Binary Reverse Engineering

'DROP TABLE team;

Jan Harm Kuipers
j.h.kuipers@student.utwente.nl
(s1240838 - UTwente)

David Stritzl
d.l.stritzl@student.utwente.nl
(s1360752 - UTwente)

April 10, 2016

1 Introduction

Reverse Engineering in the field of Computer Science and software is the process of reversing the typical steps in producing a binary executable program. The typical software creation process starts with the programmer writing a program in a high-level programming language, for example C. This program is then compiled to an assembly format suitable for the target CPU instruction set. The assembly format already closely resembles the resulting binary structure but is still human readable. The final step is converting the assembly to binary machine instructions. There are several variations in this process, for example an interpreted programming language such as Python is not compiled but translated to machine instruction on the fly by an interpreter. But in all cases the result will be binary machine instructions for a specific CPU, also known as instruction set architecture (ISA).

This report describes the application of modern binary reverse engineering techniques on an example binary program in order to show the modern reverse engineering process. First we will outline the underlying techniques used by binary reverse engineering tools for reconstructing the assembly for an arbitrary binary program. Secondly, we will cover the tools used for the reverse engineering of the example program. The cases on which the reverse engineering techniques will be applied will be described in the next section. And the final sections will consist of the result and our conclusions.

2 Techniques

The first step in the binary reverse engineering process is the reconstruction of assembly from the binary machine code. This is usually the only step in the reverse engineering process. The second step, reproducing the high-level program code the program was (presumably) written in, is usually not possible since too much information is lost in the compilation process. Information such as variable/function names and even the programming language used is usually not available and, in addition to missing information, there might also be various combinations of high level programming language statements that result in virtually identical assembly. Although it might not be possible to reconstruct the original source code from a binary program, the reconstructed assembly code provides the information needed to infer what a binary program does and is, to a certain degree, human readable. Therefore we focus on binary disassembly techniques in the following summary of the binary reverse engineering techniques used.

There are two main categories of binary disassembly techniques: static disassembly and dynamic disassembly. Static disassembly involves processing a binary executable and converting the found opcodes to instructions. Dynamic disassembly techniques involve executing a binary program and monitoring the executed instructions. An advantage of dynamic disassembly is that only reachable instructions are part of disassembler output. At the same time, this is also a disadvantage because it cannot be determined if all possible execution traces of a binary program have been monitored and therefore instructions might be missed by the disassembler.

2.1 Static Analysis

Static disassemblers come in two main flavours: linear sweep disassemblers and recursive disassemblers.

Linear sweep disassemblers such as GNUs Objdump[1] parse a binary instruction in linear order. This kind of disassembler suffers from data embedded in the binary which might be incorrectly interpreted as instructions.

Recursive disassemblers follow the control flow of the program by taking into account branch and call instructions. But the control cannot always be determined due to indirect jumps etc., which complicates disassembly.

2.2 Dynamic Analysis

Dynamic disassembly techniques involve executing a binary program and monitoring the executed instructions. An advantage of dynamic disassembly is that only reachable instructions are part of disassembler output. At the same time, this is also a disadvantage because it cannot be determined if all possible execution traces of a binary program have been monitored and therefore instructions might be missed by the disassembler.

Several dynamic binary analysis classes can be identified, for example using a debugger with either software or hardware break points (BPs), or virtual machine introspection (VMI), where a binary is run inside a virtual machine which provides access to the execution from the outside.

To mitigate main drawback of dynamic analysis, the fact that only executed binary code is analysed, techniques such as multipath execution or dynamic symbolic execution are used. These techniques attempt to execute all execution paths by forking execution at branch points.

2.3 Obfuscation

Obfuscation of binaries is a process that attempts to modify binary executables in such a way that they become hard to disassemble while limiting the impact on performance and size of the binary. Linn and Debray [2] introduced two techniques for confusing disassemblers. The first technique is inserting junk bytes at unreachable destinations in the machine code, such as directly after a jump instruction. This technique mainly affects linear sweep disassemblers, because recursive disassemblers follow control flow instructions such as jumps. The second technique replaces call instructions for normal functions with a call to a branch function. The call to a branch function executes the original function, but modifies the return address of the function to return to a chosen number of addresses after the original call instruction. This way additional junk bytes can be inserted after every call instruction. A recursive disassembler expects the control flow to resume directly after the call instruction, but with use of a branch function this is no longer the case and the disassembler will interpret the junk bytes as valid instructions (the valid instructions start several addresses after the call instruction).

Several disassembly techniques have been proposed to counter these obfuscation techniques, for example by Kreugel et al. [3].

3 Tools

In this section we will mention the tools used to reverse engineer the example binaries. Several reverse engineering tools for binary disassembly exist, with IDA Pro [4] as probably the most widely used. However, for the reverse engineering examples in the next section Hopper [5] was used, since it is cheaper and provides sufficient capabilities for the chosen examples.

3.1 Hopper

Hopper is a disassembler/decompiler originally designed for OS X. It has support for 32 and 64-bit binaries in Windows (PE), Linux (ELF) and OS X (Mach-O) formats. It supports both ARM and Intel x86-64 ISAs. Disassembly is performed statically but there is support for integration with the GDB/LLDB debugger as well. Hopper also includes a decompiler which attempts to decompile the disassembled binary to C code. As of April 2015, Hopper uses the Capstone framework [6] as its main disassembly engine.

3.2 GDB/LLDB

The Gnu Debugger (GDB) [7] and LLDB Debugger (LLDB) [8] are both debuggers that support ARM and Intel X86-64 ISAs. GDB runs on UNIX-like systems and LLDB on Linux, Windows and OS X.

4 Test Cases

For this report, multiple test cases are used. Firstly, some of our own code was used to get familiar with the different reverse engineering techniques and tools. Secondly, a so-called Crackme[9] challenge was used, where the goal is to reverse engineer and “crack” a binary executable.

4.1 Own Code

To get familiar with the techniques and tools, a program for path finding in mazes was used. The code is written in modern C++ and is well-tested by a group of fellow students. It was compiled without symbols (for as far that is possible with CLANG[10]) to make the disassembly process more challenging. Since the entire code is too large, this section focuses on only one piece of the code, that is an implementation of the depth-first search algorithm for mazes. For the analysis of this program, only Hopper was used.

4.2 Crackme

Crackme challenges exists in various forms, but all have in common that they require binary reverse engineering to solve them. The Crackme challenge solved in this report consisted of a binary program for checking serial numbers [11]. A specific serial number activates several features when checked by the program. The goal of the challenge is to generate serial numbers which activate arbitrary features when checked by the provided serial checker. The challenge consists of two parts, the first is to make the serial checker accept a serial by patching the binary program. The second is to reverse engineer the serial checker algorithm in order to generate serial numbers with arbitrary features turned on or off.

5 Results

5.1 Own Code

In Hopper, the function for depth-first was found using the string view of Hopper, which showed a reference from the procedure sub_100003d70 to string "DFS [...]". The rest of the program was analysed using the disassembly and CFG views of Hopper. The procedure starts with a normal function prologue that saves the stack pointer, allocates and, if possible, initializes variables.

```
0000000100003d70         push       rbp
   ; XREF=sub_1000022b0+187
0000000100003d71         mov        rbp, rsp
0000000100003d74         push       r15
0000000100003d76         push       r14
0000000100003d78         push       r13
0000000100003d7a         push       r12
0000000100003d7c         push       rbx
0000000100003d7d         sub        rsp, 0x58
0000000100003d81         mov        r14, rdi
0000000100003d84         mov        rax, qword [ds:imp___got____stack_chk_guard]
0000000100003d8b         mov        rax, qword [ds:rax]
0000000100003d8e         mov        qword [ss:rbp+var_30], rax
0000000100003d92         lea        rax, qword [ds:r14+0x282d]
0000000100003d99         xor        ecx, ecx
0000000100003d9b         nop        dword [ds:rax+rax]
```

The code following this is a nested loop that initializes some array value to 0. This array represents the grid for the maze, since the loops iterate to $0x28 = 40$, which is the size of the maze in the program. The inner loop seems to be partially unrolled by the optimizer and executes 5 iterations per loop cycle instead of the normal 1.

```
0000000100003da0         mov        edx, 0x28
   ; XREF=sub_100003d70+130
0000000100003da5         mov        rsi, rax
0000000100003da8         nop        dword [ds:rax+rax]

0000000100003db0         mov        word [ds:rsi-0x2801], 0x0
   ; XREF=sub_100003d70+117
0000000100003db9         mov        word [ds:rsi-0x1e01], 0x0
0000000100003dc2         mov        word [ds:rsi-0x1401], 0x0
0000000100003dcb         mov        word [ds:rsi-0xa01], 0x0
0000000100003dd4         mov        word [ds:rsi-1], 0x0
0000000100003dda         add        rsi, 0x3200
0000000100003de1         add        rdx, 0xfffffffffffffffb
0000000100003de5         jne        0x100003db0

0000000100003de7         inc        rcx
0000000100003dea         add        rax, 0x40
0000000100003dee         cmp        rcx, 0x28
0000000100003df2         jne        0x100003da0
```

Next, some more variables are initialized, including a dequeue, which is used a stack for the depth-first search algorithm. The fact that C++ symbols were not removed completely, makes the analysis easier.

```
0000000100003df4         xorps      xmm0, xmm0
0000000100003df7         movaps     xmmword [ss:rbp+var_60], xmm0
0000000100003dfb         movaps     xmmword [ss:rbp+var_70], xmm0
0000000100003dff         movaps     xmmword [ss:rbp+var_80], xmm0
0000000100003e03         lea        rdi, qword [ss:rbp+var_80]
0000000100003e07         call       imp___stubs___ZNSt3__15dequeIP11GridElementNS_9allocator
   IS2_EEE19__add_back_capacityEv
    ; std::__1::deque<GridElement*, std::__1::allocator<GridElement*> >::__add_back_capacity()
0000000100003e0c         mov        rcx, qword [ss:rbp+var_58]
0000000100003e10         mov        rax, qword [ss:rbp+var_78]
0000000100003e14         mov        rdx, qword [ss:rbp+var_60]
0000000100003e18         lea        rdi, qword [ds:rcx+rdx]
0000000100003e1c         mov        rbx, rdi
0000000100003e1f         shr        rbx, 0x9
0000000100003e23         mov        rax, qword [ds:rax+rbx*8]
0000000100003e27         and        rdi, 0x1ff
0000000100003e2e         mov        qword [ds:rax+rdi*8], r14
0000000100003e32         mov        rax, rcx
0000000100003e35         inc        rax
0000000100003e38         mov        qword [ss:rbp+var_58], rax
0000000100003e3c         je         0x1000040d7
```

After this, some loop seems to be initialized, although, without context, there is not much we can learn from this. When looking at the original source, this corresponds to a `while(!stack.empty())` and a `stack.top()` which are optimized away in the binary and the initialization of the first variables in the loop.

```
0000000100003e42         add        r14, 0x18fc0
0000000100003e49         mov        rsi, qword [ss:rbp+var_78]
0000000100003e4d         mov        rbx, qword [ds:rsi+rbx*8]
0000000100003e51         mov        r13, qword [ds:rbx+rdi*8]
0000000100003e55         mov        word [ds:r13+0x2c], 0x101
0000000100003e5c         mov        r15d, 0x1
0000000100003e62         cmp        r13, r14
0000000100003e65         je         0x100004027

0000000100003e6b         lea        rdi, qword [ds:r13+0x2d]
0000000100003e6f         nop
```

