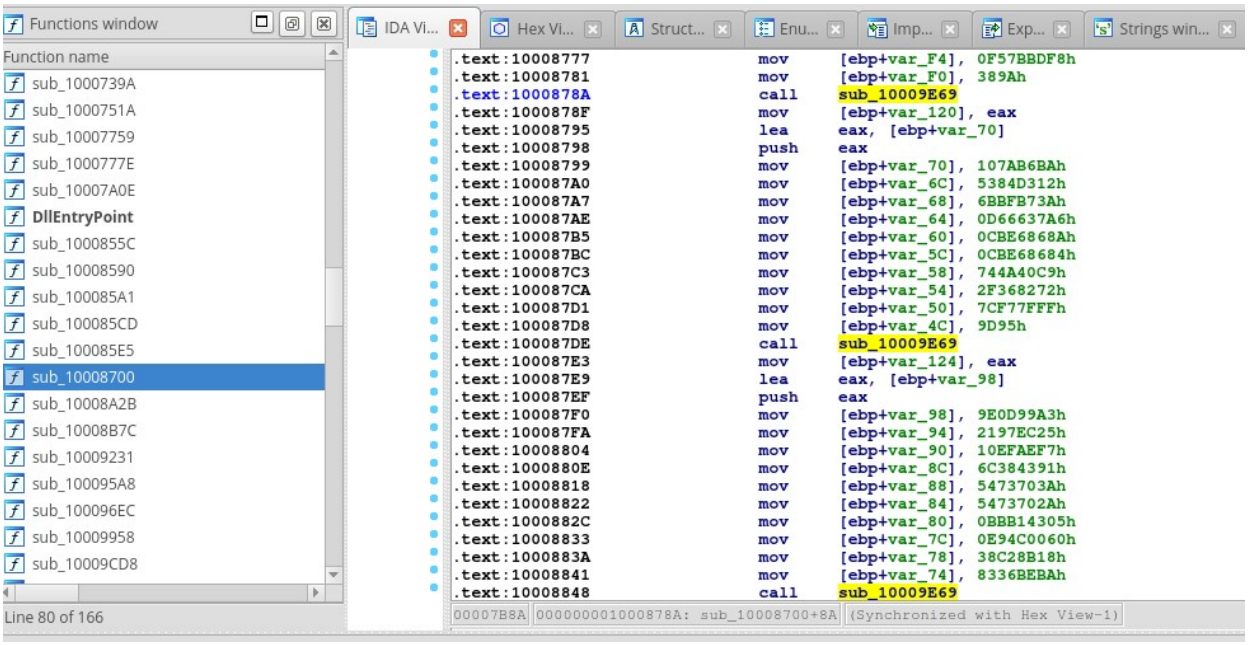
## Obfuskované textové reťazce

V prvých verziách ransomvéru boli textové reťazce uložené ako hexadecimálne štvorbytové hodnoty z Unicode reprezentácie reťazca, ktoré boli pomocou inštrukcií `mov` presunuté na správne miesto v pamäti, aby tam vytvorili celý textový reťazec. Ukážku je možné vidieť na obrázku 1 vpravo hore. V aktuálnych verziách GandCrabu je však táto obfuskácia oveľa komplikovanejšia a zahŕňa okrem iného aj dešifrovanie pomocou prúdových šifier.

V nasledujúcom texte sa pozrieme na konkrétnu vzorku ransomvéru GandCrab verzie 5.1, ktorá sa šírila v kampani zo začiatku februára tohto roka. SHA256 použitej vzorky je 6aa3f17e5f62b715908b5cb3ea462bfa6cecfd3f4d70078eabd418291a5a7b83.

Po zbežnom pohľade na disasemblované inštrukcie si môžeme všimnúť, že častokrát je volaná jedna konkrétna funkcia, v našom prípade označená nástrojom IDA ako `sub_10009E69`. Tieto volania sa objavujú tesne po sérii `mov` inštrukcií s druhým operandom v podobe štvorbytovej hodnoty. Tieto hodnoty sú presúvané do lokálnej premennej na zásobníku podobne ako v prípade prvých verzií

GandCrabu, kde sa takto vytváral Unicode textový reťazec. Tu však tieto hodnoty vyzerajú pomerne náhodne. Následne ešte pred sériou týchto `mov` inštrukcií môžeme vidieť uloženie adresy lokálnej premennej na zásobník (načítanie jej adresy zvyčajne do registra `eax` a uloženie tohto registra na zásobník), čím bude táto lokálna premenná použitá ako argument pre volanie funkcie `sub_10009E69`. Pre lepšiu predstavu je to znázornené na obrázku nižšie:




