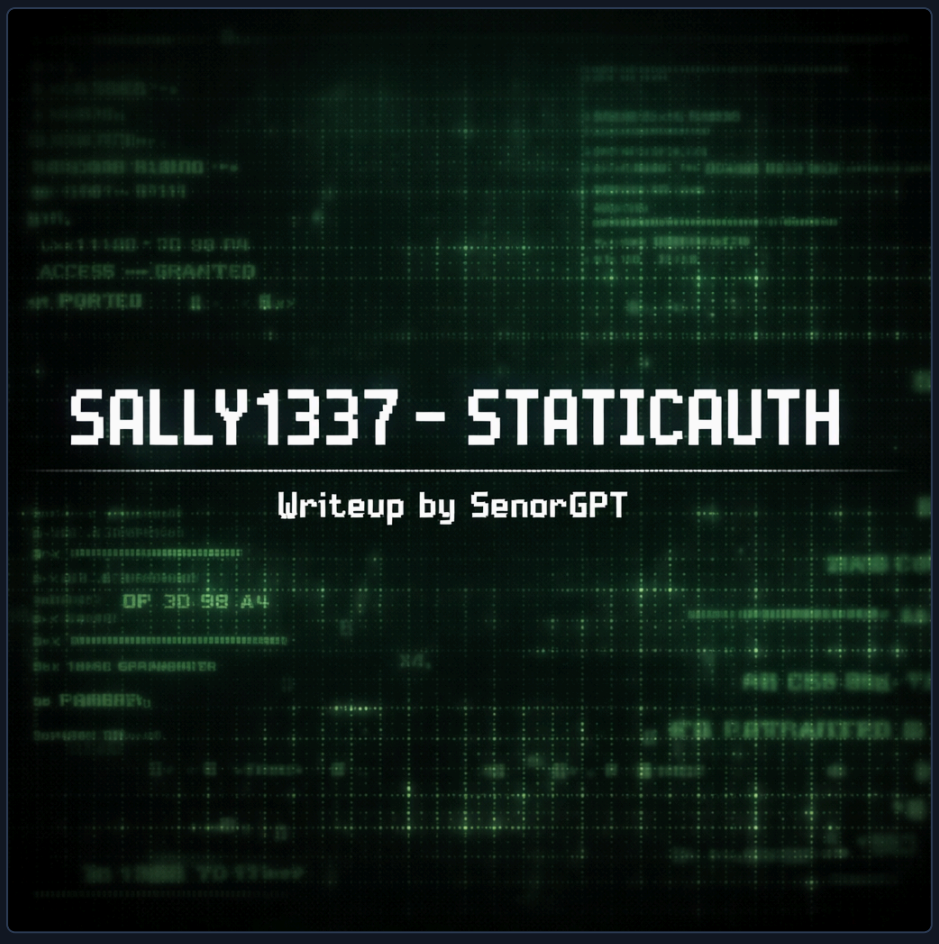
sally1337 - StaticAuth — Reverse Engineering Write-up

Challenge link: https://crackmes.one/crackme/6947f3c00c16072f40f5a2b0
Author: *sally1337*
Write-up by: *SenorGPT*
Tools used: *Binary Ninja*

Platform	Difficulty	Quality	Arch	Language
Windows	2.0	4.0	x86-64	C/C++



Status: Complete
Goal: Document a clean path from initial recon → locating key-check logic → validation/reversal strategy

---

sally1337 - StaticAuth — Reverse Engineering Write-up

---

1. Executive Summary

This crackme is a small *Windows x64* console binary that implements a "system authentication" prompt. The user is asked to enter an authentication code, with a limited number of attempts. The correct password is embedded in the binary in a lightly obfuscated form and is reconstructed at runtime using a few helper functions.

The imposed constraint for this challenge was to solve it purely using **static analysis**. No dynamic debugging, no stepping, and no tracing. I used **Binary Ninja (HLIL)** to follow the control flow, identify `main`, and reverse the flag-generation logic.

The end result of the analysis is the recovered authentication code:

▶ Click to reveal password

Static analysis alone was enough to:

- Find the input-handling and validation logic in `main`.
- Identify the helper routines that assemble the flag string.
- Confirm the expected program behaviour (attempt counter, lockout, success message).

- Verify that `goodjob123` is the correct authentication code.

1.1 Crackme Description

This is a classic, beginner-friendly reverse engineering challenge designed to teach static analysis techniques. A password is hidden within the binary using simple obfuscation methods. Your task is to analyze the executable ***without running it***, locate the obfuscated data, reconstruct the correct key, and enter it into the program.

---

2. Target Overview

2.1 UI / Behaviour

As per the rules of the *crackme*, I am to analyse it ***without running it.***

---

3. Tooling & Environment

- OS: *Windows 11*
- Decompiler: *Binary Ninja Free*

---

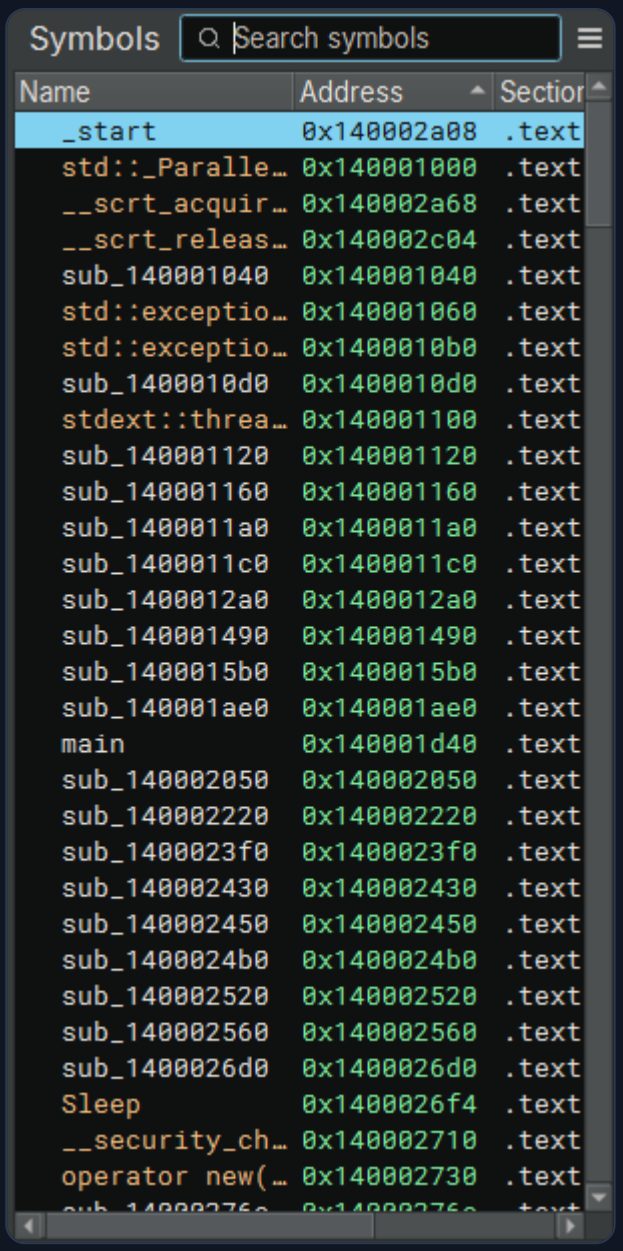
4. Static Recon

I didn't bother importing the binary into *CFF Explorer* this time as it was just purely static analysis.

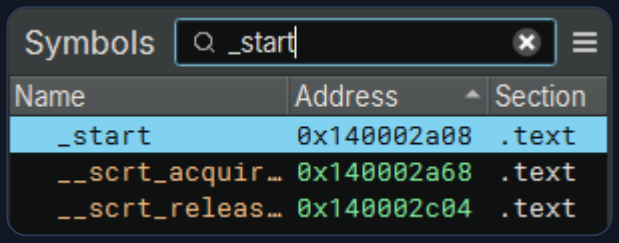
---

5. Static Analysis

Loading up the *Portable Executable* (*PE*) within *Binary Ninja* I begin by targeting the *Symbols*.