The next piece does some checks based on a condition and counts those, which seems to be the amount of neighbouring grid cells in the maze based on the state of the current cell. When compared to the source, one can see that a loop was used originally, but it was again unrolled by the optimizer. Also, the unrolled loop cycles are optimized even more, which leads to substantial differences in the assembly in the different cases.

```
0000000100003e70         cmp        byte [ds:r13+0x28], 0x0
   ; XREF=sub_100003d70+689
0000000100003e75         mov        r12d, 0x0
0000000100003e7b         jne        0x100003ea0

0000000100003e7d         mov        rbx, qword [ds:r13+8]
0000000100003e81         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003e85         mov        r12d, 0x0
0000000100003e8b         jne        0x100003ea0

0000000100003e8d         mov        dword [ss:rbp+var_40], 0x0
```

```
0000000100003e94         mov        r12d, 0x1
0000000100003e9a         nop        word [ds:rax+rax]

0000000100003ea0         cmp        byte [ds:r13+0x29], 0x0
   ; XREF=sub_100003d70+267, sub_100003d70+283
0000000100003ea5         jne        0x100003ed0

0000000100003ea7         mov        rbx, qword [ds:r13+0x10]
0000000100003eab         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003eaf         jne        0x100003ed0

0000000100003eb1         lea        r8d, dword [ds:r12+1]
0000000100003eb6         shl        r12, 0x2
0000000100003eba         lea        rbx, qword [ss:rbp+var_40]
0000000100003ebe         or         r12, rbx
0000000100003ec1         mov        dword [ds:r12], 0x1
0000000100003ec9         mov        r12d, r8d
0000000100003ecc         nop        dword [ds:rax]

0000000100003ed0         cmp        byte [ds:r13+0x2a], 0x0
   ; XREF=sub_100003d70+309, sub_100003d70+319
0000000100003ed5         jne        0x100003ef0

0000000100003ed7         mov        rbx, qword [ds:r13+0x18]
0000000100003edb         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003edf         jne        0x100003ef0

0000000100003ee1         movsxd     rbx, r12d
0000000100003ee4         inc        r12d
0000000100003ee7         mov        dword [ss:rbp+rbx*4+var_40], 0x2
0000000100003eef         nop

0000000100003ef0         cmp        byte [ds:r13+0x2b], 0x0
   ; XREF=sub_100003d70+357, sub_100003d70+367
0000000100003ef5         jne        0x100003f10

0000000100003ef7         mov        rbx, qword [ds:r13+0x20]
0000000100003efb         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003eff         jne        0x100003f10

0000000100003f01         movsxd     rbx, r12d
0000000100003f04         inc        r12d
0000000100003f07         mov        dword [ss:rbp+rbx*4+var_40], 0x3
0000000100003f0f         nop
```

Based on the count the previous section (r12d) a value is conditionally added to the dequeue. This value is one of the grid cells checked in the last section chosen by `rand()`. When compared with the original code, some of the functions seem to be inlined since some of the conditionals in this section are not in the original code.

```
0000000100003f10         test       r12d, r12d
   ; XREF=sub_100003d70+389, sub_100003d70+399
0000000100003f13         je         0x100003f90

0000000100003f15         call       imp___stubs__rand
0000000100003f1a         cdq
0000000100003f1b         idiv       r12d
0000000100003f1e         movsxd     rax, edx
0000000100003f21         mov        eax, dword [ss:rbp+rax*4+var_40]
0000000100003f25         mov        r12, qword [ds:r13+rax*8+8]
0000000100003f2a         mov        rcx, qword [ss:rbp+var_78]
0000000100003f2e         mov        rax, qword [ss:rbp+var_70]
0000000100003f32         sub        rax, rcx
```

```
0000000100003f35         mov        esi, 0x0
0000000100003f3a         je         0x100003f46

0000000100003f3c         shl        rax, 0x6
0000000100003f40         dec        rax
0000000100003f43         mov        rsi, rax

0000000100003f46         mov        rdx, qword [ss:rbp+var_60]
   ; XREF=sub_100003d70+458
0000000100003f4a         mov        rax, qword [ss:rbp+var_58]
0000000100003f4e         sub        rsi, rdx
0000000100003f51         cmp        rsi, rax
0000000100003f54         jne        0x100003f6b

0000000100003f56         lea        rdi, qword [ss:rbp+var_80]
0000000100003f5a         call       imp___stubs___ZNSt3__15dequeIP11GridElementNS_9allocator
   IS2_EEE19__add_back_capacityEv
   ; std::__1::deque<GridElement*, std::__1::allocator<GridElement*> >::__add_back_capacity()
0000000100003f5f         mov        rax, qword [ss:rbp+var_58]
0000000100003f63         mov        rcx, qword [ss:rbp+var_78]
0000000100003f67         mov        rdx, qword [ss:rbp+var_60]

0000000100003f6b         add        rdx, rax
   ; XREF=sub_100003d70+484
0000000100003f6e         mov        rsi, rdx
0000000100003f71         shr        rsi, 0x9
0000000100003f75         mov        rcx, qword [ds:rcx+rsi*8]
0000000100003f79         and        rdx, 0x1ff
0000000100003f80         mov        qword [ds:rcx+rdx*8], r12
0000000100003f84         inc        rax
0000000100003f87         mov        qword [ss:rbp+var_58], rax
0000000100003f8b         jmp        0x100003fe0
```

For the other case, for which r12d is 0, a value is removed from the stack. Again, the stack.pop() from the original code seems to be inlined, so inline operator::delete() is called.

```
0000000100003f90         mov        byte [ds:rdi], 0x0
   ; XREF=sub_100003d70+419
0000000100003f93         mov        qword [ss:rbp+var_58], rcx
0000000100003f97         mov        r8, qword [ss:rbp+var_70]
0000000100003f9b         mov        rdi, r8
0000000100003f9e         sub        rdi, rsi
0000000100003fa1         mov        esi, 0x0
0000000100003fa6         je         0x100003fb2

0000000100003fa8         shl        rdi, 0x6
0000000100003fac         dec        rdi
0000000100003faf         mov        rsi, rdi

0000000100003fb2         mov        edi, 0x1
   ; XREF=sub_100003d70+566
0000000100003fb7         sub        rdi, rax
0000000100003fba         add        rdi, rsi
0000000100003fbd         sub        rdi, rdx
0000000100003fc0         cmp        rdi, 0x400
0000000100003fc7         mov        rax, rcx
0000000100003fca         jb         0x100003fe0

0000000100003fcc         mov        rdi, qword [ds:r8-8]
0000000100003fd0         call       imp___stubs___ZdlPv
   ; operator delete(void*)
0000000100003fd5         add        qword [ss:rbp+var_70], 0xfffffffffffffff8
0000000100003fda         mov        rax, qword [ss:rbp+var_58]
```

```
0000000100003fde         nop
```

In the the following section the dequeue size (`rax`) is checked again. If it is zero, the loop will be stopped. Otherwise the variables for the next loop cycle will be initialized. This has to do with optimization, since the initialization of those variables happened at the beginning of the loop cycle in the original code.

```
0000000100003fe0         test       rax, rax
   ; XREF=sub_100003d70+539, sub_100003d70+602
0000000100003fe3         je         0x1000040d7

0000000100003fe9         mov        rsi, qword [ss:rbp+var_78]
0000000100003fed         mov        rdx, qword [ss:rbp+var_60]
0000000100003ff1         lea        rcx, qword [ds:rax-1]
0000000100003ff5         lea        rbx, qword [ds:rax+rdx-1]
0000000100003ffa         mov        rdi, rbx
0000000100003ffd         shr        rdi, 0x9
0000000100004001         mov        rdi, qword [ds:rsi+rdi*8]
0000000100004005         and        rbx, 0x1ff
000000010000400c         mov        r13, qword [ds:rdi+rbx*8]
0000000100004010         mov        word [ds:r13+0x2c], 0x101
0000000100004017         lea        rdi, qword [ds:r13+0x2d]
000000010000401b         inc        r15d
000000010000401e         cmp        r13, r14
0000000100004021         jne        0x100003e70
```

Lastly, some final status information is printed about the number of states or the path length. Interestingly enough, the check if a path has been found, has been optimized away due to the different exit points of the loop (when a path has been found or when all possibilities have been tried).

```
0000000100004027         mov        rdi, qword [ds:imp___got___ZNSt3__14coutE]
   ; XREF=sub_100003d70+245
000000010000402e         lea        rsi, qword [ds:0x10048cb4c]
   ; "DFS(states: "
0000000100004035         mov        edx, 0xc
000000010000403a         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
   std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
   (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000403f         mov        rdi, rax
0000000100004042         mov        esi, r15d
0000000100004045         call       imp___stubs___ZNSt3__113basic_ostreamIcNS_11char_traits
   IcEEElsEi
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::operator<<(int)
000000010000404a         lea        rsi, qword [ds:0x10048cb59]
   ; ", path length: "
0000000100004051         mov        edx, 0xf
0000000100004056         mov        rdi, rax
0000000100004059         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000405e         mov        rsi, qword [ss:rbp+var_58]
0000000100004062         mov        rdi, rax
0000000100004065         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEElsEm
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::operator<<(unsigned long)
000000010000406a         lea        rsi, qword [ds:0x10048cb69]
   ; ")"
```

```
0000000100004071         mov        edx, 0x1
0000000100004076         mov        rdi, rax
0000000100004079         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
   std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
   (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000407e         mov        rbx, rax
0000000100004081         mov        rax, qword [ds:rbx]
0000000100004084         mov        rsi, qword [ds:rax-0x18]
0000000100004088         add        rsi, rbx
000000010000408b         lea        rdi, qword [ss:rbp+var_48]
000000010000408f         call       imp___stubs___ZNKSt3__18ios_base6getlocEv
   ; std::__1::ios_base::getloc() const
0000000100004094         mov        rsi, qword [ds:imp___got___ZNSt3__15ctypeIcE2idE]
000000010000409b         lea        rdi, qword [ss:rbp+var_48]
000000010000409f         call       imp___stubs___ZNKSt3__16locale9use_facetERNS0_2idE
   ; std::__1::locale::use_facet(std::__1::locale::id&) const
00000001000040a4         mov        rcx, qword [ds:rax]
00000001000040a7         mov        rcx, qword [ds:rcx+0x38]
00000001000040ab         mov        esi, 0xa
00000001000040b0         mov        rdi, rax
00000001000040b3         call       rcx
00000001000040b5         mov        r14b, al
00000001000040b8         lea        rdi, qword [ss:rbp+var_48]
00000001000040bc         call       imp___stubs___ZNSt3__16localeD1Ev
   ; std::__1::locale::~locale()
00000001000040c1         movsx      esi, r14b
00000001000040c5         mov        rdi, rbx
00000001000040c8         call       imp___stubs___ZNSt3__113basic_ostreamIcNS_11char_traits
   IcEEE3putEc
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::put(char)
00000001000040cd         mov        rdi, rbx
00000001000040d0         call       imp___stubs___ZNSt3__113basic_ostreamIcNS_11char_traits
   IcEEE5flushEv
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::flush()
00000001000040d5         jmp        0x100004146
```

And the same for the other case, if no path was found:

```
00000001000040d7         mov        rdi, qword [ds:imp___got___ZNSt3__14coutE]
   ; XREF=sub_100003d70+204, sub_100003d70+627
00000001000040de         lea        rsi, qword [ds:0x10048cb39]
   ; "DFS(end not found)"
00000001000040e5         mov        edx, 0x12
00000001000040ea         call       imp___stubs___ZNSt3__124__put_character_sequenceIc
   NS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
00000001000040ef         mov        rbx, rax
00000001000040f2         mov        rax, qword [ds:rbx]
00000001000040f5         mov        rsi, qword [ds:rax-0x18]
00000001000040f9         add        rsi, rbx
00000001000040fc         lea        rdi, qword [ss:rbp+var_50]
0000000100004100         call       imp___stubs___ZNKSt3__18ios_base6getlocEv
   ; std::__1::ios_base::getloc() const
0000000100004105         mov        rsi, qword [ds:imp___got___ZNSt3__15ctypeIcE2idE]
000000010000410c         lea        rdi, qword [ss:rbp+var_50]
0000000100004110         call       imp___stubs___ZNKSt3__16locale9use_facetERNS0_2idE
   ; std::__1::locale::use_facet(std::__1::locale::id&) const
0000000100004115         mov        rcx, qword [ds:rax]
0000000100004118         mov        rcx, qword [ds:rcx+0x38]
000000010000411c         mov        esi, 0xa
```

```
0000000100004121         mov        rdi, rax
0000000100004124         call       rcx
0000000100004126         mov        r14b, al
0000000100004129         lea        rdi, qword [ss:rbp+var_50]
000000010000412d         call       imp___stubs___ZNSt3__16localeD1Ev
   ; std::__1::locale::~locale()
0000000100004132         movsx      esi, r14b
0000000100004136         mov        rdi, rbx
0000000100004139         call       imp___stubs___ZNSt3__113basic_ostreamIcNS_11char_traits
   IcEEE3putEc
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::put(char)
000000010000413e         mov        rdi, rbx
0000000100004141         call       imp___stubs___ZNSt3__113basic_ostreamIcNS_11char_traits
   IcEEE5flushEv
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::flush()
```

The rest of the procedure contains the function epilogue including the destruction and deallocation of the local variables. Especially the dequeue takes up a lot of space because its destructor is inlined there. Since the epilogue is very long and it only contains deconstruction code, it is not included here, although it can be seen in appendix A from 0x100004146 to 0x1000043f5.

