Blok 2:
Reverse engineering

Názov tímu: 42

Členovia: Nicolas Macák

Veronika Szabóová

Petra Kirschová

V úlohe 1 a 2 sme použili statickú analýzu, kde sme sledovali dekompilovaný kód v Binary Ninja a z neho sme vyčítali správny kľúč. Pri úlohách 3-6 sme využili aj statickú aj dynamickú analýzu - sledovali sme najskôr dekompilovaný kód a potom sme v debuggeri gdb pozerali na to, ako sa počas behu programu mení vstup. V úlohe 7 stačila statická analýza – pozrieť sa na dekompilovaný kód a v ňom nájsť správnu adresu pamäťového miesta, ktoré je potrebné zmeniť.

Level 1

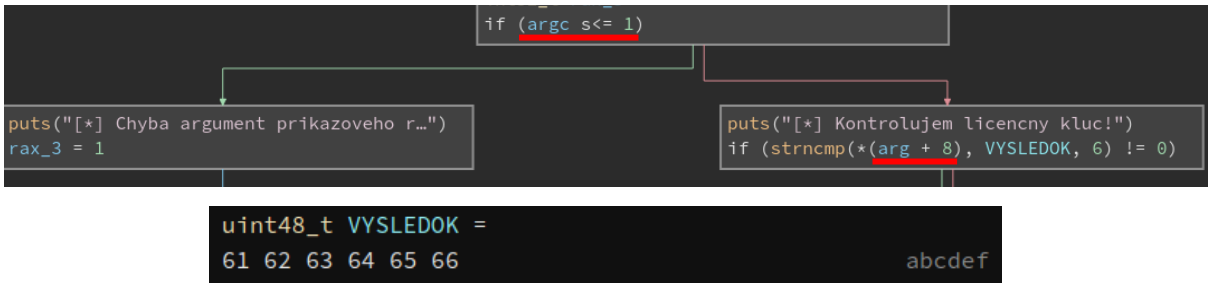
Po prvom spustení /re_level_1 sme videli, že program číta argumenty z príkazového riadku, z čoho sme usúdili, že kľúč bude potrebné zadať do programu ako argument.

/re_level_1

```
[*] Prijimam licencny kluc!
[*] Chyba argument prikazoveho riadku
```

V dekompilátori sme zobrazili program ako high-level kód. Dalo sa vyčítať, že program potrebuje 1 argument, ktorý sa porovnáva s výsledkom.

Dekompilátor




```
uint48_t VYSLEDOK =
61 62 63 64 65 66                                    abcdef
```

Premennú VYSLEDOK sa dalo zobraziť priamo ako string, ktorý sme vložili do príkazového riadku a získali sme flag.

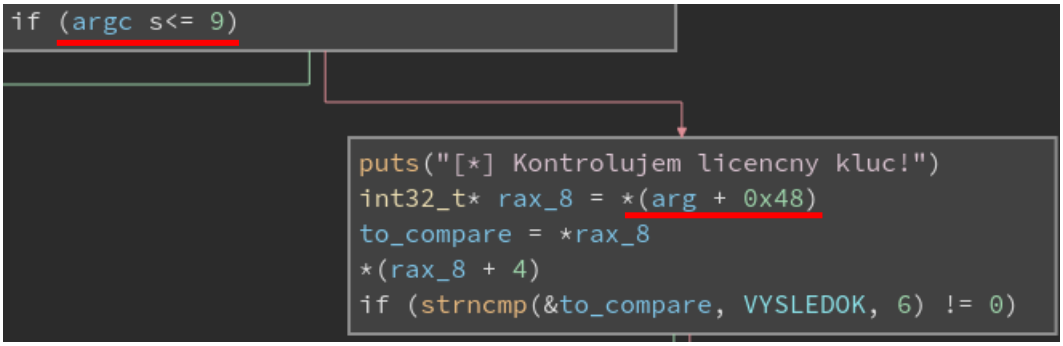
/re_level_1 abcdef

```
ctf@e6e483acc52b:~$ /re_level_1 abcdef
########################################
### BISzPP 2021
### Vitajte v ulohe /re_level_1!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise jeho obsah na obrazovku

[*] Prijimam licencny kluc!
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{TK5mcN-duncct-jgKTje-OhK2E4-FChsZf}ctf@e6e483acc52b:~$
```

Level 2

V leveli 2 sa tiež čítali argumenty z príkazového riadku, s tým rozdielom, že tu nestačilo pridať jeden argument, ale až 9 argumentov. Pri porovnávaní reťazcov sa v poli argumentov posúva o 0x48 (72) bitov, teda o 9 pamäťových miest. Kľúč sa preto musí nachádzať až na 9tej pozícii. Toto sme opäť vyčítali z dekompilátora.