*Obr 2.: Obfuskované textové reťazce v GandCrabe v5.1*

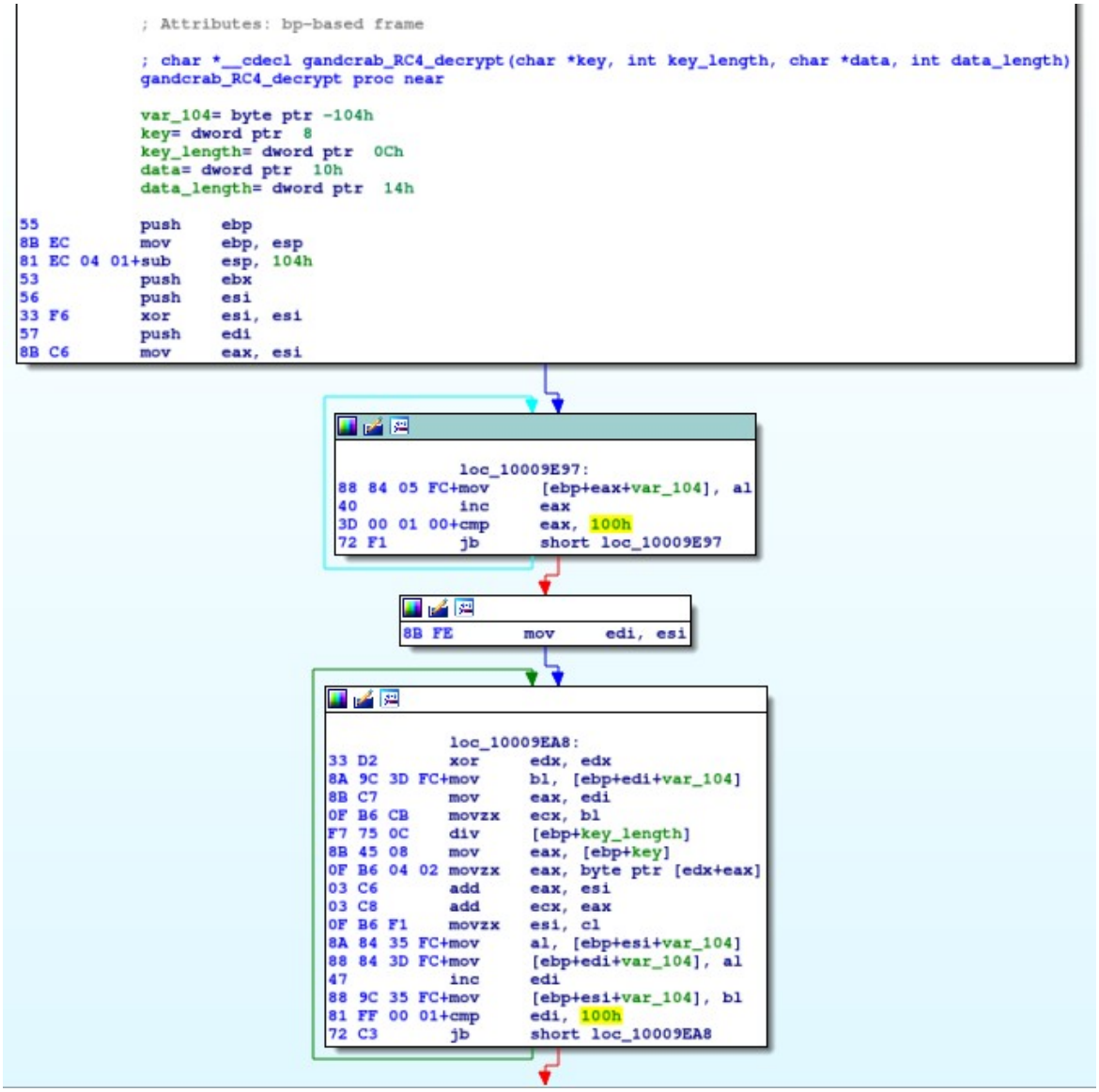
Poďme sa teraz podrobnejšie pozrieť na funkciu `sub_10009E69`. Pre názorné účely ju premenujeme na `gandcrab_decrypt_string`. Táto funkcia berie jeden argument v podobe obfuskovaného textového reťazca (typ `char *`) a extrahuje z neho (de)šifrovací kľúč, šifrovaný text a dĺžku šifrovaného textu, ktorá je tiež obfuskovaná (získava sa ako xor dvoch štvorbytových hodnôt z argumentu na vstupe). Dĺžka (de)šifrovacieho kľúča je v tomto prípade konštantná, 10 bytov. Tieto štyri hodnoty sú následne použité ako vstupné argumenty pre dešifrovaciu rutinu.

```
                ; Attributes: bp-based frame

                ; char *__cdecl gandcrab_decrypt_string(char *string)
                gandcrab_decrypt_string proc near

                string= dword ptr  8

55              push    ebp
8B EC           mov     ebp, esp
8B 4D 08        mov     ecx, [ebp+string]
8B 41 14        mov     eax, [ecx+14h]
33 41 10        xor     eax, [ecx+10h]
50              push    eax             ; data_length
8D 41 18        lea     eax, [ecx+18h]
50              push    eax             ; data
6A 10           push    10h             ; key_length
51              push    ecx             ; key
E8 05 00 00+call    gandcrab_RC4_decrypt
83 C4 10        add     esp, 10h
5D              pop     ebp
C3              retn
                gandcrab_decrypt_string endp
```