I am trying to find where the entry point is. I try the following common symbols; `entry`, `_start`, and `WinMainCRTStartup`. I get a hit on `_start`.



Double clicking it opens up the `_start` function in the *High Level IL* view.

```
140002a08    int64_t _start()

140002a0c        sub_140002ca8()
140002a15        return __scrt_common_main_seh() __tailcall
```

Binary Ninja auto labels functions starting with the prefix `sub_`. The first function is just *C Run Time* (*CRT*) and initialization code. Double clicking on `__scrt_common_main_seh` lands me in that function where there should be a call to the `main` function somewhere.

```
1400028a7        if (__scrt_initialize_crt(1) == 0)
1400029e8            sub_140002dcc(7)
1400029e8            noreturn
1400029e8
1400028ad        int64_t rsi
1400028ad        rsi.b = 0
1400028b0        char var_18 = 0
1400028ba        int64_t rbx
1400028ba        rbx.b = __scrt_acquire_startup_lock()
1400028bc        int32_t rcx = data_140006120
1400028bc
1400028c5        if (rcx == 1)
1400029f3            sub_140002dcc(7)
1400029f3            noreturn
1400029f3
1400028cd        if (rcx != 0)
140002919            rsi.b = 1
14000291c            char var_18_1 = 1
1400028cd        else
1400028cf            data_140006120 = 1
1400028cf
1400028ee            if (_initterm_e(_First: &data_140004298, _Last:
&data_1400042b0) != 0)
1400028f0                return 0xff
1400028f0
140002908            _initterm(_First: &data_140004280, _Last: &data_140004290)
14000290d            data_140006120 = 2
14000290d
140002921        rcx.b = rbx.b
140002923        __scrt_release_startup_lock(rcx.b)
140002923
140002934        if (data_1400061a8 != 0
```

```
140002934                &&
__scrt_is_nonwritable_in_current_image(&data_1400061a8) != 0)
14000294e            data_1400061a8(0, 2, 0)
14000294e
140002960        if (data_1400061a0 != 0
140002960                &&
__scrt_is_nonwritable_in_current_image(&data_1400061a0) != 0)
140002971            _register_thread_local_exe_atexit_callback(_Callback:
data_1400061a0)
140002971
140002976        _get_initial_narrow_environment()
140002983        *__p___argv()
140002991        *__p___argc()
140002993        main()
140002993
1400029a1        if (sub_140002ddc() == 0)
1400029fa            exit(_Except: 0)
1400029fa            noreturn
1400029fa
1400029a6        if (rsi.b == 0)
1400029a8            _cexit()
1400029a8
1400029b1        __scrt_uninitialize_crt(1, 0)
1400029b6        return 0
```

Bingo!

```
140002993        main()
```

Double clicking it lands us in `main`.
Boy that's a long function.

```
140001d57        void var_a8
140001d57        int64_t rax_1 = __security_cookie ^ &var_a8
140001d60        int32_t var_88 = 0
140001d71        sub_140002050(std::cout, "System authentication required.\n")
140001d76        int32_t r15 = 3
140001d7c        int64_t r14
140001d7c        r14.b = 0
140001d7c
140001d83        while (r14.b == 0)
140001d97            sub_140002050(std::cout, "\nEnter authentication code: ")
```

```
140001da0            int128_t var_80 = zx.o(0)
140001da4            char* rbx_1 = nullptr
140001da6            char* var_70_1 = nullptr
140001daa            int64_t rdi_1 = 0xf
140001daf            int64_t var_68_1 = 0xf
140001db3            var_80.b = 0
140001db9            int32_t var_88_1 = 1
140001dbc            char i_2
140001dbc            int64_t rdx_1
140001dbc            int64_t r8_1
140001dbc            i_2, rdx_1, r8_1 = _getch()
140001dc2            char i_1 = i_2
140001dc2
140001dc6            if (i_2 != 0xd)
140001e5e                char i
140001e5e
140001e5e                do
140001dd3                    if (i_1 != 8)
140001e0c                        if (i_1 - 0x20 u<= 0x5e)
140001e11                            if (rbx_1 u>= rdi_1)
140001e3a                                sub_140002560(&var_80, rdx_1, r8_1, i_1)
140001e11                            else
140001e17                                var_70_1 = &rbx_1[1]
140001e1b                                char* rax_5 = &var_80
140001e1b
140001e23                                if (rdi_1 u> 0xf)
140001e23                                    rax_5 = var_80.q
140001e23
140001e28                                *(rax_5 + rbx_1) = i_1
140001e2b                                *(rax_5 + rbx_1 + 1) = 0
140001e2b
140001e46                            sub_140002220(std::cout)
140001e4b                            rbx_1 = var_70_1
140001e4f                            rdi_1 = var_68_1
140001dd3                    else if (rbx_1 != 0)
140001ddd                        var_70_1 = rbx_1 - 1
140001de1                        int128_t* rax_2 = &var_80
140001de1
140001de9                        if (rdi_1 u> 0xf)
140001de9                            rax_2 = var_80.q
140001de9
140001dee                        *(rbx_1 - 1 + rax_2) = 0
140001e00                        sub_140002050(std::cout, &data_1400043a0)
140001e4b                        rbx_1 = var_70_1
```

```
140001e4f                        rdi_1 = var_68_1
140001e4f
140001e53                    i, rdx_1, r8_1 = _getch()
140001e59                    i_1 = i
140001e5e                while (i != 0xd)
140001e5e
140001e72            std::ostream::operator<<(this: std::cout, sub_1400023f0)
140001e7c            int128_t var_60
140001e7c            sub_1400015b0(&var_60)
140001e88            int64_t var_50
140001e88            char* rbx_3
140001e88            int64_t var_48
140001e88
140001e88            if (rbx_1 == var_50)
140001ece                int64_t rdx_5 = 0
140001ed0                rbx_3 = var_80.q
140001ed8                int128_t* r8_2 = var_60.q
140001ed8
140001edf                if (var_50 == 0)
140001f1d                label_140001f1d:
140001f1d
140001f21                    if (var_48 u<= 0xf)
140001f54                        r14.b = 1
140001f57                        sub_140001ae0()
140001f21                    else
140001f2e                        if (var_48 + 1 u>= 0x1000)
140001f30                            int128_t* rax_9 = *(r8_2 - 8)
140001f30
140001f3f                            if (r8_2 - rax_9 - 8 u> 0x1f)
140002038                                trap(0xd)
140002038
140001f49                            r8_2 = rax_9
140001f49
140001f4f                        sub_140002774(_Block: r8_2)
140001f54                        r14.b = 1
140001f57                        sub_140001ae0()
140001edf                else
140001ef0                    while (true)
140001ef0                        char* rcx_10 = &var_80
140001ef0
140001ef8                        if (rdi_1 u> 0xf)
140001ef8                            rcx_10 = rbx_3
140001ef8
140001efc                        int128_t* rax_7 = &var_60
```

```
140001efc
140001f04                        if (var_48 u> 0xf)
140001f04                            rax_7 = r8_2
140001f04
140001f0f                        if (rcx_10[rdx_5] != *(rax_7 + rdx_5))
140001f0f                            break
140001f0f
140001f15                        rdx_5 += 1
140001f15
140001f1b                        if (rdx_5 u>= var_50)
140001f1b                            goto label_140001f1d
140001f1b
140001fe9                    if (var_48 u<= 0xf)
140002018                        r15 -= 1
140002029                        sub_140002050(std::cout, "Authentication
failed.\n")
140001fe9                    else
140001ff6                        if (var_48 + 1 u>= 0x1000)
140001ff8                            int128_t* rax_12 = *(r8_2 - 8)
140001ff8
140002007                            if (r8_2 - rax_12 - 8 u> 0x1f)
140002038                                trap(0xd)
140002038
14000200d                            r8_2 = rax_12
14000200d
140002013                        sub_140002774(_Block: r8_2)
140002018                        r15 -= 1
140002029                        sub_140002050(std::cout, "Authentication
failed.\n")
140001e88            else
140001e92                if (var_48 u> 0xf)
140001e97                    void* rcx_7 = var_60.q
140001e97
140001ea2                    if (var_48 + 1 u>= 0x1000)
140001ea4                        void* rax_6 = *(rcx_7 - 8)
140001ea4
140001eb3                        if (rcx_7 - rax_6 - 8 u> 0x1f)
140002038                            trap(0xd)
140002038
140001ebd                        rcx_7 = rax_6
140001ebd
140001ec0                    sub_140002774(_Block: rcx_7)
140001ec0
140001ec5                rbx_3 = var_80.q
```

```
140002018                r15 -= 1
140002029                sub_140002050(std::cout, "Authentication failed.\n")
140002029
140001f63            if (rdi_1 u> 0xf)
140001f70                if (rdi_1 + 1 u>= 0x1000)
140001f72                    char* rax_10 = *(rbx_3 - 8)
140001f72
140001f81                    if (rbx_3 - rax_10 - 8 u> 0x1f)
14000203f                        trap(0xd)
14000203f
140001f8b                    rbx_3 = rax_10
140001f8b
140001f91                sub_140002774(_Block: rbx_3)
140001f91
140001f99            if (r15 s<= 0)
140001fa2                if (r14.b == 0)
140001fb2                    sub_140002050(std::cout, "\nMaximum attempts
reached.\n")
140001fc5                    sub_140002050(std::cout, "System locked.\n")
140001fc5
140001fa2                break
140001fa2
140001fd3        __security_check_cookie(rax_1 ^ &var_a8)
140001fe4        return 0
```

