CRACKMES.ONE

Santa's Network Admin Portal (S.N.A.P.)

Challenge Author: DonCris

Writeup by scaredandalone

Patch out or just jump past the debugger checks (there are a few, including some with timers).

In the main function, we can find the elf_id generator function, which makes it pretty easy to just create our own key generator.

Figure 1: Key_gen function call

```
loc_7FF6BA091A2B:
call    _ZNSt6chrono3_V212system_clock3nowEv ; std::chrono::_V2::system_clock::now(void)
mov     [rbp+10h+var_60], rax
lea     rdx, [rbp+10h+var_30]
lea     rax, [rbp+10h+var_60]
mov     rcx, rax
call    _ZNSt6chronomiINS_3_V212system_clockENS_8durationIxSt5ratioILx1ELx1000000000EEEES6_EENSt11common_typeIJT0_T1_EE4ty
mov     [rbp+10h+var_20], rax
lea     rdx, [rbp+10h+var_20]
lea     rax, [rbp+10h+var_68]
mov     rcx, rax
call    _ZNSt6chrono8durationIdSt5ratioILx1ELx1000EEEC1IxS1_ILx1ELx1000000000EEvEERKNS0_IT_T0_EE ; std::chrono::duration<d
lea     rax, [rbp+10h+var_50]
mov     rcx, rax
call    _Z15generate_elf_idRKNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE ; generate_elf_id(std::string const&)
mov     [rbp+10h+var_14], eax
mov     eax, [rbp+10h+pbDebuggerPresent]
test    eax, eax
jz      short loc_7FF6BA091A71
```

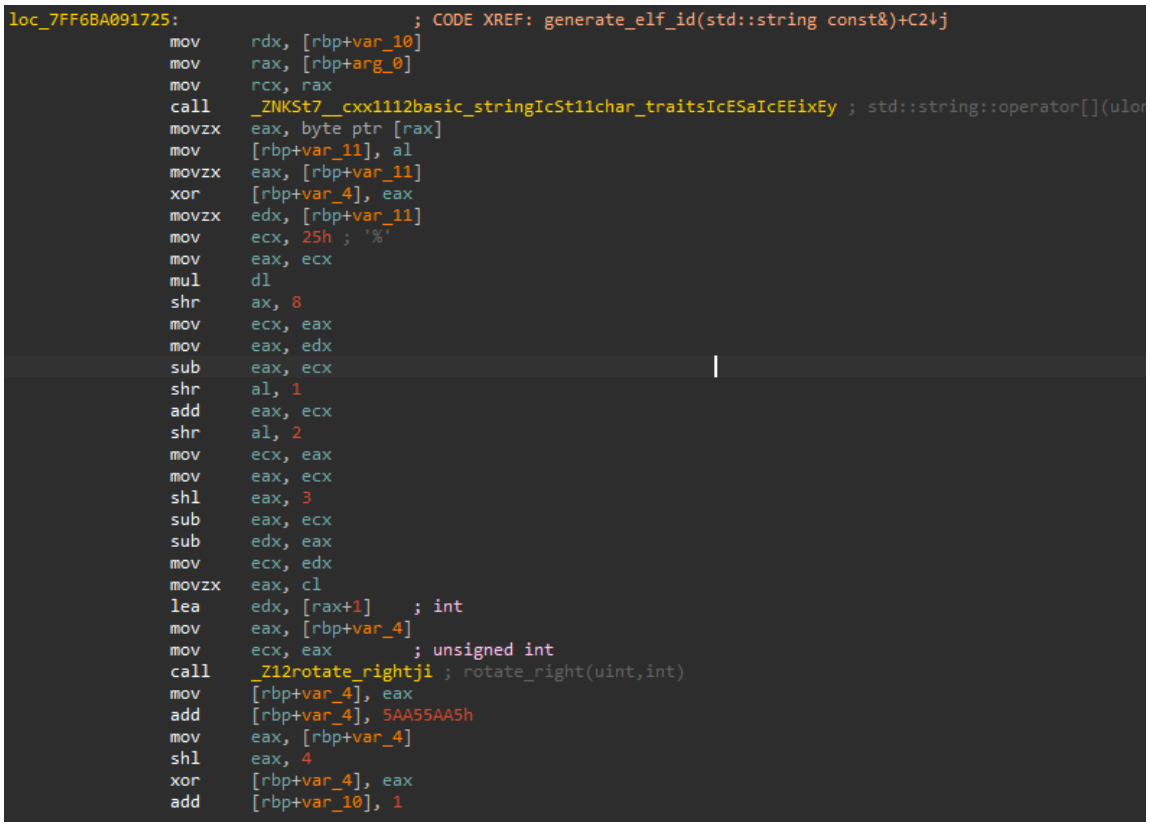
The generate_elf_id function takes in the string (elf name) as input, and does some transformations to create the key.

Three hash constants are used in the generator, the initial seed value 0xCAFEBABE, a constant added to the hash after every bit rotation 0x5AA55AA5 and finally the hash is XOR'd with 0xDEADBEEF before returning.