*Obr. 3: Funkcia na deobfuskáciu textových reťazcov, ktorá pripraví parametre pre RC4 dešifrovanie*

Dešifrovacia rutina je vlastne implementácia prúdovej šifry RC4. Opäť, pre názorné účely ju

premenujeme na `gandcrab_RC4_decrypt`. Na nasledujúcom obrázku môžeme vidieť KSA (key-scheduling algorithm) časť RC4 šifry pozostávajúcu z dvoch for-cyklov bežiacich od 0 do 255. Prvý cyklus inicializuje identickú permutáciu v poli označenom ako `var_104`, druhý for-cyklus zamieša hodnoty v tomto poli, čím definuje pseudonáhodnú permutáciu bytov. *Hodnoty 100h (256 decimálne) a štruktúra týchto dvoch for-cyklov sú dobrá nápoveda pri identifikácii použitej šifry RC4.*




*Obr. 4: RC4 dešifrovacia funkcia*

## Deobfuskovanie/dešifrovanie textových reťazcov

V tejto chvíli už vieme dosť o tom, ako funguje obfuskácia textových reťazcov v ransomvéri GandCrab, a mali by sme byť schopní dešifrovať tieto obfuskované reťazce aj bez spúšťania danej vzorky v debuggeri. Vytvoríme skript pre nástroj IDA, ktorý bude automaticky dešifrovať obfuskované reťazce. IDA skripty môžu byť napísané pomocou jazykov IDC (skriptovací jazyk podobajúci sa na C) alebo pomocou IDAPython. Podpora IDAPython je však oficiálne dostupná iba v platených verziách IDA, kdežto IDC je podporované aj vo freeware verzii. Z tohto dôvodu sme zvolili pre náš skript jazyk IDC, aby mohol byť použiteľný aj pre nadšencov a študentov bez komerčnej licencie nástroja IDA.

Najprv reimplementujeme RC4 algoritmus. To je tá jednoduchšia časť, nakoľko môžeme využiť viacero online zdrojov s implementáciou v pseudokóde, ktorú už iba prepíšeme do IDC, napríklad sa

môžeme inšpirovať Wikipediou.

Následne musíme identifikovať funkciu na deobfuskovanie textových reťazcov (vyššie uvádzanú ako `gandcrab_decrypt_string`). Táto funkcia bude veľmi často používaná, keďže bude zavolané pre každý obfuskovaný reťazec. V prípade niektorých vzoriek je dokonca najčastejšie volanou funkciou, ale ako sa ukázalo pri pohľade na ďalšie vzorky, neplatí to vo všeobecnosti. Takže na identifikovanie tejto funkcie nebude stačiť iba nájsť najpoužívanejšiu funkciu. Mohli by sme nájsť 3-5 najpoužívanejších funkcií a každú z nich skúsiť považovať za funkciu pre deobfuskáciu, ale nie je to úplne čisté riešenie a navyše, nemusí vždy fungovať. Chcelo by to iný prístup, ako identifikovať hľadanú funkciu.