Analysing the code I can see that upon running it we should be greeted with the following messages where it prompts us for an authentication code.

```
System authentication required

Enter authentication code:
```

It seems to create a while loop that loops until `r14` is toggled to *1* (*true*). I go ahead and rename `r14` to `break_loop`.

I notice that the user input is being captured one character at a time and stored into `i_2`, which I rename to `char_input`. `char_input` is being copied into `rcx_2`, which I rename to `char_input_copy`.

```
140001dbc           i_2 = _getch();
140001dc2           char rcx_2 = i_2;
```

Followed by the first conditional within the *while* loop which checks for the *carriage return* symbol `0xD` (*13*).

```
140001dc6           if (i_2 != 0xd)
```

We have another character definition, `i`. Looking further into where this character is used it seems to be doing the same task as `char_input`. So I rename it to `char_input2`.

```
140001e5e               char i;
```

This definition is followed by another *do while* loop.

```
140001e5e               do
140001e5e               {
140001dd3                   if (char_input_copy != 8)
140001dd3                   {
140001e0c                       if (char_input_copy - 0x20 <= 0x5e)
140001e0c                       {
140001e11                           if (rbx_1 >= rdi_1)
140001e3a                               sub_140002560(&var_80, rdx_1, r8_1,
char_input_copy);
140001e11                           else
140001e11                           {
140001e17                               var_70_1 = &rbx_1[1];
140001e1b                               char* rax_5 = &var_80;
140001e1b
140001e23                               if (rdi_1 > 0xf)
140001e23                                   rax_5 = (uint64_t)var_80;
140001e23
140001e28                               *(uint8_t*)(rax_5 + rbx_1) =
char_input_copy;
140001e2b                               *(uint8_t*)(rax_5 + rbx_1 + 1) = 0;
140001e11                           }
140001e11
140001e46                           sub_140002220(std::cout);
140001e4b                           rbx_1 = var_70_1;
```

```
140001e4f                           rdi_1 = var_68_1;
140001e0c                       }
140001dd3                   }
140001dd3                   else if (rbx_1)
140001dd8                   {
140001ddd                       var_70_1 = rbx_1 - 1;
140001de1                       int128_t* rax_2 = &var_80;
140001de1
140001de9                       if (rdi_1 > 0xf)
140001de9                           rax_2 = (uint64_t)var_80;
140001de9
140001dee                       *(uint8_t*)(rbx_1 - 1 + rax_2) = 0;
140001e00                       sub_140002050(std::cout, &data_1400043a0);
140001e4b                       rbx_1 = var_70_1;
140001e4f                       rdi_1 = var_68_1;
140001dd8                   }
140001dd8
140001e53                   char_input2 = _getch();
140001e59                   char_input_copy = char_input2;
140001e5e               } while (char_input2 != 0xd);
```

At the end of the loop we can see `char_input2` being used in the same way `char_input` was.

```
140001e53                   char_input2 = _getch();
140001e59                   char_input_copy = char_input2;
```

The *do while* loop also breaks when `char_input2` == `0xD`.

```
140001e5e               } while (char_input2 != 0xd);
```

The first conditional in the loop seems to check if the input character is a *backspace* (`0x8`). If it is and `rbx_1` is assigned, we do *something*. This *something* is actually just checking if we already have an input and if backspace is pressed, then delete a character off the stored input.

If it is not a *backspace character*, grab the character input, subtract `0x20` and check if it is equal or less than `0x5E`.

```
140001e0c                       if (char_input_copy - 0x20 <= 0x5e)
```

This is an interesting check.
`char_input_copy - 0x20 <= 0x5e` is mathematically the same as `0x20 <= char_input_copy <= 0x7e`.
So, with that, we can deduce that this conditional is verifying that the input character is within `0x20` (*32*) and `0x7E` (*126*). Which is the printable character set, see ASCII table for a visual reference.

If the user has entered in a valid character another check is performed to see if the length of the input buffer has reached its capacity.

```
140001e11                       if (len >= capacity)
```

If the capacity has not been reached append the current character input onto the input buffer.
Otherwise, add a null-terminator to the end of the input buffer. This is mostly boilerplate code.

```
140001e11                       if (len >= capacity)
140001e3a                           sub_140002560(&var_80, rdx_1, r8_1,
char_input_copy);
140001e11                       else
140001e11                       {
140001e17                           len_1 = &len[1];
140001e1b                           int128_t* rax_5 = &var_80;
140001e1b
140001e23                           if (capacity > 0xf)
140001e23                               rax_5 = (uint64_t)var_80;
140001e23
140001e28                           *(uint8_t*)((char*)rax_5 + len) =
char_input_copy;
140001e2b                           *(uint8_t*)((char*)rax_5 + len + 1) = 0;
140001e11                       }
```

So, with that I deduce that the first *do while* loop is just responsible for gathering the input from the user.
We can think of that whole loop as the following *Pseudo-C* code:

```
std::string input;

while (true) {
```

```
    char ch = _getch();

    if (ch == '\r') break;  // Enter

    if (ch == '\b') {
        if (!input.empty()) {
            input.pop_back();
            // print some backspace effect
        }
    } else if (isprint((unsigned char)ch)) {
        input.push_back(ch);    // grow if needed
        // print masking char / echo
    }
}
```

I do a quick scan through the rest of the while loop and a conditional at the end catches my eye.

```
140001f99            if (r15 <= 0)
140001f99            {
140001fa2                if (!(uint8_t)break_loop)
140001fa2                {
140001fb2                    sub_140002050(std::cout, "\nMaximum attempts
reached.\n");
140001fc5                    sub_140002050(std::cout, "System locked.\n");
140001fa2                }
140001fa2
140001fa2                break;
140001f99            }
```

From this it is clear that `r15` is an attempt counter of some sort, so I rename it to `attempt_counter`. It is *3* upon initialization.
If all attempts are used up the user should see the following displayed on the console.

```

Maximum attempts reached.
System locked.\n

```

The next conditional after the *do while* loop compares the length of the user input against `var_50`.

```
140001e88           if (len == var_50)
```