The original source and the corresponding assembly, disassembled by Hopper, for this code can be found in appendix A.

5.2 Crackme

Just as for the last section, we also started out with static analysis using Hopper. Since the binary was compiled without symbols, the entry function `main(...)` had to be found first.

```
                        EntryPoint:
0000000000400660         xor        ebp, ebp
0000000000400662         mov        r9, rdx
0000000000400665         pop        rsi                                         ; argument #2
0000000000400666         mov        rdx, rsp                                    ; argument #3
0000000000400669         and        rsp, 0xfffffffffffffff0
000000000040066d         push       rax
000000000040066e         push       rsp
000000000040066f         mov        r8, 0x400e20
0000000000400676         mov        rcx, 0x400db0
000000000040067d         mov        rdi, 0x400cb4
   ; argument "main" for method j___libc_start_main
0000000000400684         call       j___libc_start_main
0000000000400689         hlt
                        ; endp
```

Hopper automatically detects the binary's entry points by analyzing the binary header (ELF). Here we see a call to `__libc_start_main(...)`, where this first argument (0x400cb4) is the pointer to `main(...)`.

```
                        sub_400cb4:
0000000000400cb4         push       rbp
   ; XREF=EntryPoint+29
0000000000400cb5         mov        rbp, rsp
0000000000400cb8         sub        rsp, 0x30
0000000000400cbc         mov        dword [ss:rbp+var_24], edi
0000000000400cbf         mov        qword [ss:rbp+var_30], rsi
0000000000400cc3         mov        rax, qword [fs:0x28]
0000000000400ccc         mov        qword [ss:rbp+var_8], rax
0000000000400cd0         xor        eax, eax
```

```
0000000000400cd2         mov        edi, 0x400ef8
   ; "Crackme/keygenme by Dennis Yurichev, http://challenges.re/74",
    argument "s" for method j_puts
0000000000400cd7         call       j_puts
0000000000400cdc         mov        edi, 0xa
   ; argument "c" for method j_putchar
0000000000400ce1         call       j_putchar
0000000000400ce6         cmp        dword [ss:rbp+var_24], 0x1
0000000000400cea         jne        0x400d00

0000000000400cec         mov        edi, 0x400f35
   ; "Command line: <serial number>", argument "s" for method j_puts
0000000000400cf1         call       j_puts
0000000000400cf6         mov        edi, 0x0
   ; argument "status" for method j_exit
0000000000400cfb         call       j_exit

0000000000400d00         mov        rax, qword [ss:rbp+var_30]
   ; XREF=sub_400cb4+54
0000000000400d04         add        rax, 0x8
0000000000400d08         mov        rax, qword [ds:rax]
0000000000400d0b         add        rax, 0x1e
0000000000400d0f         mov        edx, 0xb
   ; argument "n" for method j_memcmp
0000000000400d14         mov        esi, 0x400f53
   ; "HELLO-HELLO", argument "s2" for method j_memcmp
0000000000400d19         mov        rdi, rax
   ; argument "s1" for method j_memcmp
0000000000400d1c         call       j_memcmp
0000000000400d21         test       eax, eax
0000000000400d23         je         0x400d39

0000000000400d25         mov        edi, 0x400f5f
   ; "SN format is incorrect", argument "s" for method j_puts
0000000000400d2a         call       j_puts
0000000000400d2f         mov        edi, 0x0
   ; argument "status" for method j_exit
0000000000400d34         call       j_exit

0000000000400d39         mov        rax, qword [ss:rbp+var_30]
   ; XREF=sub_400cb4+111
0000000000400d3d         add        rax, 0x8
0000000000400d41         mov        rax, qword [ds:rax]
0000000000400d44         lea        rdx, qword [ss:rbp+var_20]
0000000000400d48         mov        rsi, rdx
   ; argument #2 for method sub_400bb5
0000000000400d4b         mov        rdi, rax
   ; argument #1 for method sub_400bb5
0000000000400d4e         call       sub_400bb5
0000000000400d53         cmp        eax, 0xffffffff
0000000000400d56         jne        0x400d6c

0000000000400d58         mov        edi, 0x400f5f
   ; "SN format is incorrect", argument "s" for method j_puts
0000000000400d5d         call       j_puts
0000000000400d62         mov        edi, 0x0
   ; argument "status" for method j_exit
0000000000400d67         call       j_exit

0000000000400d6c         lea        rax, qword [ss:rbp+var_20]
   ; XREF=sub_400cb4+162
0000000000400d70         mov        rdi, rax
   ; argument #1 for method sub_40085e
0000000000400d73         call       sub_40085e
```

```
0000000000400d78         test       al, al
0000000000400d7a         je         0x400d88

0000000000400d7c         mov        edi, 0x400f76
   ; "SN valid", argument "s" for method j_puts
0000000000400d81         call       j_puts
0000000000400d86         jmp        0x400d92

0000000000400d88         mov        edi, 0x400f7f
   ; "SN is not valid", argument "s" for method j_puts, XREF=sub_400cb4+198
0000000000400d8d         call       j_puts

0000000000400d92         mov        eax, 0x0
   ; XREF=sub_400cb4+210
0000000000400d97         mov        rcx, qword [ss:rbp+var_8]
0000000000400d9b         xor        rcx, qword [fs:0x28]
0000000000400da4         je         0x400dab

0000000000400da6         call       j___stack_chk_fail

0000000000400dab         leave
   ; XREF=sub_400cb4+240
0000000000400dac         ret
                        ; endp
```

After the function prologue, where `argc` and `argv` are saved at `rbp+var_24` and `rbp+var_30`, respectively, there are a few call to libc `puts(...)` and `putchar(...)` to print out some information about the Crackme. Then, if no argument is supplied, ie. `argc == 1`, the application will print usage instructions and exit.

At `0x400d1c` there is a call to `memcpy(...)` with `argv[1] + 30`, the constant `"HELLO-HELLO"` and the length of that constant as arguments. `argv[1] + 30` is the 31st character position of the inserted serial. If these two do not match, the application will print an error and exit.

Next, there is a call to `sub_400bb5` with `argv[1]` and an array of 24 bytes as argument. If the result of this call is -1, the application will, again, print an error and exit.

Lastly, there is a call to `sub_40085e` with the array from the last call as an argument. The call returns a boolean and depending on this value either `"SN valid"` or `"SN is not valid"` will be printed.

Below, the equivalent C code is shown.

```c
char const *kHelloHello = "HELLO-HELLO";

int main(int argc, char *argv[]) { // sub_400cb4
    puts("Crackme/keygenme by Dennis Yurichev, http://challenges.re/74");
    putchar('\n');

    if(argc == 1) {
        puts("Command line: <serial number>");
        exit(0);
    }

    if(memcmp(&argv[1][30], kHelloHello, sizeof(kHelloHello) / sizeof(*kHelloHello)) != 0) {
        puts("SN format is incorrect");
        exit(0);
    }

    uint8_t result[24];
    if(sub_400bb5(argv[1], result) == -1) {
        puts("SN format is incorrect");
        exit(0);
    }

    if(sub_40085e(result)) {
        puts("SN valid");
    } else {
        puts("SN is not valid");
    }
}
```

sub_40085e is probably the most important function here, but we first have to look at sub_400bb5, since it probably does some kind of transformation to the input serial.

```
                     sub_400bb5:
0000000000400bb5         push       rbp
   ; XREF=sub_400cb4+154
0000000000400bb6         mov        rbp, rsp
0000000000400bb9         sub        rsp, 0x20
0000000000400bbd         mov        qword [ss:rbp+var_18], rdi
0000000000400bc1         mov        qword [ss:rbp+var_20], rsi
0000000000400bc5         mov        dword [ss:rbp+var_10], 0x0
0000000000400bcc         jmp        0x400bfb

0000000000400bce         mov        edx, dword [ss:rbp+var_10]
   ; XREF=sub_400bb5+74
0000000000400bd1         mov        eax, edx
0000000000400bd3         add        eax, eax
0000000000400bd5         add        eax, edx
0000000000400bd7         add        eax, eax
0000000000400bd9         cdqe
0000000000400bdb         lea        rdx, qword [ds:rax+5]
0000000000400bdf         mov        rax, qword [ss:rbp+var_18]
0000000000400be3         add        rax, rdx
0000000000400be6         movzx      eax, byte [ds:rax]
0000000000400be9         cmp        al, 0x2d
0000000000400beb         je         0x400bf7

0000000000400bed         mov        eax, 0xffffffff
0000000000400bf2         jmp        0x400cb2

0000000000400bf7         add        dword [ss:rbp+var_10], 0x1
   ; XREF=sub_400bb5+54

0000000000400bfb         cmp        dword [ss:rbp+var_10], 0x6
   ; XREF=sub_400bb5+23
0000000000400bff         jle        0x400bce

0000000000400c01         mov        dword [ss:rbp+var_C], 0x0
0000000000400c08         mov        dword [ss:rbp+var_8], 0x0
0000000000400c0f         jmp        0x400ca3

0000000000400c14         mov        edx, dword [ss:rbp+var_8]
   ; XREF=sub_400bb5+242
0000000000400c17         mov        eax, edx
0000000000400c19         add        eax, eax
0000000000400c1b         add        eax, edx
0000000000400c1d         add        eax, eax
0000000000400c1f         movsxd     rdx, eax
0000000000400c22         mov        rax, qword [ss:rbp+var_18]
0000000000400c26         add        rax, rdx
0000000000400c29         mov        rdi, rax
   ; argument #1 for method sub_400ad9
0000000000400c2c         call       sub_400ad9
```

```
0000000000400c31         mov        dword [ss:rbp+var_4], eax
0000000000400c34         cmp        dword [ss:rbp+var_4], 0xffffffff
0000000000400c38         jne        0x400c41

0000000000400c3a         mov        eax, 0xffffffff
0000000000400c3f         jmp        0x400cb2

0000000000400c41         cmp        dword [ss:rbp+var_4], 0xffffff
   ; XREF=sub_400bb5+131
0000000000400c48         jle        0x400c51

0000000000400c4a         mov        eax, 0xffffffff
0000000000400c4f         jmp        0x400cb2

0000000000400c51         mov        eax, dword [ss:rbp+var_C]
   ; XREF=sub_400bb5+147
0000000000400c54         lea        edx, dword [ds:rax+1]
0000000000400c57         mov        dword [ss:rbp+var_C], edx
0000000000400c5a         movsxd     rdx, eax
0000000000400c5d         mov        rax, qword [ss:rbp+var_20]
0000000000400c61         add        rdx, rax
0000000000400c64         mov        eax, dword [ss:rbp+var_4]
0000000000400c67         mov        byte [ds:rdx], al
0000000000400c69         mov        eax, dword [ss:rbp+var_C]
0000000000400c6c         lea        edx, dword [ds:rax+1]
0000000000400c6f         mov        dword [ss:rbp+var_C], edx
0000000000400c72         movsxd     rdx, eax
0000000000400c75         mov        rax, qword [ss:rbp+var_20]
0000000000400c79         add        rdx, rax
0000000000400c7c         mov        eax, dword [ss:rbp+var_4]
0000000000400c7f         sar        eax, 0x8
0000000000400c82         mov        byte [ds:rdx], al
0000000000400c84         mov        eax, dword [ss:rbp+var_C]
0000000000400c87         lea        edx, dword [ds:rax+1]
0000000000400c8a         mov        dword [ss:rbp+var_C], edx
0000000000400c8d         movsxd     rdx, eax
0000000000400c90         mov        rax, qword [ss:rbp+var_20]
0000000000400c94         add        rdx, rax
0000000000400c97         mov        eax, dword [ss:rbp+var_4]
0000000000400c9a         sar        eax, 0x10
0000000000400c9d         mov        byte [ds:rdx], al
0000000000400c9f         add        dword [ss:rbp+var_8], 0x1

0000000000400ca3         cmp        dword [ss:rbp+var_8], 0x7
   ; XREF=sub_400bb5+90
0000000000400ca7         jle        0x400c14

0000000000400cad         mov        eax, 0x0

0000000000400cb2         leave
   ; XREF=sub_400bb5+61, sub_400bb5+138, sub_400bb5+154
0000000000400cb3         ret
                        ; endp
```

In the prologue, the pointer to the serial is stored in `rbp+var_18`, the pointer to the (result) array is stored at `rbp+var_20`.