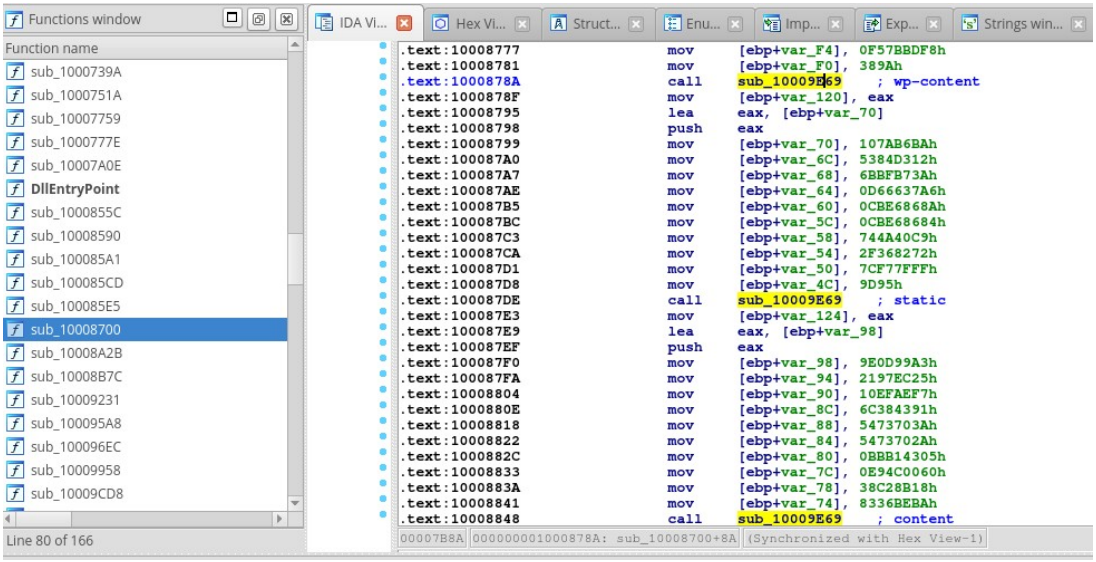
Môžeme zistiť niekoľko jej vlastností, na základe ktorých ju budeme vedieť lepšie identifikovať. Napríklad to, že je často volaná, môže byť jedna z týchto vlastností. V analyzovaných vzorkách bola táto funkcia volaná 158 až 185-krát. Na základe toho môžeme predpokladať, že nami hľadaná funkcia bude volaná aspoň 100-krát (radšej menej, nech máme rezervu, v kombinácii s ďalšími vlastnosťami ale zmenšíme pravdepodobnosť chybnej identifikácie). Ďalej, keďže jediným účelom tejto funkcie je príprava parametrov pre RC4 dešifrovanie zo vstupného argumentu (obfuskovaného reťazca), môžeme očakávať, že táto funkcia bude pomerne krátka. Zo vzoriek to vyzerá, že jej dĺžka je 30 bytov. Môžeme teda predpokladať, že dĺžka nami hľadanej funkcie bude podobne malá, povedzme že najviac 37 bytov (opäť s rezervou). Ako už bolo spomínané, účelom tejto funkcie je pripraviť 4 parametre pre RC4 dešifrovanie a zavolať dešifrovaciu funkciu. To znamená, že nami hľadaná funkcia by mala obsahovať presne jednu inštrukciu `call` a 5 inštrukcií `push` (jeden `push` je ešte súčasťou prológu tejto funkcie).

Na základe vyššie uvedených 4 vlastností (dĺžka funkcie, počet volaní, počet inštrukcií `push` a `call`) by sme mali byť schopní identifikovať nami hľadanú funkciu na deobfuskovanie textových reťazcov. *Alternatívny prístup by mohol spočívať v identifikácii funkcie pre RC4 dešifrovanie a následne preskúmanie funkcií, z ktorých je volaná.*

Teraz je pred nami ťažšia časť, a to je rekonštrukcia argumentu pre funkciu `gandcrab_decrypt_string`: pre každé volanie tejto funkcie musíme spätne nájsť, ktorý register je uložený na zásobník a akú adresu obsahuje (offset k hodnote v registri `ebp`). Následne musíme prehľadať okolité `mov` inštrukcie, ktoré vytvárajú textový reťazec, nájsť maximálny použitý offset k registru `ebp` a zo znalostí týchto dvoch offsetov určíme dĺžku obfuskovaného textového reťazca. Následne v našom skripte môžeme inicializovať reťazec danej dĺžky a jednotlivé znaky v ňom prepísať na základe hodnôt z `mov` inštrukcií. Celé je to kvôli tomu, že dané inštrukcie `mov` nemusia byť v takom poradí, v akom sú jednotlivé časti obfuskovaného reťazca. Pokojne sa môže stať, že najprv jedna inštrukcia `mov` nastaví 4 byty na konci reťazca a nasledujúca inštrukcia nastaví 4 byty niekde v strede reťazca, takže nemôžeme iba jednoducho pospájať všetky nájdené 4-bytové hodnoty z `mov` inštrukcií za seba.

Nakoniec, zo zrekonštruovaného obfuskovaného reťazca musíme vyextrahovať parametre pre RC4 dešifrovanie: kľúč, dĺžka šifrovaného reťazca a samotný šifrovaný reťazec. Urobíme to rovnako, ako to robí funkcia `gandcrab_decrypt_string`. Potom už môžeme dešifrovať textový reťazec a zobrazíme dešifrovanú hodnotu napríklad v podobe komentára v disasemblovanom kóde a ešte ju môžeme aj zalogovať do IDA konzoly.