If the length does not match then the authentication fails and the user will have `Authentication failed.` output to the console.

```
140001e88           else
140001e88           {
140001e92               if (var_48 > 0xf)
140001e92               {
140001e97                   void* rcx_7 = (uint64_t)var_60;
140001e97
140001ea2                   if (var_48 + 1 >= 0x1000)
140001ea2                   {
140001ea4                       void* rax_6 = *(uint64_t*)((char*)rcx_7 - 8);
140001ea4
140001eb3                       if ((char*)rcx_7 - rax_6 - 8 > 0x1f)
140002038                           trap(0xd);
140002038
140001ebd                       rcx_7 = rax_6;
140001ea2                   }
140001ea2
140001ec0                   sub_140002774(rcx_7);
140001e92               }
140001e92
140001ec5               rbx_2 = (uint64_t)var_80;
140002018               attempt_counter -= 1;
140002029               sub_140002050(std::cout, "Authentication failed.\n");
140001e88           }
```

Upon further analysis, it seems that `var_50` is actually the length of the flag that is being compared against, while `var_60` is the actual flag. I go ahead and rename them `flag_len` and `flag` respectively.

If the length does match, then the following code gets executed.

```
140001ece               int64_t rdx_5 = 0;
```

```
140001ed0                rbx_2 = (uint64_t)var_80;
140001ed8                int128_t* r8_2 = (uint64_t)var_60;
140001ed8
140001edf                if (!var_50)
140001edf                {
140001f1d                label_140001f1d:
140001f1d
140001f21                    if (var_48 <= 0xf)
140001f21                    {
140001f54                        (uint8_t)break_loop = 1;
140001f57                        sub_140001ae0();
140001f21                    }
140001f21                    else
140001f21                    {
140001f2e                        if (var_48 + 1 >= 0x1000)
140001f2e                        {
140001f30                            int128_t* rax_9 = *(uint64_t*)((char*)r8_2 -
8);
140001f30
140001f3f                            if ((char*)r8_2 - rax_9 - 8 > 0x1f)
140002038                                trap(0xd);
140002038
140001f49                            r8_2 = rax_9;
140001f2e                        }
140001f2e
140001f4f                        sub_140002774(r8_2);
140001f54                        (uint8_t)break_loop = 1;
140001f57                        sub_140001ae0();
140001f21                    }
140001edf                }
140001edf                else
140001edf                {
140001ef0                    while (true)
140001ef0                    {
140001ef0                        char* rcx_10 = &var_80;
140001ef0
140001ef8                        if (capacity > 0xf)
140001ef8                            rcx_10 = rbx_2;
140001ef8
140001efc                        int128_t* rax_7 = &var_60;
140001efc
140001f04                        if (var_48 > 0xf)
140001f04                            rax_7 = r8_2;
140001f04
```

```
140001f0f                       if (rcx_10[rdx_5] != *(uint8_t*)((char*)rax_7 +
rdx_5))
140001f0f                           break;
140001f0f
140001f15                       rdx_5 += 1;
140001f15
140001f1b                       if (rdx_5 >= var_50)
140001f1b                           goto label_140001f1d;
140001ef0                   }
140001ef0
140001fe9                   if (var_48 <= 0xf)
140001fe9                   {
140002018                       attempt_counter -= 1;
140002029                       sub_140002050(std::cout, "Authentication
failed.\n");
140001fe9                   }
140001fe9                   else
140001fe9                   {
140001ff6                       if (var_48 + 1 >= 0x1000)
140001ff6                       {
140001ff8                           int128_t* rax_12 = *(uint64_t*)((char*)r8_2
- 8);
140001ff8
140002007                           if ((char*)r8_2 - rax_12 - 8 > 0x1f)
140002038                               trap(0xd);
140002038
14000200d                           r8_2 = rax_12;
140001ff6                       }
140001ff6
140002013                       sub_140002774(r8_2);
140002018                       attempt_counter -= 1;
140002029                       sub_140002050(std::cout, "Authentication
failed.\n");
140001fe9                   }
140001edf               }
```

Function `sub_14001ae0` prints out the validation successful message, so I rename it to `validation_successful`. It seems that the important check is the following *if* statement.

```
140001f21                   if (var_48 <= 0xf)
140001f21                   {
```

```
140001f54                        (uint8_t)break_loop = 1;
140001f57                        validation_successful();
140001f21                    }
140001f21                    else
140001f21                    {
140001f2e                        if (var_48 + 1 >= 0x1000)
140001f2e                        {
140001f30                            int128_t* rax_9 = *(uint64_t*)((char*)r8_2 -
8);
140001f30
140001f3f                            if ((char*)r8_2 - rax_9 - 8 > 0x1f)
140002038                                trap(0xd);
140002038
140001f49                            r8_2 = rax_9;
140001f2e                        }
140001f2e
140001f4f                        sub_140002774(r8_2);
140001f54                        (uint8_t)break_loop = 1;
140001f57                        validation_successful();
140001f21                    }
```

So in order to get to the validation successful branch, we need to pass the following conditionals.

```
140001e88            if (len == var_50)

140001edf                if (!var_50)
```

As it seems that the final *if else* doesn't matter as both branches go towards a successful validation.
Which means we now have another function to decode. For readability I rename the function `sub_1400015b0` to `get_flag`.

```
140001e7c            sub_1400015b0(&var_60);
```