At `0x400bcc` to `0x400bff` we can see a loop from $i = 0$ to $i = 6$ (inclusive). Inside the loop, there is a comparison with `var_18[i * 6 + 5]` and `0x2d` which seems to be the `’-’` character. If, for some iteration, `’-’` is not found, the serial is invalid and `-1` will be returned.

Next, from `0x400c0f` to `0x400ca7` there is another loop from $i = 0$ to $i = 7$ (inclusive). Before the loop, `rbp+var_C` and `rbp+var_8` are set to 0. Inside the loop, `sub_400ad9` is called with `var_18[i * 6]` as an argument. If its result is `-1` or larger than 0xffffff, -1 will be returned. Otherwise, the result will be written to `var_20`:

```
var_20[var_C + 0] = (res >> 0x0) & 0xff
var_20[var_C + 1] = (res >> 0x8) & 0xff
var_20[var_C + 2] = (res >> 0x10) & 0xff
var_C += 3
```

After that 0 is returned, indicating that the transformation was successful. The equivalent C code can be seen below:

```
int decode_serial(char const arg[], uint8_t result[]) { // sub_400bb5
    // check dashes
    for(int i = 0; i <= 6; i++) {
        if(arg[i * 6 + 5] != ’-’) {
            return -1;
        }
    }

    int result_index = 0;
    for(int i = 0; i <= 7; i++) {
        int const val = sub_400ad9(&arg[i * 6]);

        if(val == -1) {
            return -1;
        }

        // check for overflow
        if(val > 0xffffff) {
            return -1;
        }

        // write to result
        result[result_index++] = (uint8_t)(val >> 0x00);
        result[result_index++] = (uint8_t)(val >> 0x08);
        result[result_index++] = (uint8_t)(val >> 0x10);
    }

    return 0;
}
```

Next comes sub_400ad9:

```
                     sub_400ad9:
0000000000400ad9         push       rbp
   ; XREF=sub_400bb5+119
0000000000400ada         mov        rbp, rsp
0000000000400add         sub        rsp, 0x28
0000000000400ae1         mov        qword [ss:rbp+var_28], rdi
0000000000400ae5         mov        rax, qword [ss:rbp+var_28]
0000000000400ae9         movzx      eax, byte [ds:rax]
0000000000400aec         movsx      eax, al
0000000000400aef         mov        edi, eax
   ; argument #1 for method sub_400a9f
0000000000400af1         call       sub_400a9f
0000000000400af6         mov        dword [ss:rbp+var_14], eax
0000000000400af9         mov        rax, qword [ss:rbp+var_28]
0000000000400afd         add        rax, 0x1
0000000000400b01         movzx      eax, byte [ds:rax]
0000000000400b04         movsx      eax, al
0000000000400b07         mov        edi, eax
   ; argument #1 for method sub_400a9f
0000000000400b09         call       sub_400a9f
0000000000400b0e         mov        dword [ss:rbp+var_10], eax
0000000000400b11         mov        rax, qword [ss:rbp+var_28]
0000000000400b15         add        rax, 0x2
```

```
0000000000400b19         movzx      eax, byte [ds:rax]
0000000000400b1c         movsx      eax, al
0000000000400b1f         mov        edi, eax
   ; argument #1 for method sub_400a9f
0000000000400b21         call       sub_400a9f
0000000000400b26         mov        dword [ss:rbp+var_C], eax
0000000000400b29         mov        rax, qword [ss:rbp+var_28]
0000000000400b2d         add        rax, 0x3
0000000000400b31         movzx      eax, byte [ds:rax]
0000000000400b34         movsx      eax, al
0000000000400b37         mov        edi, eax
   ; argument #1 for method sub_400a9f
0000000000400b39         call       sub_400a9f
0000000000400b3e         mov        dword [ss:rbp+var_8], eax
0000000000400b41         mov        rax, qword [ss:rbp+var_28]
0000000000400b45         add        rax, 0x4
0000000000400b49         movzx      eax, byte [ds:rax]
0000000000400b4c         movsx      eax, al
0000000000400b4f         mov        edi, eax
   ; argument #1 for method sub_400a9f
0000000000400b51         call       sub_400a9f
0000000000400b56         mov        dword [ss:rbp+var_4], eax
0000000000400b59         cmp        dword [ss:rbp+var_14], 0xffffffff
0000000000400b5d         je         0x400b77

0000000000400b5f         cmp        dword [ss:rbp+var_10], 0xffffffff
0000000000400b63         je         0x400b77

0000000000400b65         cmp        dword [ss:rbp+var_C], 0xffffffff
0000000000400b69         je         0x400b77

0000000000400b6b         cmp        dword [ss:rbp+var_8], 0xffffffff
0000000000400b6f         je         0x400b77

0000000000400b71         cmp        dword [ss:rbp+var_4], 0xffffffff
0000000000400b75         jne        0x400b7e

0000000000400b77         mov        eax, 0xffffffff
   ; XREF=sub_400ad9+132, sub_400ad9+138, sub_400ad9+144, sub_400ad9+150
0000000000400b7c         jmp        0x400bb3

0000000000400b7e         mov        eax, dword [ss:rbp+var_4]
   ; XREF=sub_400ad9+156
0000000000400b81         imul       edx, eax, 0x19a100
0000000000400b87         mov        eax, dword [ss:rbp+var_8]
0000000000400b8a         imul       eax, eax, 0xb640
0000000000400b90         add        edx, eax
0000000000400b92         mov        eax, dword [ss:rbp+var_C]
0000000000400b95         imul       eax, eax, 0x510
0000000000400b9b         lea        ecx, dword [ds:rdx+rax]
0000000000400b9e         mov        edx, dword [ss:rbp+var_10]
0000000000400ba1         mov        eax, edx
0000000000400ba3         shl        eax, 0x3
0000000000400ba6         add        eax, edx
0000000000400ba8         shl        eax, 0x2
0000000000400bab         lea        edx, dword [ds:rcx+rax]
0000000000400bae         mov        eax, dword [ss:rbp+var_14]
0000000000400bb1         add        eax, edx

0000000000400bb3         leave
   ; XREF=sub_400ad9+163
0000000000400bb4         ret
                        ; endp
```

After the prologue, where the argument, the current segment of the serial, is stored in rbp+var_28, sub_400a9f is called 5 times, with var_28[0], var_28[1], var_28[2], var_28[3], var_28[4] as arguments for each call. The results are stored in rbp+var_14, rbp+var_10, rbp+var_C, rbp+var_8 and rbp+var_4, respectively. If one of the results equals -1, -1 will be returned. Otherwise, all values will be added, multiplied, similar to the following calculation:

```
c0 + (c1 + (c2 + (c3 + (c4) * 36) * 36) * 36) * 36
```

The result of this calculation will be returned. The equivalent C code can be seen below:

```
int decode_segment(char const segment[]) { // sub_400ad9
    int c0 = sub_400a9f(segment[0]);
    int c1 = sub_400a9f(segment[1]);
    int c2 = sub_400a9f(segment[2]);
    int c3 = sub_400a9f(segment[3]);
    int c4 = sub_400a9f(segment[4]);

    if(c0 == -1 || c1 == -1 || c2 == -1 || c3 == -1 || c4 == -1) {
        return -1;
    }

    return c0 + (c1 + (c2 + (c3 + (c4) * 36) * 36) * 36) * 36;
}
```

Next comes sub_400ad9.

```
                     sub_400a9f:
0000000000400a9f         push       rbp
   ; XREF=sub_400ad9+24, sub_400ad9+48, sub_400ad9+72, sub_400ad9+96, sub_400ad9+120
0000000000400aa0         mov        rbp, rsp
0000000000400aa3         mov        eax, edi
0000000000400aa5         mov        byte [ss:rbp+var_4], al
0000000000400aa8         cmp        byte [ss:rbp+var_4], 0x2f
0000000000400aac         jle        0x400abd

0000000000400aae         cmp        byte [ss:rbp+var_4], 0x39
0000000000400ab2         jg         0x400abd

0000000000400ab4         movsx      eax, byte [ss:rbp+var_4]
0000000000400ab8         sub        eax, 0x30
0000000000400abb         jmp        0x400ad7

0000000000400abd         cmp        byte [ss:rbp+var_4], 0x40
   ; XREF=sub_400a9f+13, sub_400a9f+19
0000000000400ac1         jle        0x400ad2

0000000000400ac3         cmp        byte [ss:rbp+var_4], 0x5a
0000000000400ac7         jg         0x400ad2

0000000000400ac9         movsx      eax, byte [ss:rbp+var_4]
0000000000400acd         sub        eax, 0x37
0000000000400ad0         jmp        0x400ad7

0000000000400ad2         mov        eax, 0xffffffff
   ; XREF=sub_400a9f+34, sub_400a9f+40

0000000000400ad7         pop        rbp
   ; XREF=sub_400a9f+28, sub_400a9f+49
0000000000400ad8         ret
                        ; endp
```

After the prologue, where the argument is stored at rbp+var_4, var_4 is compared with 0x2f(’0’ - 1) and 0x39(’9’). If var_4 lies between these values, var_4 - ’0’ will be returned. Otherwise `var_4` is compared with `0x40(’A’ - 1)` and `0x5a(’Z’)`. If `var_4` lies between these values, `var_4 - ’A’` will be returned. Otherwise If `var_4` lies between these values, `-1` will be returned. The equivalent C code can be seen below:

```
int decode_char(char c) { // sub_400a9f
    if(c >= ’0’ && c <= ’9’) {
        return c - ’0’;
    }

    if(c >= ’A’ && c <= ’Z’) {
        return c - ’A’ + 10;
    }

    return -1;
}
```