Dekompilátor



Premenná VYSLEDOK obsahuje kľúč ako string, ktorý sme vložili do príkazového riadku ako 9ty argument. Zvyšné argumenty sme doplnili.

/re_level_2 * * * * * * * * xqelpp

```
ctf@9dc614bde7f7:~$ /re_level_2 * * * * * * * * xqelpp
########################################
### BISzPP 2021
### Vitajte v ulohe /re_level_2!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise jeho obsah na obrazovku

[*] Prijimam licencny kluc!
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{dWfNQ9-b4Ii9o-IVJ7j7-UgZg8A-093mU3}ctf@9dc614bde7f7:~$
```

Level 3

V tejto úlohe sa kľúč načítava zo súboru. Jeho názov (**ctf**) sme vyčítali z Binary Ninja.

Dekompilátor

```
int32_t rax_6 = open(data_214a, 0, 0)
int32_t rax_7
if (rax_6 s< 0)
```

```
data_214a:
                                63 74 66 00 00 00          ctf...
```

Výsledok teraz nedokážeme priamo preložiť na string, pretože je uložený v bajtovej podobe.

```
uint48_t VYSLEDOK =
92 84 85 86 87 9d
```

Okrem toho sme vyčítali, že súbor sa nečíta od začiatku, ale posúva sa o 8-znakový offset. Preto prvých 8 bajtov vyplníme znakmi, ktoré budú preskočené, a až po nich nasleduje kľúč.

```
lseek(rax_6, 8, 0)
read(rax_6, &var_16, 6)
int32_t var_20_1 = 0
```

Počas debuggovania pomocou gdb sme zistili, že obsah súboru ctf prechádza cez funkciu, ktorá ho cyklicky XOR-uje a nakoniec ho transformuje na reťazec **drspqk**.

92	64	d
84	72	r
85	73	s
86	70	p
87	71	q
9d	6b	k
	F6	

Teda súbor **ctf** bude obsahovať : `12345678drspqk`

/re_level_3

```
ctf@a27d24d343f1:~$ vim ctf
ctf@a27d24d343f1:~$ /re_level_3
########################################
### BISzPP 2021
### Vitajte v ulohe /re_level_3!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny
kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise je
ho obsah na obrazovku

[*] Prijimam licencny kluc!
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{eZ8lY6-O6A0c8-E4WaFm-2LoTQH-LQ3na3}ctf@a27d24d343f1:~$
```

Level 4

Funguje na podobnom princípe, ako level 3. Na začiatku sa program snaží otvoriť súbor „**hackerman**”. Ak sa to podarí, preskočí prvých 0x10 (16B), za tým očakáva kľúč dĺžky 8B.

Dekompilátor

```
int32_t rax_6 = open("hackerman", 0, 0)
int32_t rax_7
if (rax_6 s< 0)
```

```
lseek(rax_6, 0x10, 0)
read(rax_6, &var_20, 8)
```

Načítaný kľúč zo súboru sa porovnáva s reťazcom „**%,+..-06**”.

```
lea     rsi, [rel VYSLEDOK]  {"%,+..-06"}
```

Pri debuggovaní sme videli, že sa reťazec postupne menil na **trolling**.

gdb /re_level_4** pre reťazec **„%,+..-06“

```
RAX  0x7
RBX  0x56094af0e440 (__libc_csu_init) ← endbr64
RCX  0x7f5143c32142 (read+18) ← cmp    rax, -0x1000 /* 'H=' */
RDX  0x67
RDI  0x3
RSI  0x7ffd86b97d28 ← '%,+..-06trolling'
R8   0x1c
R9   0x21
R10  0x0
R11  0x246
R12  0x56094af0e160 (_start) ← endbr64
R13  0x7ffd86b97e30 ← 0x1
R14  0x0
R15  0x0
RBP  0x7ffd86b97d40 ← 0x0
RSP  0x7ffd86b97cf0 ← 0xc2
*RIP 0x56094af0e395 (main+332) ← add    dword ptr [rbp - 0x24], 1
```

Preto sme v súbore prvých 16 bajtov vyplnili a za ne sme dopísali kľúč **trolling**.

*Súbor **hackerman***

```
1234567890123456trolling
```

/re_level_4

```
[ctf@594df0b95989:~$ vim hackerman
[ctf@594df0b95989:~$ /re_level_4
########################################
### BISzPP 2021
### Vitajte v ulohe /re_level_4!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny
kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise je
ho obsah na obrazovku

[*] Prijimam licencny kluc!
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{D323Pe-5TRkhb-A1fufJ-toC4mH-eWitHW}ctf@594df0b95989:~$
```

Level 5

Kľúč sa v tejto úlohe tiež načítava zo súboru. Jeho názov je **feictf**. Program pri čítaní súboru preskočí prvých 0x20 (32B) a za nimi očakáva kľúč. Preto prvých 32B musíme vyplniť znakmi a až potom pridať kľúč.

