Bakeware

Challenge Description

Grandma had her secret family recipe stolen! All she has left now is a weird and rusty file. Please figure out what happened to the recipe, so we can get back to baking!

Sounds like a rust binary, I don't have much experience reversing rust binaries but how bad could it be...

This was a REV challenge from the Brunner 2025 CTF. We are given the ELF file Bakeware and an encrypted message in the file Grandmas_Secret_Baking_Family_Recipe.enc

Reverse Engineering with Binja and GDB

I did some high-level review static review in Binary Ninja (Binja) and confirmed the binary takes input from a file called `Grandmas_Secret_Baking_Family_Recipe.txt` and encrypts to an output file `Grandmas_Secret_Baking_Family_Recipe.enc` but not sure how the encryption is performed yet but looks like AES.

After running the binary initially it prints out *"Secret recipe not found. Nothing to steal :("*

If we inspect the decompiliation in Binary Ninja this instruction seems to be the culprit for checking some input against this pre-calculated string. If you run it dynamically under GDB we confirm this is a hard code sha256 hash.

```
26 @ 00008d2a  _$LT$$RF$alloc..string.....Digest$GT$::digest::hac6b4adf5efc5232(&var_7d8, &var_408)
27 @ 00008d30  int64_t rax_2 = var_408.q
28 @ 00008d55  uint64_t rax_4 = var_7d8
29 @ 00008d62  int64_t r14 = var_7d0.q
30 @ 00008d76  close(fd)
31 @ 00008d80  int64_t var_968 = r14
32 @ 00008d84  int64_t var_7c8
33 @ 00008d84  if (var_7c8 == 0x40 && bcmp(r14, "502ff05a7b51b76e740b19cc4957ad11…", 0x40) == 0)
```

sha256 hash of my test input in Grandmas_Secret_Baking_Family_Recipe.txt:

```
► 0x55555555cd62 <Bakeware::main+258>    mov    r14, qword ptr [rsp + 0x198]    R14, [0x7fffffffd648] => 0x5555555b8b30 ◂— 'da588e6b2b3e6692eeedcd077d851bfe3d3026b1882ef74f40...'
```

Comparison of my input to the hard-coded hash value:

```
► 0x55555555cd99 <Bakeware::main+313>    call   qword ptr [rip + 0x59f29]    <__memcmp_sse4_1>
        rdi: 0x5555555b8b30 ◂— 'da588e6b2b3e6692eeedcd077d851bfe3d3026b1882ef74f40981c01e551dba2'
        rsi: 0x55555555a49b5 ◂— '502ff05a7b51b76e740b19cc4957ad118897a25becbb87fcb662a14b2e56a5d9Secre
        rdx: 0x40
        rcx: 0x7ffff7d14f67 (close+23) ◂— cmp rax, -0x1000 /* 'H=' */
```

I decided to [patch](#) the binary with `NOP` instructions so we can continue the program execution. Here is the patched binary with `NOP` Instructions to skip over this `CMP` instruction:

```
00008d8a  488d35247c0400     lea     rsi, [rel data_509b5]  {"502ff05a7b51b76e740b19cc4957ad11…"}
00008d91  ba40000000         mov     edx, 0x40
00008d96  4c89f7             mov     rdi, r14
00008d99  ff15299f0500       call    qword [rel bcmp]
00008d9f  90                 nop
00008da0  90                 nop
00008da1  90                 nop
00008da2  90                 nop
00008da3  90                 nop
00008da4  90                 nop
00008da5  90                 nop
00008da6  90                 nop
00008da7  488dbc2448010000   lea     rdi, [rsp+0x148 {var_820}]
```

Now we can run the binary and it will complete execution with any input:

```
┌─[alain@EZTING:~/D/rev_bakeware]─[12:06:36 PM]─[V:appsec]
└─>$ cat Grandmas_Secret_Baking_Family_Recipe.txt
anyvalueyouwant
┌─[alain@EZTING:~/D/rev_bakeware]─[12:06:40 PM]─[V:appsec]
└─>$ ./bakeware_nop
Data exfiltrated to: Grandmas_Secret_Baking_Family_Recipe.enc
┌─[alain@EZTING:~/D/rev_bakeware]─[12:06:45 PM]─[V:appsec]
└─>$
```