This leaves us with `sub_40085e` which was also called from `main(...)`.

```
                        sub_40085e:
000000000040085e         push       rbp
   ; XREF=sub_400cb4+191
000000000040085f         mov        rbp, rsp
0000000000400862         sub        rsp, 0x20
0000000000400866         mov        qword [ss:rbp+var_18], rdi
000000000040086a         mov        rax, qword [ss:rbp+var_18]
000000000040086e         mov        edx, 0x4
   ; argument "n" for method j_memcmp
0000000000400873         mov        esi, 0x400e38
   ; argument "s2" for method j_memcmp
0000000000400878         mov        rdi, rax
   ; argument "s1" for method j_memcmp
000000000040087b         call       j_memcmp
0000000000400880         test       eax, eax
0000000000400882         je         0x40088e

0000000000400884         mov        eax, 0x0
0000000000400889         jmp        0x400a9d

000000000040088e         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+36
0000000000400892         add        rax, 0x4
0000000000400896         movzx      eax, byte [ds:rax]
0000000000400899         movzx      eax, al
000000000040089c         shl        eax, 0x8
000000000040089f         mov        edx, eax
00000000004008a1         mov        rax, qword [ss:rbp+var_18]
00000000004008a5         add        rax, 0x5
00000000004008a9         movzx      eax, byte [ds:rax]
00000000004008ac         movzx      eax, al
00000000004008af         or         eax, edx
00000000004008b1         mov        word [ss:rbp+var_A], ax
00000000004008b5         mov        rax, qword [ss:rbp+var_18]
00000000004008b9         movzx      eax, byte [ds:rax+6]
00000000004008bd         mov        byte [ss:rbp+var_C], al
00000000004008c0         mov        rax, qword [ss:rbp+var_18]
00000000004008c4         movzx      eax, byte [ds:rax+7]
00000000004008c8         mov        byte [ss:rbp+var_B], al
00000000004008cb         mov        eax, 0x0
00000000004008d0         call       sub_40074d
00000000004008d5         cmp        ax, word [ss:rbp+var_A]
00000000004008d9         jbe        0x4008e5

00000000004008db         mov        eax, 0x0
```

```
00000000004008e0         jmp        0x400a9d

00000000004008e5         mov        eax, 0x0
   ; XREF=sub_40085e+123
00000000004008ea         call       sub_40074d
00000000004008ef         cmp        ax, word [ss:rbp+var_A]
00000000004008f3         jne        0x400936

00000000004008f5         mov        eax, 0x0
00000000004008fa         call       sub_40077c
00000000004008ff         cmp        al, byte [ss:rbp+var_C]
0000000000400902         jbe        0x40090e

0000000000400904         mov        eax, 0x0
0000000000400909         jmp        0x400a9d

000000000040090e         mov        eax, 0x0
   ; XREF=sub_40085e+164
0000000000400913         call       sub_40077c
0000000000400918         cmp        al, byte [ss:rbp+var_C]
000000000040091b         jne        0x400936

000000000040091d         mov        eax, 0x0
0000000000400922         call       sub_4007ab
0000000000400927         cmp        al, byte [ss:rbp+var_B]
000000000040092a         jbe        0x400936

000000000040092c         mov        eax, 0x0
0000000000400931         jmp        0x400a9d

0000000000400936         cmp        word [ss:rbp+var_A], 0x7df
   ; XREF=sub_40085e+149, sub_40085e+189, sub_40085e+204
000000000040093c         jbe        0x40094a

000000000040093e         cmp        byte [ss:rbp+var_C], 0x0
0000000000400942         je         0x40094a

0000000000400944         cmp        byte [ss:rbp+var_B], 0x0
0000000000400948         jne        0x400954

000000000040094a         mov        eax, 0x0
   ; XREF=sub_40085e+222, sub_40085e+228
000000000040094f         jmp        0x400a9d

0000000000400954         cmp        word [ss:rbp+var_A], 0x834
   ; XREF=sub_40085e+234
000000000040095a         ja         0x400968

000000000040095c         cmp        byte [ss:rbp+var_C], 0xc
0000000000400960         ja         0x400968

0000000000400962         cmp        byte [ss:rbp+var_B], 0x1f
0000000000400966         jbe        0x400972

0000000000400968         mov        eax, 0x0
   ; XREF=sub_40085e+252, sub_40085e+258
000000000040096d         jmp        0x400a9d

0000000000400972         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+264
0000000000400976         mov        rax, qword [ds:rax+0x10]
000000000040097a         mov        qword [ss:rbp+var_8], rax
000000000040097e         mov        rax, qword [ss:rbp+var_18]
0000000000400982         mov        edx, 0x18
```

```
   ; argument #3 for method sub_4007da
0000000000400987         mov        rsi, rax
   ; argument #2 for method sub_4007da
000000000040098a         mov        edi, 0x0
   ; argument #1 for method sub_4007da
000000000040098f         call       sub_4007da
0000000000400994         cmp        rax, qword [ss:rbp+var_8]
0000000000400998         je         0x4009a4

000000000040099a         mov        eax, 0x0
000000000040099f         jmp        0x400a9d

00000000004009a4         movzx      ecx, byte [ss:rbp+var_B]
   ; XREF=sub_40085e+314
00000000004009a8         movzx      edx, byte [ss:rbp+var_C]
00000000004009ac         movzx      eax, word [ss:rbp+var_A]
00000000004009b0         mov        esi, eax
00000000004009b2         mov        edi, 0x400e40
   ; "Expiration date: \%04d-\%02d-\%02d\\n", argument "format" for method j_printf
00000000004009b7         mov        eax, 0x0
00000000004009bc         call       j_printf
00000000004009c1         mov        rax, qword [ss:rbp+var_18]
00000000004009c5         add        rax, 0x8
00000000004009c9         movzx      eax, byte [ds:rax]
00000000004009cc         movzx      eax, al
00000000004009cf         and        eax, 0x40
00000000004009d2         test       eax, eax
00000000004009d4         je         0x4009e2

00000000004009d6         mov        edi, 0x400e61
   ; "Feature A: ON", argument "s" for method j_puts
00000000004009db         call       j_puts
00000000004009e0         jmp        0x4009ec

00000000004009e2         mov        edi, 0x400e6f
   ; "Feature A: OFF", argument "s" for method j_puts, XREF=sub_40085e+374
00000000004009e7         call       j_puts

00000000004009ec         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+386
00000000004009f0         add        rax, 0x9
00000000004009f4         movzx      eax, byte [ds:rax]
00000000004009f7         movzx      eax, al
00000000004009fa         and        eax, 0x1
00000000004009fd         test       eax, eax
00000000004009ff         je         0x400a0d

0000000000400a01         mov        edi, 0x400e7e
   ; "Feature B: ON", argument "s" for method j_puts
0000000000400a06         call       j_puts
0000000000400a0b         jmp        0x400a17

0000000000400a0d         mov        edi, 0x400e8c
   ; "Feature B: OFF", argument "s" for method j_puts, XREF=sub_40085e+417
0000000000400a12         call       j_puts

0000000000400a17         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+429
0000000000400a1b         add        rax, 0xa
0000000000400a1f         movzx      eax, byte [ds:rax]
0000000000400a22         movzx      eax, al
0000000000400a25         and        eax, 0x2
0000000000400a28         test       eax, eax
0000000000400a2a         je         0x400a38
```

```
0000000000400a2c         mov        edi, 0x400e9b
   ; "Feature C: ON", argument "s" for method j_puts
0000000000400a31         call       j_puts
0000000000400a36         jmp        0x400a42

0000000000400a38         mov        edi, 0x400ea9
   ; "Feature C: OFF", argument "s" for method j_puts, XREF=sub_40085e+460
0000000000400a3d         call       j_puts

0000000000400a42         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+472
0000000000400a46         add        rax, 0xb
0000000000400a4a         movzx      eax, byte [ds:rax]
0000000000400a4d         movzx      eax, al
0000000000400a50         and        eax, 0x8
0000000000400a53         test       eax, eax
0000000000400a55         je         0x400a63

0000000000400a57         mov        edi, 0x400eb8
   ; "Feature D: ON", argument "s" for method j_puts
0000000000400a5c         call       j_puts
0000000000400a61         jmp        0x400a6d

0000000000400a63         mov        edi, 0x400ec6
   ; "Feature D: OFF", argument "s" for method j_puts, XREF=sub_40085e+503
0000000000400a68         call       j_puts

0000000000400a6d         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_40085e+515
0000000000400a71         add        rax, 0xc
0000000000400a75         movzx      eax, byte [ds:rax]
0000000000400a78         movzx      eax, al
0000000000400a7b         and        eax, 0x1
0000000000400a7e         test       eax, eax
0000000000400a80         je         0x400a8e

0000000000400a82         mov        edi, 0x400ed5
   ; "Feature E: ON", argument "s" for method j_puts
0000000000400a87         call       j_puts
0000000000400a8c         jmp        0x400a98

0000000000400a8e         mov        edi, 0x400ee3
   ; "Feature E: OFF", argument "s" for method j_puts, XREF=sub_40085e+546
0000000000400a93         call       j_puts

0000000000400a98         mov        eax, 0x1
   ; XREF=sub_40085e+558

0000000000400a9d         leave
   ; XREF=sub_40085e+43, sub_40085e+130, sub_40085e+171, sub_40085e+211, sub_40085e+241,
    sub_40085e+271, sub_40085e+321
0000000000400a9e         ret
                        ; endp
```

After the prologue, where the argument, the decoded serial, is stored in rbp+var_18, memcmp(...) is called with the arguments var_18, 0xbebaadde (0xdeadbabe in big endian format) and 4. If not equals, false will be returned. Otherwise, some of the values from the array var_18 will be OR'ed and saved in variables:

```
var_A = var_18[5] | var_18[4] << 8;
var_C = var_18[6];
var_B = var_18[7];
```

Next, sub_40074d is called.

```
                        sub_40074d:
000000000040074d         push       rbp
   ; XREF=sub_40085e+114, sub_40085e+140
000000000040074e         mov        rbp, rsp
0000000000400751         sub        rsp, 0x10
0000000000400755         mov        edi, 0x0
   ; argument "tloc" for method j_time
000000000040075a         call       j_time
000000000040075f         mov        qword [ss:rbp+var_10], rax
0000000000400763         lea        rax, qword [ss:rbp+var_10]
0000000000400767         mov        rdi, rax
   ; argument "clock" for method j_localtime
000000000040076a         call       j_localtime
000000000040076f         mov        qword [ss:rbp+var_8], rax
0000000000400773         mov        rax, qword [ss:rbp+var_8]
0000000000400777         mov        eax, dword [ds:rax+0x14]
000000000040077a         leave
000000000040077b         ret
                           ; endp
```

sub_40074d only calls some libc functions, time(...) and localtime(...), to get the current time in a time_t, which is stored in rbp+var_8. Then *(var_8+0x14) is returned. By looking at the libc TIME.H we find that this is the tm_year field. The equivalent C code can be seen below:

```
int get_year() { // sub_40074d
    time_t const timer = time(NULL);
    return localtime(&timer)->tm_year;
}
```

So, getting back to sub_40085e and the call to sub_40074d, the result is compared to var_A. If it is larger than var_A, false will be returned. Otherwise, at 0x4008ea sub_40074d will be called and compared to var_A again. If equal, some more checks will be performed: sub_40077c, which seems to almost the equal to sub_40074d, but then for the current month, will then be called and compared to var_C. If it is larger than var_C, false will be returned. Otherwise, sub_40077c will be called and compared to var_C again. If equal sub_4007ab will be called, which seems to almost the equal to sub_40074d, but then for the current day. Its result will be compared to var_B. If it is larger than var_B, false will be returned. With these function calls, one can assume that var_A, var_C and var_B stand for year, month and day, respectively. The equivalent pseudo code can be seen below:

```
if(sub_40074d() > var_A) {
    return false;
}

if(sub_40074d() == var_A) {
    if(sub_40077c() > var_C) {
        return false;
    }

    if(sub_40077c() == var_C && sub_4007ab() > var_B) {
        return false;
    }
}
```

After this, starting at 0x400936 some more checks are performed. The equivalent pseudo code can be seen below:

```
if(var_A > 2015 && var_C != 0 && var_B != 0) {
    if(var_A <= 2100 && var_C <= 12 && var_B <= 31) {
        // ...
    } else {
        return false;
    }
} else {
    return false;
}
```