Figure 2: initial seed value 0xCAFEBABE

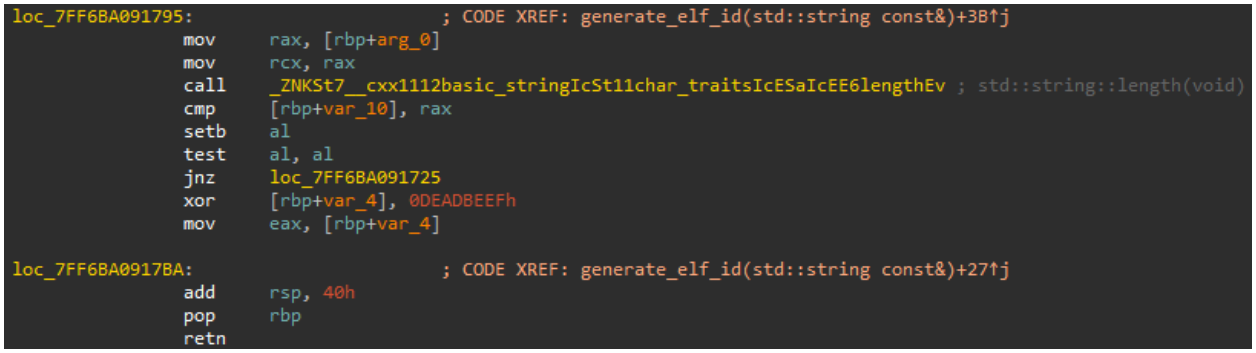
```
loc_7FF6BA091714:                       ; CODE XREF: generate_elf_id(std::string const&)+20↑j
                mov     [rbp+var_4], 0CAFEBABEh
                mov     [rbp+var_10], 0
                jmp     short loc_7FF6BA091795
```

Figure 3: generate_elf_id main loop (0x5AA55AA5 and rotate_right)



```
loc_7FF6BA091725:                       ; CODE XREF: generate_elf_id(std::string const&)+C2↓j
                mov     rdx, [rbp+var_10]
                mov     rax, [rbp+arg_0]
                mov     rcx, rax
                call    _ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEixEy ; std::string::operator[](ulo
                movzx   eax, byte ptr [rax]
                mov     [rbp+var_11], al
                movzx   eax, [rbp+var_11]
                xor     [rbp+var_4], eax
                movzx   edx, [rbp+var_11]
                mov     ecx, 25h ; '%'
                mov     eax, ecx
                mul     dl
                shr     ax, 8
                mov     ecx, eax
                mov     eax, edx
                sub     eax, ecx
                shr     al, 1
                add     eax, ecx
                shr     al, 2
                mov     ecx, eax
                mov     eax, ecx
                shl     eax, 3
                sub     eax, ecx
                sub     edx, eax
                mov     ecx, edx
                movzx   eax, cl
                lea     edx, [rax+1]    ; int
                mov     eax, [rbp+var_4]
                mov     ecx, eax        ; unsigned int
                call    _Z12rotate_rightji ; rotate_right(uint,int)
                mov     [rbp+var_4], eax
                add     [rbp+var_4], 5AA55AA5h
                mov     eax, [rbp+var_4]
                shl     eax, 4
                xor     [rbp+var_4], eax
                add     [rbp+var_10], 1
```

Figure 4: final XOR (0xDEADBEEF)



```
loc_7FF6BA091795:                       ; CODE XREF: generate_elf_id(std::string const&)+3B↑j
                mov     rax, [rbp+arg_0]
                mov     rcx, rax
                call    _ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE6lengthEv ; std::string::length(void)
                cmp     [rbp+var_10], rax
                setb    al
                test    al, al
                jnz     loc_7FF6BA091725
                xor     [rbp+var_4], 0DEADBEEFh
                mov     eax, [rbp+var_4]

loc_7FF6BA0917BA:                       ; CODE XREF: generate_elf_id(std::string const&)+27↑j
                add     rsp, 40h
                pop     rbp
                retn
```

Since we have all the logic, just rewrite the functionality in your favourite programming language.

I used C++, and have included my code below:

```
#include <iostream>
#include <string>
#include <cstdint>



uint32_t rotate_right(uint32_t value, int shift) {
    shift = shift & 31;
    return (value >> shift) | (value << (32 - shift));
}

uint32_t generate_elf_id(const std::string& name) {
    if (name.length() == 0) return 0;

    uint32_t hash = 0xCAFEBABE;

    for (size_t i = 0; i < name.length(); i++) {
        uint8_t c = name[i];

        hash ^= c;

        int rotation = (c % 7) + 1;
        hash = rotate_right(hash, rotation);

        hash += 0x5AA55AA5;
        hash ^= (hash << 4);
    }

    hash ^= 0xDEADBEEF;
    return hash;
}

int main() {
    std::string elf_name;
    std::cout << "[AUTH] Enter Elf Username: ";
    std::getline(std::cin, elf_name);

    uint32_t elf_id = generate_elf_id(elf_name);

    std::cout << "[SYSTEM] Generated Elf ID: " << std::dec << elf_id << "\n\n";

    return 0;
}
```

Now we can generate a valid Elf ID for any Elf username.

(Make sure to use uint32_t to preserve correct bit rotation behavior).