Teraz to môžeme dať všetko dokopy a máme funkčný IDC skript pre dešifrovanie textových reťazcov v ransomvéri GandCrab. Po jeho spustení dostaneme nasledovný výstup:




*Obr. 5: Dešifrované textové reťazce v disasemblovanom kóde*

```
1000e97b: xref to decrypt function 10009e69
"RAMDISK" (length: 0x8)

1000edb7: xref to decrypt function 10009e69
"Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko" (length: 0x1fc5)

1000eef8: xref to decrypt function 10009e69
"HTTP/1.1" (length: 0xa)

1000f933: xref to decrypt function 10009e69
"ENCRYPTED BY GANDCRAB %s" (length: 0x1c)

1000fa59: xref to decrypt function 10009e69
"ĝg@U@cq@@@]a;z" (length: 0xe)

1000fad5: xref to decrypt function 10009e69
"DEAR %s, " (length: 0xa)

1000fb35: xref to decrypt function 10009e69
"DEAR USER, " (length: 0xc)

1000fced: xref to decrypt function 10009e69
"YOUR FILES ARE UNDER STRONG PROTECTION BY OUR SOFTWARE. IN ORDER TO RESTORE IT YOU MUST BUY DECRYPTOR" (length: 0x68)

1000fe2d: xref to decrypt function 10009e69
"For further steps read %s-DECRYPT.%s that is located in every Mncrypted folder" (length: 0x4f)

IDC
```

*Obr. 6: Dešifrované textové reťazce, výstup skriptu*

## Záver

Ukázali a analyzovali sme spôsob, akým ransomvér GandCrab obfuskuje textové reťazce, ktoré používa. Okrem analýzy sme aj predstavili spôsob, ako nájsť v disasemblovanom kóde funkciu pre deobfuskovanie týchto reťazcov a pre automatizáciu deobfuskácie sme vytvorili IDC skript pre nástroj IDA.

Zo zoznamu deobfuskovaných textových reťazcov v Prílohe 1 je napríklad zrejmé, o aké programy sa GandCrab zaujíma. Ukončí všetky bežiace Office programy, pretože by mohli blokovať šifrovanie a prepísanie otvorených súborov. Taktiež kontroluje prítomnosť antivírových programov. A okrem iného sú tu aj zmienky o tom, že sa jedná o GandCrab (kľúče, správy, inštrukcie pre dešifrovanie,...). Aj z textových reťazcov sa dá urobiť aspoň základná predstava o funkcionalite skúmanej vzorky.

## Zdroje

- IDC skript na dešifrovanie textových reťazcov v GandCrabe
- Pôvodný článok o dešifrovaní textových reťazcov v GandCrabe
- VirusTotal analýza vzorky GandCrab
- Intezer analýza vzorky GandCrab
- Personalizovaná vzorka GandCrab