If both checks resolve to true, sub_4007da is called with 0, var_18 (the decoded serial) and 24 as arguments. The result is then compared with var_18[16]. If not equal, false will be returned. Otherwise, the serial is assumed valid, the status of the serial is printed, ie. the expiration date and the turned on features according to certain bits in the serial, and true is returned. The equivalent pseudo code can be seen below:

```
if(sub_4007da(0, var_18, 24) != var_18[16]) {
    return false;
}

printf("Expiration date: \%04d-\%02d-\%02d\n", var_A, var_C, var_B);

if((var_18[8] & 0x40) != 0) {
    puts("Feature A: ON");
} else {
    puts("Feature A: OFF");
}

if((var_18[9] & 0x1) != 0) {
    puts("Feature B: ON");
} else {
    puts("Feature B: OFF");
}

if((var_18[10] & 0x2) != 0) {
    puts("Feature C: ON");
} else {
    puts("Feature C: OFF");
}

if((var_18[11] & 0x8) != 0) {
    puts("Feature D: ON");
} else {
    puts("Feature D: OFF");
}

if((var_18[12] & 0x1) != 0) {
    puts("Feature D: ON");
} else {
    puts("Feature D: OFF");
}

return true;
```

Lastly, we will look at sub_4007da.

```
                        sub_4007da:
00000000004007da        push      rbp
   ; XREF=sub_40085e+305
00000000004007db        mov       rbp, rsp
00000000004007de        mov       qword [ss:rbp+var_18], rdi
00000000004007e2        mov       qword [ss:rbp+var_20], rsi
```

```
00000000004007e6         mov        dword [ss:rbp+var_24], edx
00000000004007e9         not        qword [ss:rbp+var_18]
00000000004007ed         jmp        0x400848

00000000004007ef         mov        rax, qword [ss:rbp+var_20]
   ; XREF=sub_4007da+121
00000000004007f3         lea        rdx, qword [ds:rax+1]
00000000004007f7         mov        qword [ss:rbp+var_20], rdx
00000000004007fb         movzx      eax, byte [ds:rax]
00000000004007fe         movzx      eax, al
0000000000400801         xor        qword [ss:rbp+var_18], rax
0000000000400805         mov        dword [ss:rbp+var_4], 0x0
000000000040080c         jmp        0x400842

000000000040080e         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_4007da+108
0000000000400812         and        eax, 0x1
0000000000400815         test       rax, rax
0000000000400818         je         0x400833

000000000040081a         mov        rax, qword [ss:rbp+var_18]
000000000040081e         shr        rax, 0x1
0000000000400821         mov        rdx, rax
0000000000400824         movabs     rax, 0x42f0e1eb0badbad0
000000000040082e         xor        rax, rdx
0000000000400831         jmp        0x40083a

0000000000400833         mov        rax, qword [ss:rbp+var_18]
   ; XREF=sub_4007da+62
0000000000400837         shr        rax, 0x1

000000000040083a         mov        qword [ss:rbp+var_18], rax
   ; XREF=sub_4007da+87
000000000040083e         add        dword [ss:rbp+var_4], 0x1

0000000000400842         cmp        dword [ss:rbp+var_4], 0x7
   ; XREF=sub_4007da+50
0000000000400846         jle        0x40080e

0000000000400848         mov        eax, dword [ss:rbp+var_24]
   ; XREF=sub_4007da+19
000000000040084b         lea        edx, dword [ds:rax-1]
000000000040084e         mov        dword [ss:rbp+var_24], edx
0000000000400851         test       eax, eax
0000000000400853         jne        0x4007ef

0000000000400855         mov        rax, qword [ss:rbp+var_18]
0000000000400859         not        rax
000000000040085c         pop        rbp
000000000040085d         ret
                        ; endp
```

In the prologue, the arguments are stored in `rbp+var_18`, `rbp+var_20` and `rbp+var_24`, respectively. Directly after that `var_18` is inverted with a bitwise NOT. From `0x4007ed` to `0x400853` there is a loop from `i = var_24` to `0` (exclusive). Inside the loop, `var_18` is XOR'ed with the value at position `i` of the serial: `var_18 ^= var_20[var_24]`. After that, `var_20` is incremented to the next position of the serial. Next, still inside the loop, comes another loop from `j = 0` to `7` (inclusive). For each iteration, if the rightmost bit is set, `var_18` will be shifted to the right and XOR'ed with a random constant. Otherwise, it is only shifted to the right. After the nested loops finish, `var_18` is inverted, again, with a bitwise NOT and returned. Looking at `sub_4007da`, it seems like some kind of checksum or obfuscation function. The equivalent C code can be seen below:

```c
uint64_t const kUnknownConstant = 0x42f0e1eb0badbad0;

uint8_t calculate_checksum(uint64_t var_18, uint8_t const *var_20, size_t var_24) { // sub_4007da
    var_18 = ~var_18;

    for(int i = var_24; i != 0; i--) {
        var_18 ^= *var_20;
        var_20++;

        for(int j = 0; j <= 7; j++) {
            if((var_18 & 0x1) != 0) {
                var_18 >>= 1;
                var_18 ^= kUnknownConstant;
            } else {
                var_18 >>= 1;
            }
        }
    }

    return ~var_18;
}
```

After putting all the above parts together and improving the variable and functions names, we have an functionally equivalent program in C. It is shown in appendix B. Although not provided here, with this code, it is fairly easy to create a keygen to solve the given challenge.

6 Encountered Problems

6.1 External Libraries

External libraries like the C or C++ standard library still use symbols even if the program using them is compiled without symbols. It not only makes function calls easier to analyze, but it also makes it easier to find the more interesting parts of the code by looking where a certain external function, like, for instance, `printf(...)`, is called. Although this is not a problem for the analysis process, it can be for the author of the binary. Solutions could be minimizing the use of external libraries with symbols or obfuscating the code flow, for instance, with jump tables as described by Linn and Debray [2].

6.2 Optimizer

The optimizer makes the analysis and decompilation process harder for humans and tools, like Hopper, by, among others, inlining functions, unrolling loops or reordering of instructions. The only solution for this problem is probably improvement of the analysis tools.

6.3 Missing Type Information

In the C code of the Crackme, some wrong assumptions about types were made, since Hopper only assumes `int` most of the time. By looking closer at the different instruction used for a variable these mistakes could be fixed. LLDB was also used for some of the more challenging errors.

7 Conclusion

In this report we have shown the basic process of binary reverse engineering. We introduced the different available techniques and tools and applied them on two example binaries. The first example case explored what a C++ program looks like after disassembly and tried to find the assembly constructs corresponding to high level C++ statements. In the second example the binary reverse engineering process was applied to 'crack' a serial key checking program. The examples have shown that modern reverse engineering tools perform well on non-obfuscated code, compiled with reasonable levels of optimization. Further testing will have to prove if the tools, such as Hopper, perform equally well on obfuscated code.

References

[1] Wikipedia, *Objdump — wikipedia, the free encyclopedia*, [Online; accessed 10-April-2016], 2016. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Objdump&oldid=706272555`.

[2] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proceedings of the 10th ACM conference on Computer and communications security*, ACM, 2003, pp. 290–299.

[3] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *USENIX security Symposium*, vol. 13, 2004, pp. 18–18.

[4] H. Rays, *Ida pro*, [Online; accessed 10-April-2016]. [Online]. Available: `https://www.hex-rays.com/products/ida/index.shtml`.

[5] *Hopper*, [Online; accessed 10-April-2016]. [Online]. Available: `http://www.hopperapp.com/`.

[6] *Capstone*, [Online; accessed 10-April-2016]. [Online]. Available: `http://www.capstone-engine.org/`.

[7] Wikipedia, *Gnu debugger*, [Online; accessed 10-April-2016]. [Online]. Available: `https://en.wikipedia.org/wiki/GNU_Debugger`.

[8] ——, *Lldb debugger*, [Online; accessed 10-April-2016]. [Online]. Available: `https://en.wikipedia.org/wiki/GNU_Debugger`.

[9] ——, *Crackme — wikipedia, the free encyclopedia*, [Online; accessed 10-April-2016], 2015. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Crackme&oldid=696504078`.

[10] *Clang: A c language family frontend for llvm*, 2016. [Online]. Available: `http://clang.llvm.org/` (visited on 04/10/2016).

[11] *Crackme challenge*, [Online; accessed 10-April-2016]. [Online]. Available: `http://challenges.re/74/`.

A Source

Listing 1: `src/depthFirstSearch.cc`

```
void Grid::depthFirstSearch() {
    { // reset visited and marked flags
        for(int y = 0; y < GRID_SIZE; y++) {
            for(int x = 0; x < GRID_SIZE; x++) {
                grid[x][y].visited = false;
                grid[x][y].marked = false;
            }
        }
    }

    std::stack<GridElement *> stack;

    stack.push(&grid[0][0]);
    GridElement *end_element = &grid[GRID_SIZE - 1][GRID_SIZE - 1];

    int states = 0;
    while(!stack.empty()) {
        GridElement *current_element = stack.top();

        current_element->visited = true;
        current_element->marked = true;

        states++;

        if(current_element == end_element) {
            // found end, so stop
            break;
        }

        int n_unvisited_directions = 0;
        Direction unvisited_directions[N_DIRECTIONS];
        for(int direction = 0; direction < N_DIRECTIONS; direction++) {
            if(!current_element->walls[direction]
               && !current_element->neighbours[direction]->visited) {
                unvisited_directions[n_unvisited_directions++] = (Direction)direction;
            }
        }

        if(n_unvisited_directions == 0) {
            current_element->marked = false;
            stack.pop();
            continue;
        }

        Direction next_direction = unvisited_directions[std::rand() % n_unvisited_directions];
        GridElement *next_element = current_element->neighbours[next_direction];
        stack.push(next_element);
    }

    // print result
    if(stack.empty()) {
        std::cout << "DFS (end not found)" << std::endl;
    } else {
        std::cout << "DFS (states: " << states << ", path length: " << stack.size() << ")"
            << std::endl;
    }
}
```