5.1 sub_1400015b0 - get_flag

It seems there is a common trend that these functions are quite long in the decompiler.

```
1400015b0    {
1400015b0        void** result = arg1;
1400015c0        int128_t* var_10 = arg1;
1400015c6        int32_t var_38 = 0;
1400015cc        *(uint128_t*)arg1 = {0};
1400015cf        arg1[1] = 0;
1400015d3        *(uint64_t*)((char*)arg1 + 0x18) = 0xf;
1400015db        *(uint8_t*)arg1 = 0;
1400015dd        int32_t var_38_1 = 1;
1400015e8        void* var_30;
1400015e8        int128_t* rax;
1400015e8        int64_t r8;
1400015e8        rax = sub_1400012a0(&var_30);
1400015e8
1400015f3        if (*(uint64_t*)((char*)rax + 0x18) > 0xf)
1400015f5            rax = *(uint64_t*)rax;
1400015f5
1400015f8        char r9 = *(uint8_t*)rax;
1400015fc        char* rcx_1 = result[2];
140001600        int64_t rdx = result[3];
140001600
140001607        if (rcx_1 >= rdx)
14000162b            sub_140002560(result, rdx, r8, r9);
140001607        else
140001607        {
14000160d            result[2] = &rcx_1[1];
140001611            void** result_1 = result;
140001611
140001618            if (rdx > 0xf)
14000161a                result_1 = *(uint64_t*)result;
14000161a
14000161d            *(uint8_t*)(rcx_1 + result_1) = r9;
140001621            *(uint8_t*)(rcx_1 + result_1 + 1) = 0;
140001607        }
140001607
140001639        int64_t var_18;
140001639
140001639        if (var_18 <= 0xf)
```

```
140001639            goto label_140001672;
140001639
14000163b        void* rax_2 = var_30;
140001649        void* rcx_3;
140001649
140001649        if (var_18 + 1 < 0x1000)
140001649        {
140001666            rcx_3 = rax_2;
140001669        label_140001669:
140001669            sub_140002774(rcx_3);
140001672        label_140001672:
140001672            int128_t* rax_5;
140001672            int64_t r8_1;
140001672            rax_5 = sub_1400012a0(&var_30);
140001672
14000167d            if (*(uint64_t*)((char*)rax_5 + 0x18) > 0xf)
14000167f                rax_5 = *(uint64_t*)rax_5;
14000167f
140001682            char r9_1 = *(uint8_t*)((char*)rax_5 + 1);
140001687            char* rcx_5 = result[2];
14000168b            int64_t rdx_4 = result[3];
14000168b
140001692            if (rcx_5 >= rdx_4)
1400016b6                sub_140002560(result, rdx_4, r8_1, r9_1);
140001692            else
140001692            {
140001698                result[2] = &rcx_5[1];
14000169c                void** result_2 = result;
14000169c
1400016a3                if (rdx_4 > 0xf)
1400016a5                    result_2 = *(uint64_t*)result;
1400016a5
1400016a8                *(uint8_t*)(rcx_5 + result_2) = r9_1;
1400016ac                *(uint8_t*)(rcx_5 + result_2 + 1) = 0;
140001692            }
140001692
1400016c4            if (var_18 <= 0xf)
1400016c4                goto label_1400016fd;
1400016c4
1400016c6            void* rax_7 = var_30;
1400016d4            void* rcx_7;
1400016d4
1400016d4            if (var_18 + 1 < 0x1000)
1400016d4            {
```

```
1400016f1                rcx_7 = rax_7;
1400016f4            label_1400016f4:
1400016f4                sub_140002774(rcx_7);
1400016fd            label_1400016fd:
1400016fd                int128_t* rax_10;
1400016fd                int64_t r8_2;
1400016fd                rax_10 = sub_1400012a0(&var_30);
1400016fd
140001708                if (*(uint64_t*)((char*)rax_10 + 0x18) > 0xf)
14000170a                    rax_10 = *(uint64_t*)rax_10;
14000170a
14000170d                char r9_2 = *(uint8_t*)((char*)rax_10 + 2);
140001712                int64_t rcx_9 = result[2];
140001716                int64_t rdx_8 = result[3];
140001716
14000171d                if (rcx_9 >= rdx_8)
140001741                    r8_2 = sub_140002560(result, rdx_8, r8_2, r9_2);
14000171d                else
14000171d                {
140001723                    result[2] = rcx_9 + 1;
140001727                    void** result_3 = result;
140001727
14000172e                    if (rdx_8 > 0xf)
140001730                        result_3 = *(uint64_t*)result;
140001730
140001733                    *(uint8_t*)((char*)result_3 + rcx_9) = r9_2;
140001737                    *(uint8_t*)((char*)result_3 + rcx_9 + 1) = 0;
14000171d                }
14000171d
14000174f                if (var_18 <= 0xf)
14000174f                    goto label_140001788;
14000174f
140001751                void* rax_12 = var_30;
14000175f                void* rcx_11;
14000175f
14000175f                if (var_18 + 1 < 0x1000)
14000175f                {
14000177c                    rcx_11 = rax_12;
14000177f                label_14000177f:
14000177f                    r8_2 = sub_140002774(rcx_11);
140001788                label_140001788:
140001788                    var_30 = {0};
14000178c                    int64_t var_18_1 = 0xf;
140001794                    int32_t var_38_2 = 3;
```

```
14000179b                   __builtin_strncpy(&var_30, "djob", 5);
1400017a2                   int64_t var_20_1 = 4;
1400017ae                   int64_t rcx_12 = result[2];
1400017b2                   int64_t rdx_12 = result[3];
1400017b2
1400017b9                   if (rcx_12 >= rdx_12)
1400017dd                       r8_2 = sub_140002560(result, rdx_12, r8_2,
0x64);
1400017b9                   else
1400017b9                   {
1400017bf                       result[2] = rcx_12 + 1;
1400017c3                       void** result_4 = result;
1400017c3
1400017ca                       if (rdx_12 > 0xf)
1400017cc                           result_4 = *(uint64_t*)result;
1400017cc
1400017cf                       *(uint16_t*)((char*)result_4 + rcx_12) = 0x64;
1400017b9                   }
1400017b9
1400017e6                   var_30 = {0};
1400017ea                   int64_t var_18_2 = 0xf;
1400017f2                   int32_t var_38_3 = 5;
1400017f9                   __builtin_strncpy(&var_30, "djob", 5);
140001800                   int64_t var_20_2 = 4;
14000180c                   int64_t rcx_14 = result[2];
140001810                   int64_t rdx_13 = result[3];
140001810
140001817                   if (rcx_14 >= rdx_13)
14000183b                       r8_2 = sub_140002560(result, rdx_13, r8_2,
0x6a);
140001817                   else
140001817                   {
14000181d                       result[2] = rcx_14 + 1;
140001821                       void** result_5 = result;
140001821
140001828                       if (rdx_13 > 0xf)
14000182a                           result_5 = *(uint64_t*)result;
14000182a
14000182d                       *(uint16_t*)((char*)result_5 + rcx_14) = 0x6a;
140001817                   }
140001817
140001844                   var_30 = {0};
140001848                   int64_t var_18_3 = 0xf;
140001850                   int32_t var_38_4 = 9;
```

```
140001857                   __builtin_strncpy(&var_30, "djob", 5);
14000185e                   int64_t var_20_3 = 4;
14000186a                   int64_t rcx_16 = result[2];
14000186e                   int64_t rdx_14 = result[3];
14000186e
140001875                   if (rcx_16 >= rdx_14)
140001899                       r8_2 = sub_140002560(result, rdx_14, r8_2,
0x6f);
140001875                   else
140001875                   {
14000187b                       result[2] = rcx_16 + 1;
14000187f                       void** result_6 = result;
14000187f
140001886                       if (rdx_14 > 0xf)
140001888                           result_6 = *(uint64_t*)result;
140001888
14000188b                       *(uint16_t*)((char*)result_6 + rcx_16) = 0x6f;
140001875                   }
140001875
1400018a2                   var_30 = {0};
1400018a6                   int64_t var_18_4 = 0xf;
1400018ae                   int32_t var_38_5 = 0x11;
1400018b5                   __builtin_strncpy(&var_30, "djob", 5);
1400018bc                   int64_t var_20_4 = 4;
1400018c8                   int64_t rcx_18 = result[2];
1400018cc                   int64_t rdx_15 = result[3];
1400018cc
1400018d3                   if (rcx_18 >= rdx_15)
1400018f7                       sub_140002560(result, rdx_15, r8_2, 0x62);
1400018d3                   else
1400018d3                   {
1400018d9                       result[2] = rcx_18 + 1;
1400018dd                       void** result_7 = result;
1400018dd
1400018e4                       if (rdx_15 > 0xf)
1400018e6                           result_7 = *(uint64_t*)result;
1400018e6
1400018e9                       *(uint16_t*)((char*)result_7 + rcx_18) = 0x62;
1400018d3                   }
1400018d3
1400018fc                   int32_t var_38_6 = 1;
140001907                   int64_t* rax_19;
140001907                   int64_t r8_3;
140001907                   rax_19 = sub_140001490(&var_30);
```

```
140001907
140001912                   if (rax_19[3] > 0xf)
140001914                       rax_19 = *(uint64_t*)rax_19;
140001914
140001917                   char r9_3 = *(uint8_t*)rax_19;
14000191b                   int64_t rcx_21 = result[2];
14000191f                   int64_t rdx_16 = result[3];
14000191f
140001926                   if (rcx_21 >= rdx_16)
14000194a                       sub_140002560(result, rdx_16, r8_3, r9_3);
140001926                   else
140001926                   {
14000192c                       result[2] = rcx_21 + 1;
140001930                       void** result_8 = result;
140001930
140001937                       if (rdx_16 > 0xf)
140001939                           result_8 = *(uint64_t*)result;
140001939
14000193c                       *(uint8_t*)((char*)result_8 + rcx_21) = r9_3;
140001940                       *(uint8_t*)((char*)result_8 + rcx_21 + 1) = 0;
140001926                   }
140001926
140001958                   if (var_18_4 <= 0xf)
140001958                       goto label_140001991;
140001958
14000195a                   void* rax_21 = var_30;
140001968                   void* rcx_23;
140001968
140001968                   if (var_18_4 + 1 < 0x1000)
140001968                   {
140001985                       rcx_23 = rax_21;
140001988                   label_140001988:
140001988                       sub_140002774(rcx_23);
140001991                   label_140001991:
140001991                       int64_t* rax_24;
140001991                       int64_t r8_4;
140001991                       rax_24 = sub_140001490(&var_30);
140001991
14000199c                       if (rax_24[3] > 0xf)
14000199e                           rax_24 = *(uint64_t*)rax_24;
14000199e
1400019a1                       char r9_4 = *(uint8_t*)((char*)rax_24 + 1);
1400019a6                       int64_t rcx_25 = result[2];
1400019aa                       int64_t rdx_20 = result[3];
```

```
1400019aa
1400019b1                       if (rcx_25 >= rdx_20)
1400019d5                           sub_140002560(result, rdx_20, r8_4, r9_4);
1400019b1                       else
1400019b1                       {
1400019b7                           result[2] = rcx_25 + 1;
1400019bb                           char* result_9 = result;
1400019bb
1400019c2                           if (rdx_20 > 0xf)
1400019c4                               result_9 = *(uint64_t*)result;
1400019c4
1400019c7                           result_9[rcx_25] = r9_4;
1400019cb                           result_9[rcx_25 + 1] = 0;
1400019b1                       }
1400019b1
1400019e3                       if (var_18_4 <= 0xf)
1400019e3                           goto label_140001a1c;
1400019e3
1400019e5                       void* rax_26 = var_30;
1400019f3                       void* rcx_27;
1400019f3
1400019f3                       if (var_18_4 + 1 < 0x1000)
1400019f3                       {
140001a10                           rcx_27 = rax_26;
140001a13                       label_140001a13:
140001a13                           sub_140002774(rcx_27);
140001a1c                       label_140001a1c:
140001a1c                           int64_t* rax_29;
140001a1c                           int64_t r8_5;
140001a1c                           rax_29 = sub_140001490(&var_30);
140001a1c
140001a27                           if (rax_29[3] > 0xf)
140001a29                               rax_29 = *(uint64_t*)rax_29;
140001a29
140001a2c                           char r9_5 = *(uint8_t*)((char*)rax_29 + 2);
140001a31                           char* rcx_29 = result[2];
140001a35                           int64_t rdx_24 = result[3];
140001a35
140001a3c                           if (rcx_29 >= rdx_24)
140001a6b                               sub_140002560(result, rdx_24, r8_5,
r9_5);
140001a3c                           else
140001a3c                           {
140001a42                               result[2] = &rcx_29[1];
```

```
140001a42
140001a4a                                if (rdx_24 <= 0xf)
140001a4a                                {
140001a5d                                    *(uint8_t*)(rcx_29 + result) = r9_5;
140001a61                                    *(uint8_t*)(rcx_29 + result + 1) =
0;
140001a4a                                }
140001a4a                                else
140001a4a                                {
140001a4c                                    void* rax_31 = *(uint64_t*)result;
140001a4f                                    *(uint8_t*)(rcx_29 + rax_31) = r9_5;
140001a53                                    *(uint8_t*)(rcx_29 + rax_31 + 1) =
0;
140001a4a                                }
140001a3c                            }
140001a3c
140001a79                            if (var_18_4 <= 0xf)
140001acf                                return result;
140001acf
140001a7b                            void* rax_33 = var_30;
140001a7b
140001a89                            if (var_18_4 + 1 < 0x1000)
140001a89                            {
140001abd                                sub_140002774(rax_33);
140001acf                                return result;
140001a89                            }
140001a89
140001a8b                            int64_t rcx_31 = *(uint64_t*)((char*)rax_33
- 8);
140001a8b
140001a9a                            if ((char*)rax_33 - rcx_31 - 8 <= 0x1f)
140001a9a                            {
140001aa0                                sub_140002774(rcx_31);
140001ab2                                return result;
140001a9a                            }
1400019f3                        }
1400019f3                        else
1400019f3                        {
1400019f5                            rcx_27 = *(uint64_t*)((char*)rax_26 - 8);
1400019f5
140001a04                            if ((char*)rax_26 - rcx_27 - 8 <= 0x1f)
140001a04                                goto label_140001a13;
1400019f3                        }
140001968                    }
```

```
140001968                    else
140001968                    {
14000196a                        rcx_23 = *(uint64_t*)((char*)rax_21 - 8);
14000196a
140001979                        if ((char*)rax_21 - rcx_23 - 8 <= 0x1f)
140001979                            goto label_140001988;
140001968                    }
14000175f                }
14000175f                else
14000175f                {
140001761                    rcx_11 = *(uint64_t*)((char*)rax_12 - 8);
140001761
140001770                    if ((char*)rax_12 - rcx_11 - 8 <= 0x1f)
140001770                        goto label_14000177f;
14000175f                }
1400016d4            }
1400016d4            else
1400016d4            {
1400016d6                rcx_7 = *(uint64_t*)((char*)rax_7 - 8);
1400016d6
1400016e5                if ((char*)rax_7 - rcx_7 - 8 <= 0x1f)
1400016e5                    goto label_1400016f4;
1400016d4            }
140001649        }
140001649        else
140001649        {
14000164b            rcx_3 = *(uint64_t*)((char*)rax_2 - 8);
14000164b
14000165a            if ((char*)rax_2 - rcx_3 - 8 <= 0x1f)
14000165a                goto label_140001669;
140001649        }
140001ab8        trap(0xd);
1400015b0    }
```