## Príloha 1: Zoznam deobfuskovaných textových reťazcov

```
"Global7a94U8TAO7zVm5qzEjzks" (length: 0x26)
"Global%s.luck" (length: 0x10)
"@hashbreaker Daniel J. Bernstein let's dance salsa <3" (length: 0x36)
"@hashbreaker :)))" (length: 0x12)
"A" (length: 0x2)
"onenote.exe" (length: 0xc)
"outlook.exe" (length: 0xc)
"powerpnt.exe" (length: 0xe)
"steam.exe" (length: 0xa)
"thebat.exe" (length: 0xc)
"thebat64.exe" (length: 0xe)
"thunderbird.exe" (length: 0x10)
"visio.exe" (length: 0xa)
"winword.exe" (length: 0xc)
"wordpad.exe" (length: 0xc)
"runas" (length: 0x6)
"---BEGIN GANDCRAB KEY---" (length: 0x1a)
"---END GANDCRAB KEY---" (length: 0x18)
"---BEGIN PC DATA---" (length: 0x14)
"---END PC DATA---" (length: 0x12)
"pc_user" (length: 0x8)
"pc_name" (length: 0x8)
"pc_group" (length: 0xa)
"av" (length: 0x4)
"pc_lang" (length: 0x8)
"pc_keyb" (length: 0x8)
"os_major" (length: 0xa)
"os_bit" (length: 0x8)
"ransom_id" (length: 0xa)
"hdd" (length: 0x4)
"ip" (length: 0x4)
"ransom_id=" (length: 0xc)
"SOFTWARE_data" (length: 0x18)
"public" (length: 0x8)
"private" (length: 0x8)
"open" (length: 0x6)
"Keyboard Layout" (length: 0x18)
"00000419" (length: 0xa)
"/c timeout -c 5 & del "%s" /f /q" (length: 0x22)
"open" (length: 0x6)
"cmd.exe" (length: 0x8)
"pc_user" (length: 0x8)
"pc_name" (length: 0x8)
"pc_group" (length: 0xa)
"av" (length: 0x4)
"pc_lang" (length: 0x8)
"pc_keyb" (length: 0x8)
"os_major" (length: 0xa)
"os_bit" (length: 0x8)
"ransom_id" (length: 0xa)
"hdd" (length: 0x4)
"ip" (length: 0x4)
"&id=" (length: 0x6)
"&sub_id=" (length: 0xa)
"&version=" (length: 0xa)
"&action=call" (length: 0xe)
"%s/%s/%s/%s.%s" (length: 0x10)
"wp-content" (length: 0xc)
"static" (length: 0x8)
```

```
   "content" (length: 0x8)
   "includes" (length: 0xa)
   "data" (length: 0x6)
   "uploads" (length: 0x8)
   "news" (length: 0x6)
   "jpg" (length: 0x4)
   "png" (length: 0x4)
   "gif" (length: 0x4)
   "bmp" (length: 0x4)
   "im" (length: 0x4)
   "de" (length: 0x4)
   "ka" (length: 0x4)
   "ke" (length: 0x4)
   "am" (length: 0x4)
   "so" (length: 0x4)
   "fu" (length: 0x4)
   "se" (length: 0x4)
   "da" (length: 0x4)
   "he" (length: 0x4)
   "me" (length: 0x4)
   "mo" (length: 0x4)
   "th" (length: 0x4)
   "zu" (length: 0x4)
   "images" (length: 0x8)
   "pictures" (length: 0xa)
   "image" (length: 0x6)
   "graphic" (length: 0x8)
   "assets" (length: 0x8)
   "pics" (length: 0x6)
   "imgs" (length: 0x6)
   "tmp" (length: 0x4)
   "http://%s" (length: 0xa)
   "POST" (length: 0x6)
   "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko" (length:
0xc5)
   "GET" (length: 0x4)
   "HTTP/1.1" (length: 0xa)
   "%s%s-DECRYPT.txt" (length: 0x12)