Listing 2: `src/depthFirstSearch.txt`

```
                     sub_100003d70:
0000000100003d70         push       rbp
   ; XREF=sub_1000022b0+187
0000000100003d71         mov        rbp, rsp
0000000100003d74         push       r15
0000000100003d76         push       r14
0000000100003d78         push       r13
0000000100003d7a         push       r12
0000000100003d7c         push       rbx
0000000100003d7d         sub        rsp, 0x58
0000000100003d81         mov        r14, rdi
0000000100003d84         mov        rax, qword [ds:imp___got____stack_chk_guard]
0000000100003d8b         mov        rax, qword [ds:rax]
0000000100003d8e         mov        qword [ss:rbp+var_30], rax
0000000100003d92         lea        rax, qword [ds:r14+0x282d]
0000000100003d99         xor        ecx, ecx
0000000100003d9b         nop        dword [ds:rax+rax]

0000000100003da0         mov        edx, 0x28
   ; XREF=sub_100003d70+130
0000000100003da5         mov        rsi, rax
0000000100003da8         nop        dword [ds:rax+rax]

0000000100003db0         mov        word [ds:rsi-0x2801], 0x0
   ; XREF=sub_100003d70+117
0000000100003db9         mov        word [ds:rsi-0x1e01], 0x0
0000000100003dc2         mov        word [ds:rsi-0x1401], 0x0
0000000100003dcb         mov        word [ds:rsi-0xa01], 0x0
0000000100003dd4         mov        word [ds:rsi-1], 0x0
0000000100003dda         add        rsi, 0x3200
0000000100003de1         add        rdx, 0xfffffffffffffffb
0000000100003de5         jne        0x100003db0

0000000100003de7         inc        rcx
0000000100003dea         add        rax, 0x40
0000000100003dee         cmp        rcx, 0x28
0000000100003df2         jne        0x100003da0

0000000100003df4         xorps      xmm0, xmm0
0000000100003df7         movaps     xmmword [ss:rbp+var_60], xmm0
0000000100003dfb         movaps     xmmword [ss:rbp+var_70], xmm0
0000000100003dff         movaps     xmmword [ss:rbp+var_80], xmm0
0000000100003e03         lea        rdi, qword [ss:rbp+var_80]
0000000100003e07         call       imp___stubs___ZNSt3__15dequeIP11GridElementNS_9allocator
   IS2_EEE19__add_back_capacityEv
    ; std::__1::deque<GridElement*, std::__1::allocator<GridElement*> >::__add_back_capacity()
0000000100003e0c         mov        rcx, qword [ss:rbp+var_58]
0000000100003e10         mov        rax, qword [ss:rbp+var_78]
0000000100003e14         mov        rdx, qword [ss:rbp+var_60]
0000000100003e18         lea        rdi, qword [ds:rcx+rdx]
0000000100003e1c         mov        rbx, rdi
0000000100003e1f         shr        rbx, 0x9
0000000100003e23         mov        rax, qword [ds:rax+rbx*8]
0000000100003e27         and        rdi, 0x1ff
0000000100003e2e         mov        qword [ds:rax+rdi*8], r14
0000000100003e32         mov        rax, rcx
0000000100003e35         inc        rax
0000000100003e38         mov        qword [ss:rbp+var_58], rax
0000000100003e3c         je         0x1000040d7

0000000100003e42         add        r14, 0x18fc0
0000000100003e49         mov        rsi, qword [ss:rbp+var_78]
0000000100003e4d         mov        rbx, qword [ds:rsi+rbx*8]
0000000100003e51         mov        r13, qword [ds:rbx+rdi*8]
```

```
0000000100003e55         mov        word [ds:r13+0x2c], 0x101
0000000100003e5c         mov        r15d, 0x1
0000000100003e62         cmp        r13, r14
0000000100003e65         je         0x100004027

0000000100003e6b         lea        rdi, qword [ds:r13+0x2d]
0000000100003e6f         nop

0000000100003e70         cmp        byte [ds:r13+0x28], 0x0
   ; XREF=sub_100003d70+689
0000000100003e75         mov        r12d, 0x0
0000000100003e7b         jne        0x100003ea0

0000000100003e7d         mov        rbx, qword [ds:r13+8]
0000000100003e81         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003e85         mov        r12d, 0x0
0000000100003e8b         jne        0x100003ea0

0000000100003e8d         mov        dword [ss:rbp+var_40], 0x0
0000000100003e94         mov        r12d, 0x1
0000000100003e9a         nop        word [ds:rax+rax]

0000000100003ea0         cmp        byte [ds:r13+0x29], 0x0
   ; XREF=sub_100003d70+267, sub_100003d70+283
0000000100003ea5         jne        0x100003ed0

0000000100003ea7         mov        rbx, qword [ds:r13+0x10]
0000000100003eab         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003eaf         jne        0x100003ed0

0000000100003eb1         lea        r8d, dword [ds:r12+1]
0000000100003eb6         shl        r12, 0x2
0000000100003eba         lea        rbx, qword [ss:rbp+var_40]
0000000100003ebe         or         r12, rbx
0000000100003ec1         mov        dword [ds:r12], 0x1
0000000100003ec9         mov        r12d, r8d
0000000100003ecc         nop        dword [ds:rax]

0000000100003ed0         cmp        byte [ds:r13+0x2a], 0x0
   ; XREF=sub_100003d70+309, sub_100003d70+319
0000000100003ed5         jne        0x100003ef0

0000000100003ed7         mov        rbx, qword [ds:r13+0x18]
0000000100003edb         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003edf         jne        0x100003ef0

0000000100003ee1         movsxd     rbx, r12d
0000000100003ee4         inc        r12d
0000000100003ee7         mov        dword [ss:rbp+rbx*4+var_40], 0x2
0000000100003eef         nop

0000000100003ef0         cmp        byte [ds:r13+0x2b], 0x0
   ; XREF=sub_100003d70+357, sub_100003d70+367
0000000100003ef5         jne        0x100003f10

0000000100003ef7         mov        rbx, qword [ds:r13+0x20]
0000000100003efb         cmp        byte [ds:rbx+0x2c], 0x0
0000000100003eff         jne        0x100003f10

0000000100003f01         movsxd     rbx, r12d
0000000100003f04         inc        r12d
0000000100003f07         mov        dword [ss:rbp+rbx*4+var_40], 0x3
0000000100003f0f         nop

```

```
0000000100003f10         test       r12d, r12d
   ; XREF=sub_100003d70+389, sub_100003d70+399
0000000100003f13         je         0x100003f90

0000000100003f15         call       imp___stubs__rand
0000000100003f1a         cdq
0000000100003f1b         idiv       r12d
0000000100003f1e         movsxd     rax, edx
0000000100003f21         mov        eax, dword [ss:rbp+rax*4+var_40]
0000000100003f25         mov        r12, qword [ds:r13+rax*8+8]
0000000100003f2a         mov        rcx, qword [ss:rbp+var_78]
0000000100003f2e         mov        rax, qword [ss:rbp+var_70]
0000000100003f32         sub        rax, rcx
0000000100003f35         mov        esi, 0x0
0000000100003f3a         je         0x100003f46

0000000100003f3c         shl        rax, 0x6
0000000100003f40         dec        rax
0000000100003f43         mov        rsi, rax

0000000100003f46         mov        rdx, qword [ss:rbp+var_60]
   ; XREF=sub_100003d70+458
0000000100003f4a         mov        rax, qword [ss:rbp+var_58]
0000000100003f4e         sub        rsi, rdx
0000000100003f51         cmp        rsi, rax
0000000100003f54         jne        0x100003f6b

0000000100003f56         lea        rdi, qword [ss:rbp+var_80]
0000000100003f5a         call       imp___stubs___ZNSt3__15dequeIP11GridElementNS_9allocator
   IS2_EEE19__add_back_capacityEv
   ; std::__1::deque<GridElement*, std::__1::allocator<GridElement*> >::__add_back_capacity()
0000000100003f5f         mov        rax, qword [ss:rbp+var_58]
0000000100003f63         mov        rcx, qword [ss:rbp+var_78]
0000000100003f67         mov        rdx, qword [ss:rbp+var_60]

0000000100003f6b         add        rdx, rax
   ; XREF=sub_100003d70+484
0000000100003f6e         mov        rsi, rdx
0000000100003f71         shr        rsi, 0x9
0000000100003f75         mov        rcx, qword [ds:rcx+rsi*8]
0000000100003f79         and        rdx, 0x1ff
0000000100003f80         mov        qword [ds:rcx+rdx*8], r12
0000000100003f84         inc        rax
0000000100003f87         mov        qword [ss:rbp+var_58], rax
0000000100003f8b         jmp        0x100003fe0

0000000100003f90         mov        byte [ds:rdi], 0x0
   ; XREF=sub_100003d70+419
0000000100003f93         mov        qword [ss:rbp+var_58], rcx
0000000100003f97         mov        r8, qword [ss:rbp+var_70]
0000000100003f9b         mov        rdi, r8
0000000100003f9e         sub        rdi, rsi
0000000100003fa1         mov        esi, 0x0
0000000100003fa6         je         0x100003fb2

0000000100003fa8         shl        rdi, 0x6
0000000100003fac         dec        rdi
0000000100003faf         mov        rsi, rdi

0000000100003fb2         mov        edi, 0x1
   ; XREF=sub_100003d70+566
0000000100003fb7         sub        rdi, rax
0000000100003fba         add        rdi, rsi
0000000100003fbd         sub        rdi, rdx
```

```
0000000100003fc0         cmp        rdi, 0x400
0000000100003fc7         mov        rax, rcx
0000000100003fca         jb         0x100003fe0

0000000100003fcc         mov        rdi, qword [ds:r8-8]
0000000100003fd0         call       imp___stubs___ZdlPv
   ; operator delete(void*)
0000000100003fd5         add        qword [ss:rbp+var_70], 0xfffffffffffffff8
0000000100003fda         mov        rax, qword [ss:rbp+var_58]
0000000100003fde         nop

0000000100003fe0         test       rax, rax
   ; XREF=sub_100003d70+539, sub_100003d70+602
0000000100003fe3         je         0x1000040d7

0000000100003fe9         mov        rsi, qword [ss:rbp+var_78]
0000000100003fed         mov        rdx, qword [ss:rbp+var_60]
0000000100003ff1         lea        rcx, qword [ds:rax-1]
0000000100003ff5         lea        rbx, qword [ds:rax+rdx-1]
0000000100003ffa         mov        rdi, rbx
0000000100003ffd         shr        rdi, 0x9
0000000100004001         mov        rdi, qword [ds:rsi+rdi*8]
0000000100004005         and        rbx, 0x1ff
000000010000400c         mov        r13, qword [ds:rdi+rbx*8]
0000000100004010         mov        word [ds:r13+0x2c], 0x101
0000000100004017         lea        rdi, qword [ds:r13+0x2d]
000000010000401b         inc        r15d
000000010000401e         cmp        r13, r14
0000000100004021         jne        0x100003e70

0000000100004027         mov        rdi, qword [ds:imp___got___ZNSt3__14coutE]
   ; XREF=sub_100003d70+245
000000010000402e         lea        rsi, qword [ds:0x10048cb4c]
   ; "DFS (states: "
0000000100004035         mov        edx, 0xc
000000010000403a         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000403f         mov        rdi, rax
0000000100004042         mov        esi, r15d
0000000100004045         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEElsEi
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::operator<<(int)
000000010000404a         lea        rsi, qword [ds:0x10048cb59]
   ; ", path length: "
0000000100004051         mov        edx, 0xf
0000000100004056         mov        rdi, rax
0000000100004059         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000405e         mov        rsi, qword [ss:rbp+var_58]
0000000100004062         mov        rdi, rax
0000000100004065         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEElsEm
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::operator<<(unsigned long)
000000010000406a         lea        rsi, qword [ds:0x10048cb69]
   ; ")"
0000000100004071         mov        edx, 0x1
0000000100004076         mov        rdi, rax
0000000100004079         call       imp___stubs___ZNSt3__124__put_character_sequence
```

```
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
000000010000407e         mov        rbx, rax
0000000100004081         mov        rax, qword [ds:rbx]
0000000100004084         mov        rsi, qword [ds:rax-0x18]
0000000100004088         add        rsi, rbx
000000010000408b         lea        rdi, qword [ss:rbp+var_48]
000000010000408f         call       imp___stubs___ZNKSt3__18ios_base6getlocEv
   ; std::__1::ios_base::getloc() const
0000000100004094         mov        rsi, qword [ds:imp___got___ZNSt3__15ctypeIcE2idE]
000000010000409b         lea        rdi, qword [ss:rbp+var_48]
000000010000409f         call       imp___stubs___ZNKSt3__16locale9use_facetERNS0_2idE
   ; std::__1::locale::use_facet(std::__1::locale::id&) const
00000001000040a4         mov        rcx, qword [ds:rax]
00000001000040a7         mov        rcx, qword [ds:rcx+0x38]
00000001000040ab         mov        esi, 0xa
00000001000040b0         mov        rdi, rax
00000001000040b3         call       rcx
00000001000040b5         mov        r14b, al
00000001000040b8         lea        rdi, qword [ss:rbp+var_48]
00000001000040bc         call       imp___stubs___ZNSt3__16localeD1Ev
   ; std::__1::locale::~locale()
00000001000040c1         movsx      esi, r14b
00000001000040c5         mov        rdi, rbx
00000001000040c8         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEE3putEc
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::put(char)
00000001000040cd         mov        rdi, rbx
00000001000040d0         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEE5flushEv
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::flush()
00000001000040d5         jmp        0x100004146

00000001000040d7         mov        rdi, qword [ds:imp___got___ZNSt3__14coutE]
   ; XREF=sub_100003d70+204, sub_100003d70+627
00000001000040de         lea        rsi, qword [ds:0x10048cb39]
   ; "DFS (end not found)"
00000001000040e5         mov        edx, 0x12
00000001000040ea         call       imp___stubs___ZNSt3__124__put_character_sequence
   IcNS_11char_traitsIcEEEERNS_13basic_ostreamIT_T0_EES7_PKS4_m
    ; std::__1::basic_ostream<char, std::__1::char_traits<char> >&
    std::__1::__put_character_sequence<char, std::__1::char_traits<char> >
    (std::__1::basic_ostream<char, std::__1::char_traits<char> >&, char const*, unsigned long)
00000001000040ef         mov        rbx, rax
00000001000040f2         mov        rax, qword [ds:rbx]
00000001000040f5         mov        rsi, qword [ds:rax-0x18]
00000001000040f9         add        rsi, rbx
00000001000040fc         lea        rdi, qword [ss:rbp+var_50]
0000000100004100         call       imp___stubs___ZNKSt3__18ios_base6getlocEv
   ; std::__1::ios_base::getloc() const
0000000100004105         mov        rsi, qword [ds:imp___got___ZNSt3__15ctypeIcE2idE]
000000010000410c         lea        rdi, qword [ss:rbp+var_50]
0000000100004110         call       imp___stubs___ZNKSt3__16locale9use_facetERNS0_2idE
   ; std::__1::locale::use_facet(std::__1::locale::id&) const
0000000100004115         mov        rcx, qword [ds:rax]
0000000100004118         mov        rcx, qword [ds:rcx+0x38]
000000010000411c         mov        esi, 0xa
0000000100004121         mov        rdi, rax
0000000100004124         call       rcx
0000000100004126         mov        r14b, al
0000000100004129         lea        rdi, qword [ss:rbp+var_50]
000000010000412d         call       imp___stubs___ZNSt3__16localeD1Ev
```

```
   ; std::__1::locale::~locale()
0000000100004132         movsx      esi, r14b
0000000100004136         mov        rdi, rbx
0000000100004139         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEE3putEc
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::put(char)
000000010000413e         mov        rdi, rbx
0000000100004141         call       imp___stubs___ZNSt3__113basic_ostream
   IcNS_11char_traitsIcEEE5flushEv
   ; std::__1::basic_ostream<char, std::__1::char_traits<char> >::flush()

0000000100004146         mov        rbx, qword [ss:rbp+var_78]
   ; XREF=sub_100003d70+869
000000010000414a         mov        r14, qword [ss:rbp+var_70]
000000010000414e         mov        rdi, qword [ss:rbp+var_60]
0000000100004152         mov        rax, rdi
0000000100004155         shr        rax, 0x9
0000000100004159         lea        rcx, qword [ds:rbx+rax*8]
000000010000415d         xor        edx, edx
000000010000415f         mov        rax, r14
0000000100004162         sub        rax, rbx
0000000100004165         mov        esi, 0x0
000000010000416a         je         0x1000041b0

000000010000416c         mov        rdx, rdi
000000010000416f         and        rdx, 0x1ff
0000000100004176         shl        rdx, 0x3
000000010000417a         add        rdx, qword [ds:rcx]
000000010000417d         add        rdi, qword [ss:rbp+var_58]
0000000100004181         mov        rsi, rdi
0000000100004184         shr        rsi, 0x9
0000000100004188         and        rdi, 0x1ff
000000010000418f         shl        rdi, 0x3
0000000100004193         add        rdi, qword [ds:rbx+rsi*8]
0000000100004197         mov        rsi, rdi
000000010000419a         jmp        0x1000041b0

000000010000419c         mov        rdx, qword [ds:rcx+8]
   ; XREF=sub_100003d70+1112
00000001000041a0         add        rcx, 0x8
00000001000041a4         nop        word [cs:rax+rax]

00000001000041b0         cmp        rdx, rsi
   ; XREF=sub_100003d70+1018, sub_100003d70+1066, sub_100003d70+1110
00000001000041b3         je         0x1000041ca

00000001000041b5         add        rdx, 0x8
00000001000041b9         mov        rdi, rdx
00000001000041bc         sub        rdi, qword [ds:rcx]
00000001000041bf         cmp        rdi, 0x1000
00000001000041c6         jne        0x1000041b0

00000001000041c8         jmp        0x10000419c

00000001000041ca         mov        qword [ss:rbp+var_58], 0x0
   ; XREF=sub_100003d70+1091
00000001000041d2         sar        rax, 0x3
00000001000041d6         cmp        rax, 0x3
00000001000041da         jb         0x100004208

00000001000041dc         nop        dword [ds:rax]

00000001000041e0         mov        rdi, qword [ds:rbx]
   ; XREF=sub_100003d70+1174
```

```
00000001000041e3         call       imp___stubs___ZdlPv
   ; operator delete(void*)
00000001000041e8         mov        rbx, qword [ss:rbp+var_78]
00000001000041ec         add        rbx, 0x8
00000001000041f0         mov        qword [ss:rbp+var_78], rbx
00000001000041f4         mov        r14, qword [ss:rbp+var_70]
00000001000041f8         mov        rax, r14
00000001000041fb         sub        rax, rbx
00000001000041fe         sar        rax, 0x3
0000000100004202         cmp        rax, 0x2
0000000100004206         ja         0x1000041e0

0000000100004208         cmp        rax, 0x2
   ; XREF=sub_100003d70+1130
000000010000420c         jne        0x100004218

000000010000420e         mov        qword [ss:rbp+var_60], 0x200
0000000100004216         jmp        0x100004226

0000000100004218         cmp        rax, 0x1
   ; XREF=sub_100003d70+1180
000000010000421c         jne        0x100004226

000000010000421e         mov        qword [ss:rbp+var_60], 0x100

0000000100004226         cmp        rbx, r14
   ; XREF=sub_100003d70+1190, sub_100003d70+1196
0000000100004229         je         0x100004263

000000010000422b         nop        dword [ds:rax+rax]

0000000100004230         mov        rdi, qword [ds:rbx]
   ; XREF=sub_100003d70+1231
0000000100004233         call       imp___stubs___ZdlPv
   ; operator delete(void*)
0000000100004238         add        rbx, 0x8
000000010000423c         cmp        r14, rbx
000000010000423f         jne        0x100004230

0000000100004241         mov        rcx, qword [ss:rbp+var_78]
0000000100004245         mov        rax, qword [ss:rbp+var_70]
0000000100004249         cmp        rax, rcx
000000010000424c         je         0x100004263

000000010000424e         lea        rdx, qword [ds:rax-8]
0000000100004252         sub        rdx, rcx
0000000100004255         not        rdx
0000000100004258         and        rdx, 0xfffffffffffffff8
000000010000425c         add        rdx, rax
000000010000425f         mov        qword [ss:rbp+var_70], rdx

0000000100004263         mov        rdi, qword [ss:rbp+var_80]
   ; XREF=sub_100003d70+1209, sub_100003d70+1244
0000000100004267         test       rdi, rdi
000000010000426a         je         0x100004271

000000010000426c         call       imp___stubs___ZdlPv
   ; operator delete(void*)

0000000100004271         mov        rax, qword [ds:imp___got____stack_chk_guard]
   ; XREF=sub_100003d70+1274
0000000100004278         mov        rax, qword [ds:rax]
000000010000427b         cmp        rax, qword [ss:rbp+var_30]
000000010000427f         jne        0x1000043df
```

```

0000000100004285        add       rsp, 0x58
0000000100004289        pop       rbx
000000010000428a        pop       r12
000000010000428c        pop       r13
000000010000428e        pop       r14
0000000100004290        pop       r15
0000000100004292        pop       rbp
0000000100004293        ret

00000001000043df        call      imp___stubs____stack_chk_fail
   ; XREF=sub_100003d70+1295
```