Instead of breaking this function down bit by bit, I will just go through the interesting parts.

It seems to initialize a pointer `rax` which gets passed to another function `sub_1400012a0`.

```
1400015e8        rax, r8 = sub_1400012a0(&var_30)
```

Time to break down sub_1400012a0. It seems that it just returns the string `goo`. Therefore, `rax == goo`, so I rename it to `flag_piece_goo`.

Skipping a bunch of boilerplate code, I land on `label_140001788`.

```
140001788                label_140001788:
140001788                    var_30.o = zx.o(0)
14000178c                    int64_t var_18_1 = 0xf
140001794                    int32_t var_38_2 = 3
14000179b                    __builtin_strncpy(dest: &var_30, src: "djob", count:
5)
1400017a2                    int64_t var_20_1 = 4
1400017ae                    int64_t rcx_12 = arg1[1].q
1400017b2                    int64_t rdx_12 = *(arg1 + 0x18)
```

Which seems to be appending `djob` onto `var_30`, which I rename to `flag`. So now, `flag` / `var_30` should be `goodjob` as it appended `djob` onto `goo` (the result of the `get_goo` function).

Tracing the logic paste more boilerplate, I land on `label_140001991`.

```
140001991                        int64_t* rax_27
140001991                        int64_t r8_4
140001991                        rax_27, r8_4 = sub_140001490(&flag)
```

After analysing sub_140001490 it seems that it just returns the string `123`. So, our `flag` variable should now be `goodjob123`.
Tracing the logic further, it seems that this is the last transformation that is done on the flag before the it is returned.

With that, let's test the flag out.