   "%s-DECRYPT.txt" (length: 0x14)
   "%s.KRAB" (length: 0x8)
   "ntdlldll" (length: 0xc)
   "NtSetInformationFile" (length: 0x18)
   "d9sktop.ini" (length: 0xe)
   "autorun.inf" (length: 0xc)
   "ntuser.dat" (length: 0xc)
   "iconcache.db" (length: 0xe)
   "bootsect.bak" (length: 0xe)
   "boot.ini" (length: 0xa)
   "ntuser.dat.log" (length: 0x10)
   "thumbs.db" (length: 0xa)
   "-DECRYPT.txt" (length: 0xe)
   "-DECRYPT.html" (length: 0xe)
   "%s-DECRYPT.html" (length: 0x10)
   "%s-DECRYPT.txt" (length: 0x10)
   "KRAB-DECRYPT.html" (length: 0x12)
   "CRAB-DECRYPT.html" (length: 0x12)
   "KRAB-DECRYPT.txt" (length: 0x12)
   "CRAB-DECRYPT.txt" (length: 0x12)
   "ntldr" (length: 0x6)
   "NTDETECT.COM" (length: 0xe)
   "Bootfont.bin" (length: 0xe)
   "SQL" (length: 0x4)
```

```
   "%s%x%x%x%x.lock" (length: 0x10)
   "" (length: 0xe)
   "" (length: 0xe)
   "" (length: 0x8)
   "Files" (length: 0x10)
   "Browser" (length: 0xe)
   "Users" (length: 0xc)
   "Settings" (length: 0x12)
   "" (length: 0xa)
   "=" (length: 0x2)
   "&" (length: 0x2)
   "undefined" (length: 0xa)
   "AVP.EXE" (length: 0x8)
   "ekrn.exe" (length: 0xa)
   "avgnt.exe" (length: 0xa)
   "ashDisp.exe" (length: 0xc)
   "NortonAntiBot.exe" (length: 0x12)
   "Mcshield.exe" (length: 0xe)
   "avengine.exe" (length: 0xe)
   "cmdagent.exe" (length: 0xe)
   "smc.exe" (length: 0x8)
   "persfw.exe" (length: 0xc)
   "pccpfw.exe" (length: 0xc)
   "fsguiexe.exe" (length: 0xe)
   "cfp.exe" (length: 0x8)
   "msmpeng.exe" (length: 0xc)
   "SYSTEM" (length: 0x34)
   "WORKGROUP" (length: 0xa)
   "Contro6 Panel" (length: 0x1e)
   "LocaleName" (length: 0xc)
   "Keyboard Layout" (length: 0x18)
   "00000419" (length: 0xa)
   "productName" (length: 0xc)
   "SOFTWARE6432NodeNT" (length: 0x3a)
   "productName" (length: 0xc)
   "error" (length: 0x6)
   "x86" (length: 0x4)
   "HARDWARE" (length: 0x30)
   "ProcessorNameString" (length: 0x14)
   "Identifier" (length: 0xc)
   "ntdll.dll" (length: 0xa)
   "RtlComputeCrc32" (length: 0x10)
   "UNKNOWN" (length: 0x8)
   "NO_ROOT_DIR" (length: 0xc)
   "REMOVABLE" (length: 0xa)
   "FIXED" (length: 0x7)
   "REMOTE" (length: 0x8)
   "CDROM" (length: 0x7)
   "RAMDISK" (length: 0x8)
   "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko" (length:
0x1fc5)
   "HTTP/1.1" (length: 0xa)
   "ENCRYPTED BY GANDCRAB %s" (length: 0x1c)
   "DEAR %s, " (length: 0xa)
   "DEAR USER, " (length: 0xc)
   "YOUR FILES ARE UNDER STRONG PROTECTION BY OUR SOFTWARE. IN ORDER TO RESTORE IT
YOU MUST BUY DECRYPTOR" (length: 0x68)
   "For further steps read %s-DECRYPT.%s that is located in every Mncrypted folder"
(length: 0x4f)
```