B Crackme Reverse Engineered Source

Listing 3: `src/challenge74.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <time.h>

char const *kHelloHello = "HELLO-HELLO";
uint32_t const kDeadBabe = 0xbebaadde;
uint64_t const kUnknownConstant = 0x42f0e1eb0badbad0;

int get_year() { // sub_40074d
    time_t const timer = time(NULL);
    return localtime(&timer)->tm_year;
}

int get_month() { // sub_40077c
    time_t const timer = time(NULL);
    return localtime(&timer)->tm_mon;
}

int get_day() { // sub_4007ab
    time_t const timer = time(NULL);
    return localtime(&timer)->tm_mday;
}

uint8_t calculate_checksum(uint64_t value, uint8_t const *current, size_t length) { // sub_4007da
    value = ~value;

    for(int i = length; i != 0; i--) {
        value ^= *current;
        current++;

        for(int j = 0; j <= 7; j++) {
            if((value & 0x1) != 0) {
                value >>= 1;
                value ^= kUnknownConstant;
            } else {
                value >>= 1;
            }
        }
    }

    return ~value;
```

```
}

bool validate_serial(uint8_t const arg[]) { // sub_40085e
    if(memcmp(arg, &kDeadBabe, 4) != 0) {
        return false;
    }

    uint16_t const year = arg[5] | arg[4] << 8;
    uint8_t const month = arg[6];
    uint8_t const day = arg[7];

    if(get_year() > year) {
        return false;
    }

    if(get_year() == year) {
        if(get_month() > month) {
            return false;
        }

        if(get_month() == month && get_day() > day) {
            return false;
        }
    }

    if(year > 2015 && month != 0 && day != 0) {
        if(year <= 2100 && month <= 12 && day <= 31) {
            if(calculate_checksum(0, arg, 24) != arg[16]) {
                return false;
            }

            printf("Expiration date: %04d-%02d-%02d\n", year, month, day);

            if((arg[8] & 0x40) != 0) {
                puts("Feature A: ON");
            } else {
                puts("Feature A: OFF");
            }

            if((arg[9] & 0x1) != 0) {
                puts("Feature B: ON");
            } else {
                puts("Feature B: OFF");
            }

            if((arg[10] & 0x2) != 0) {
                puts("Feature C: ON");
            } else {
                puts("Feature C: OFF");
            }

            if((arg[11] & 0x8) != 0) {
                puts("Feature D: ON");
            } else {
                puts("Feature D: OFF");
            }

            if((arg[12] & 0x1) != 0) {
                puts("Feature D: ON");
            } else {
                puts("Feature D: OFF");
            }

            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

int decode_char(char c) { // sub_400a9f
    if(c >= ’0’ && c <= ’9’) {
        return c - ’0’;
    }

    if(c >= ’A’ && c <= ’Z’) {
        return c - ’A’ + 10;
    }

    return -1;
}

int decode_segment(char const segment[]) { // sub_400ad9
    int c0 = decode_char(segment[0]);
    int c1 = decode_char(segment[1]);
    int c2 = decode_char(segment[2]);
    int c3 = decode_char(segment[3]);
    int c4 = decode_char(segment[4]);

    if(c0 == -1 || c1 == -1 || c2 == -1 || c3 == -1 || c4 == -1) {
        return -1;
    }

    return c0 + (c1 + (c2 + (c3 + (c4) * 36) * 36) * 36) * 36;
}

int decode_serial(char const arg[], uint8_t result[]) { // sub_400bb5
    // check dashes
    for(int i = 0; i <= 6; i++) {
        if(arg[i * 6 + 5] != ’-’) {
            return -1;
        }
    }

    int result_index = 0;
    for(int i = 0; i <= 7; i++) {
        int const val = decode_segment(&arg[i * 6]);

        if(val == -1) {
            return -1;
        }

        // check for overflow
        if(val > 0xffffff) {
            return -1;
        }

        // write to result
        result[result_index++] = (uint8_t)(val >> 0x00);
        result[result_index++] = (uint8_t)(val >> 0x08);
        result[result_index++] = (uint8_t)(val >> 0x10);
    }

    return 0;
}

```

```c
int main(int argc, char *argv[]) { // sub_400cb4
    puts("Crackme/keygenme by Dennis Yurichev, http://challenges.re/74");
    putchar(’\n’);

    if(argc == 1) {
        puts("Command line: <serial number>");
        exit(0);
    }

    if(memcmp(&argv[1][30], kHelloHello, sizeof(kHelloHello) / sizeof(*kHelloHello)) != 0) {
        puts("SN format is incorrect");
        exit(0);
    }

    uint8_t result[24];
    if(decode_serial(argv[1], result) == -1) {
        puts("SN format is incorrect");
        exit(0);
    }

    if(validate_serial(result)) {
        puts("SN valid");
    } else {
        puts("SN is not valid");
    }
}
```