



Strings deobfuscation in pseudocode with microcode manipulation

04/07/2023

Case Study



■ Triangle DB

- iOS malware

■ Analysis

- <https://securelist.com/triangledb-triangulation-implant/110050/>

■ Sample

- fd9e97cfb55f9cfb5d3e1388f712edd952d902f23a583826ebe55e9e322f730f

Case Study



```
id __cdecl -[CRConfig getBuildArchitecture](CRConfig *self, SEL a2)
{
    return +[CRConfig unmungeHexString:](&OBJC_CLASS__CRConfig, "unmungeHexString:", CFSTR("61137e487c19"));
}
```



arm64e

Microcode manipulation

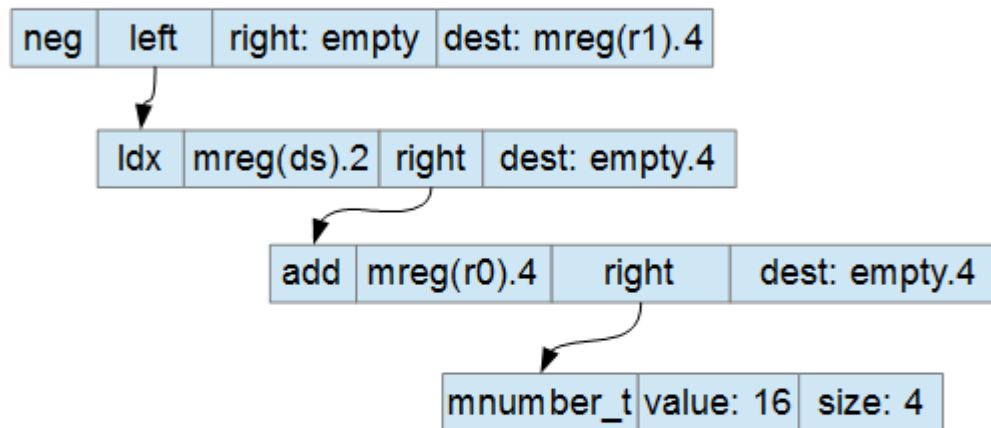


- **Intermediate representation between assembly and C**
 - Write 1 assembly → microcode per architecture
 - Apply the same optimizations to microcode for all architectures
- **Called pcode in Ghidra**
 - Java API :(

Micro-instruction in IDA



- **1 opcode, multiple operands**
 - Instructions can also be operands



Writing the plugin



- **optinsn_t plugin to optimize individual instructions**
- **Wait for maturity MMAT_CALLS**
- **Check if the instruction is a call to unmungeHexString**
- **Fetch the string**
- **Decode it**
- **Replace the microcode**

Finding the call



- **Objective-C calls are done with objc_msgSend and a string**
 - IDA is smart enough to replace it with the direct function call
 - But we are before this optimization pass

```
id __cdecl -[CRConfig getBuildArchitecture](CRConfig *self, SEL a2)
{
    return +[CRConfig unmungeHexString:](&OBJC_CLASS__CRConfig, "unmungeHexString:", CFSTR("61137e487c19"));
}
```

```
objc_msgSend = ida_name.get_name_ea(idaapi.BADADDR, "_objc_msgSend")
unmunge = b'unmungeHexString:'

class UnmungeHexStrings(optinsn_t):
    def func(self, blk: mblock_t, ins: minsn_t) -> bool:
        if blk.mba.maturity == MMAT_CALLS:
            call, mop = ins.find_ins_op()

            # Check that we have a call to objc_msgSend
            if not call: return 0
            if not call.l.t == mop_v: return 0
            if not call.l.g == objc_msgSend: return 0

            # Check args type and that it is unmungeHexString
            f = call.d.f
            if not len(f.args) == 3: return 0
            if not f.args[1].t == mop_a or not f.args[2].t == mop_a: return 0
            if not get_strlit_contents(f.args[1].a.g) == unmunge: return 0

            # Get string data from CFSTR
            addr = get_qword(f.args[2].a.g + 16)
            h = bytes.fromhex(get_strlit_contents(addr).decode("ascii"))

            # Decode the string
            s = decode(h)

            # Replace the string
            ins.l._make_strlit(s)
            return 1
        return 0
```


Result



```
id __cdecl -[CRConfig getBuildArchitecture](CRConfig *self, SEL a2)
{
    return "arm64e";
}
```

More examples



■ Hexrays vs Obfuscating compiler

- <https://hex-rays.com/blog/hex-rays-microcode-api-vs-obfuscating-compiler/>

■ 7 days to lift

- <https://blog.ret2.io/2020/07/22/ida-pro-avx-decompiler/>

■ d810

- <https://eshard.com/posts/d810-deobfuscation-ida-pro>
- <https://eshard.com/posts/D810-a-journey-into-control-flow-unflattening>



■ SDK Hexrays:

- <https://hex-rays.com/products/decompiler/manual/sdk/index.shtml>

■ VM used by microcode

- <https://hex-rays.com/products/decompiler/manual/sdk/vmpage.shtml>

■ Decompiler Internals

- <https://i.blackhat.com/us-18/Thu-August-9/us-18-Guilfanov-Compiler-Internals-Microcode-wp.pdf>
- <https://www.youtube.com/watch?v=T-YkhNElvng>

■ Microcode in pictures:

- <https://hex-rays.com/blog/microcode-in-pictures/>



<https://www.linkedin.com/company/synacktiv>

<https://twitter.com/synacktiv>

Our publications : <https://synacktiv.com>