We can see a function in Binary Ninja called `Bakeware::get_key_part` which sounds like it may be generating the encryption key. We can inspect it dynamically in GDB and we will have the entire key used for encryption revealed to us piece by piece. Below shows the concatenated key in GDB:

```
► 0x55555555d1df <Bakeware::main+1407>    call   qword ptr [rip + 0x5984b]    <__rust_dealloc>
        rdi: 0x5555555b8b80 ◂— 0x55555554f
        rsi: 1
        rdx: 1
        rcx: 0x5555555b8ae0 ◂— 'OTHellOTotallyStealGoodRecipes!!'
```

At this point we know the binary takes input from a file and encrypts it with this key. We can also determine in Binary Ninja that AES256 encryption is used via the AES rust crate (needed to google as I'm not that familiar with rust)

```
Symbols   aes:
Name                                                                        Address      Section
aes::autodetect::aes_intrinsics::init_get::init_inner::cpuid::h4…          0x00000ca00  .text
aes::autodetect::aes_intrinsics::init_get::init_inner::cpuid_cou…          0x00000ca20  .text
aes::autodetect::aes_intrinsics::init_get::init_inner::hd8075192…          0x000006180  .text
aes::ni::aes256::inv_expanded_keys::h6025cfbcbfed022f.llvm.39699…          0x000007cb0  .text
aes::soft::fixslice::aes256_encrypt::hc9bb03be56ce6e82                     0x00000aa80  .text
aes::soft::fixslice::aes256_key_schedule::hda8c6ed16e648178                0x00000a170  .text
aes::soft::fixslice::bitslice::hdf31250f4ef1210f                           0x00000b670  .text
```

The last piece we would need would be to know the encryption mode and initialization vector (IV) if possible. I took an educated guess as part of my analysis that the IV was `1234567890123456`.

Binary Ninja view suggesting this could be possible IV:

```
if (aes::autodetect::aes_intrinsics::STORAGE::hc8673b1237a96b79_1 != 1 && (zx.d(aes::autodetect
    aes::soft::fixslice::aes256_key_schedule::hda8c6ed16e648178(&var_408, var_400)
memcpy(&var_7d8, &var_408, 0x3c0)
int128_t var_418
__builtin_strncpy(dest: &var_418, src: "1234567890123456", n: 0x10)
```

Assumptions and Solution

We have enough known values to decrypt the flag now
Cipher: Given
Key: OTHelloTotallyStealGoodRecipes!!
Encryption: AES
Mode: Appears to be maybe CBC, this is my working assumption.

A well crafted prompt from an AI should whip-up a potential solver script now if you use these known values.

Script

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

# Read the ciphertext from file
with open("Grandmas_Secret_Baking_Family_Recipe.enc", "rb") as f:
    ciphertext = f.read()

# Known values
key = b'OTHellOTotallyStealGoodRecipes!!'  # 32 bytes for AES-256
iv = b'1234567890123456'  # 16-byte IV

# Decrypt
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted = cipher.decrypt(ciphertext)

# Try PKCS#7 unpadding first
try:
    plaintext = unpad(decrypted, 16)
    print("✅ Decrypted and unpadded (PKCS#7):", plaintext.decode())
except ValueError:
    # If padding fails, try stripping nulls
    print("⚠️ PKCS#7 padding failed. Trying null-stripping.")
    plaintext = decrypted.rstrip(b'\x00')
    try:
        print("Decrypted (null-stripped):", plaintext.decode())
    except:
        print("Could not decode plaintext.")
```

Decrypted Message and Flag:

```
python solver.py
✅ Decrypted and unpadded (PKCS#7): Oh, the good ol' days of baking and eatin' cookies and cake all day long. I finally think you are old enough for me to share my favourite recipe with you.
Please keep this safe for generations to come.

The recipe for the perfect brunsviger:
- 20g yeast
- 1dl milk
- 40g butter
- 1 egg
- 40g sugar
- 0.5 tsp salt
- 250g flour

To bake it you simply just:
brunner{Gr4ndm4_sh0u1d_R34lL7_l34rn_b3tt3r_0ps3c}
```