Dekompilátor

```
int32_t rax_6 = open("feictf", 0, 0)
int32_t rax_7
if (rax_6 s< 0)
```

```
lseek(rax_6, 0x20, 0)
read(rax_6, &var_18, 8)
```

Do súboru sme najskôr uložili nájdený kľúč „**rfsagcep**“ a sledoval sme, ako sa pri debuggovaní mení.

```
lea     rsi, [rel VYSLEDOK]  {"rfsagcep"}
```

Zistili sme, že tento reťazec sa otočil, teda z „**rfsagcep“** vzniklo **„pecgasfr“**. Preto je potrebné, aby sme do súboru uložili otočený reťazec **„pecgasfr“**.

*Súbor **feictf***

```
123456789012345678901234567890123456789012pecgasfr
```

/re_level_5

```
ctf@49948b1942ff:~$ vim feictf
ctf@49948b1942ff:~$ /re_level_5
########################################
### BISzPP 2021
### Vitajte v ulohe /re_level_5!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny
kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise je
ho obsah na obrazovku

[*] Prijimam licencny kluc!
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{LOqUUz-gr5RtL-bxcj81-NcgtYB-ilNx32}ctf@49948b1942ff:~$
```

Level 6

Komunikačným kanálom bol v tejto úlohe socket, čo sme vyčítali z dekompilovaného kódu cez Binary Ninja. Program vytvorí vo file systéme súbor reprezentujúci socket s názvom „epic“, na ktorom počúva komunikáciu.

Dekompilátor

```
int32_t socket_fd = socket(AF_LOCAL, SOCK_STREAM, 0)
int16_t socket_addr = 1
void socket_file
memcpy(&socket_file, "epic", 5)
bind(socket_fd, &socket_addr, 0x10)
listen(socket_fd, 1)
int32_t socket_fd_2 = accept(socket_fd, 0, nullptr)
```

Pre zapisovanie na socket sme vytvorili C script *client.c*, ktorý sa spustil zároveň s programom *./re_level_6*.

client.c

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>

int main( int argc, char **argv ) {

      printf( "Zapis na socket:\n\n" );

      // inicializacia
      int socket_fd = socket(AF_LOCAL, SOCK_STREAM, 0); // file descriptor socketu

      struct sockaddr_un servaddr;
      bzero( &servaddr, sizeof( servaddr ) );            // vynulovanie adresy
      servaddr.sun_family = AF_LOCAL;                    // typ = AF_LOCAL
      strcpy( servaddr.sun_path, "epic" );               // nazov = "epic"

      // pripojenie na socket
      connect(socket_fd, (struct sockaddr *) &servaddr, sizeof(servaddr));

      // posiela sa po jednom znaku
      while( !feof( stdin ) ) {
         char send = getchar();
         write( socket_fd, &send , sizeof(char));
      }

      close(socket_fd);

      return 0;
}
```

zdroj: http://software.hpclab.ceid.upatras.gr/opsys/opsys/project/server_client.htm

Výsledok obsahuje reťazec v hex tvare, ktorý sa dá preložiť na string „**AJNA^RBD**“.

```
uint64_t VYSLEDOK = 0x4442525e414e4a41
```

Avšak pred porovnaním sa s reťazcom robili v programe rôzne úpravy, ktoré sme trasovali pomocou gdb debuggera. Pri poslaní reťazca „AJNA^RBD“ sme v debuggeri videli, že sa tento reťazec postupne modifikuje na „**jaejuyio“.**

gdb /re_level_6 pre „AJNA^RBD“

```
*RAX  0x55cf7e1b96b0 ◂— 'AJNA^RBD'
```

...

```
*RAX  0x55cf7e1b96b5 ◂— 0x444252 /* 'RBD' */
```

...

