KEYEDUP WRITEUP

Category: Reverse Engineering
Points 200
Challenge Author: mburk4
CTF: NICCTF 2026

By : Lelkiramkeel

Desc:

An internal license verification utility from NIC has been leaked. It prompts for a short key and performs a series of checks before deciding whether to grant access.
Analyze the binary, determine what it is really validating, and recover the correct key to obtain the flag.

Solution:

Running a basic file type recognition, we find that the file is a 64-bit ELF LSB pie executable, stripped, and dynamically linked.

```
$ file keyedup
keyedup: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=fd5acab516824089672773d8be14e040682fd972, for
GNU/Linux 3.2.0, stripped

$ ./keyedup
Enter license key: 'test'
Invalid key
```

.

Using **Ghidra** we to decompile the file we get that the input undergoes a series of encryption and compared against some hardcoded hex

```
61 58 47 e5 bd 34 1b 92 f9 00 8f dd 7d 8c 53 1a
```

Futher analysis, reveals the following decompiled sudo C code:

```
void FUN_001012d0(long param_1,long param_2,long param_3)
```

```
{
  byte bVar1;
  char cVar2;
  long lVar3;
  char cVar4;

  if (param_3 != 0) {
    cVar4 = '\0';
    lVar3 = 0;
    do {
      bVar1 = (char)lVar3 + 0x42U ^ *(byte *)(param_1 + lVar3);
      cVar2 = (bVar1 << 3 | bVar1 >> 5) + cVar4;
      cVar4 = cVar4 + '\a';
      *(char *)(param_2 + lVar3) = cVar2;
      lVar3 = lVar3 + 1;
    } while (param_3 != lVar3);
  }
  return;
}
```

As it can be seen the following layers of encryption:

- **XOR operation**: It takes your input byte and XORs it with a rolling value:
- **Bitwise Rotation (ROL 3):** It performs a left circular shift (rotation) by 3 bits.
- **Addition with Offset:** It adds a rolling "salt" value (`cVar4`) that starts at `0` and increases by `7` (`\a` is the bell character, which is hex `0x07`) every iteration

Exploit

The following python script attempts to reverse the encryption layer and print out the correct license key

```
# the 0xFF is to maintain 8 bits as in c
def solver():
   target = [
       0x61, 0x58, 0x47, 0xe5, 0xbd, 0x34, 0x1b, 0x92,
       0xf9, 0x00, 0x8f, 0xdd, 0x7d, 0x8c, 0x53, 0x1a
   ]

   key = ""
   salt = 0
   # Reversing the encryption method
   for i in range(16):
```

```
        # reverse the salt addition last_byte -salt
        val = (target[i] - salt) & 0xFF

        # reverse ROL3 by performing ROR3
        val = ((val >> 3) | (val << 5)) & 0xFF

        # reverse XOR
        char_code = val ^ (i + 0x42) & 0xFF
        key += chr(char_code)

        # update the salt
        salt = (salt + 7) & 0xFF

    return key

print(f"{solver()}")
```

The result is the license:

```
        #Output: nic_reverse_king
```

The following Bash script extracts the flag

```
#!/usr/bin/env bash


#this bash script extracts the flag and places it in a flag.txt file
if [ ! -f 'exploit.py' ]; then
    echo "ERROR: file: exploit.py: not in working directory"

elif [ ! -f 'keyedup' ] && [ ! -x 'keyedup' ]; then
    echo "ERROR: file: keyedup: not in working directory OR no 'x' in file
permissions"

else
    flag=$( python3 exploit.py | ./keyedup )

    echo "${flag}" > flag.txt
    echo "Flag: ${flag} "
fi
```

The Flag is

```
#Flag: NICCTF26{Z25pa19lc3JldmVyX2Npbg==}
```

Conclusion:

This challenge demonstrates binary reverse engineering techniques including string extraction, disassembly analysis, and reversing custom cryptographic transformations.