## Príloha 2: zdrojový kód IDC skriptu

```
#include <idc.idc>

static RC4_decrypt(key, key_length, data, length){
    //print(key, length);
    auto i, w, b, K, index, esi, eax, al, edi, edx, dl, ebx, bl, ecx, text;
    auto S = "";

    //Key-scheduling algorithm
    for (i=0; i<256; i++){
        S[i] = i;
    }
    w = 0;
    for (i=0; i<256; i++){
        w = (w + ord(S[i]) + ord(key[i % key_length])) % 256;

        b = S[i];
        S[i] = S[w];
        S[w] = b;
    }

    //decrypting, xor ciphertext with pseudorandom stream of bytes
    i = w = 0;
    for (index=0; index<length; index++) {
        i = (i+1) % 256;
        w = (w+ord(S[i])) % 256;

        b = S[i];
        S[i] = S[w];
        S[w] = b;

        K = ord(S[ord(S[i])+ord(S[w]) % 256]) % 256;

        data[index] = ord(data[index]) ^ K;
    }

    return data;
}

static wchar2ascii(str) {
    auto i;
    auto ret = "";
    for (i=i; i<strlen(str); i = i + 2) {
        ret = ret + str[i];
    }
    return ret;
}

static str2dword(str) {
    auto i, ret;
    ret = 0;
    for (i=3; i>=0; i--) {
        ret = ret*0x100 + ord(str[i]);
    }
    return ret;
}

static get_push_reg(addr) {
    // find argument (register name) for string decrypt function
    // in the form of "push %reg"
    while (GetMnem(addr) != "push") {
        addr = FindCode(addr, SEARCH_UP | SEARCH_NEXT);
    }

    return GetOpnd(addr, 0);
}

static get_reg_offset(addr, reg) {
    // find the instruction which sets the value of the pushed register
    // in the form of "lea, %reg, [ebp-offset]"
    do {
        addr = FindCode(addr, SEARCH_UP | SEARCH_NEXT);
    } while ((GetMnem(addr) != "lea") || (GetOpnd(addr,0) != reg));

    return GetOperandValue(addr,1);
}
```

```
static get_start_addr_of_string(addr, offset) {
    // find the address of the instruction which sets the first bytes of reconstructed string
    // in the form of "mov [ebp-offset], imm"
    do {
        addr = FindCode(addr, SEARCH_UP | SEARCH_NEXT);
    }
    while ((GetMnem(addr) != "mov") || (GetOpType(addr,1) != o_imm) || (GetOperandValue(addr,0) !=
offset));

    return addr;
}

static get_max_offset(addr1, addr2) {
    // find the max offset (aka end of the reconstructed string) across the "mov [ebp-offset], imm"
instructions
    auto addr, offset, max = -0x1fffffff;
    for (addr = addr1; addr < addr2; addr = FindCode(addr, SEARCH_DOWN | SEARCH_NEXT)) {
        if ((GetMnem(addr) == "mov") && (GetOpType(addr,1) == o_imm)) {
            offset = GetOperandValue(addr,0);
            if (offset > max) {
                max = offset;
            }
        }
    }
    return max;
}

static get_string_argument(ref) {
    auto addr, reg, ebp_offset, ebp_max_offset, arglength, argument;
    // get register name from the instruction "push %reg"
    reg = get_push_reg(ref);

    // get ebp offset stored in register by instruction "lea, %reg, [ebp-offset]"
    // aka start of the reconstructed string argument
    ebp_offset = get_reg_offset(ref, reg);

    // find the instruction which sets the first bytes of reconstructed string
    addr = get_start_addr_of_string(ref, ebp_offset);

    // find the max ebp offset, aka end of the reconstructed string argument
    ebp_max_offset = get_max_offset(addr, ref);
    arglength = ebp_max_offset - ebp_offset;
    argument = strfill('\x00', arglength);

    // reconstruct string argument from instruction like "mov [ebp-offset], value"
    auto offset, value;
    for (addr; addr < ref; addr=FindCode(addr,SEARCH_DOWN|SEARCH_NEXT)) {
        // instruction like "mov [ebp-offset], value"
        if ((GetMnem(addr) == "mov") && (GetOpType(addr,1) == o_imm)) {
            offset = GetOperandValue(addr,0);
            // set value for desired string argument starting at [ebp-ebp_offset]
            if (offset-ebp_offset >= 0) {
                value = GetOperandValue(addr,1);
                //Message(" %08lx\t[%d] = %08x\n", addr, offset-ebp_offset, value);
                //Message(" %08lx\t%s\n", addr, GetDisasm(addr));

                //convert dword value to bytes in string
                auto i;
                for (i=0; i<4; i++) {
                    argument[offset-ebp_offset+i] = value & 0xFF;
                    value = value>>8;
                }
            }
        }
    }

    return argument;
}

static count_xrefs(addr) {
    auto ref, count;
    count = 0;
    ref = RfirstB(addr);
    while (ref!=BADADDR) {
        count = count + 1;
        ref = RnextB(addr,ref);
```

```
    }

    return count;
}

static find_decrypt_function() {
    // find the address of the string decryption function with following conditions:
    //   it is short (up to 0x25 bytes)
    //   it is heavily used (at least 100 xrefs)
    //   it contains exactly one call instruction (for calling RC4 decryption routine)
    //   it contains exactly 5 push instructions (one push ebp from prologue and 4 arguments for RC4)
    auto func_start, func_end, found;
    found = 0;
    func_start = NextFunction(0);
    while ((func_start!=BADADDR) && (found == 0)) {
        func_start = NextFunction(func_start);
        func_end = FindFuncEnd(func_start);
        if (func_end-func_start < 0x25) {
            auto  addr, push_count, call_count;
            push_count = 0;
            call_count = 0;
            addr = func_start;
            while (addr < func_end) {
                if (GetMnem(addr) == "push") push_count = push_count + 1;
                if (GetMnem(addr) == "call") call_count = call_count + 1;
                addr = FindCode(addr, SEARCH_DOWN | SEARCH_NEXT);
            }
            if ((push_count == 5) && (call_count == 1) && (count_xrefs(func_start) > 100))  {
                found = 1;
            }
        }
    }
    return func_start;
}

static main() {
    Message("\n=====GandCrab String Decryptor=====\n\n");
    auto ea, ref;
    // string decrypt function
    ea = find_decrypt_function();
    // get first xref to calling decrypt function
    ref = RfirstB(ea);
    while (ref!=BADADDR) {
    //ref = 0x10006866;
        Message("%08lx: xref to decrypt function %08lx \n", ref, ea);
        // find xrefs to calling decrypt function
        if ((XrefType() == fl_CN) || (XrefType() == fl_CF)) {
            // reconstructs string argument
            auto argument = get_string_argument(ref);
            // parse parameters from string argument
            auto key_length = 0x10;
            auto key = substr(argument,0,key_length);
            auto data = substr(argument,key_length+8,-1);
            auto length = str2dword(substr(argument,key_length,key_length+4)) ^
str2dword(substr(argument,key_length+4,key_length+8));
            //print(length);
            // if the strings is too long, there may be an error, or it can be long binary data
            if (length < 0x10000) {
                auto text = RC4_decrypt(key, key_length, data, length);
                auto plaintext;
                // simple check for widechar string
                if (text[1] == '\x00') {
                    plaintext = wchar2ascii(text);
                } else {
                    plaintext = text;
                }
                // prints decrypted string to output windows
                Message("\"%s\" (length: 0x%x)\n\n", plaintext, strlen(plaintext));  ;
                // puts comment at the call of the decryption function
                MakeComm(ref, plaintext);
            }
        }
        ref = RnextB(ea,ref);
    }
}
```