```
R8    0x55cf7e1b96b0 ◂— 0x60079756a65616a /* 'jaejuy' */
```

...

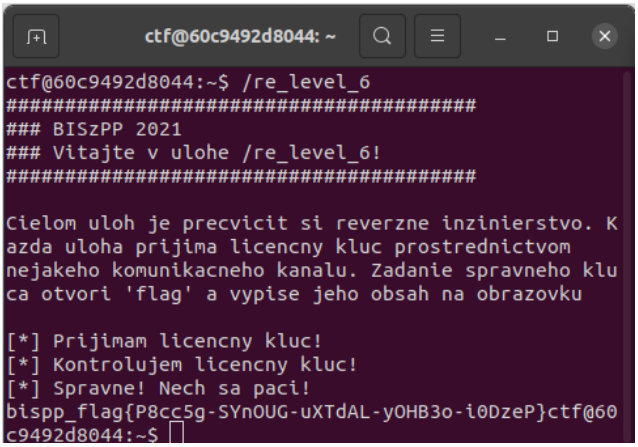
```
R8    0x55cf7e1b96b0 ◂— 'jaejuyio'
```

...

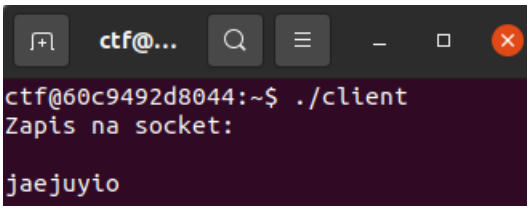
```
RDI   0x55cf7e1b96b0 ◂— 'jaejuyio'
RSI   0x55cf7c5e7010 (VYSLEDOK) ◂— 'AJNA^RBD'
```

Takto modifikovaný reťazec sa nakoniec porovnal s „AJNA^RBD“, čiže pre tento vstup sme flag nezískali, ale odhalili sme reálny kľúč ***jaejuyio***, pomocou ktorého sme dokázali vypísať flag, pretože reťazec „jaejuyio“ sa v programe zmenil na očakávaný „AJNA^RBD“.

/re_level_6



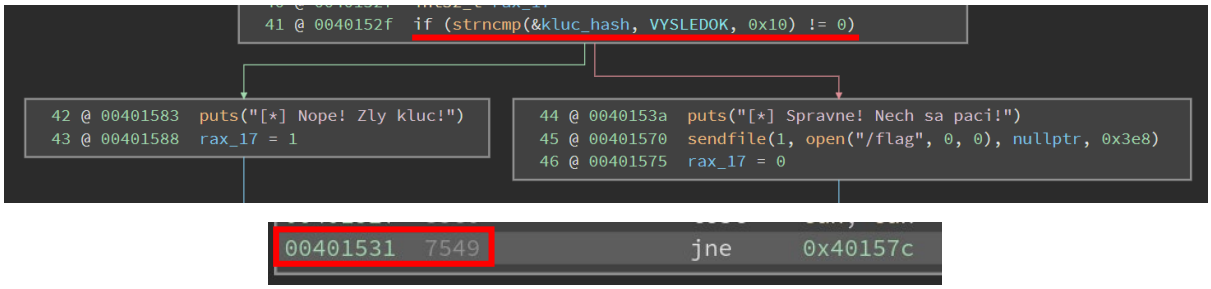
./client: jaejuyio



Level 7

V leveli 7 sme potrebovali zmeniť 1 bajt tak, aby sa obišla kontrola kľúča. V dekompilátori Binary Ninja sme videli, že zahashovaný vstup sa porovnáva s výsledkom, pričom keď strcmp vráti číslo rôzne od 0, program končí a nezískame flag.

Dekompilátor



V assembly je toto porovnanie vykonané pomocou inštrukcie **jne.** Keďže môžeme v programe zmeniť jeden bajt, ak prepíšeme **jne** na **je** dosiahneme, že program spadne do „pravej časti“, teda vypíše flag, keď bude zadaný kľúč nesprávny.

Adresa inštrukcie, ktorú treba zmeniť, je **0x401531**, opcode **jne = 75** sme zmenili na **je** = **74**.

/re_level_7** zmena bajtu na **74** na adrese **0x401531

```
ctf@d5f1eb07343f:~$ /re_level_7
########################################a##
### BISzPP 2021
### Vitajte v ulohe /re_level_7!
########################################

Cielom uloh je precvicit si reverzne inzinierstvo. Kazda uloha prijima licencny kluc prostrednictvom
nejakeho komunikacneho kanalu. Zadanie spravneho kluca otvori 'flag' a vypise jeho obsah na obrazovku
POZOR:
Tato uloha je specialna v tom ze na kontrolu licencneho kluca pouziva jednosmernu hashovaciu funkciu MD5,
co znamena ze dany kluc nevieme zreverzovat (mozeme iba skusat BF). Uloha vsak umozni modifikovat jeden
lubovolny bajt v programe (adresy su v tomto pripade staticke), cim mozeme napr. uplne zrusit kontrolu kl
uca
a nebude zalezat aku hodnotu zadame (inymi slovami program 'crackneme'). Choose wisely!

[*] Nastavujem opravnenia na zapis!
[*] Zadajte adresu bajtu, ktory si zelate zmenit v hexa tvare (napr. 4012bb):
401531
[*] Zadajte hodnotu bajtu v hexa tvare (napr. aa, b6, ff, ...) na adrese 0x401531:
74
[*] Upravujem program!
[*] Prijimam licencny kluc!
xxx
[*] Kontrolujem licencny kluc!
[*] Spravne! Nech sa paci!
bispp_flag{aSztFc-8MRqZz-t0bhDE-BEPp5C-s1ZEPL}ctf@d5f1eb07343f:~$
```