5.2 sub_1400012a0

```
1400012b9        void var_68
1400012b9        int64_t rax_1 = __security_cookie ^ &var_68
1400012c4        int128_t* var_40 = arg1
1400012ce        int16_t var_48 = 0x6f67
1400012d5        char var_46 = 0x6f
1400012da        int64_t var_38 = (zx.o(0)).q
1400012e0        int64_t var_28 = 0
1400012ea        int64_t rax_2 = operator new(3)
1400012ef        var_38 = rax_2
1400012f4        var_38:8.q = rax_2
140001310        memmove(dest: rax_2, src: &var_48, count: 3)
140001315        var_38:8.q = rax_2 + 3
14000131a        int64_t rdi = var_38
140001322        *arg1 = zx.o(0)
140001326        arg1[1].q = 0
14000132a        *(arg1 + 0x18) = 0xf
14000132a
140001335        if (rdi != rax_2 + 3)
14000133f            int64_t rbp_1 = rax_2 + 3 - rdi
140001342            int64_t rsi_1 = 0x7fffffffffffffff
140001342
14000134f            if (rbp_1 u> 0x7fffffffffffffff)
14000147c                sub_1400011a0()
14000147c                noreturn
14000147c
140001359            int128_t* rax_7
140001359
140001359            if (rbp_1 u<= 0xf)
1400013eb                rsi_1 = 0xf
1400013f0                rax_7 = nullptr
140001359            else
140001362                int64_t rax_4 = rbp_1 | 0xf
140001369                uint64_t rax_5
140001369
140001369                if (rax_4 u<= 0x7fffffffffffffff)
14000138c                    rsi_1 = rax_4
14000138c
140001397                    if (rax_4 u< 0x16)
140001397                        rsi_1 = 0x16
140001397
1400013a2                    if (rsi_1 != -1)
```

```
1400013b7                       if (rsi_1 + 1 u>= 0x1000)
1400013b9                           rax_5 = rsi_1 + 0x28
1400013b9
1400013c0                           if (rax_5 u> rsi_1 + 1)
1400013c0                               goto label_140001378
1400013c0
140001482                           stdext::threads::_Throw_lock_error()
140001482                           noreturn
140001482
1400013dd                       rax_7 = operator new(rsi_1 + 1)
1400013e2                       *arg1 = rax_7
1400013e5                       *(arg1 + 0x18) = rsi_1
1400013a2                   else
1400013a4                       rax_7 = nullptr
1400013a7                       *arg1 = 0
1400013aa                       *(arg1 + 0x18) = rsi_1
140001369               else
14000136b                   rax_5 = -0x7fffffffffffffd9
140001378               label_140001378:
140001378                   int64_t rax_6 = operator new(rax_5)
140001378
140001383                   if (rax_6 == 0)
14000138a                       trap(0xd)
14000138a
1400013cc                   rax_7 = (rax_6 + 0x27) & 0xfffffffffffffff0
1400013d0                   *(rax_7 - 8) = rax_6
1400013d4                   *arg1 = rax_7
1400013d7                   *(arg1 + 0x18) = rsi_1
1400013d7
1400013f3           int128_t* rbx_1 = arg1
1400013f3
1400013fa           if (rsi_1 u> 0xf)
1400013fa               rbx_1 = rax_7
1400013fa
140001407           memmove(dest: rbx_1, src: rdi, count: rbp_1.d)
14000140c           arg1[1].q = rbp_1
140001410           *(rbx_1 + rbp_1) = 0
140001414           rdi = var_38
140001335       else
140001337           *arg1 = 0
140001337
14000141c       if (rdi != 0)
14000142d           if (rax_2 + 3 - rdi u>= 0x1000)
14000142f               int64_t rax_9 = *(rdi - 8)
```

```
14000142f
14000143e                if (rdi - rax_9 - 8 u> 0x1f)
14000147a                    trap(0xd)
14000147a
140001444                rdi = rax_9
140001444
14000144a            sub_140002774(_Block: rdi)
14000144a
14000145a        __security_check_cookie(rax_1 ^ &var_68)
140001474        return arg1
```

At the top we see `var_48` being assigned `0x6F67`. `0x67 == g` and `0x68 == o`.

```
1400012ce        int16_t var_48 = 0x6f67
1400012d5        char var_46 = 0x6f
```

These three bytes are then copied into `rax_2`, so `rax_2 == goo`.

```
140001310        memmove(dest: rax_2, src: &var_48, count: 3)
```

Note: The majority of `sub_1400012a0` is not part of the actual challenge logic. After extracting the meaningful constant bytes, `0x6f676f == goo`, the remainder of the function consists entirely of **C++ standard library implementation details**, not custom obfuscation.

In particular:

- Large conditionals checking values like `0x7FFFFFFFFFFFFFFF`, `0x16`, `0x1000`
- Branches involving heap metadata and page-aligned allocations
- Calls to `operator new`, `operator delete`, and internal allocator helpers
- Stack-protector setup/teardown (`__security_cookie`, `__security_check_cookie`)
- Alignment arithmetic (`& 0xFFFFFFFFFFFFFFE0`)
- Fallback allocator paths (`trap(0xd)`, `stdext::threads::_Throw_lock_error`)

None of these operations reference:

- user input
- expected authentication data
- constants that contribute to the password
- any transformation logic that affects correctness

They are simply the compiler-generated machinery for constructing a `std::string`, handling *Small-String Optimization* (*SSO*), and freeing temporary buffers.

Therefore, after extracting the 3-byte literal `goo` from the stack variables and confirming that this is the only meaningful data produced by the function, the remainder of the code was safely ignored as **allocator and runtime boilerplate**.

In simplified form, the entire function reduces to:

```
std::string out = "goo";
return out;
```

This matches exactly what the surrounding code expects, and no other values produced in the function influence the authentication logic.

With that, I rename the function to `get_goo` (d son).

6.3 sub_140001490

Finally, a short one!

```
140001499        int64_t* _Str = arg1
1400014a1        __builtin_memset(dest: arg1, ch: 0, count: 0x20)
1400014a4        void* rdi = nullptr
1400014a6        _Str.d = 0x333231
1400014be        uint64_t rax = strlen(&_Str)
1400014c3        int64_t rbp = 0x7fffffffffffffff
1400014c3
1400014d3        if (rax u> 0x7fffffffffffffff)
14000159f            sub_1400011a0()
14000159f            noreturn
14000159f
```

```
1400014dd       if (rax u<= 0xf)
1400014e2           arg1[2] = rax
1400014eb           arg1[3] = 0xf
1400014f6           memcpy(dest: arg1, src: &_Str, count: rax.d)
1400014fe           *(rax + arg1) = 0
14000150a           return arg1
14000150a
14000150b       int64_t rax_2 = rax | 0xf
140001512       uint64_t rax_3
140001512
140001512       if (rax_2 u<= 0x7fffffffffffffff)
140001537           rbp = rax_2
140001537
14000153d           if (rax_2 u< 0x16)
14000153d               rbp = 0x16
14000153d
140001548           if (rbp != -1)
140001551               if (rbp + 1 u>= 0x1000)
140001553                   rax_3 = rbp + 0x28
140001553
14000155a                   if (rax_3 u> rbp + 1)
14000155a                       goto label_140001521
14000155a
1400015a5                   stdext::threads::_Throw_lock_error()
1400015a5                   noreturn
1400015a5
140001571               rdi = operator new(rbp + 1)
140001512       else
140001514           rax_3 = -0x7fffffffffffffd9
140001521       label_140001521:
140001521           int64_t rax_4 = operator new(rax_3)
140001521
140001529           if (rax_4 == 0)
140001530               trap(0xd)
140001530
140001562           rdi = (rax_4 + 0x27) & 0xfffffffffffffff0
140001566           *(rdi - 8) = rax_4
140001577       *arg1 = rdi
14000157f       arg1[2] = rax
140001586       arg1[3] = rbp
14000158a       memcpy(dest: rdi, src: &_Str, count: rax.d)
140001592       *(rax + rdi) = 0
14000159e       return arg1
```

It seems that all this function is doing is returning a string, which upon first assumption we think is `321`.
We see the bytes `0x333231`, which are the bytes that are getting written.

```
_Str.d = 0x333231
```

But since it's on *little-endian*, the value laid out in memory would be

```
0x31   0x32   0x33   0x00
 '1'    '2'    '3'   '\0'
```

So the actual string getting returned here is `123`.

