

# Understanding BlackMatter's API Hashing

By Jan Gru

Archived: 2026-04-06 01:06:18 UTC

For the calculation of the API hash each character is added up one by another. In each iteration a seeded ROR-13-operation is performed, as the following figure illustrates.

```

*****
*                               *
*                               *
*****
uint __fastcall calcFuncHash(undefined4...
    EAX:4    <RETURN>
    ECX:4    param_1
    EDX:4    param_2
    byte *   Stack[0x4...funcName      XREF[1]: 004010a0(R)
    uint     Stack[0x8...modHash       XREF[1]: 0040109d(R)
    undefined1 HASH:5ff6...curChar
    calcFuncHash                        XREF[1]: resolveHashedImport

00401096 55          PUSH    EBP
00401097 8b ec       MOV     EBP,ESP
00401099 52          PUSH    param_2
0040109a 56          PUSH    ESI
0040109b 33 c0       XOR     EAX,EAX
0040109d 8b 55 0c    MOV     param_2,dword ptr [EBP + modHash]
004010a0 8b 75 08    MOV     ESI,dword ptr [EBP + funcName]

LAB_004010a3                                XREF[1]: 004010b1(j)
004010a3 ac          LODSB   ESI
004010a4 80 c6 61    ADD     param_2,0x61
004010a7 80 ee 61    SUB     param_2,0x61
004010aa c1 ca 0d    ROR     param_2,0xd
004010ad 03 d0      ADD     param_2,EAX
004010af 85 c0      TEST   EAX,EAX
004010b1 75 f0      JNZ    LAB_004010a3
004010b3 8b c2      MOV     EAX,param_2
004010b5 5e        POP     ESI
004010b6 5a        POP     param_2
004010b7 5d        POP     EBP
004010b8 c2 08 00    RET    0x8

```

Figure 3: Algorithm to calculate the API hash

Because of the fact, that the hash of the module name is used as a seed, a two step process has to be employed to construct the final API hash for a single function.

First, the module name is hashed in a similar manner with a seed of 0. This happens in the function at `004010bb`, which is not shown here. It is looped over the characters, which are transformed to lower case. In each iteration a rotation by 13 bits of the dword value resulting from the previous iteration is performed and the current character value is added. This leads to the following Python implementation:

```

def calc_mod_hash(modname):
    mask = 0xFFFFFFFF
    h = 0

```

```
for c in modname + "\x00":
    cc = ord(c)
    if (0x40 < cc and cc < 0x5b):
        cc = (cc | 0x20) & mask
    h = (h >> 0xd) | (h << 0x13)
    h = (h + cc) & mask

return h
```

The resulting hash of the module name is then used as a seed for the similar but simpler function presented at fig. 3, which finally calculates the actual function hash. The following Python code shows the logic found in this function at `00401096` :

```
def calc_func_hash(modhash, funcname):
    mask = 0xFFFFFFFF
    h = modhash
    for c in funcname + "\x00":
        cc = ord(c)
        h = (h >> 0xd) | (h << 0x13)
        h = (h + cc) & mask

    return h
```

**Note:** It is important to add the nullbyte, so that for a function name of  $n$  characters,  $n+1$  ROR-operations are performed.<sup>Z</sup>

In summary this leads to the following calculation of a function hash as it is used by *BlackMatter*:

```
def get_api_hash(modname, funcname):
    return calc_func_hash(calc_mod_hash(modname), funcname)
```

Let's test it:

```
mn = "kernel32.dll"
fn = "GetProcAddress"
print(hex(get_api_hash(mn, fn)))

mn = "kernel32.dll"
fn = "LoadLibraryA"
print(hex(get_api_hash(mn, fn)))
```

```
#+Result
: 0xbb93705c
```

: 0x27d05eb2

Indeed, both hashes can be found in the binary, as fig. 3 shows:

```
26 |  
27 | if (LOADLIBRARY == (int *)0x0) {  
28 |     LOADLIBRARY = (int *)0x27d05eb2;  
29 |     LOADLIBRARY = resolveHashedImport(0x27d05eb2);  
30 | }  
31 | if (GETPROCADDRESS == (int *)0x0) {  
32 |     GETPROCADDRESS = (int *)0xbb93705c;  
33 |     GETPROCADDRESS = resolveHashedImport(0xbb93705c);  
34 | }
```

Figure 4: Function hashes of LoadLibraryA and GetProcAddress

Actually only  $0x5d6015f \wedge 0x22065fed$ , which results in  $0x27d05eb2$  can be found, since all API hashes are stored XORed with  $0x22065fed$  and are XORed again with this value before a comparison with the calculated hash.

---

Source: <https://blog.digital-investigations.info/2021-08-05-understanding-blackmatters-api-hashing.html>