5.4 validation_successful

```
140001ae0    int64_t validation_successful()

140001ae0    {
140001ae0        sub_140002050(std::cout, "\nVerification successful.\n\n");
140001b12        sub_140002050(std::cout, "Secure data accessed:\n");
140001b25        sub_140002050(std::cout, "----------------------\n");
140001b38        sub_140002050(std::cout, "Location: Grid 47-C\n");
140001b4b        sub_140002050(std::cout, "Access code: 8891-3324\n");
140001b5e        sub_140002050(std::cout, "Status: Active\n");
140001b71        sub_140002050(std::cout, "Timestamp: Verified\n\n");
140001b84        sub_140002050(std::cout, "Clearing in 15 seconds...\n");
140001b84
140001d08        for (int32_t i = 0xf; i > 0; i -= 1)
140001d08        {
140001bf7            std::ostream::operator<<(
140001bf7 
 sub_140002050(std::ostream::operator<<(sub_140002050(std::cout, u"\r")),
140001bf7                    u" "),
140001bf7                sub_140002430);
140001c02            int64_t arg_8;
140001c02            sub_1400011c0(&arg_8);
140001c07            int64_t rbx_1 = arg_8;
140001c0f            int64_t rbx_2;
140001c0f
```

```
140001c0f           rbx_2 = rbx_1 > 0x7fffffffc46535fe ? 0x7fffffffffffffff :
rbx_1 + 0x3b9aca00;
140001c0f
140001c20           while (true)
140001c20           {
140001c20               int64_t rax_4 = _Query_perf_frequency();
140001c29               int64_t rax_5 = _Query_perf_counter();
140001c39               int64_t rax_6;
140001c39
140001c39               if (rax_4 != 0x989680)
140001c39               {
140001c48                   int64_t rax_14;
140001c48                   int64_t rcx_18;
140001c48
140001c48                   if (rax_4 != 0x16e3600)
140001c48                   {
140001c94                       int64_t rax_15;
140001c94                       int64_t rdx_4;
140001c94                       rdx_4 = HIGHQ((int128_t)rax_5);
140001c94                       rax_15 = LOWQ((int128_t)rax_5);
140001ca5                       rax_14 = (int128_t)(COMBINE(rdx_4, rax_15) %
rax_4 * 0x3b9aca00)
140001ca5                           / rax_4;
140001ca8                       rcx_18 = COMBINE(rdx_4, rax_15) / rax_4 *
0x3b9aca00;
140001c48                   }
140001c48                   else
140001c48                   {
140001c4d                       rax_14 = rax_5 % 0x16e3600 * 0x3b9aca00 /
0x16e3600;
140001c8b                       rcx_18 = rax_5 / 0x16e3600 * 0x3b9aca00;
140001c48                   }
140001c48
140001caf                   rax_6 = rax_14 + rcx_18;
140001c39               }
140001c39               else
140001c3b                   rax_6 = rax_5 * 0x64;
140001c3b
140001cb5               if (rbx_2 == rax_6)
140001cb5                   break;
140001cb5
140001cb7               if (rbx_2 < rax_6)
140001cb7                   break;
140001cb7
```

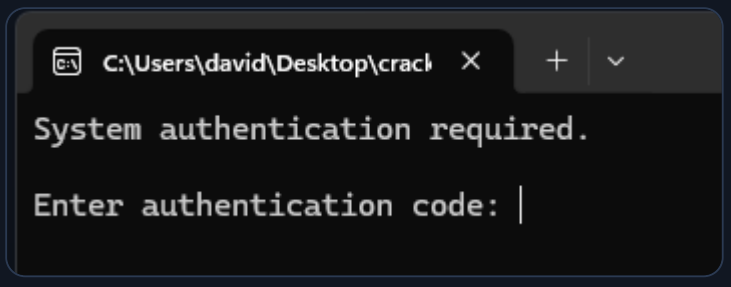
```
140001cbc                int64_t rcx_21 = rbx_2 - rax_6;
140001cbc
140001cc2                if (rcx_21 < 0x4e94914f0001)
140001cc2                {
140001cd8                    int64_t rdx_10 = rcx_21 / 0xf4240;
140001cd8
140001cf3                    if (rdx_10 * 0xf4240 < rcx_21)
140001cf5                        rdx_10 += 1;
140001cf5
140001cfa                    Sleep((uint32_t)rdx_10);
140001cc2                }
140001cc2                else
140001ccb                    Sleep(0x5265c00);
140001c20            }
140001d08        }
140001d08
140001d15        system("cls");
140001d39        /* tailcall */
140001d39        return sub_140002050(std::cout, "Session terminated.\n");
140001ae0    }
```

---

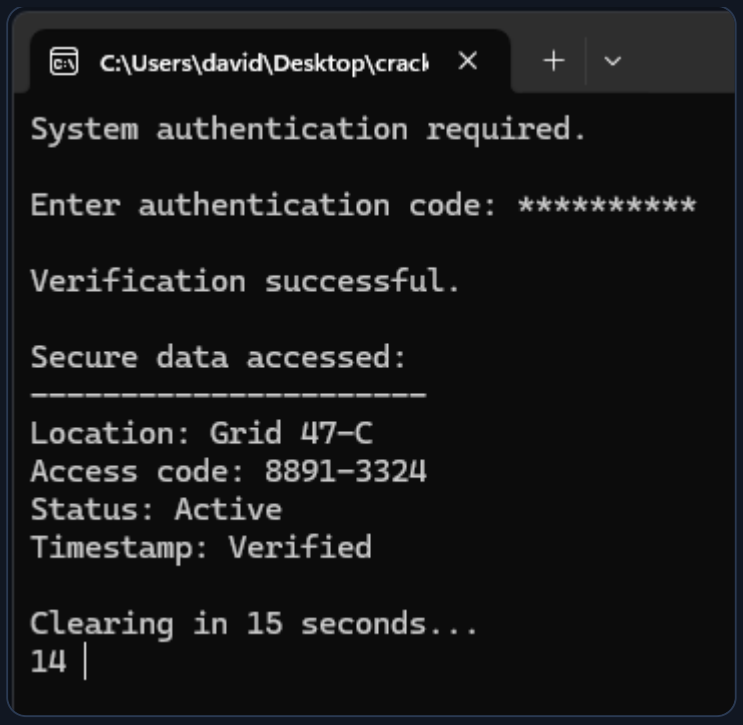
6. Testing The Flag

With the static analysis complete, we try out the flag we found `goodjob123` as well as track the output to see if it matches the findings from the static analysis.

Starting the crackme reveals that we traced the initialization output logic correctly.



Entering the flag we found, `goodjob123` reveals that our static analysis was on point!



---

7. Conclusion

This challenge implements a small *MSVC*-built console program that prompts the user for an authentication code and compares it against a statically constructed flag string. The flag is assembled in a deliberately noisy way using helper routines (`get_goo`, `get_flag`, and a small builder for `"123"`), but semantically it's straightforward: build `"goodjob123"` and compare it to the user's input.

Why this was a good static-analysis exercise:
The crackme forces you to:

- Navigate *CRT* start-up to find `main`.
- Read and interpret *High Level IL / disassembly*.
- Recognise boilerplate *SSO* string initialisation and growth logic.
- Isolate the small "interesting" parts of a long decompiled function and ignore allocator / housekeeping noise.

What I'd improve next time:

- Be more deliberate about identifying and mentally collapsing "standard library" patterns early, so I can focus attention on the real logic sooner.

- Take more structured notes on helper functions (`get_goo`, `get_flag`) up front, since recognising those patterns made the solution fall into place.
- Cross-check string layouts with a minimal C++ std::string model to make the *SSO* fields and offsets even more obvious.

Lessons learned:

- Even when a function looks huge in the decompiler, often 80–90% of it is generic runtime or container management. The actual obfuscation / logic is small and localised.
- Simple "manual obfuscation" (splitting a string across helpers, copying bytes around, using *SSO* internals) is more about intimidating beginners than providing real security.
- Static-only reversing is very achievable for this kind of challenge and it forces you to build a stronger mental model of the code compared to relying on a debugger from the start.

The final solution is the recovered password:

```
goodjob123
```

Appendix A — ASCII Table

ASCII table for reference. Source: https://www.asciitable.com/

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com